
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

NET401

Deliver great experiences with QUIC on Amazon CloudFront

Jim Roskind

VP, Distinguished Engineer
Amazon.com

Mahmoud Ragab

Engineering Manager
Snap Inc

Why is Jim Roskind talking about QUIC?

- Architected/designed/led development of QUIC
  - “Quick UDP internet connections” evolved into IETF HTTP/3 Standard
- 1995: Netscape on browser/server and security
  - Helped design SSL 2.0 (TLS 1.0); designed signed Java; was Netscape’s “Java Security Architect”
- 2008: Worked at Google on making Chrome (go faster)
  - Proposed/architected/designed . . .
    - Implemented metrics infrastructure (central to QUIC)
    - Implemented DNS pre-resolution, TCP pre-connection
- 2016: Amazon VP/Distinguished Engineer

QUIC: “Quick UDP Internet Connections”

- Protocol to supplant HTTP and more
- Cryptographic privacy and tamper resistance
  - TLS
- Multiplexes requests (like HTTP/2)
  - One congestion-controlled flow
  - Improved latency, reduced variance
- Sequenced UDP packets
  - Evolving congestion control

. . . but why do we need/want/use it??

QUIC is all about speed to user: Latency

- Faster and reliable connections
  - Fewer round trips to connect
  - Reduces time-to-first-byte
  - Reduces latency variance in delivered bytes
  - Better web performance in congested networks

- Amazon CloudFront supports HTTP/3!
  - Available worldwide
  - Full TLS Security

  . . . and you should enable it!

Don’t keep the customer waiting
Give customers what they want!

Overview

Context/justification

Background history of HTTP latency/bandwidth

Problems

Solutions

SnapChat deployment using Amazon CloudFront

Q&A

Context for developing a new protocol

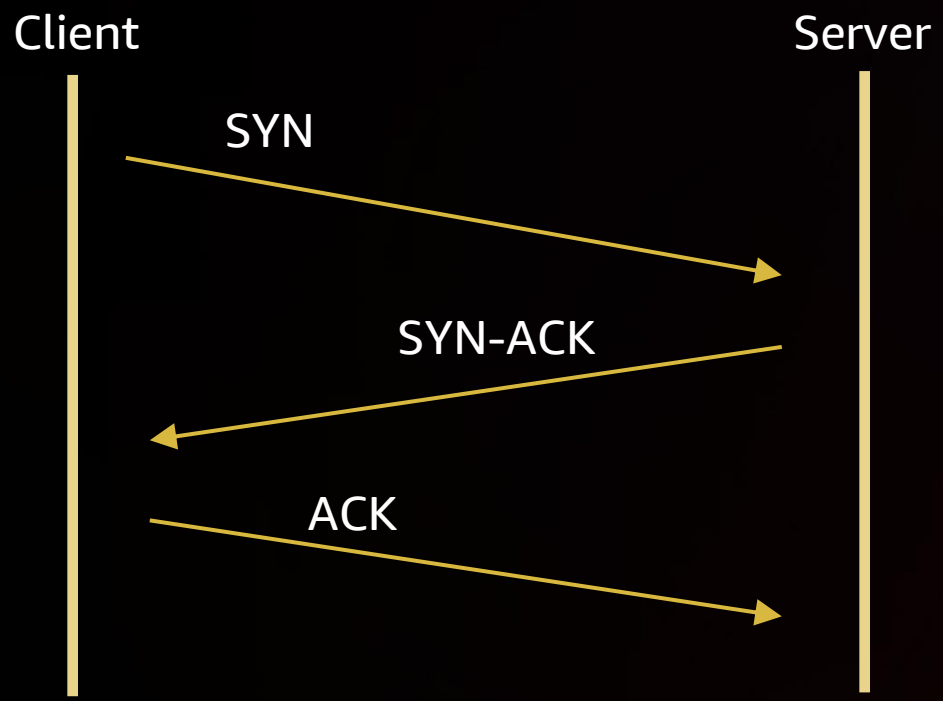
- Customer (obsessed?) measurement focus
  - Don’t keep the customer waiting: **Reduce latency**
- Stand on shoulders of (protocol) giants
  - Expertise history/support (TLS)
  - Recent development of SPDY (HTTP/2)
  - Customer metrics infrastructure (histograms)
- Luck

Details of pre-existing problems

RTT up to 400ms (almost a half second!)

