NetApp

Towards securing the Internet of Things with QUIC

Lars Eggert
Technical Director, Networking
2020-2-23

QUIC: a fast, secure, evolvable transport protocol for the Internet

- **Fast** — **better user experience** than TCP/TLS for HTTP and other content
- **Secure** — **always-encrypted** end-to-end security, resist pervasive monitoring
- **Evolvable** — prevent network from ossifying, **can deploy new versions** easily
- **Transport** — **support TCP semantics & more** (realtime media, etc.)
  provide better abstractions, avoid known TCP issues

NetApp

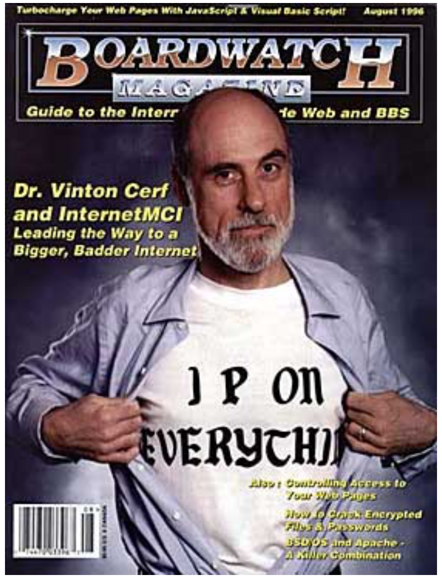
What motivated QUIC?

NetApp

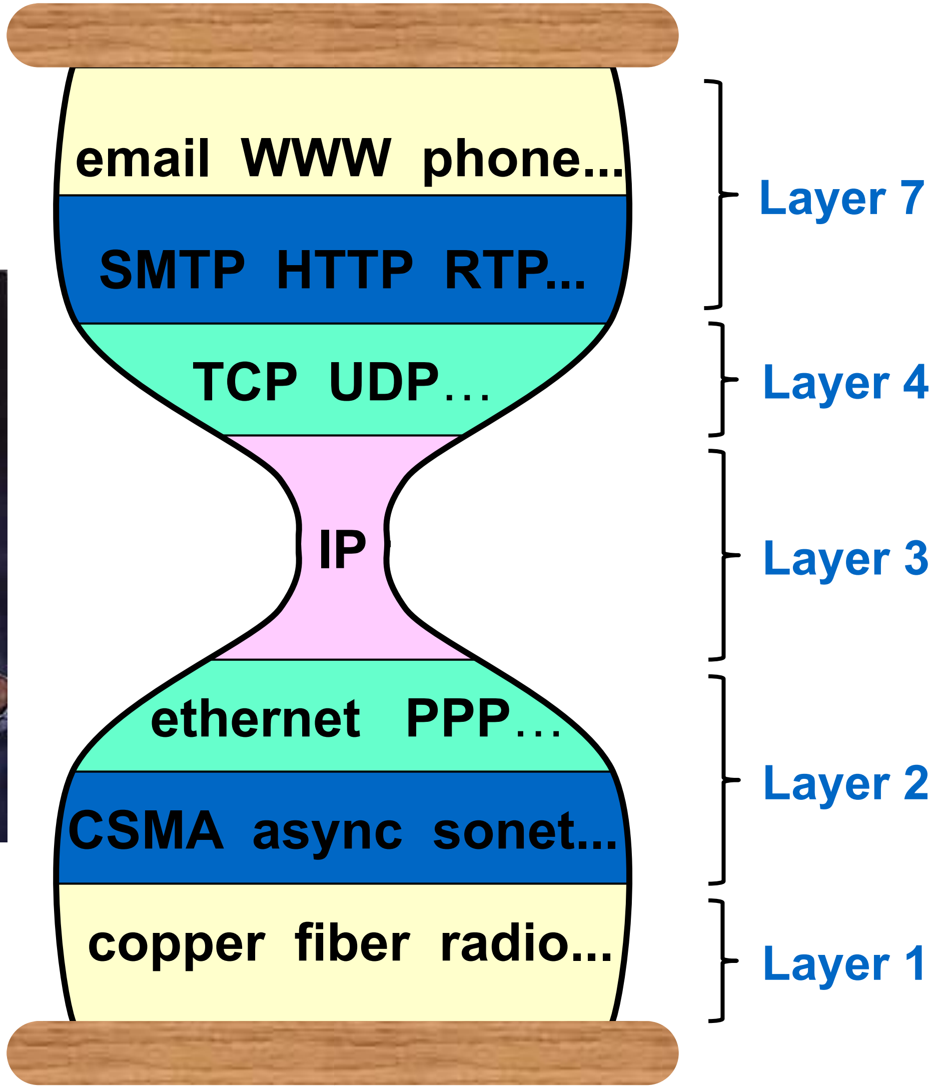
The Internet hourglass

Classical version

- Inspired by OSI “seven-layer” model
  - Minus presentation (6) and session (5)
- “IP on everything”
  - All link tech looks the same (approx.)
- **Transport layer** provides communication abstractions to apps
  - Unicast/multicast
  - Multiplexing
  - Streams/messages
  - Reliability (full/partial)
  - Flow/congestion control
  - …



Boardwatch Magazine, Aug. 1994.



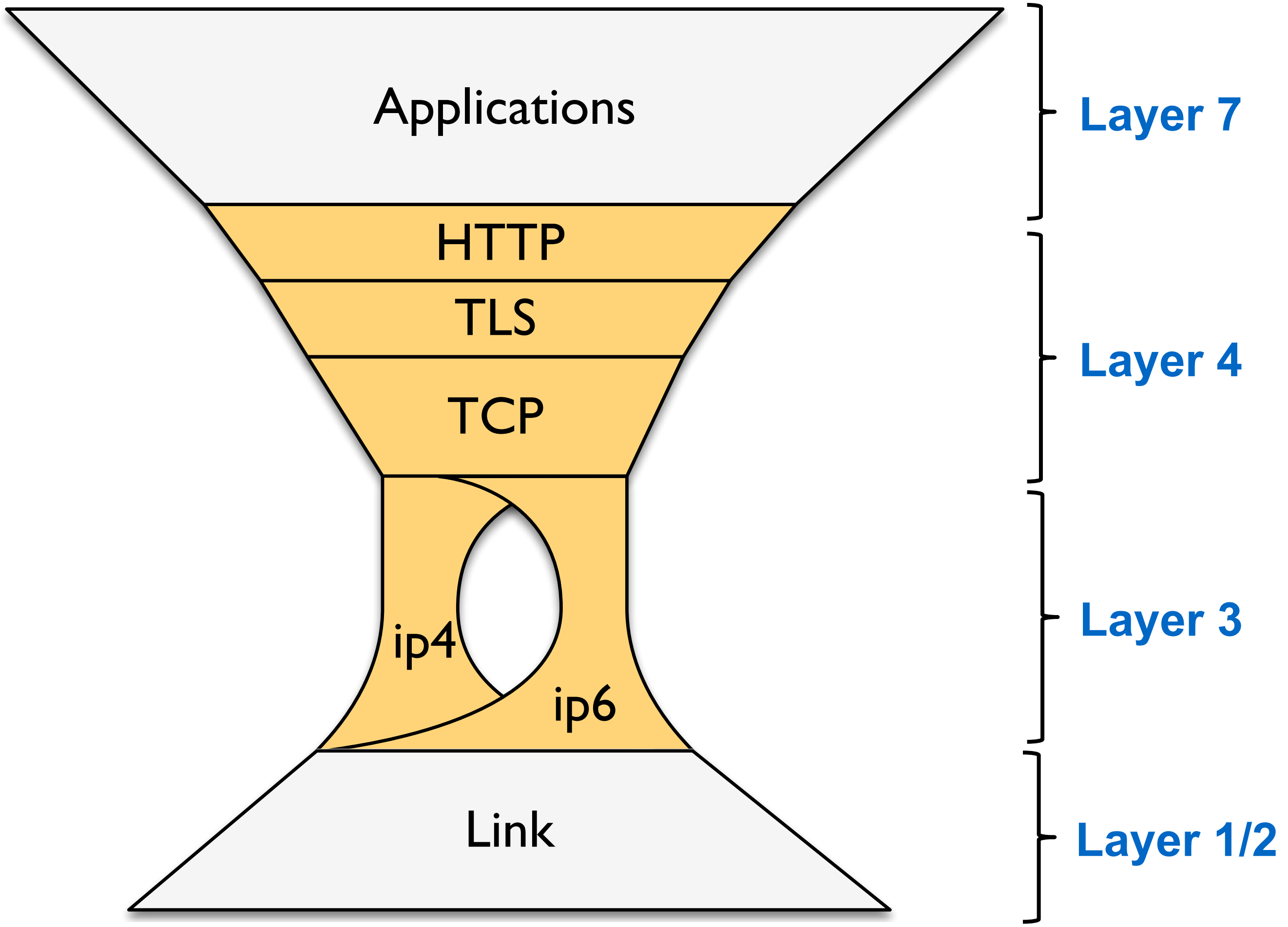
Steve Deering. Watching the Waist of the Protocol Hourglass. Keynote, IEEE ICNP 1998, Austin, TX, USA. http://www.ieee-icnp.org/1998/Keynote.ppt

NetApp

The Internet hourglass

2015 version (ca.)

- The waist has split: **IPv4** and **IPv6**
- **TCP** is drowning out UDP
- **HTTP** and **TLS** are *de facto* part of transport
- Consequence: **web apps** on IPv4/6

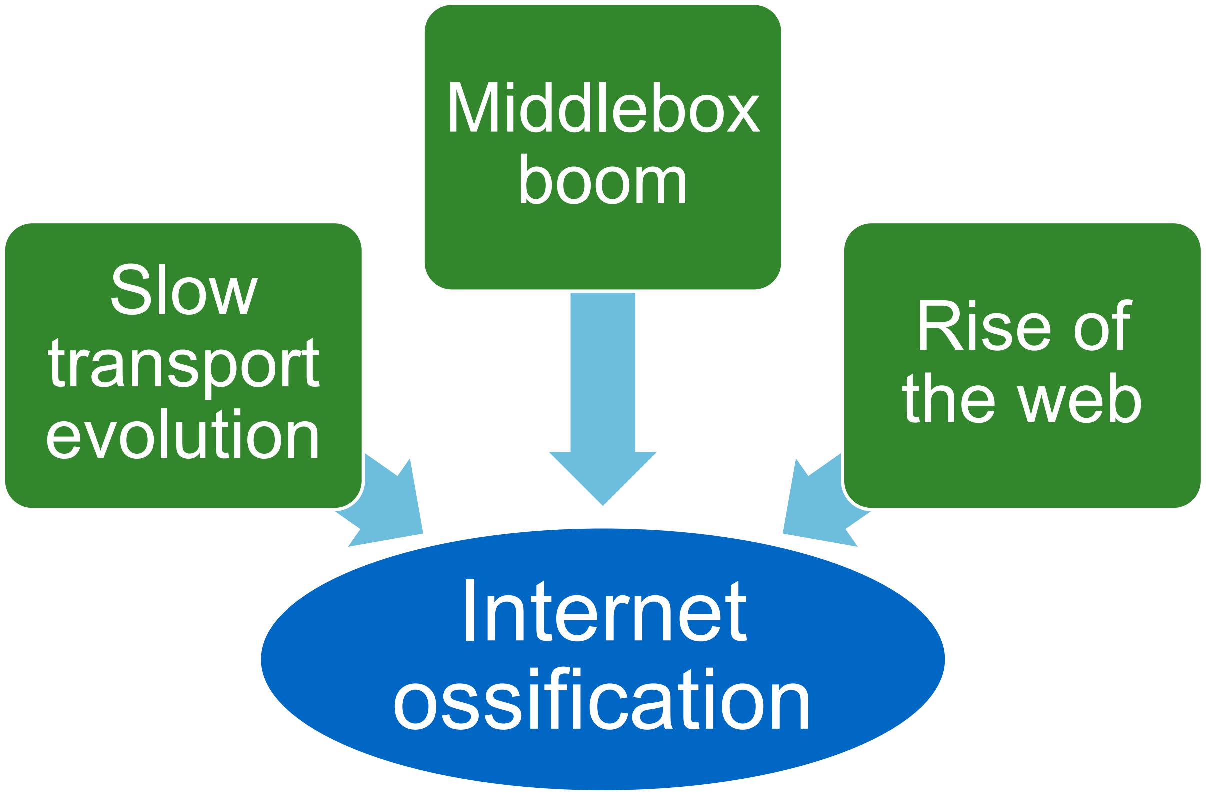


B. Trammell and J. Hildebrand, "Evolving Transport in the Internet," in *IEEE Internet Computing*, vol. 18, no. 5, pp. 60-64, Sept.-Oct. 2014.

NetApp

What happened?

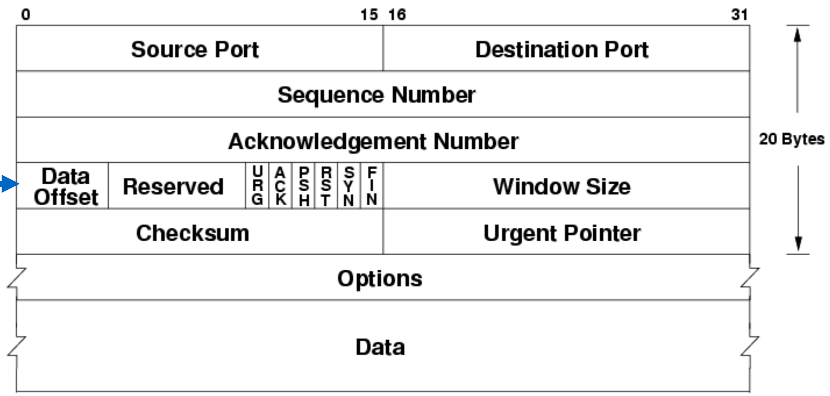
- **Transport slow to evolve** (esp. TCP)
  - Fundamentally difficult problem
- **Network made assumptions** about what (TCP) traffic looked like & how it behaved
- Tried to "help" and "manage"
  - TCP "accelerators" & firewalls, DPI, NAT, etc.
- **The web happened**
  - Almost all content on HTTP(S)
  - Easier/cheaper to develop for & deploy on
  - Amplified by mobile & cloud
  - Baked-in client/server assumption



NetApp

TCP is not aging well

- **We're hitting hard limits** (e.g., TCP option space)
  - 40B total (15 * 4B - 20)
  - Used: SACK-OK (2), timestamp (10), window Scale (3), MSS (4)
  - Multipath needs 12, Fast-Open 6-18…
- **Incredibly difficult to evolve**, c.f. Multipath TCP
  - New TCP must look like old TCP, otherwise it gets dropped
  - TCP is already very complicated
- **Slow upgrade cycles** for new TCP stacks (kernel update required)
  - Better with more frequent update cycles on consumer OS
  - Still high-risk and invasive (reboot)
- **TCP headers not encrypted** or even authenticated – middleboxes can still meddle
  - TCP-MD5 and TCP-AO in practice only used for (some) BGP sessions