- Speed of light
- US cross-continent speed-of-light RTT was 20ms
  - 60–100ms with packet handling :-/
- Target < 200ms
  - Insist on the highest standards!
- Can’t afford many round trips on critical path!
  - Each HTTP/1.0 request uses a fresh TCP connection
  - www.example.com: 150 HTTP requests on home page!

TCP connection establishment



Time-consuming RTT (round trip time)

400ms average to India!

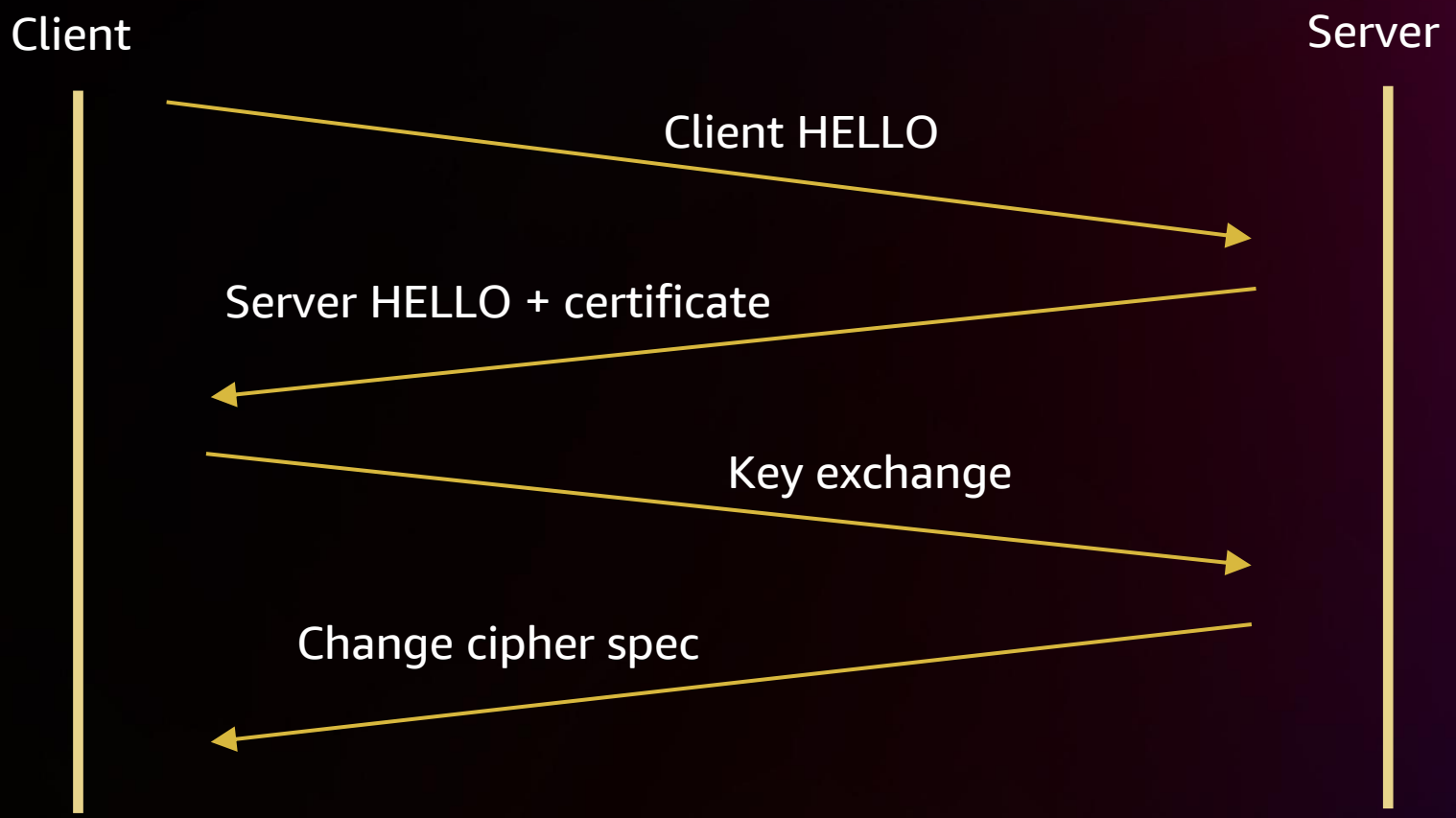
50ms median in USA

Why does TCP have to wait for SYN-ACK?

- SYN-flood (DOS) attacks
- Blowback attacks
- Servers **can't** be very trusting

- Security is lurking even at the TCP level
  - [TCP Fast Open tried to help . . .]