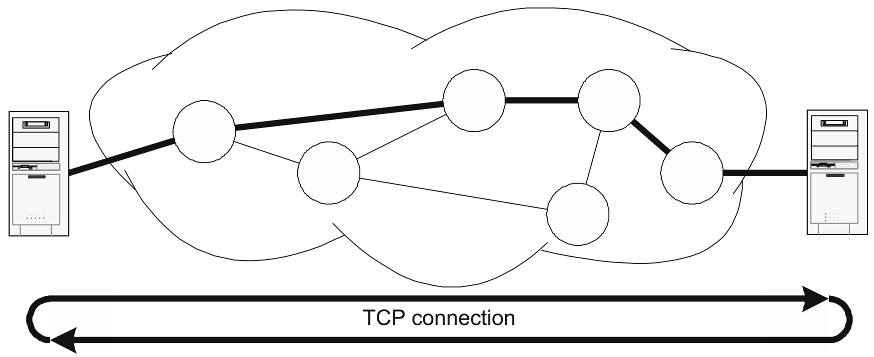


By Ere at Norwegian Wikipedia (Own work) [Public domain], via Wikimedia Commons

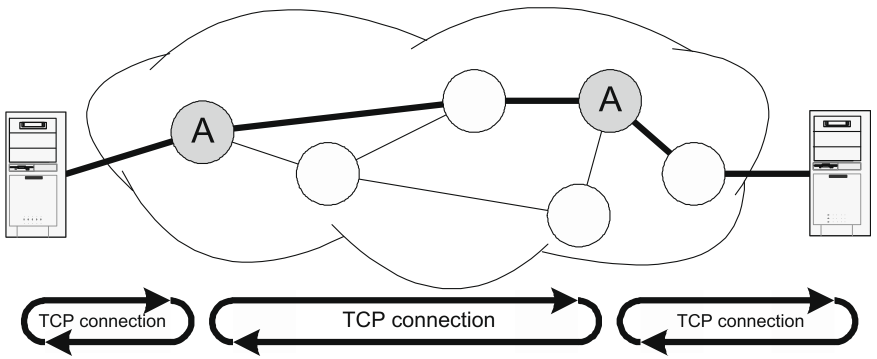
NetApp

Middleboxes meddle

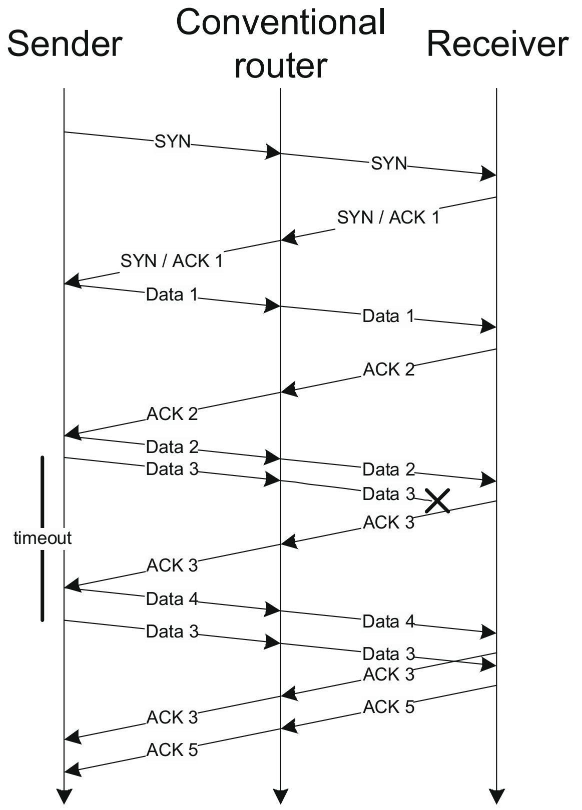
Example: TCP accelerators



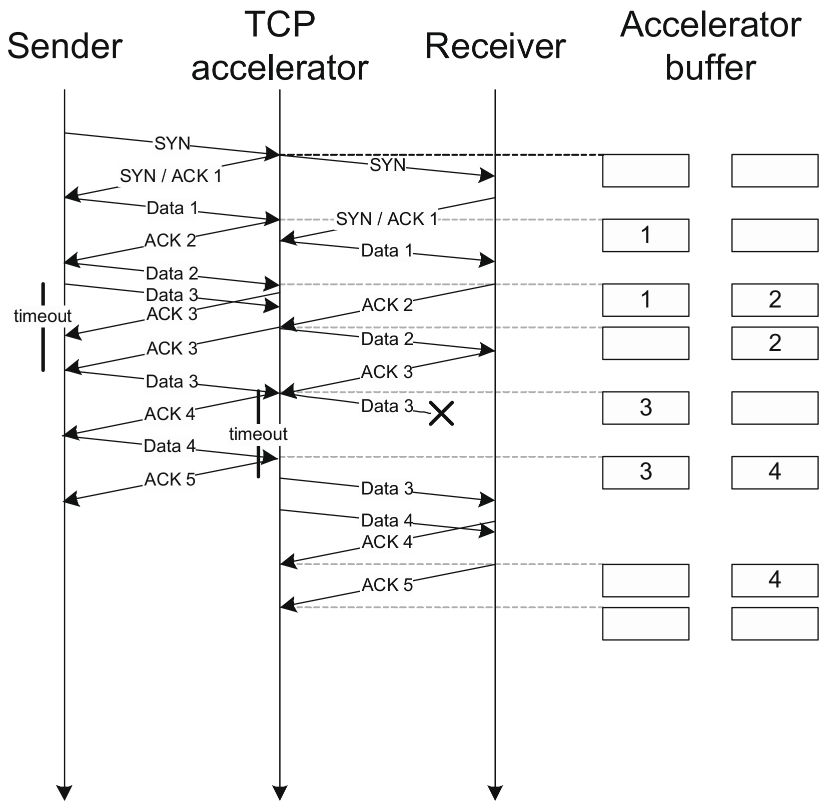
(a) Conventional TCP Connection



(b) Accelerated TCP Connection



(a) Conventional TCP Connection

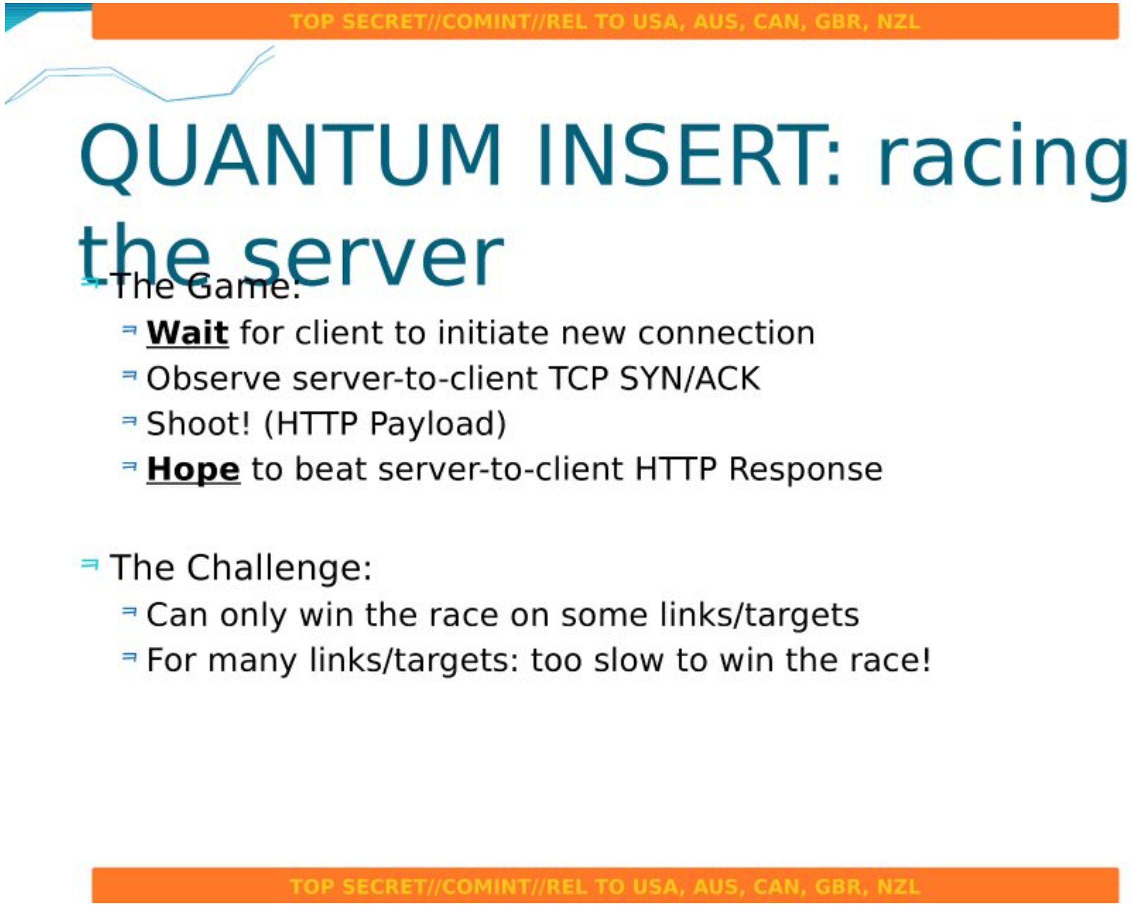


(b) Accelerated TCP Connection

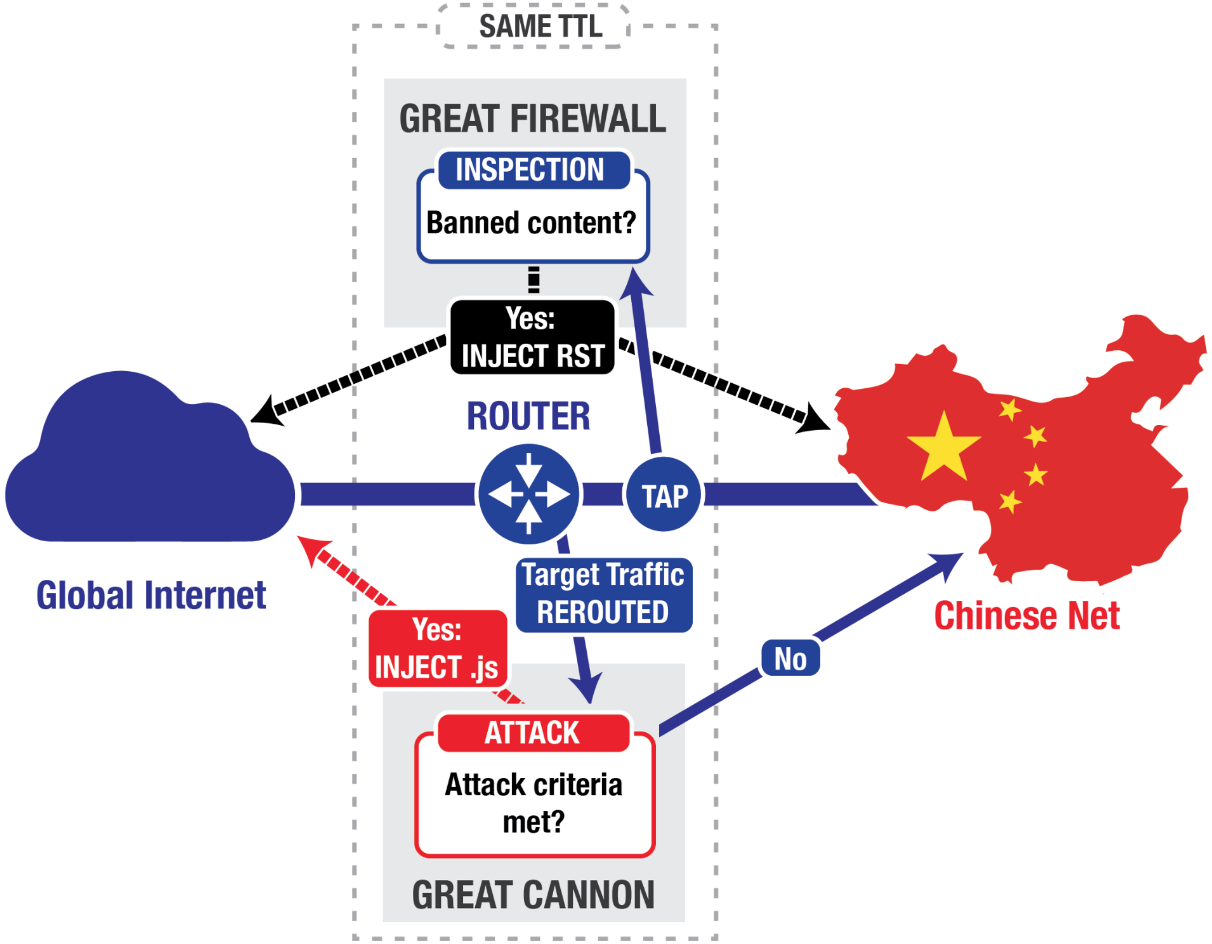
Sameer Ladiwala, Ramaswamy Ramaswamy, and Tilman Wolf. Transparent TCP acceleration. Computer Communications, Volume 32, Issue 4, 2009, pages 691-702.

Middleboxes meddle

Example: Nation states attacking end users or services



QFIRE Pilot Lead. NSA/Technology Directorate. QFIRE pilot report. 2011.



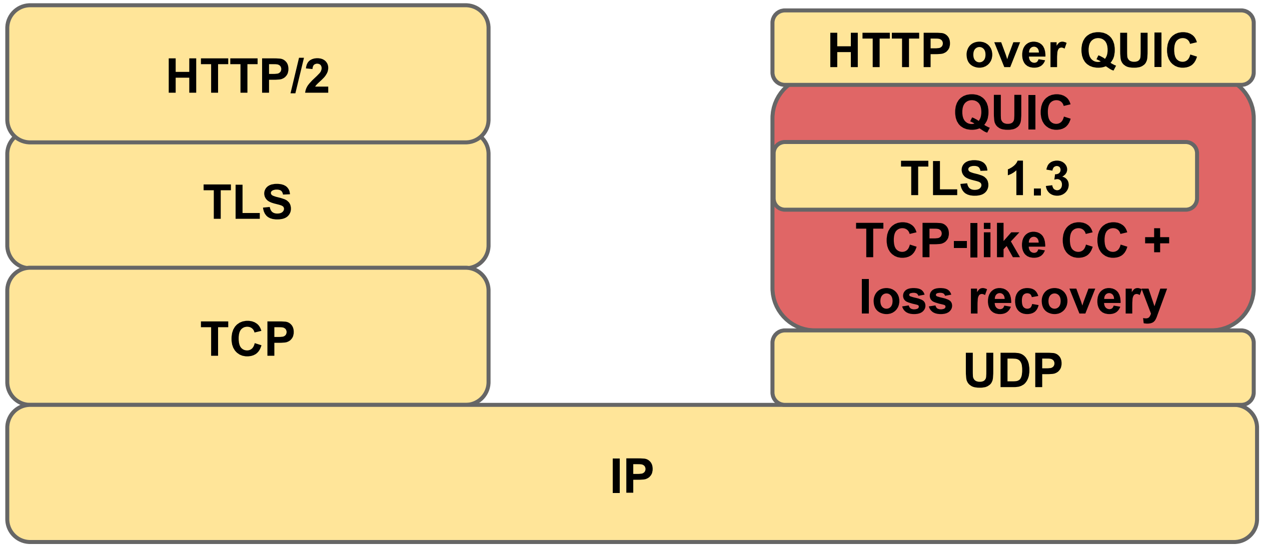
B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson. An Analysis of China's "Great Cannon". 5th USENIX FOCI Workshop, 2015.

NetApp

QUIC components

QUIC in the stack

- Integrated transport stack on top of UDP
- Replaces TCP and some part of HTTP; reuses TLS-1.3
- Initial target application: HTTP/2
- Prediction: many others will follow



J. Iyengar. QUIC Tutorial A New Internet Transport/ IETF-98 Tutorial, 2017.

Why UDP?

UDP

- TCP hard to evolve
- Other protocols blocked by middleboxes (SCTP, etc.)
- **UDP is all we have left**
- Not without problems!
  - Many middleboxes ossified on “UDP is for DNS”
  - Enforce short binding timeouts, etc.
  - Short-term issue with hardware NIC offloading
- Also, benefits
  - Can deploy in userspace (no kernel update needed)
  - Can offer alternative transport types (partial reliability, etc.)

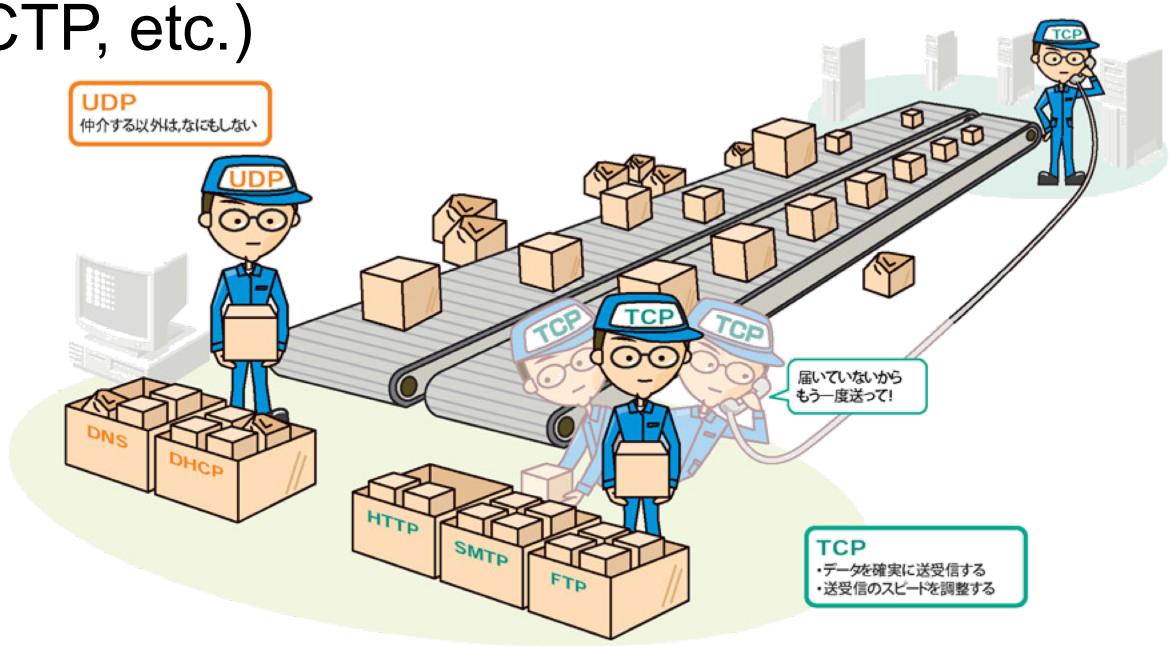


Image from http://itpro.nikkeibp.co.jp

NetApp

Why congestion control?