TLS adds another RTT+ after TCP connects



HTTP/1.1 supported pipelining

- (Try to) Reuse TCP/TLS connection
  - Minor problem: In-order-response
- Major problem: Broken implementations
  - Some servers clear input buffer after replying!?!
- No way to use it in the wild
  - Usable in datacenters (both endpoints controlled)
  - “In order replies” ➔ head-of-line (HOL) blocking

Circumventing TCP costs

- Multiple parallel HTTP (TCP) connections
- Servers don’t like being overloaded by impatient browsers
  - Limit browser to 6 connections per server (domain!)
- Big sites define “extra domains” (to hog bandwidth!)
  - www.example.com fetched 150 resources, 17 distinct domains
- Flows on one connection hurt other connections (latency variance!)

Standardization helps modernize TCP

- Initial congestion window bumped to 12–16 packets

. . . but still . . .

- Parallel streams battle for bandwidth
  - Variance in latency/packet loss is high
- HTTP resends client **header** info for **each** connection
  - Bandwidth is wasted, increasing congestion

HTTP/2 (SPDY) multiplexed streams

- Kudos to Mike Belshe and Roberto Peon!

- Each HTTP/GET or HTTP/PUT is a multiplexed stream
  - Large PUT won’t delay transmitting other GETs
- Responses are multiplexed/interspersed in any order
  - Servers respond ASAP (out-of-order!)

- Prioritize JavaScript and style sheet responses

SPDY (HTTP/2) removes redundancy

- Data compression of common headers
  - Send UserAgent string only once
  - Send cookies only once
  - Compress content across streams

- Streams share single flow congestion window
  - No fighting/variance among streams

HTTP/2 on TCP: Weaknesses

- TCP initial congestion window of 2 packets was terrible
  - Harmed HTTP/2 more than parallel HTTP/1
- 1–2% packet loss on internet
  - AIMD (additive increase, multiplicative decrease)
  - N% backoff harmed HTTP/2 more than HTTP/1
- In-order TCP delivery ➔ HOL blocking
  - SPDY used TCP, so kernel withheld packets
- TLS induced HOL blocking also (cipher block chaining)

QUIC: Quick UDP internet connections

TCP and TLS were holding HTTP back

- We wanted to avoid HOL blocking
  - TCP and TLS induced HOL blocking
- We wanted congestion control to evolve faster
  - Stop arguing with operating systems vendors
  - 5–10 year deploy/test is too slow

Was a new protocol feasible?

- Must handle packets separately
  - Critical to reduce HOL blocking
- Define a new IP protocol? (avoid UDP?)
  - Nope: Must use UDP

- Can clients reach web host servers using UDP?
- Network address translation (NAT) versus UDP?
- Load balancers versus UDP?
- Can servers handle high UDP loads?