CC

- Functional CC is **absolute requirement** for operation over real networks
  - UDP has no CC
- First approach: **take what works for TCP, apply to QUIC**
- Consequence: need
  - Segment/packet numbers
  - Acknowledgments (ACKs)
  - Round-trip time (RTT) estimators
  - etc.
- Not an area of large innovation at present
  - This will change
  - <your PhD goes here>

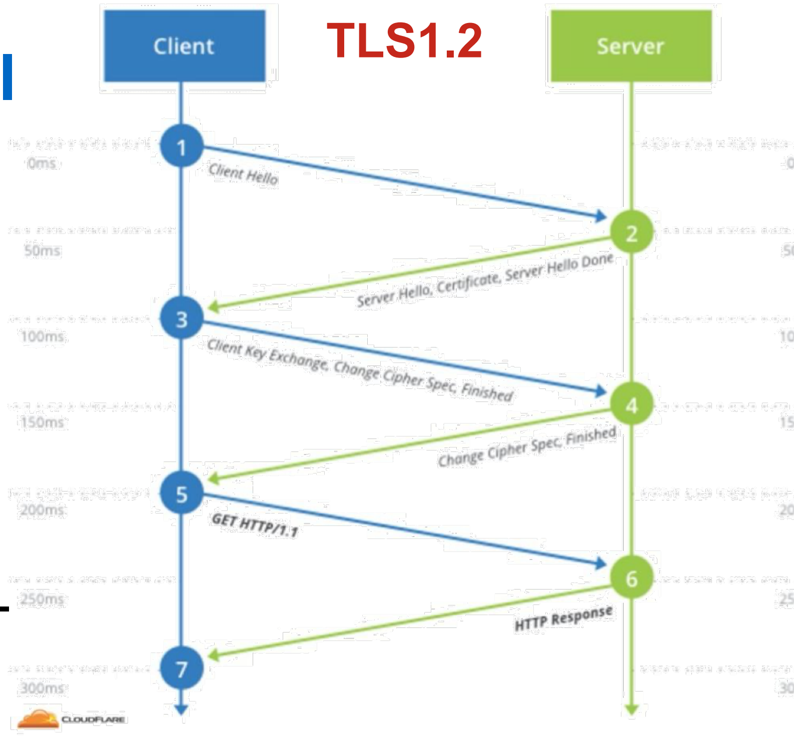
Image from People's Daily, http://people.cn/

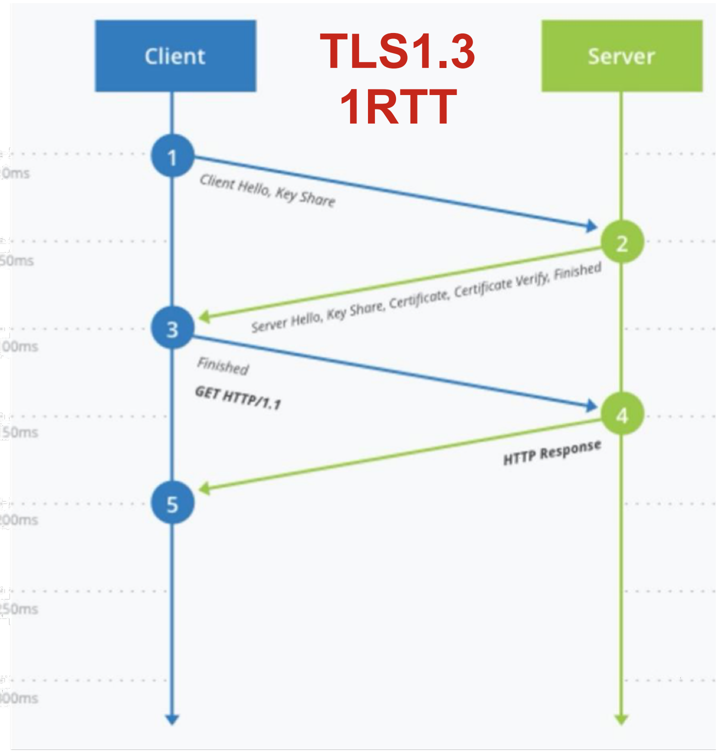
NetApp

Why transport-layer security (TLS)?

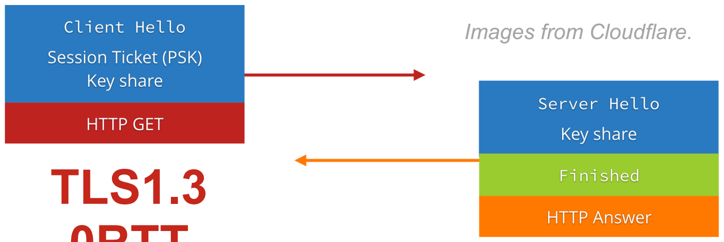
TLS

- **End-to-end security is critical**
  - To protect users
  - To prevent network ossification
- TLS is very widely used
  - Can leverage all community R&D
  - Can leverage the PKI
- **Don't want custom security** – too much to get wrong
  - Even TLS keeps having issues
  - But TLS 1.3 removes a lot of cruft
- And benefit from new TLS features
  - E.g., 0-RTT handshakes (inspired by gQUIC-crypto)





Images from Cloudflare.

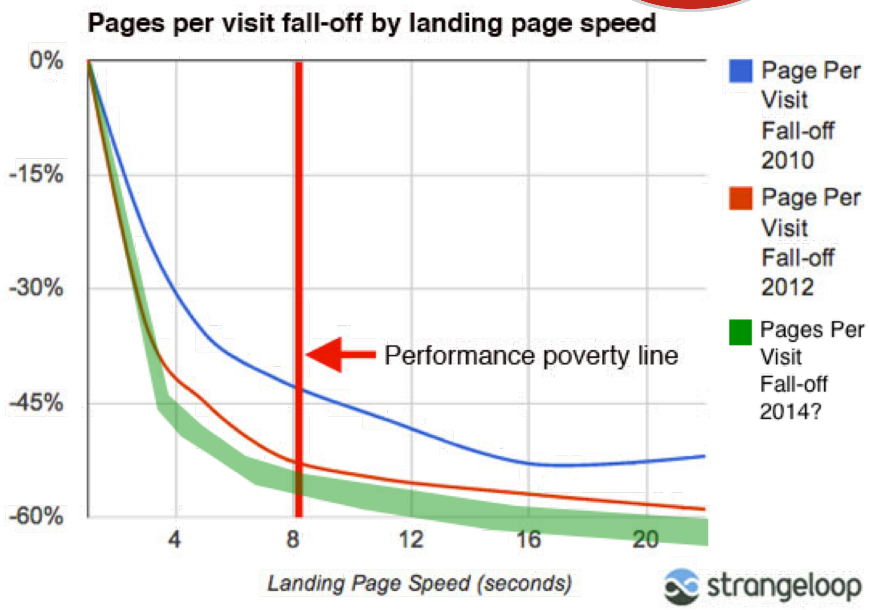


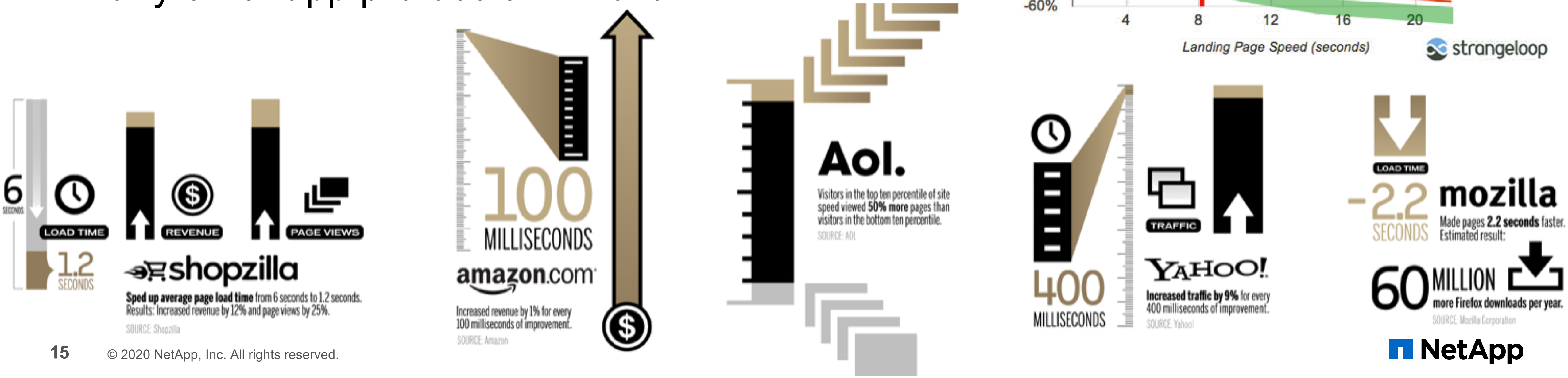
TLS1.3
0RTT

NetApp

Why HTTP?

- Because that's where the **impact** is
  - Web industry incredibly interested in improved UE and security
- Rapid update cycles for browsers, servers, CDNs, etc.
  - Can deploy and update QUIC quickly
- Many other app protocols will follow





NetApp

QUIC

Selected aspects

NetApp

Minimal network-visible header

- With QUIC, the network sees:
  - Packet **type** (partially obfuscated)
  - QUIC **version** (only in long packet header)
  - Destination **CID**
  - Packet **number** (obfuscated)
- With TCP, also
  - ACK numbers, ECN information
  - Timestamps
  - Windows & scale factors
- Also, entire QUIC header is **authenticated**, i.e., not modifiable

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| DCID Len (8)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0..160)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| SCID Len (8)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0..160)               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

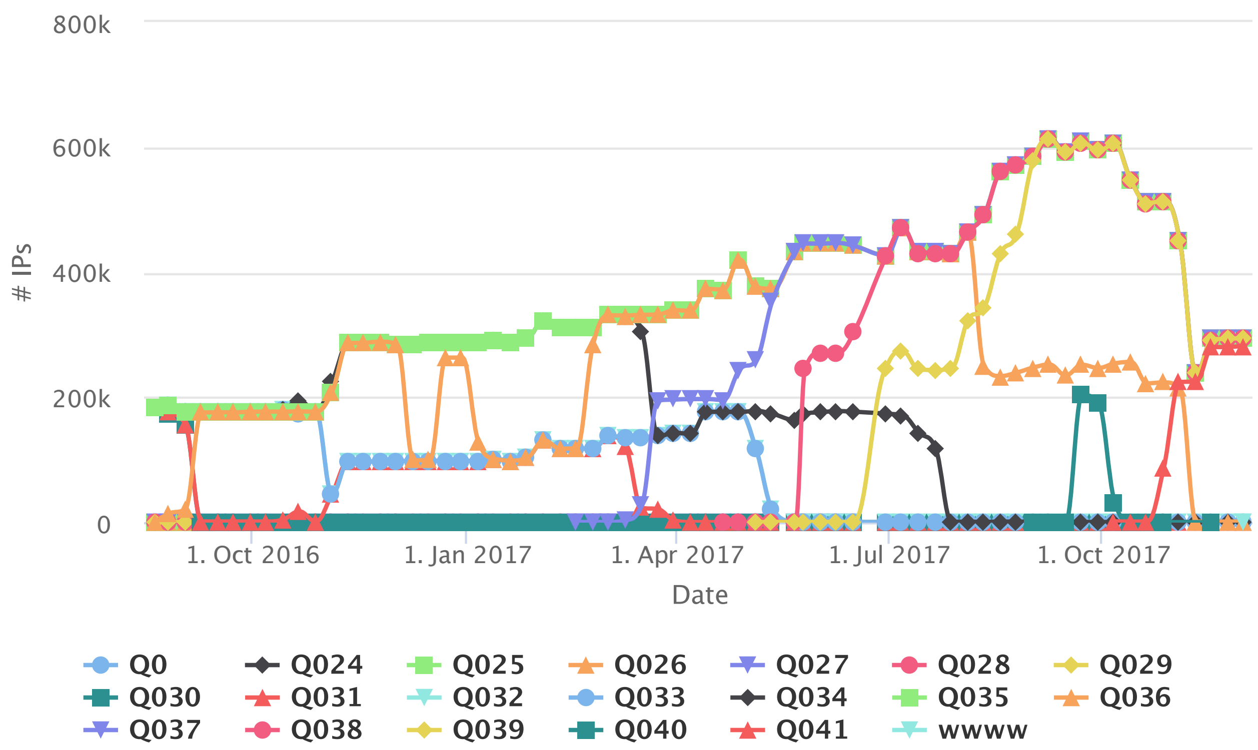
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|0|1|S|R|R|K|P P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Destination Connection ID (0..160)           ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Packet Number (8/16/24/32)              ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Protected Payload (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


NetApp

Version negotiation

(Currently under re-design)

- 32-bit version field
  - IP: 8 bits, TCP: 0 bits
- Allows rapid deployment of new versions
  - Plus, vendor-proprietary versions
- Very few **protocol invariants**
  - Location and lengths of version and CIDs in LH
  - Location and lengths of CID in SH (if present)
  - Version negotiation server response
  - Etc. (details under discussion)
- Everything else is version-dependent
  - But must **grease** unused codepoints!



Source: RWTH QUIC Measurements: https://quic.comsys.rwth-aachen.de/

1-RTT vs. 0-RTT handshakes

- **QUIC client can send 0-RTT data in first packets**
  - Using new TLS 1.3 feature
- Except for very first contact between client and server
  - Requires 1-RTT handshake (same latency as TCP w/o TLS)
- **Huge latency win in many cases** (faster than TCP)
  - HTTPS: 7 messages
  - QUIC 1-RTT or TCP: 5 messages
  - QUIC 0-RTT: 2 messages
- Also helps with
  - Tolerating NAT re-bindings
  - Connection migration to different physical interface
- But only for **idempotent** data

NetApp

Everything else is frames

- Inside the crypto payload,
  QUIC carries a sequence of frames
  - Encrypted = can change between versions
- Frames can come in **any order**
- Frames carry **control data** and **payload data**
- Payload data is carried in **STREAM** frames
  - Most other frames carry control data
- Packet acknowledgment blocks in **ACK** frames

- PADDING
- PING
- **ACK**
- RESET_STREAM
- STOP_SENDING
- CRYPTO
- NEW_TOKEN
- **STREAM**
- MAX_DATA
- MAX_STREAM_DATA
- MAX_STREAMS
- DATA_BLOCKED
- STREAM_DATA_BLOCKED
- STREAMS_BLOCKED
- NEW_CONNECTION_ID
- RETIRE_CONNECTION_ID
- PATH_CHALLENGE
- PATH_RESPONSE
- CONNECTION_CLOSE
- HANDSHAKE_DONE

NetApp

Stream multiplexing

- A QUIC **connection** multiplexes potentially many **streams**
  - Congestion control happens at the connection level
  - Connections are also flow controlled
- **Streams**
  - Carry units of application data
  - Can be uni- or bidirectional
  - Can be opened by client or server
  - Are flow controlled
  - Currently, always reliably transmitted (partial reliability coming soon)
- Number of open streams is negotiated over time (as are stream windows)
- Stream prioritization is up to application

QUIC on IoT devices

Why? Reuse & leverage

NetApp

Warpcore

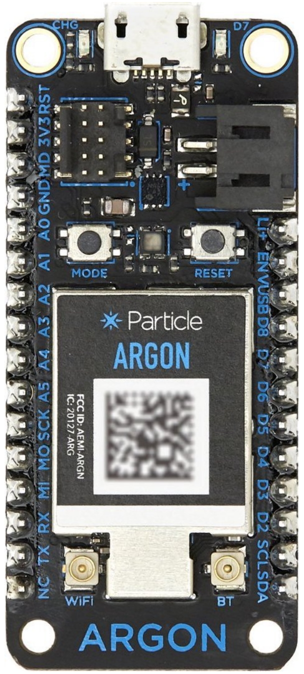
- **Minimal, BSD-licensed, zero-copy UDP/IP/Eth stack**
- Meant to run on netmap, can use Socket API as fallback
- 3700 LoC (+ 3000 LoC w/netmap), C
- Exports generic zero-copy API
- **Device OS has LWIP** = just works (after some patch submissions)
- **RIOT has GNRC** = needs own backend
  - RIOT port of LWIP unfortunately broken
  - GNRC lacks key features (poll/select, IPv4, etc.)

Quant

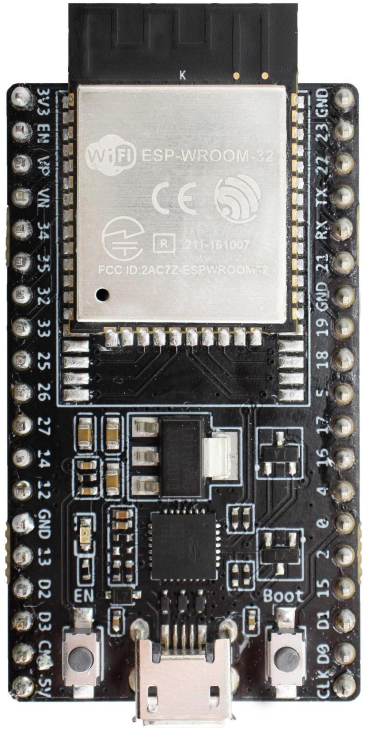
- **QUIC *transport* stack** (i.e., no H3)
  - Focus: high-perf datacenter networking
  - Client and server modes
  - 10,300 LoC, C
- **Warpcore for UDP**, otherwise uses:
  - **khash** (from klib, modified)
  - **timing wheels** (Ahern's timeout.c, modified)
  - **tree.h** (from FreeBSD, modified)
  - **bitset.h** (from FreeBSD, modified)
  - **picotls** (Kazuho Oku)
    - **cifra**
    - **micro-ecc**

NetApp

System hardware and software



Particle Argon	Platform	ESP32-DevKitC V4
Nordic Semiconductor nRF52840	**SoC**	ESP32-D0WDQ6
ARM Cortex-M4F	**CPU**	Tensilica Xtensa LX6
32-bit	**Instruction set**	32-bit
64 MHz	**Clock speed**	240 MHz
IEEE 754 single-precision	**FPU**	IEEE 754 single-precision
ARM TrustZone CryptoCell-310	**HW crypto**	AES, SHA, RSA, and ECC
256 KB	**RAM**	520 KB
1 MB (+ 4 MB SPI)	**Flash**	4 MB
4 KB EEPROM (emulated)	**Other mem.**	96 B e-Fuse
IEEE 802.11 b/g/n	**WLAN**	IEEE 802.11 b/g/n
Device OS 1.4.3	**OS**	RIOT-OS 2019.10
arm-none-eabi-gcc 5.3.1	**Toolchain**	xtensa-esp32-elf-gcc 5.2.0



NetApp

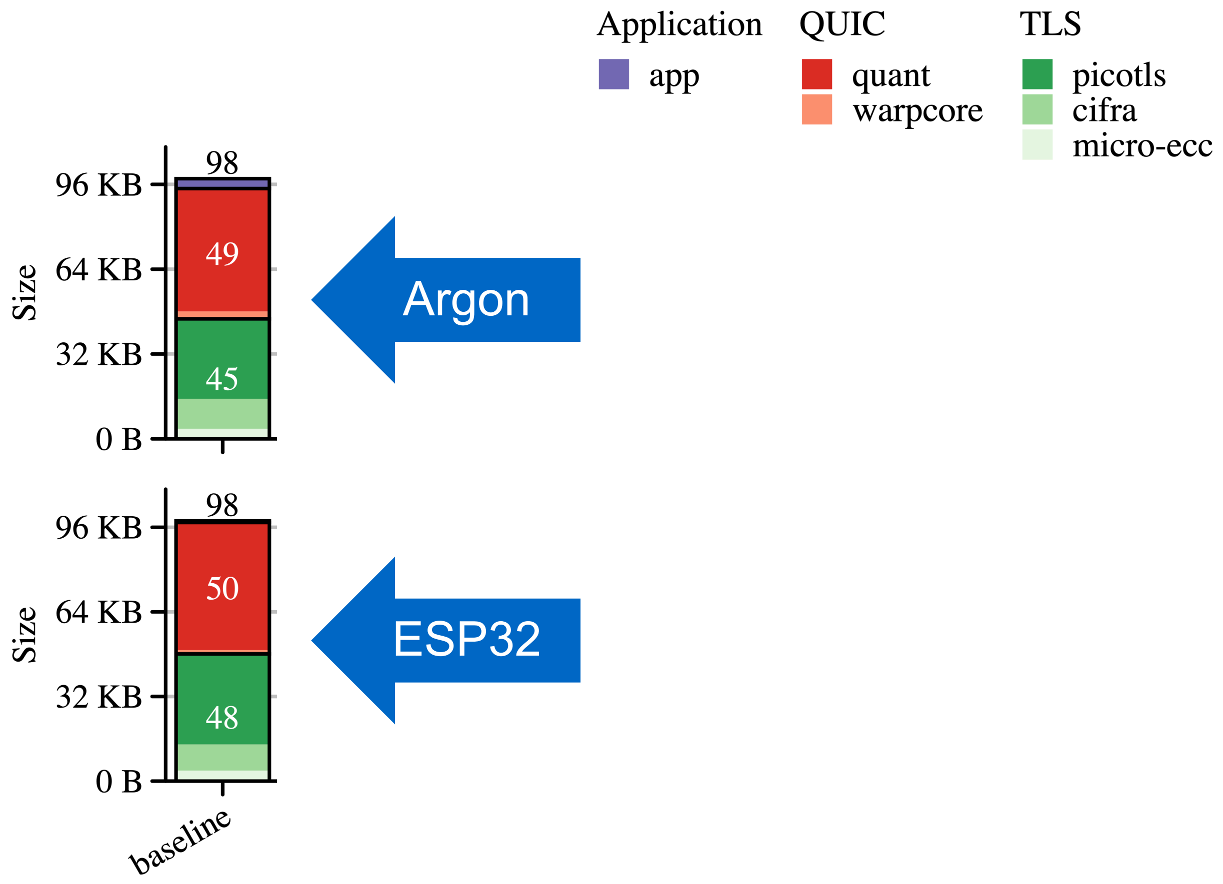
Measurements

Code and static data size

NetApp

Build size: baseline

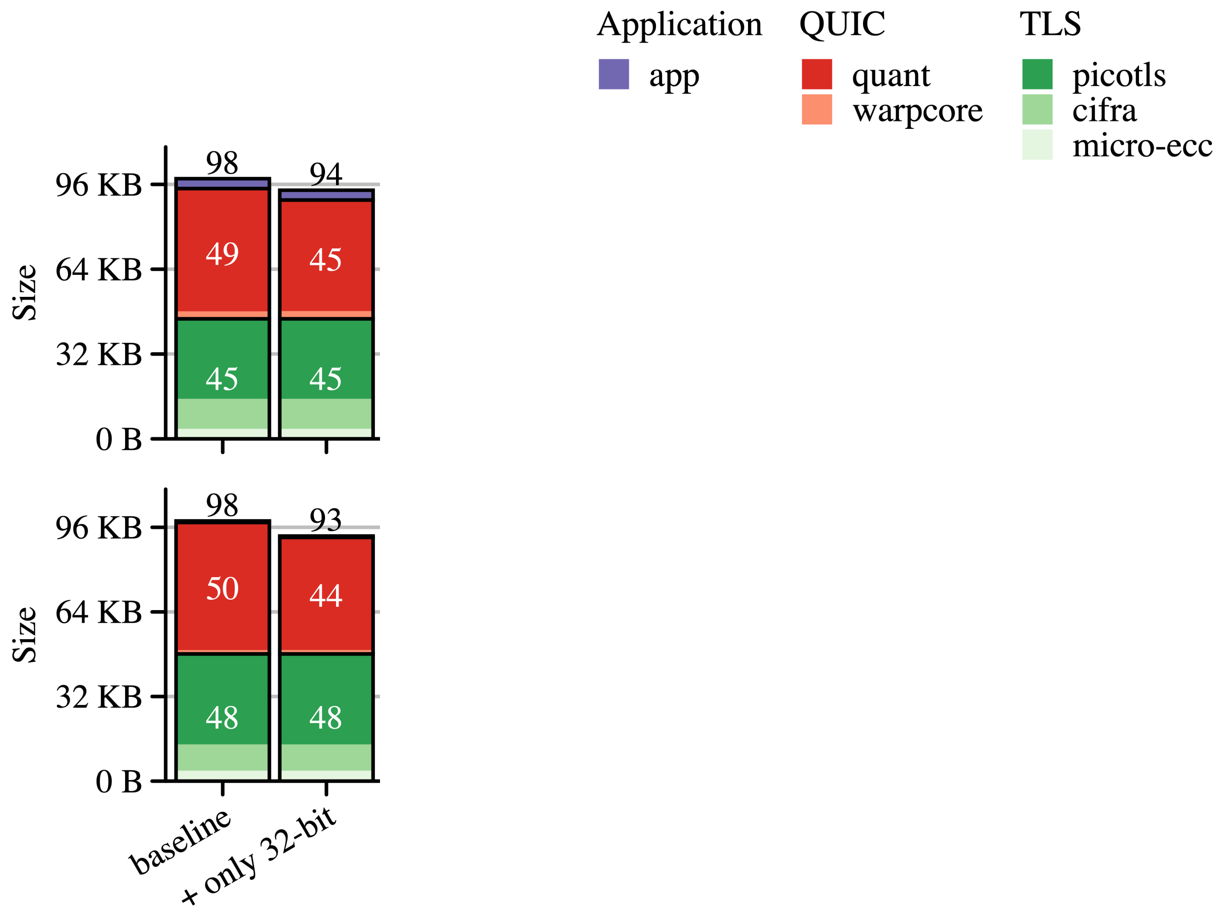
- Compiled code and static data size
- Application
  - Argon app has more features, hence larger
- QUIC
  - Already only uses single-precision FP
- TLS



NetApp

Build size: 32-bit optimizations

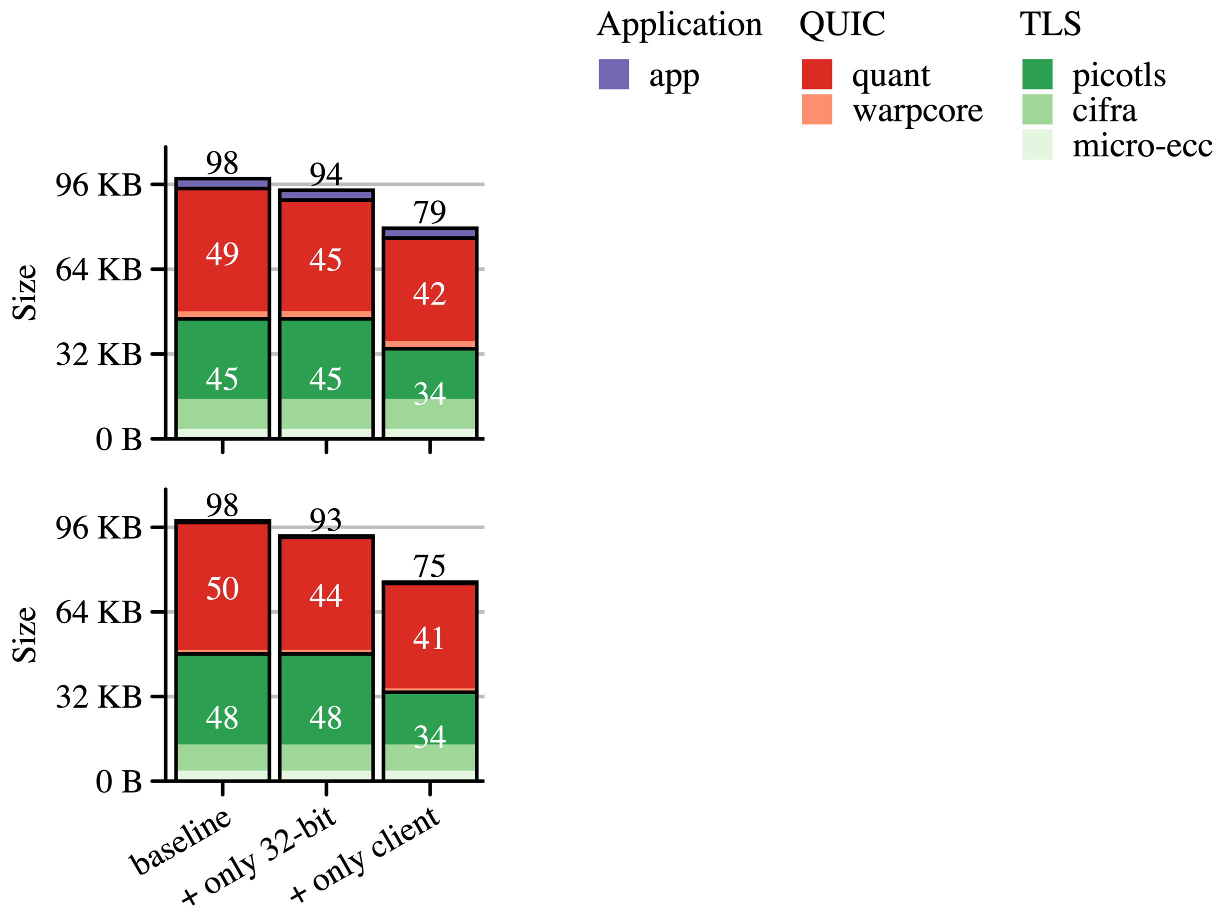
- **Eliminate costliest 64-bit math**, i.e., division and modulus
  - All are by constants, can multiply by magic number and right shift
- **Use 32-bit width** for many internal variables, e.g.,
  - Packet numbers
  - Window sizes
  - RTT (μs)
- **Not *fully* spec-conformant**, but unlikely to matter in practice for IoT



NetApp

Build sizes: client-only mode

- **Disable server functionality**
- Unlikely to be of much use for IoT, esp. when battery-powered
- Also makes client use zero-length CIDs
- Large gain at the TLS layer!