Measure: UDP reachability for our customers

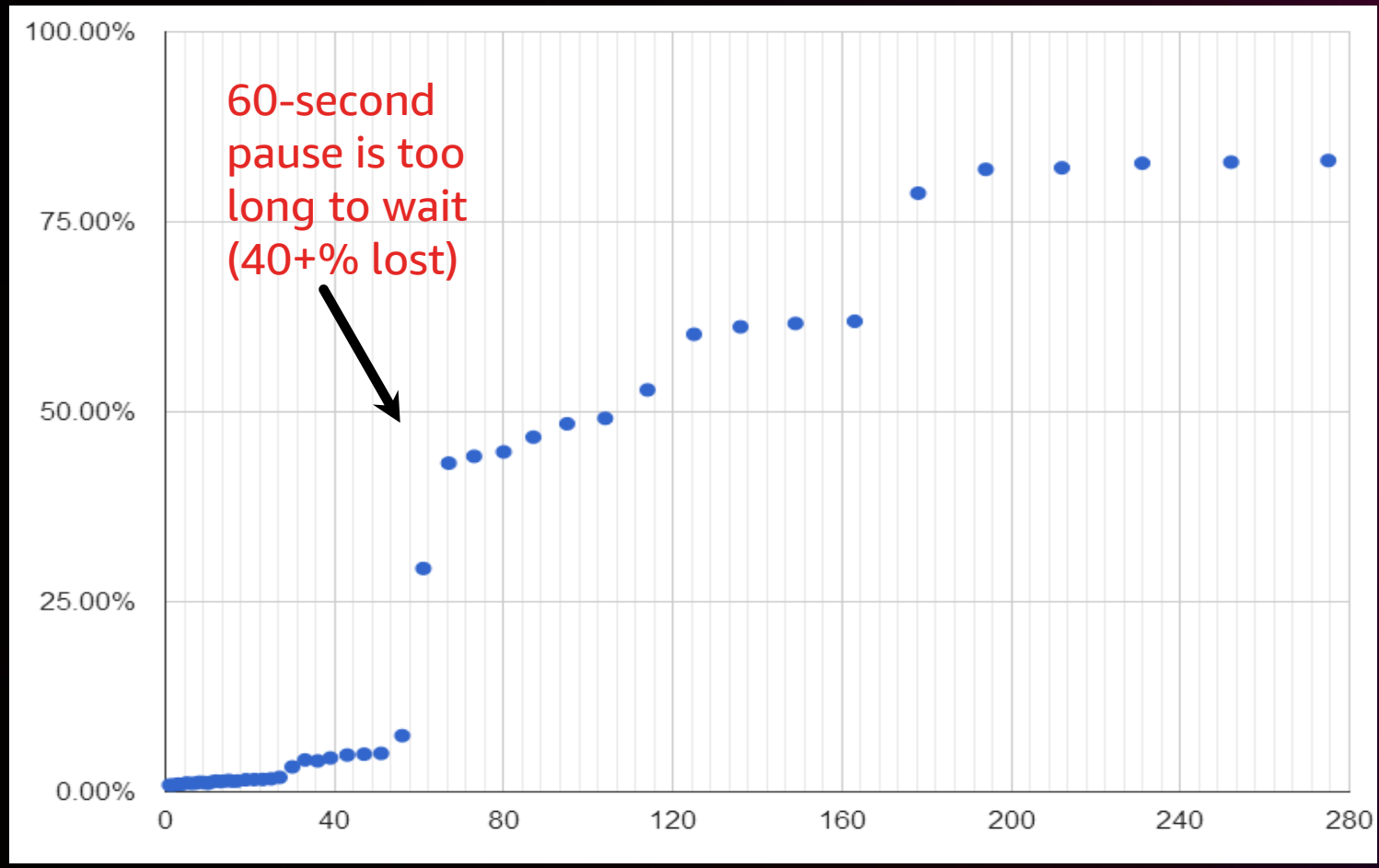
- Test reachability (around the world!)
- 93.5% of Chrome customers could reach Google via UDP
  - Fallback gracefully to TCP!
- “If you build it, they will come” (my optimistic mantra!)
  - We **must** create compelling advantages in QUIC

Measure: How do NATs handle UDP traffic?

- TCP has a FIN packet to drive NAT teardown

- NATs just “time out” and tear down UDP NAT binding
  - What is the timeout for actual customers??
  - How long can server pause/delay before NAT loses binding and packet is lost??
    - **We must test/measure**

Probability of losing server response



Seconds of delay before server attempts to respond

NAT timeout complications

- Must tolerate NAT timeout/rebinding
  - Need embedded connection ID

- Load-balancing complication

Can servers handle large UDP loads?

- TCP handling is highly evolved
  - TCP offload to hardware network interface cards (NICs) is common
  - Kernel hackers *assured* me UDP could be scaled up
- Load balancers would need to inspect connection ID
  - QUIC proof-of-concept needed before they would hack . . .
    . . . but they promised it could be done
- “Perpetual optimism is a force multiplier!” – Colin Powell

Can we bring value to customer?

- Can we (securely) start an encrypted stream faster?
- Can we befriend TCP congestion control?
- Can we improve on congestion control?
  - Reduce packet loss?
- Can we avoid HOL blocking?

Can we start an encrypted stream faster?

- Parallel discussion drove TCP FastOpen
  - Send a cryptographic token to attest to source IP ownership
  - Don’t let perfect be the enemy of the good
- TLS *rarely* changes server certificate
  - Speculate that certificate hasn’t changed!
- Measured: 75% of the time test servers accepted this
  - We **can** do better than TCP+TLS!