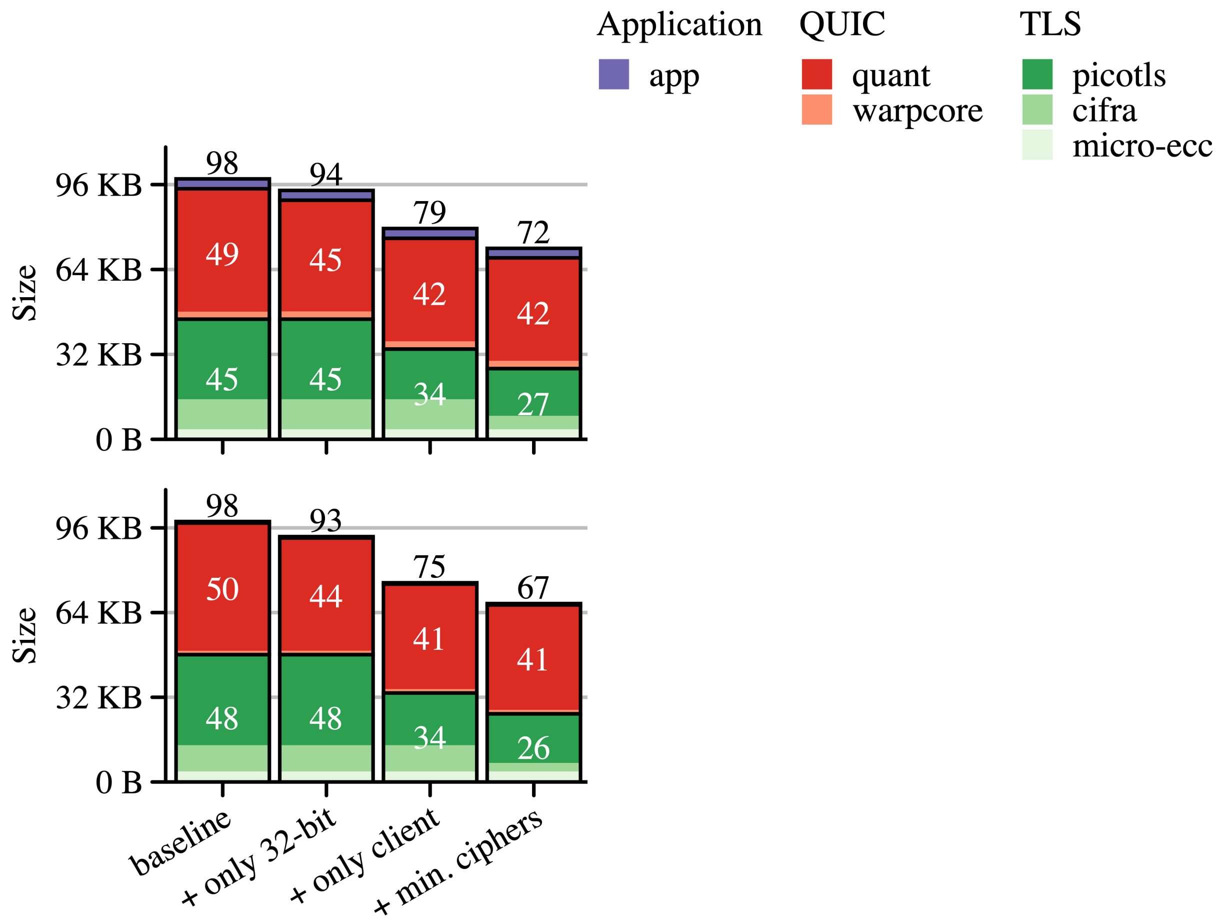


- (Server-only mode: future work)

NetApp

Build sizes: minimally-required crypto

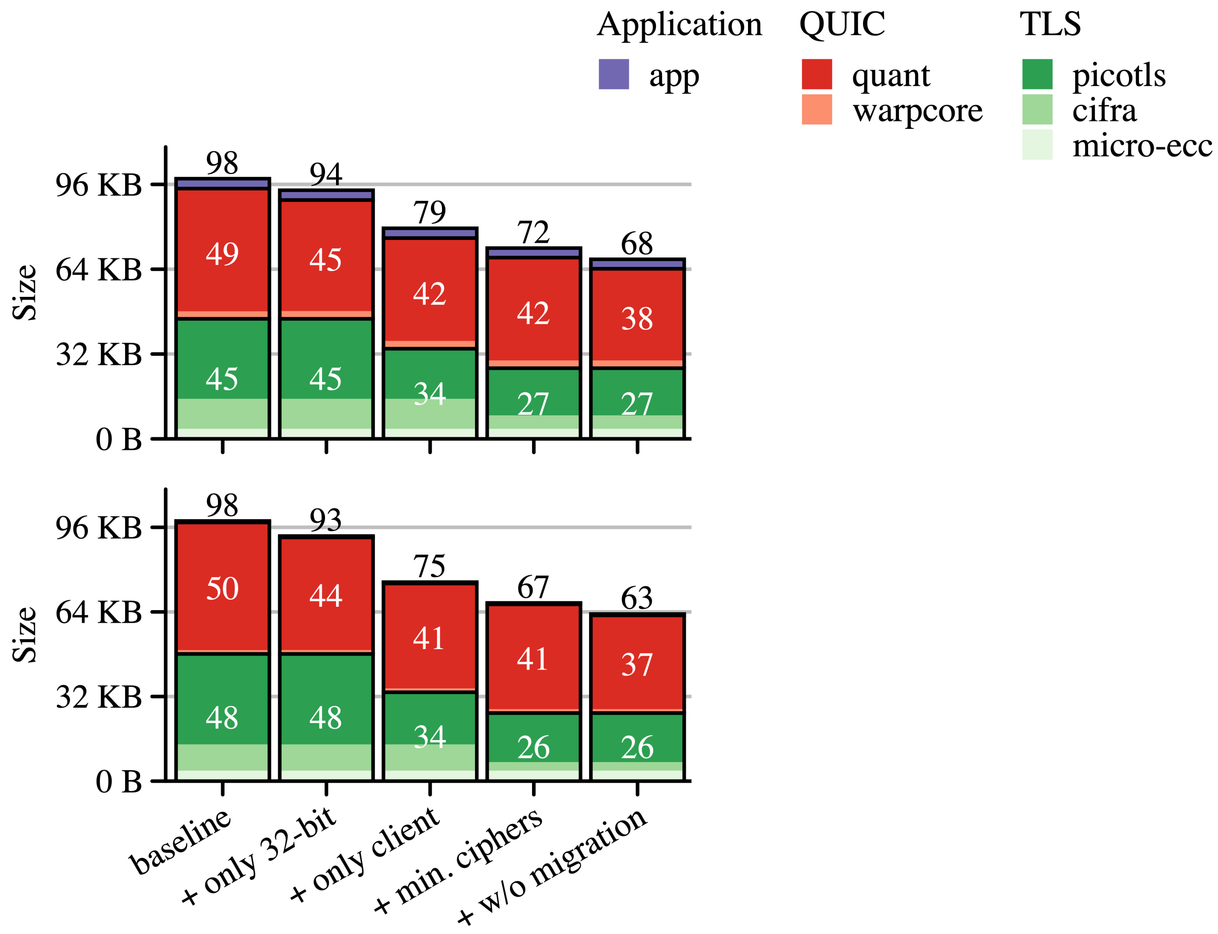
- **Disable non-required crypto**, leaving
  - TLS_AES_128_GCM_SHA256 cipher suite
  - secp256r1 key exchange
- More gains at the TLS layer!
- **Could fully eliminate cifra & micro-ecc** *if* HW crypto was accessible from OSs...
- Together, reductions of 25-30% so far, without much loss in functionality
- Can save more by turning off functionality...



NetApp

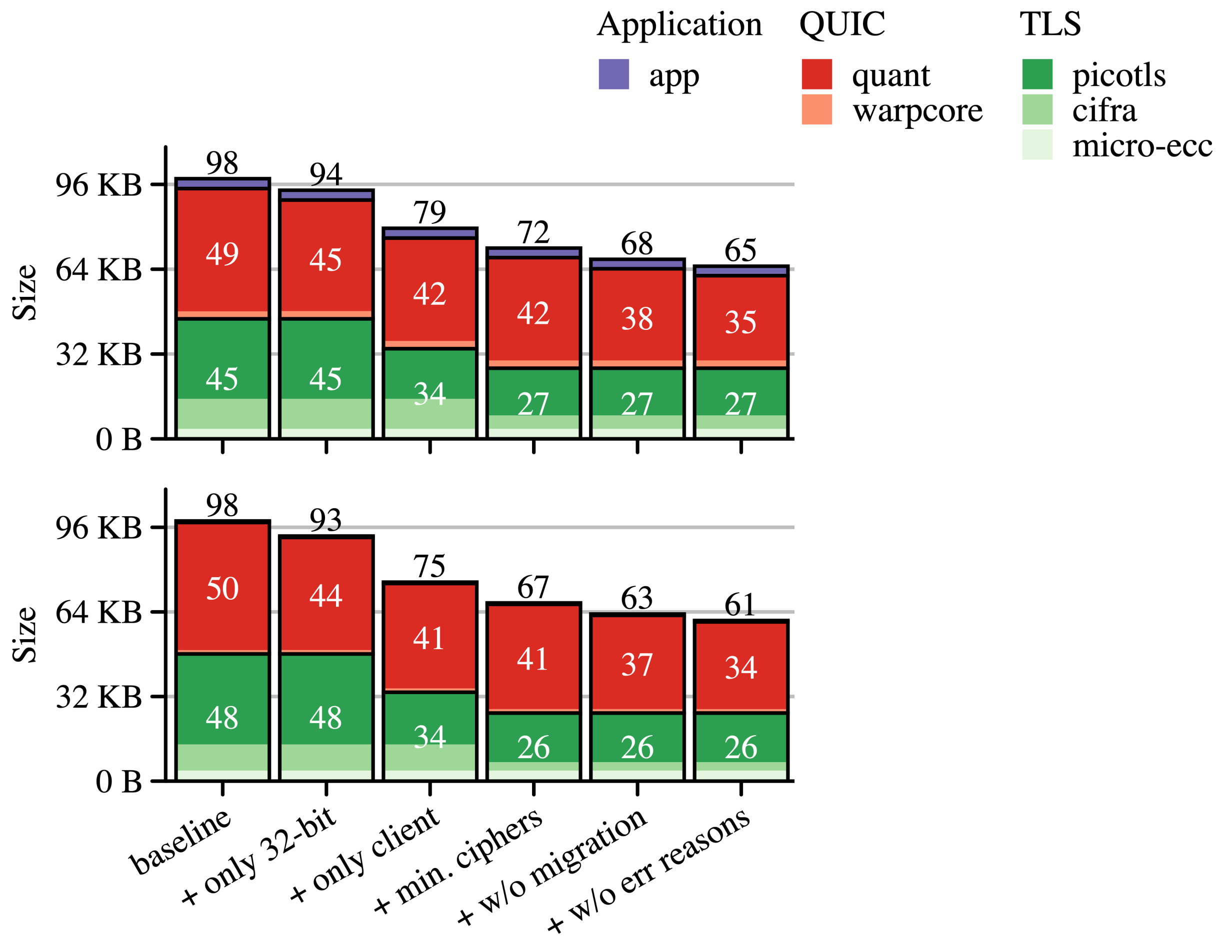
Build sizes: no migration

- **Connection migration** = switching an established connection to a new path
- Likely **unnecessary for IoT** usage



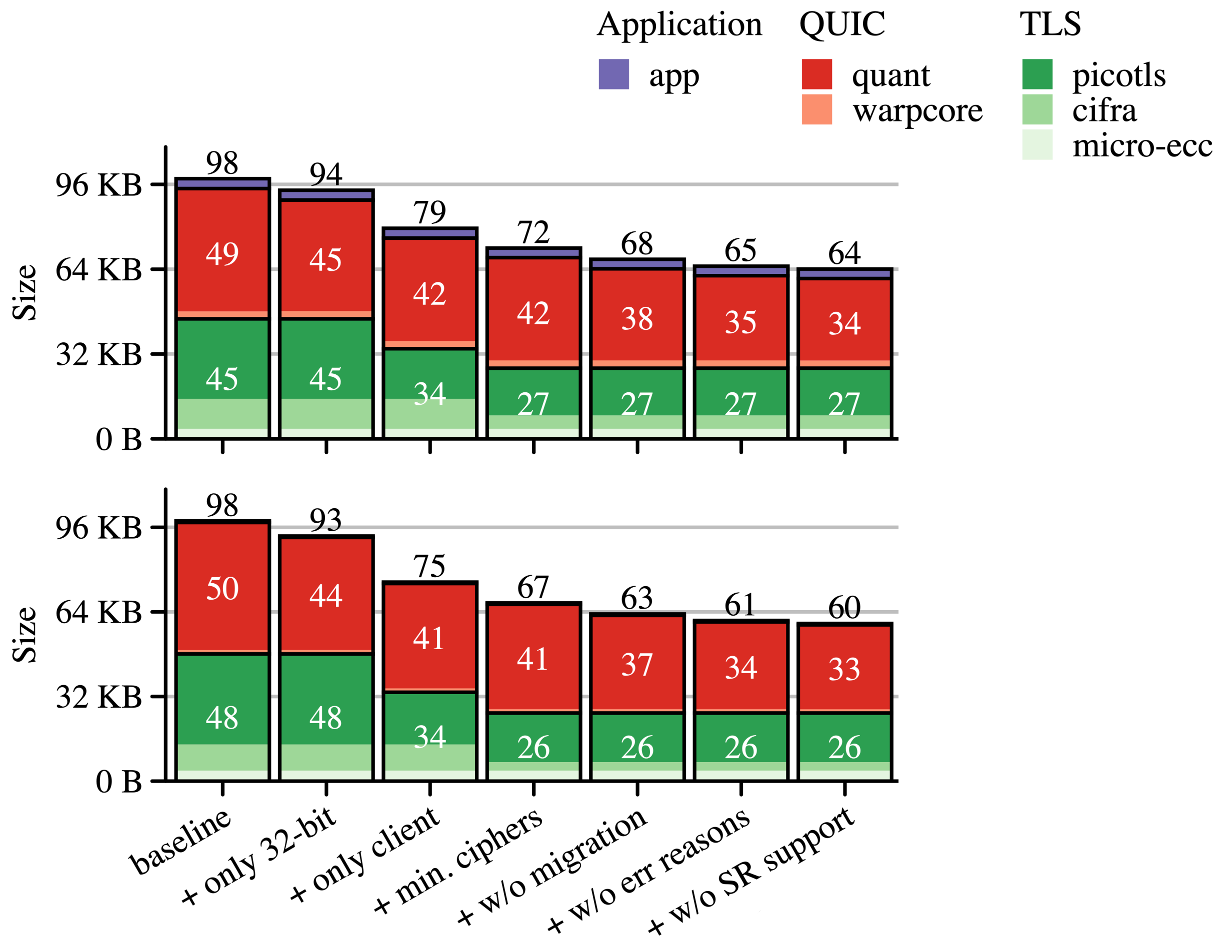
Build sizes: no error reasons

- QUIC allows **plaintext “reason” strings** in CONNECTION_CLOSE frames
- No protocol usage, only for human consumption
- Quant by default uses those heavily & verbosely
- So don’t



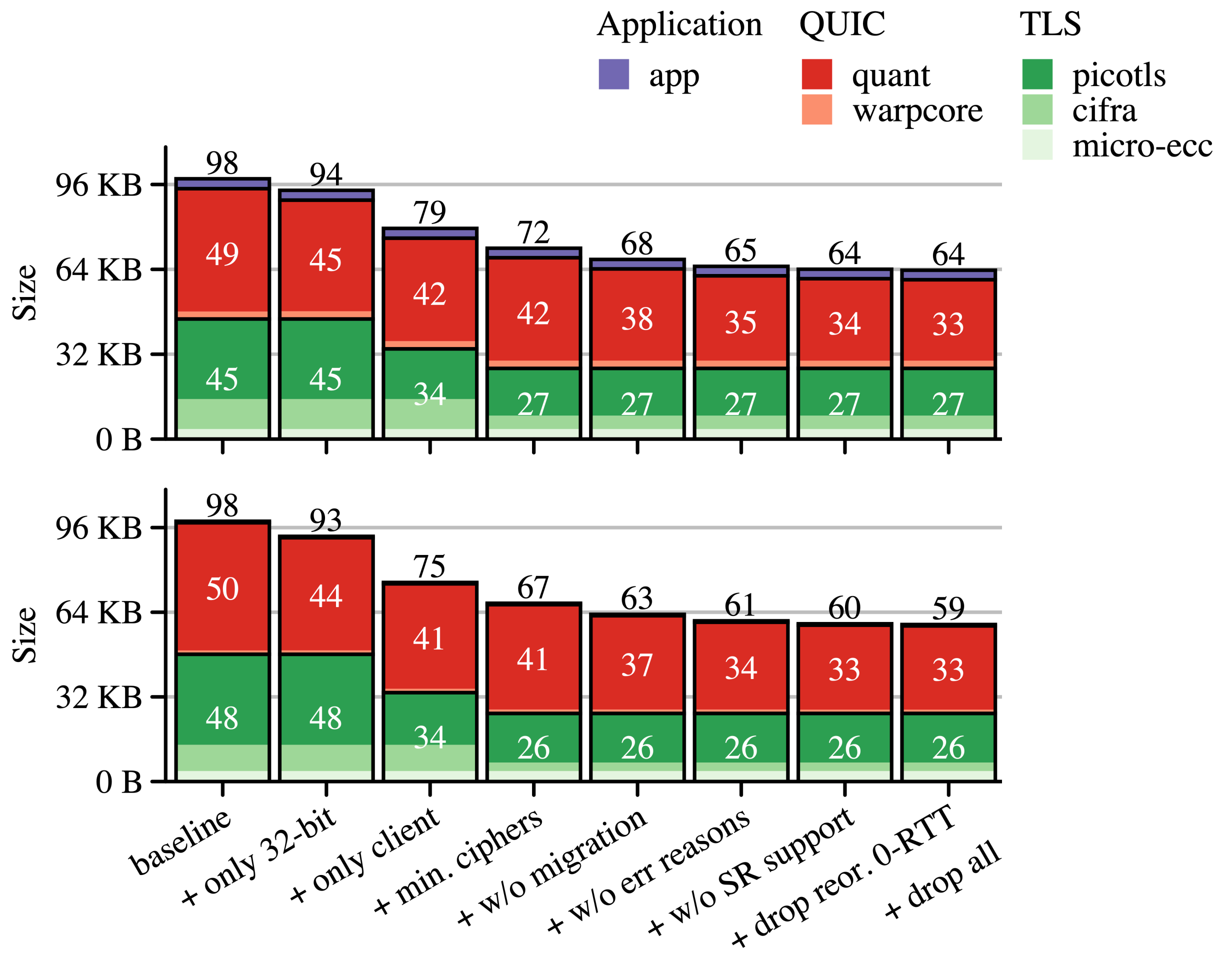
Build sizes: no stateless resets

- **Stateless reset** = signal to peer that local end has no more state for a connection
- To handle, need to be able to identify *which* connection RX'ed SR is for
- **Tradeoff:** handle SR vs. needlessly RTX