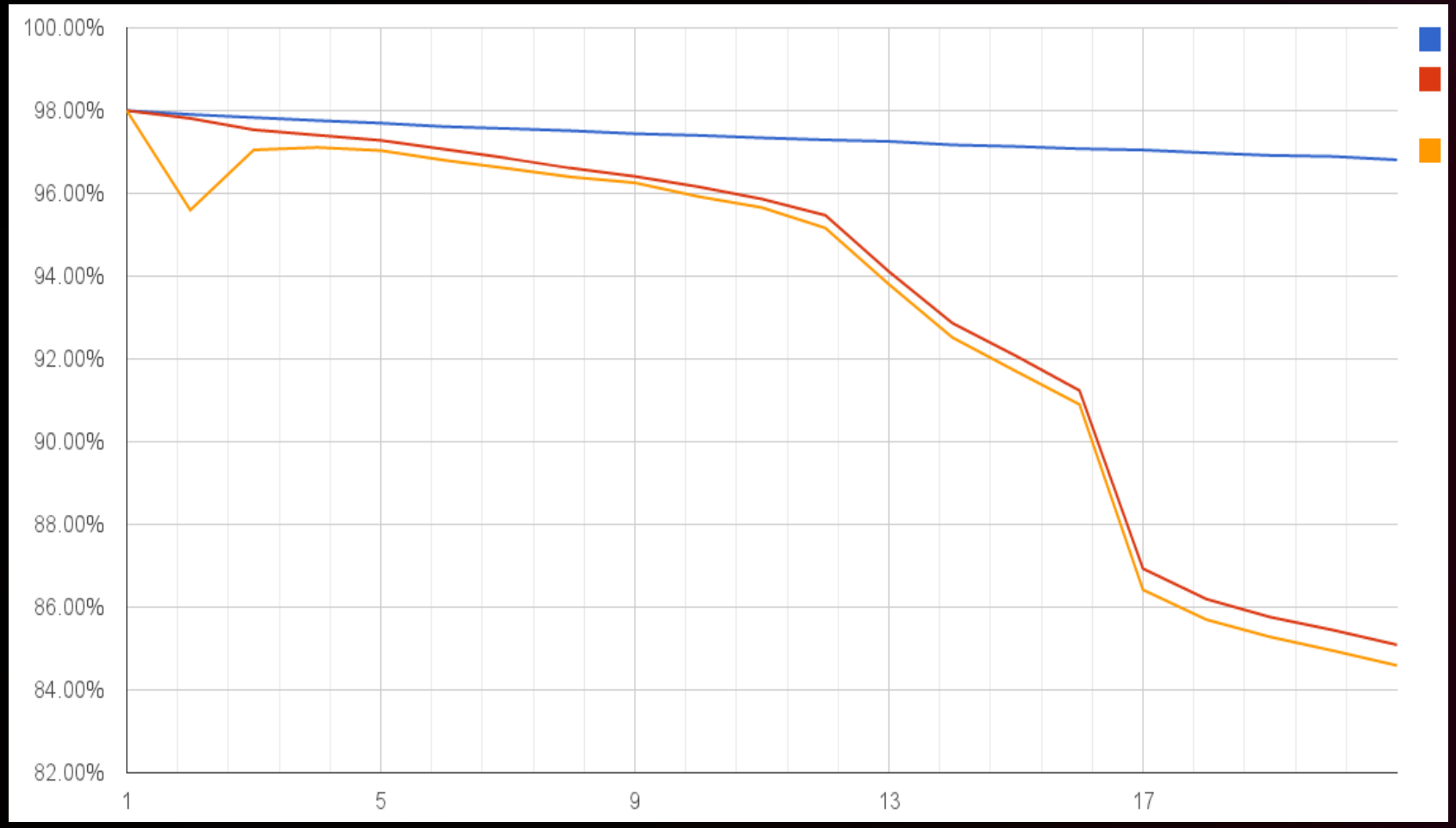
Can we befriend TCP congestion control?

- Rule 1: Don’t break the internet!
- Rule 2: If you help TCP, it won’t return the favor
  - TCP grabs all available bandwidth