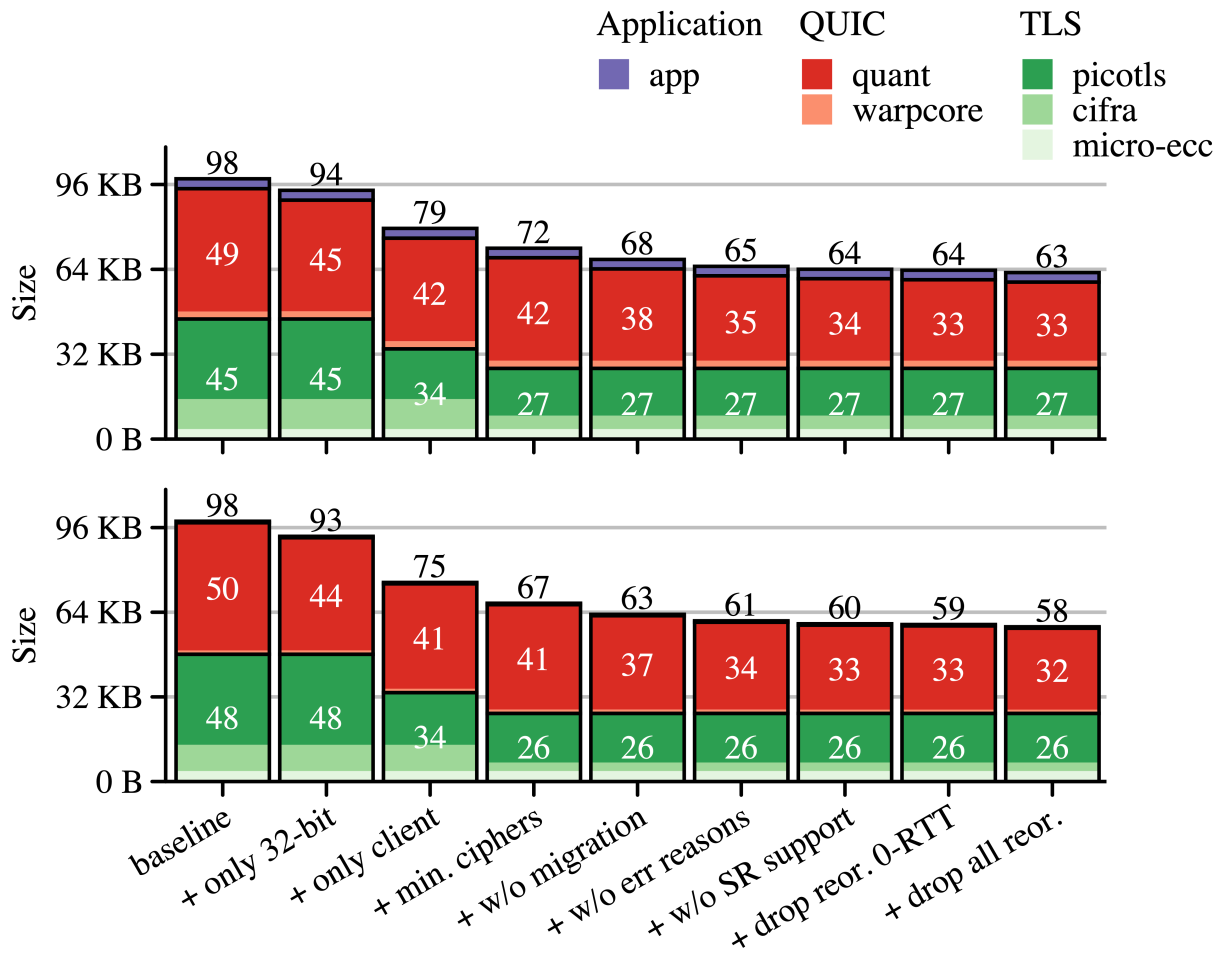
Build sizes: drop reordered 0-RTT

- Caching 0-RTT packets arriving out-of-order can avoid RTX
- Also has an overhead
- **Tradeoff:** cache vs. force RTX



Build sizes: drop *all* reordered data

- Caching *any* out-of-order CRYPTO or STREAM data can avoid RTX
- Also has an overhead
- **Tradeoff:** cache vs. force RTX



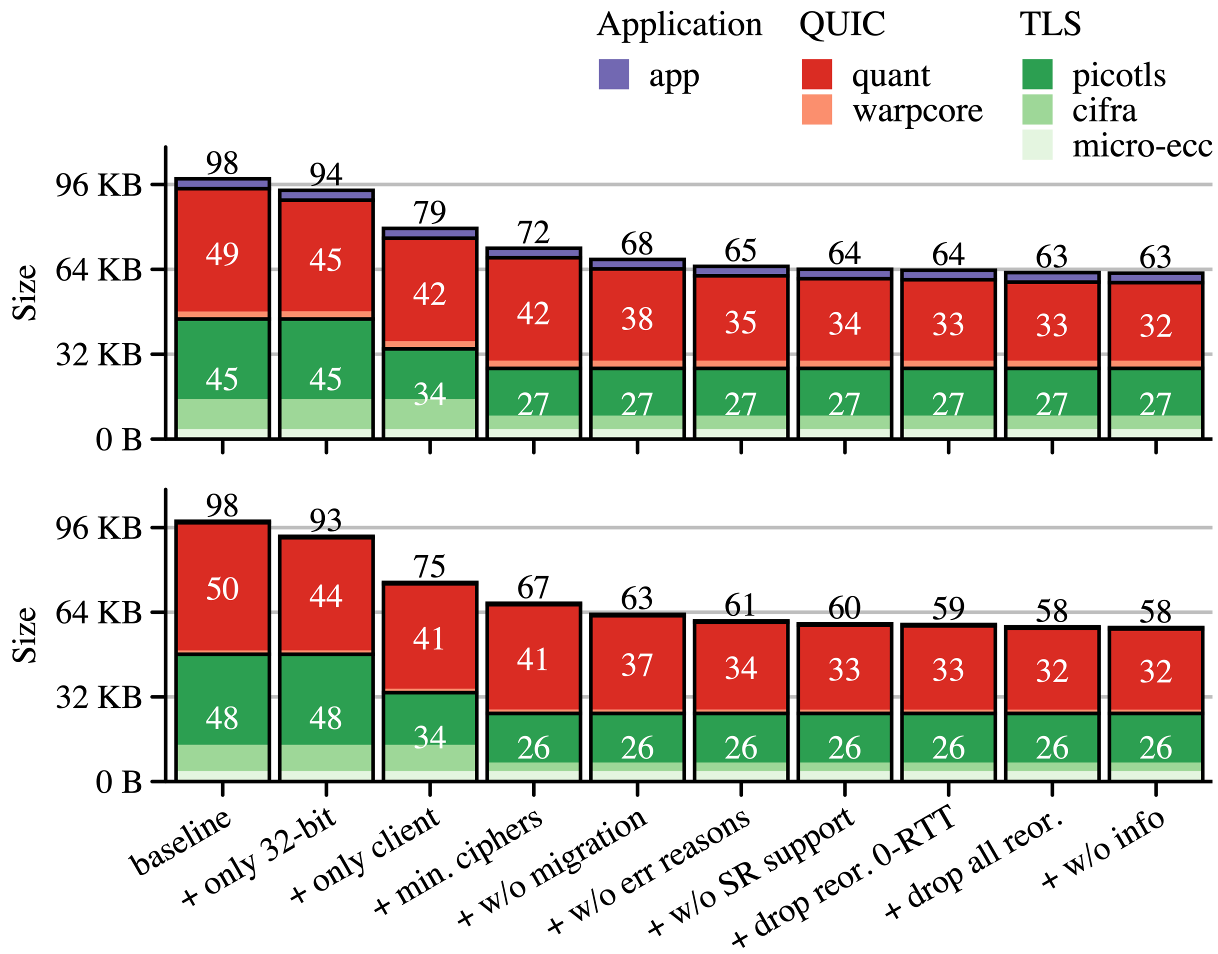
NetApp

Build sizes: don’t maintain connection info

- Quant maintains a TCP_INFO-like struct about each connection:

```
pkts_in_valid = 40
pkts_in_invalid = 0
pkts_out = 10
pkts_out_lost = 0
pkts_out_rtx = 0
rtt = 0.049 (min = 0.000, max = 0.087, var = 0.027)
cwnd = 14840 (max = 14840)
ssthresh = 0
pto_cnt = 0
frame                  code       out        in
PADDING                0x00      2941      1214
PING                   0x01         1         1
ACK                    0x02         6         7
CRYPTO                 0x06         3         5
NEW_TOKEN              0x07         0         3
STREAM                 0x08         1        29
MAX_STREAM_DATA        0x11         1         0
NEW_CONNECTION_ID      0x18         3         1
RETIRE_CONNECTION_ID   0x19         1         2
CONNECTION_CLOSE_APP   0x1d         1         1
HANDSHAKE_DONE         0x1e         0         2
strm_frms_in_seq = 33
strm_frms_in_ooo = 1
strm_frms_in_dup = 0
strm_frms_in_ign = 0
```

- Don’t do that



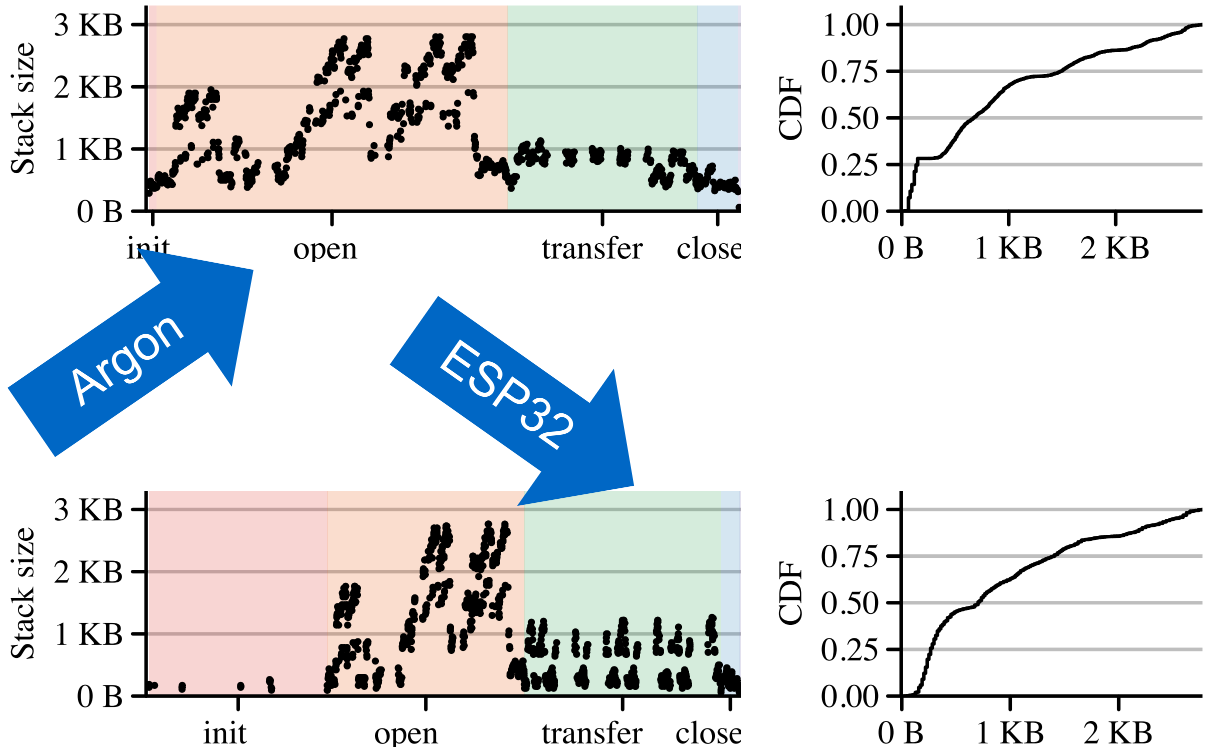
Measurements

Stack and heap usage

NetApp

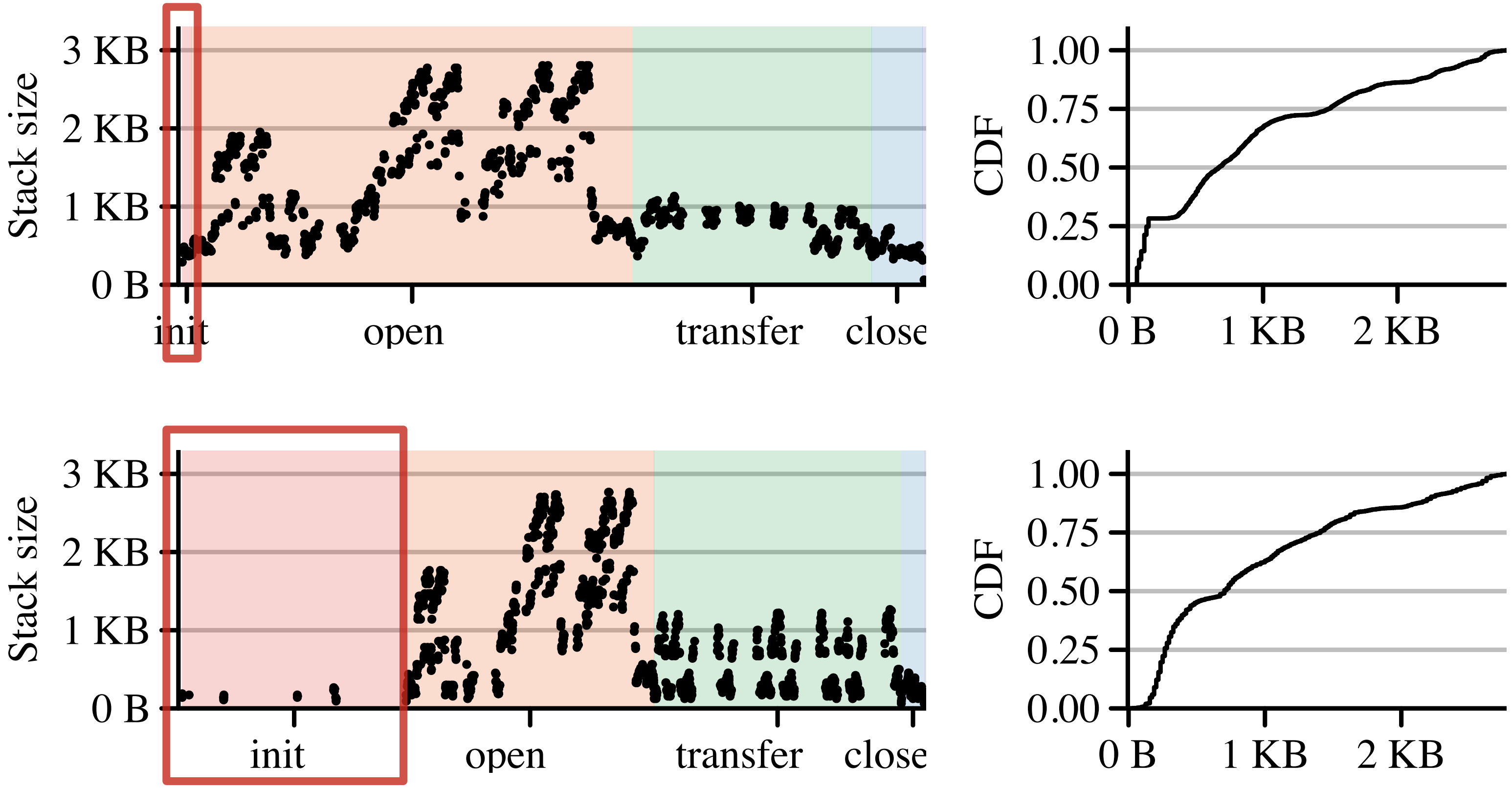
Stack and heap usage

- **Instrumented binaries** to log stack and heap usage on function enter/exit
- cifra and micro-cc ***NOT*** instrumented
  - Too many small functions, too much log data
- Shown results are therefore **lower bounds**
  - Approximate the case *if* HW did crypto
- **Time units not shown** on purpose
  - Run takes tens of seconds due to 112.5Kb/s serial
- **Random 20%** of data points plotted to reduce overplotting



Stack usage: init phase

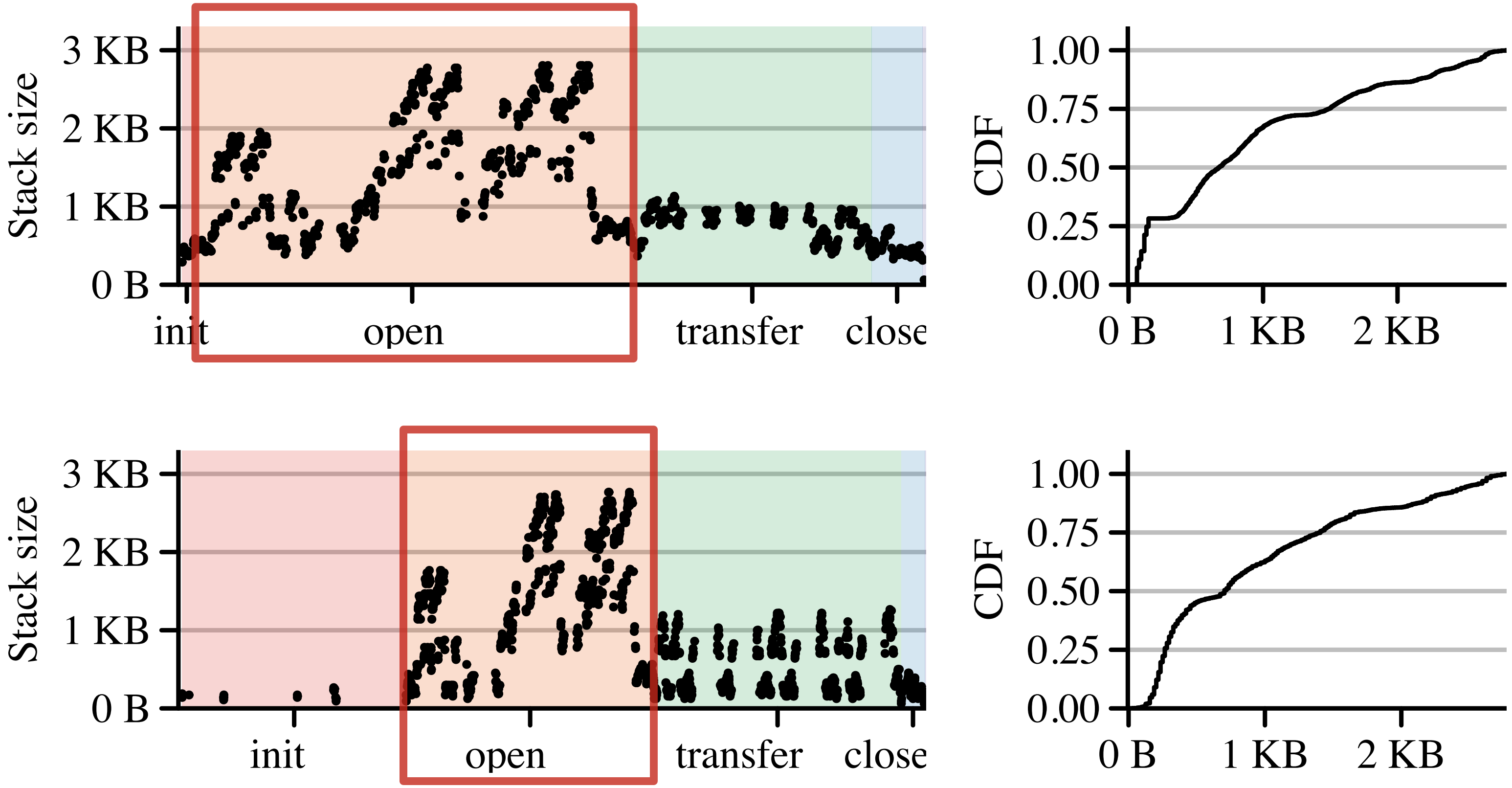
- Quant and Warpcore initialization
- On ESP32, includes WLAN association = longer duration
- Minimal stack usage,few 100s of B



NetApp

Stack usage: open phase

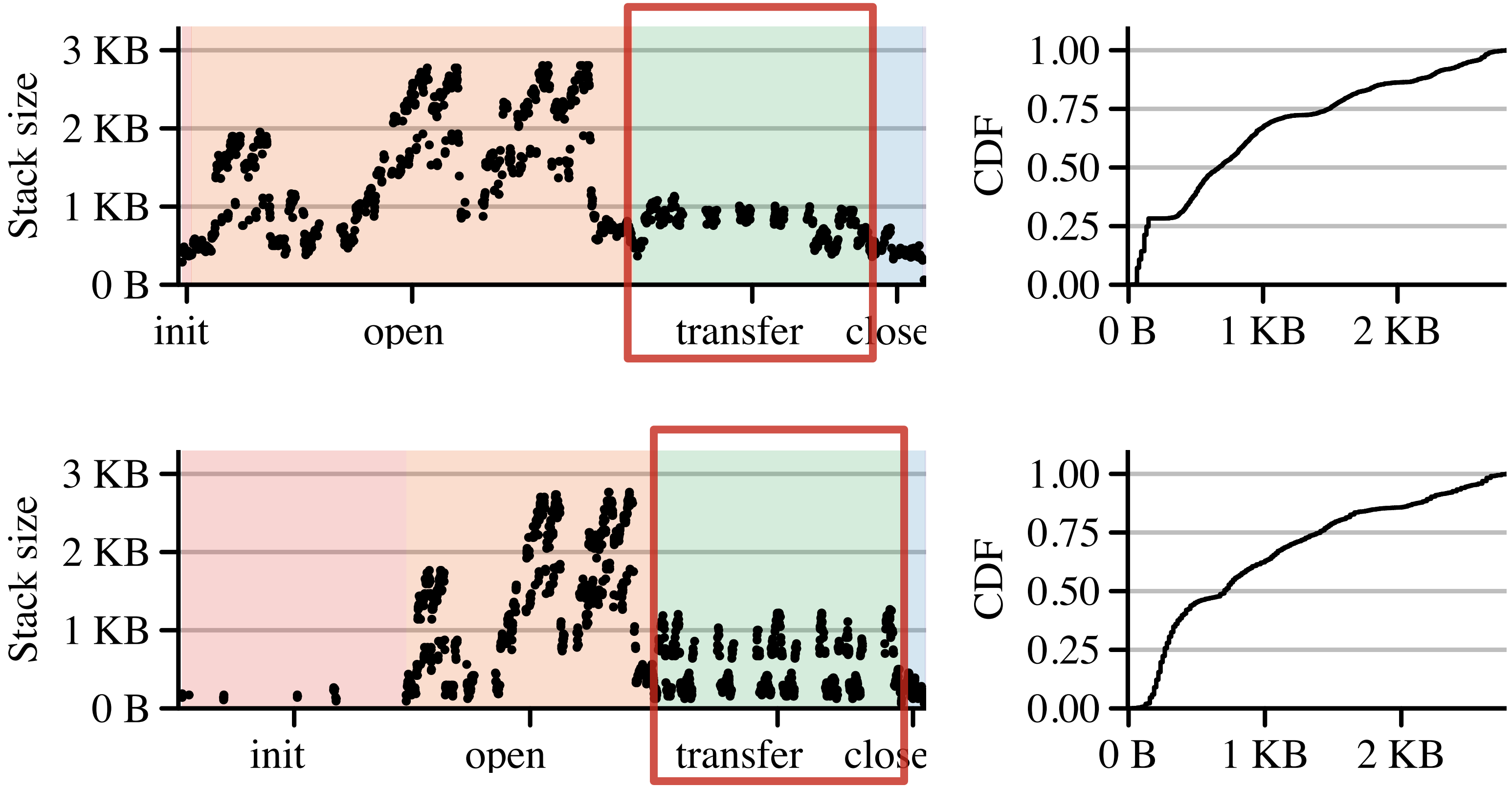
- Open connection to server
- **Public key crypto** as part of handshake
- **Stack usage peaks** at almost 3 KB
- **Not great** for IoT usage
  - 1 KB RIOT stack default
  - 6 KB Device OS stack default
- Optimizations needed
  - picotls uses stack-allocated buffers



NetApp

Stack usage: transfer phase

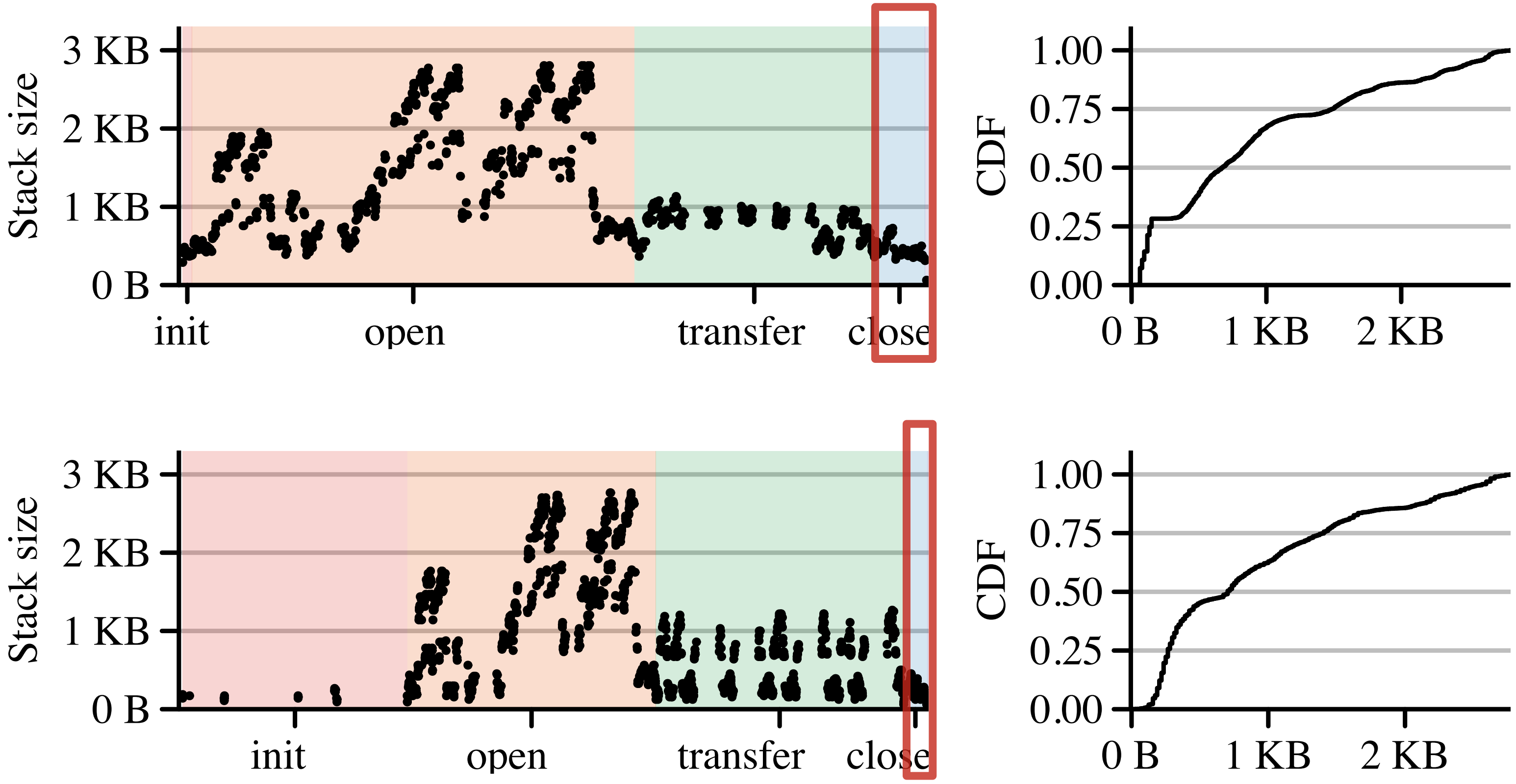
- RX data from server
- **Symmetric crypto**
- **Stack usage is lower** at around 1 KB
- **Still not super-great** for IoT
- Optimizations needed



Stack usage: close phase

- Close connection with server and de-init
- **Stack usage dropping** down to initial values

- Overall, unfortunately, **peak stack usage** is what matters

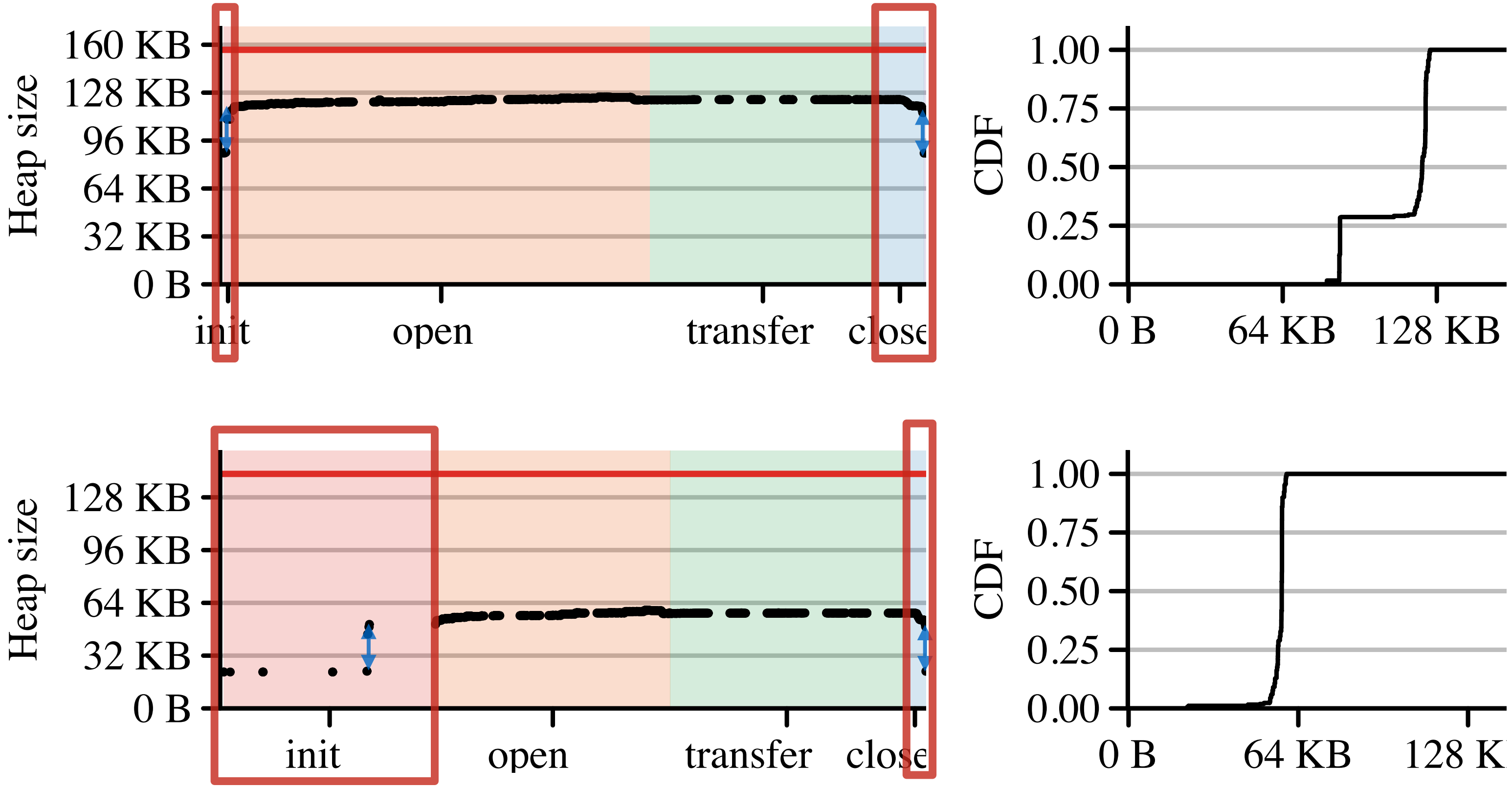


NetApp

Heap usage

- **Heap usage jumps** on allocation/deallocation of packet buffers
- 15 buffers @ 1500 B each