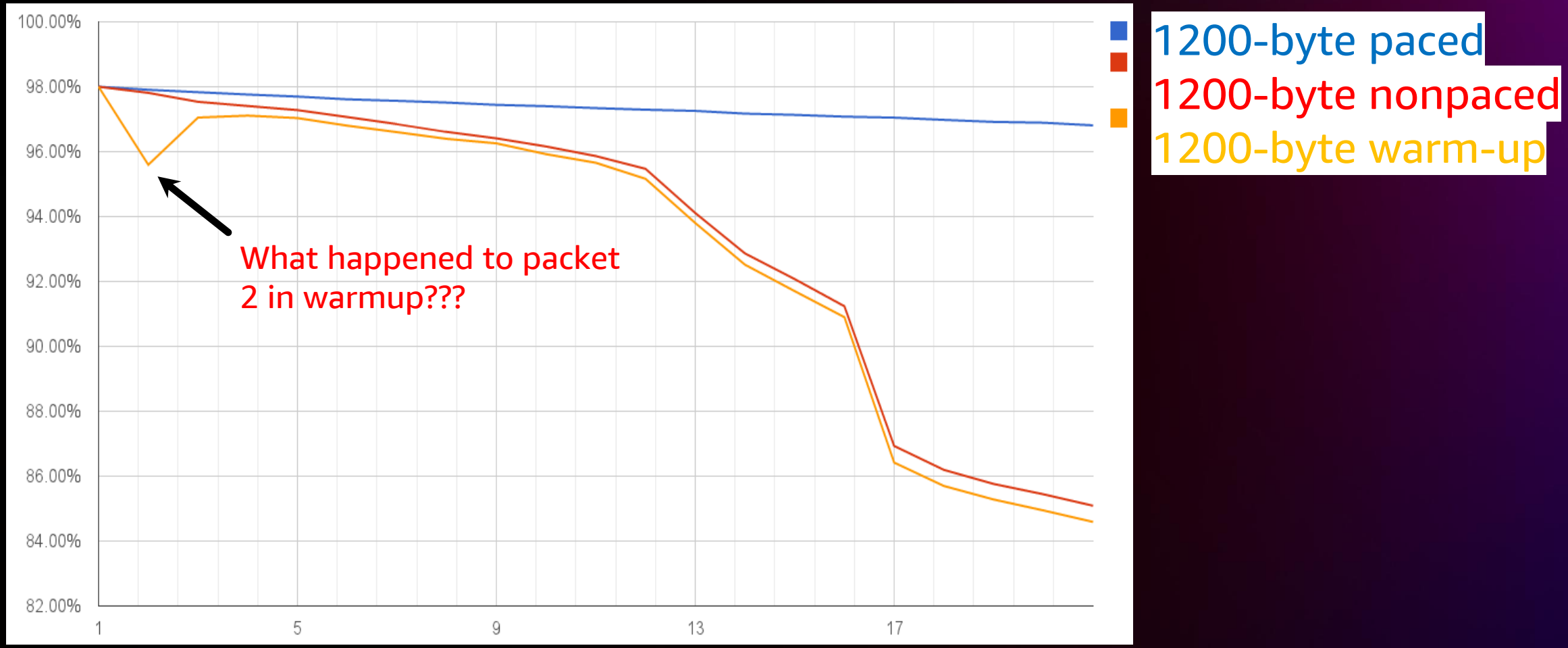
- Initial congestion control will use TCP style

- Why do we “burst transmit” congestion windows?
  - Spreading out traffic, or “pacing” might help . . .
  - But we need to measure . . .