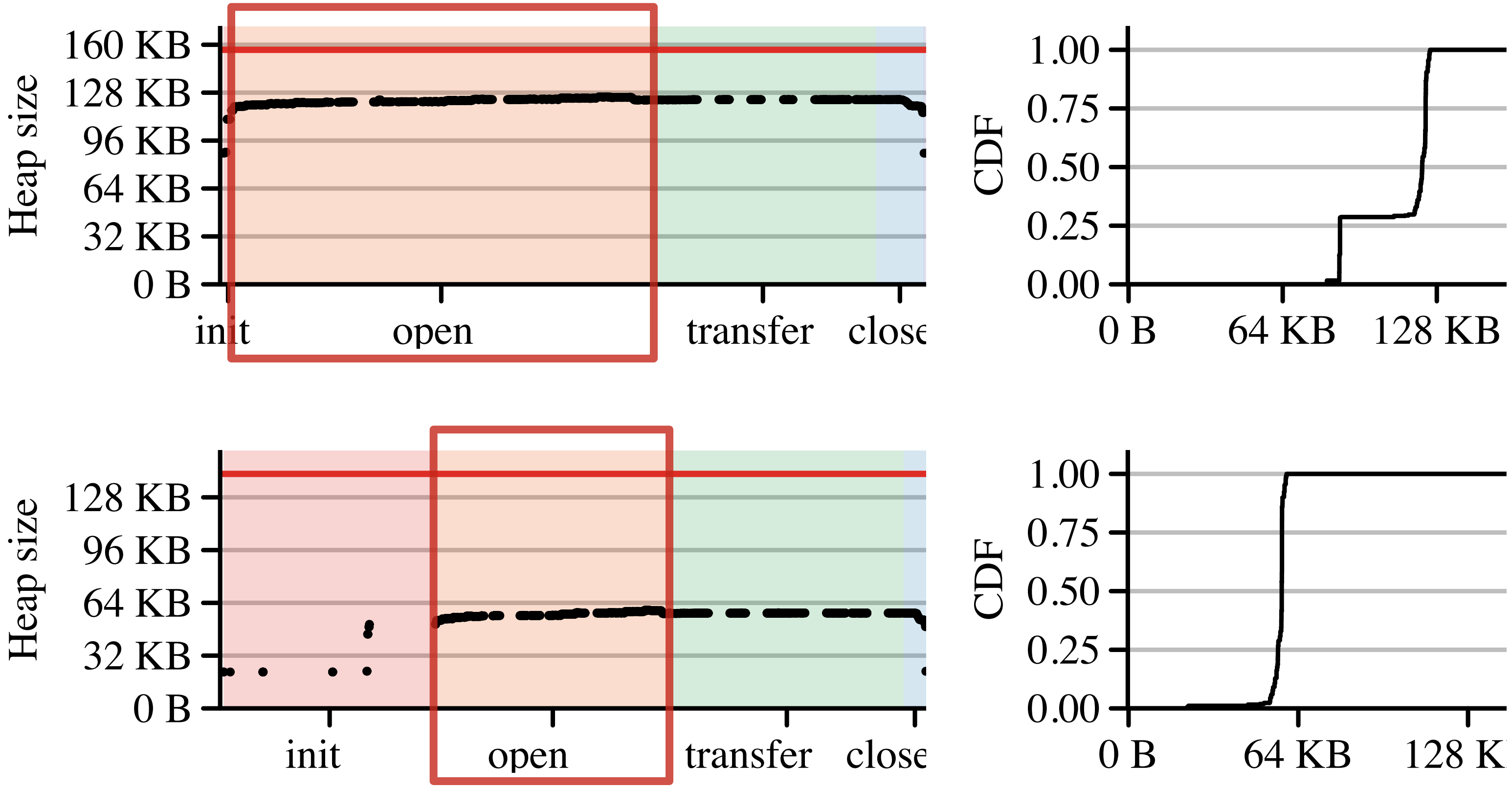
- Baseline heap usage on Argon much higher
  - DeviceOS executing in background



NetApp

Heap usage

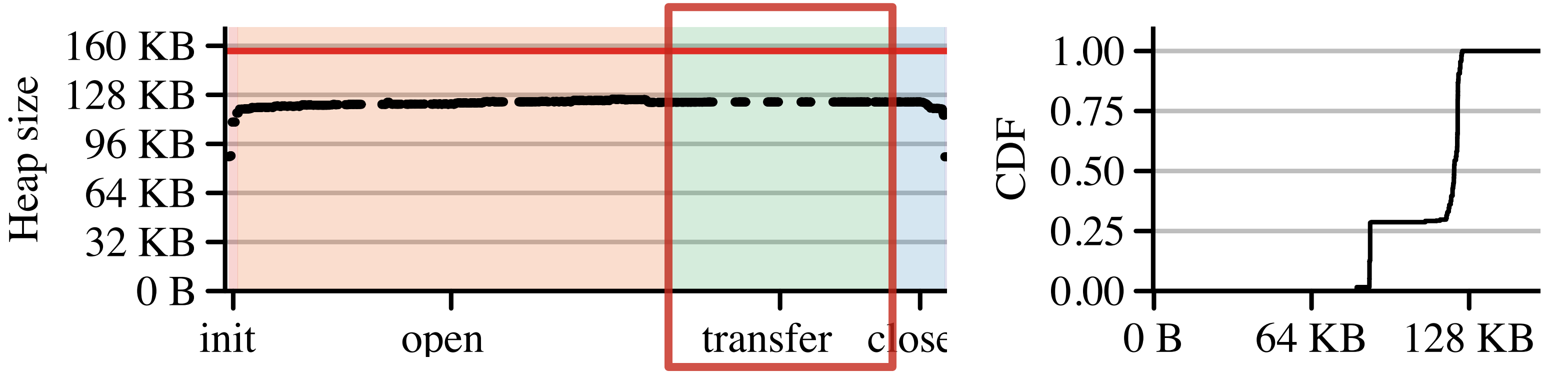
- During open phase, **slight increase in heap**
- Allocation of additional per-connection dynamic state

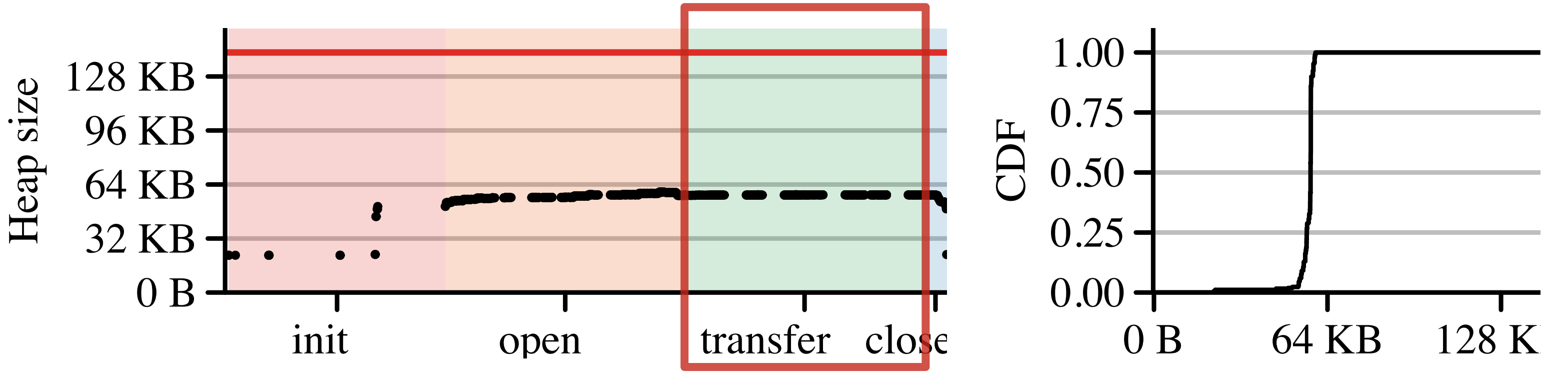


NetApp

Heap usage

- **Flat heap usage** during transfer phase
- Nice!



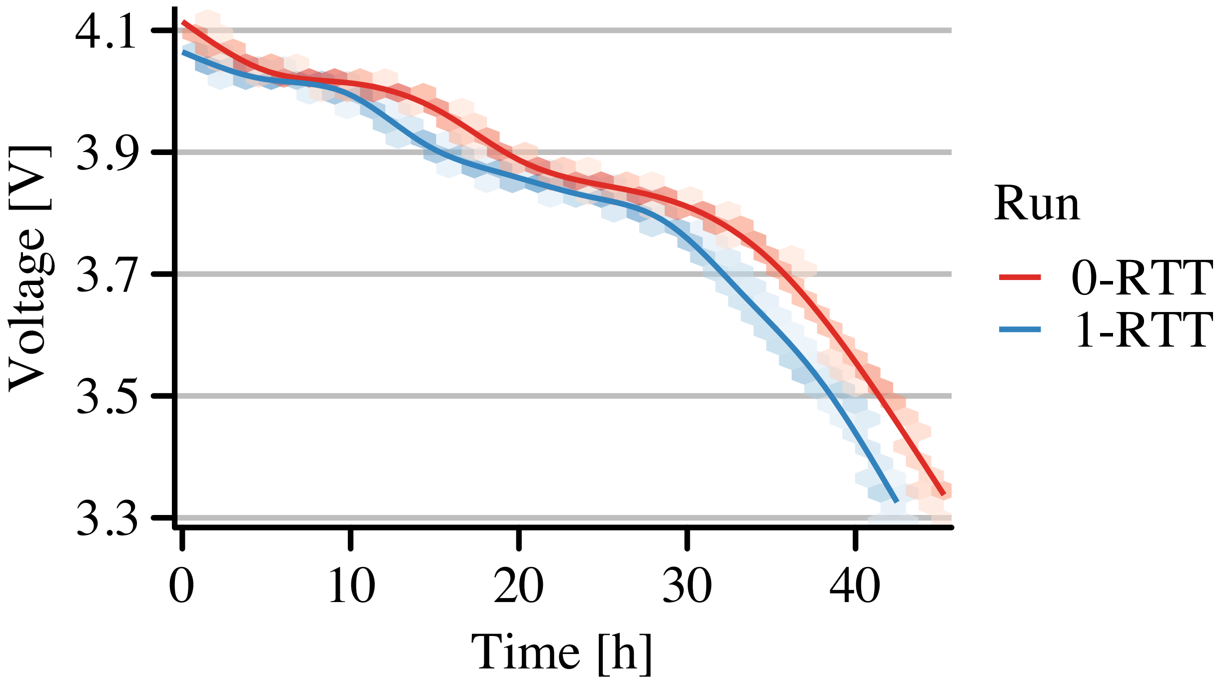


Measurements

Energy and performance

Energy measurements

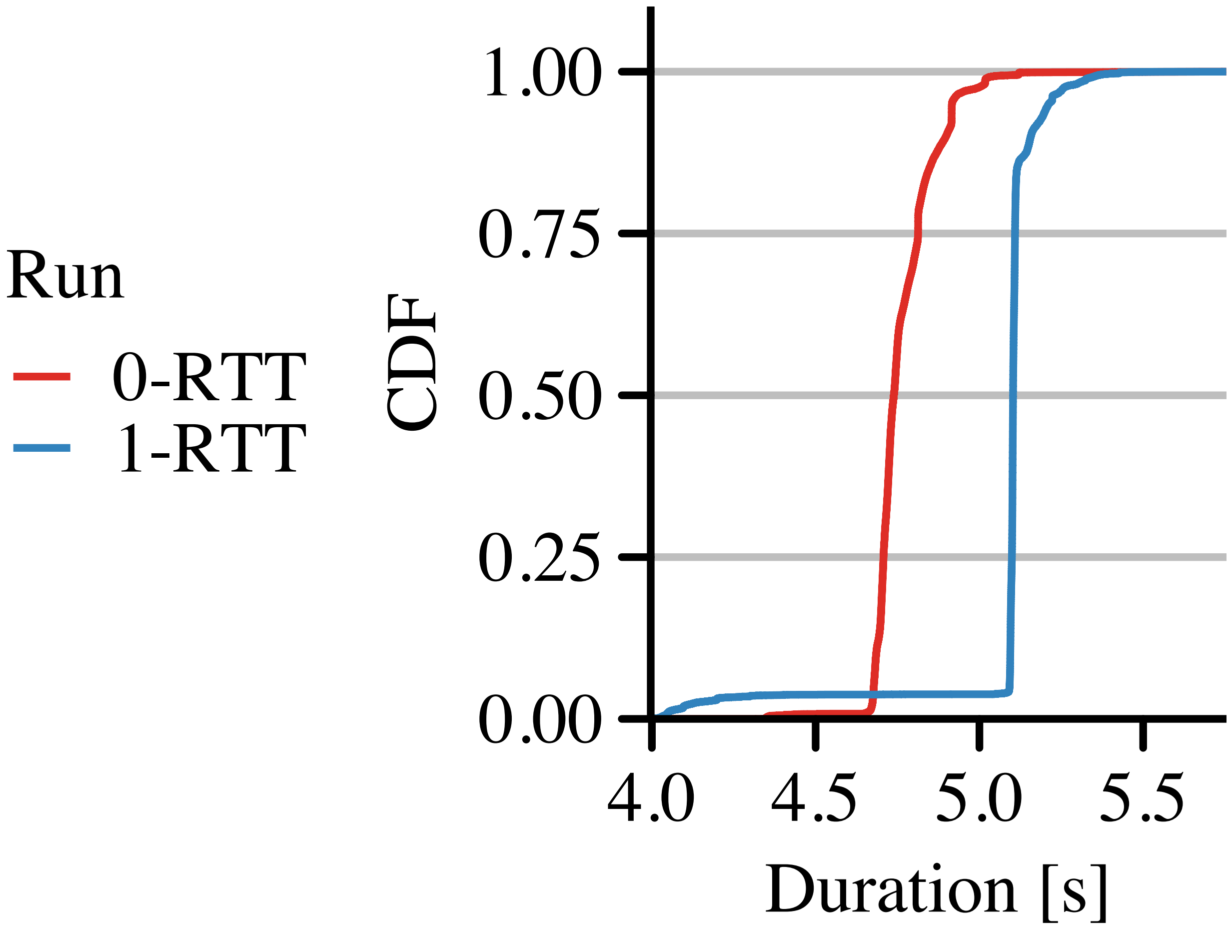
- Argon with 2000 mAh 3.7 V LiPo battery
- **Two runs** after full charges
  - **Only 1-RTT** connections
  - (Initial 1-RTT followed by) **only 0-RTT** connections
- Ran for ~2.5 days non-stop
  - 29,338 1-RTT connections (~0.90 J/conn)
  - 31,844 0-RTT connections (~0.83 J/conn)
- **Very preliminary!**
  - Argon-internal voltage reporting is coarse
  - Single run only
  - Hesitant to draw conclusions



NetApp

Performance measurements

- Data from the same runs used for energy measurements
- Median 1-RTT connection took 5.10 s
- Median 0-RTT connection took 4.74 s
- Open questions
  - Why does 0-RTT show more of a slope?
  - Why is 1-RTT sometimes faster? (Loss?)



NetApp

Future work

Lots and lots

Future work

Measurements

- Measure **data upload**
- **Vary parameters** of measurement
  - Object sizes, streams, connections, etc.
- **Compare** against other protocols
  - TCP, TLS/TCP, CoAP, MQTT, etc.
- **Compare** different IoT boards
- More accurate **energy measurements**

Implementation

- Add **H3** binding & measure
- Make **picotls** not use stack buffers
- **Better data structures** w/less heap churn
- Use **HW crypto** (performance & energy)
- **Drop 0-RTT** to shrink code size?
- IP over **BLE or 802.15.4** instead of WLAN
  - WLAN on ESP32 is 115 KB (45% of OS size)
- Can we scale down to **16-bit controllers**?

NetApp

NetApp

Thank you

Questions later?
lars@netapp.com