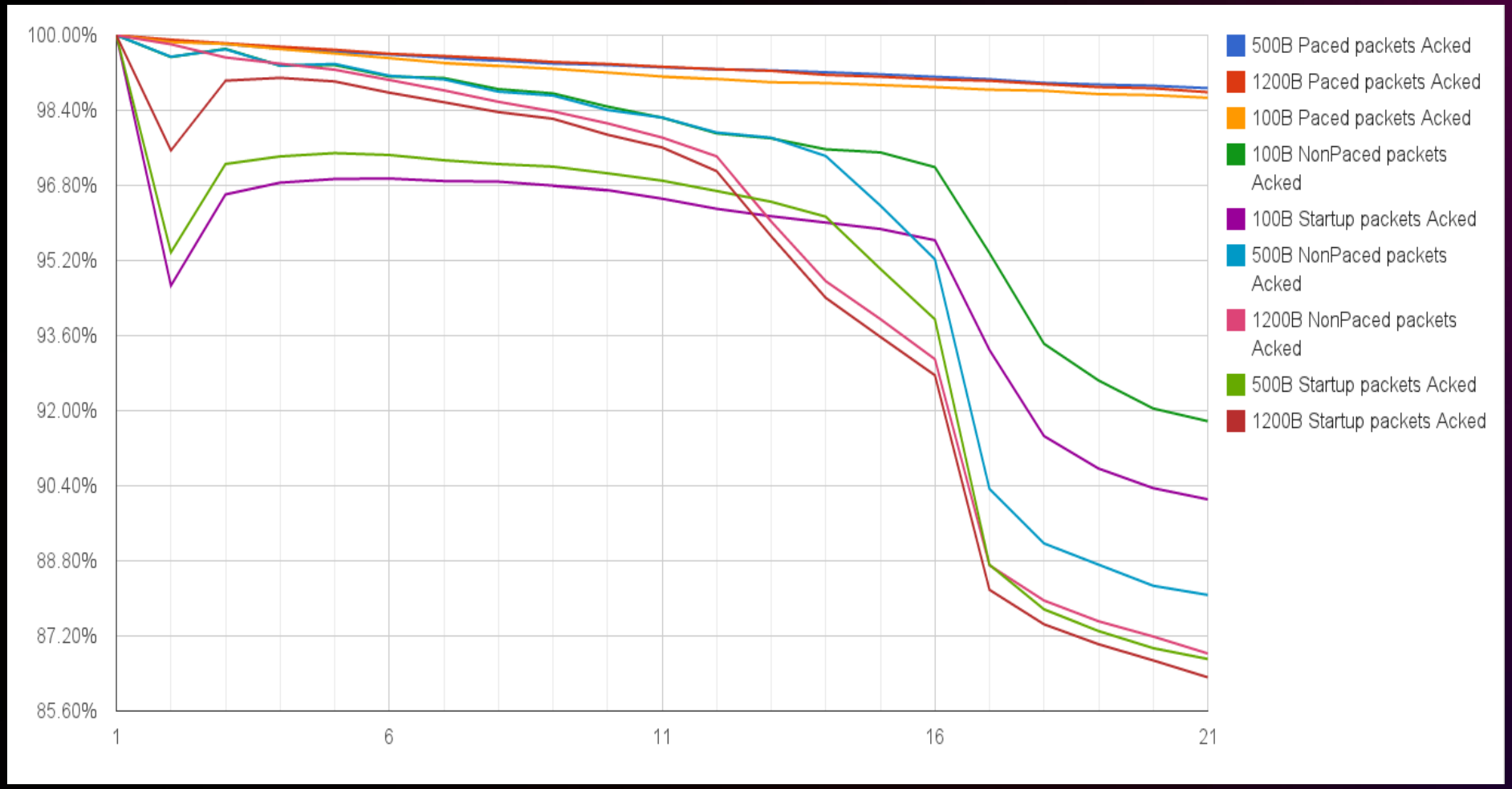
Paced vs. streaming: Probability of ACK



Paced vs. streaming: Probability of ACK

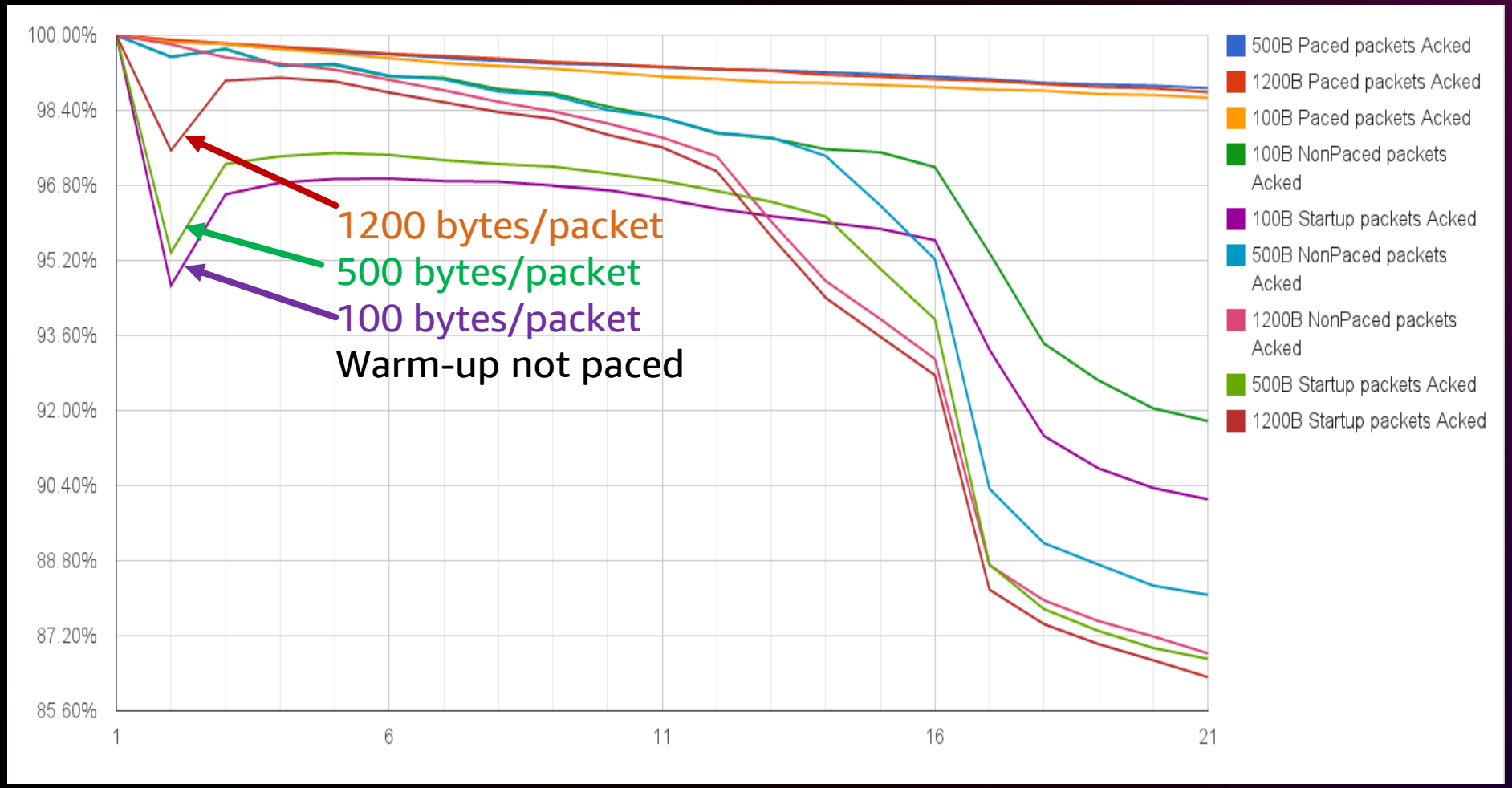


Relative packet-ACK probability



Compare 1200- vs. 500- vs. 100-byte packets; paced vs. nonpaced vs. warm-up

Relative packet-ACK probability



Compare 1200- vs. 500- vs. 100-byte packets; paced vs. nonpaced vs. warm-up

QUIC: Loose ends

QUIC from 50,000 feet: Adopt, migrate, use

- How browsers discover QUIC support
- Clients re-test QUIC viability
- Faster than TCP+TLS connection using speculation

- Limited blast radius for a lost packet
- Integrated encryption with packet sequence numbers
  - Avoid cipher-block chaining

Browser discovery of QUIC server support

- Prior HTTPS over TCP can indicate QUIC support
- Browser (Chrome) remembers the offer
  - Browser remembers Server’s public key certificate
  - Browser remembers signed IP-address-used token

- Future connections go faster than the first one
  - Browser “locally optimizes” for customer’s needs

Can QUIC reach the server each day?

- Browser races QUIC HELLO against TCP/TLS connect
  - If there is no UDP response, TCP proceeds

- If QUIC connection works, TCP is abandoned

What happens after QUIC HELLO?

- Packets and streams are bidirectional
  - ACKs piggyback with real data
  - Congestion control algorithms will evolve

Head of line blocking: No more

- Streams generally honor UDP packet boundaries
  - Packet loss (resend) usually slows only one stream
  - Multiplexing better than HTTP/2
    - Impossible in TCP, but explicit with QUIC/UDP
- Packets are individually numbered/encrypted
  - No cipher block chaining (CBC)
  - Each packet decrypted separately!
- HOL blocking has minimal blast radius
  - YouTube reported big improvement in rebuffer rate

How are packets acknowledged?

- Packets are ACKed in a control stream
  - All packets are encrypted
  - Malicious third party can’t inject a TCP-style FIN
- Selective ACK can hitchhike with “real data” packets
  - Third party can’t even deduce packet-loss stats!
  - TLS running on TCP exposes ACKs, creating vulnerabilities
    - Optimistic ACK attack anyone?

How are packet losses handled?

- QUIC “rebundles” lost **data** in new packet
  - QUIC never retransmits lost **packets**
  - QUIC can better understand packet losses
  - TCP retransmits **all** lost packets

- Constantly advancing packet numbers helps!
  - Congestion/receive window limits outstanding numbers

Results: IETF QUIC emerged as HTTP/3

- Google QUIC supported 90% of all Chrome traffic
  - Years of testing worldwide
  - AWS, Google, and others implement IETF QUIC standards
- Faster connection, reduced latency variance
  - Customers around the world benefit

Summary of QUIC, and now HTTP/3

What is HTTP/3?

- Next generation HTTP protocol and more
  - Replaces TCP+TLS+HTTP/2 using UDP packets
- Provides guaranteed delivery
- Faster cryptographic connection establishment
- Multiplexing of streams
  - One congestion-controlled flow
  - Improved latency with reduced variance
- Evolving congestion control
- Amazon CloudFront supports IETF QUIC, as do browsers!

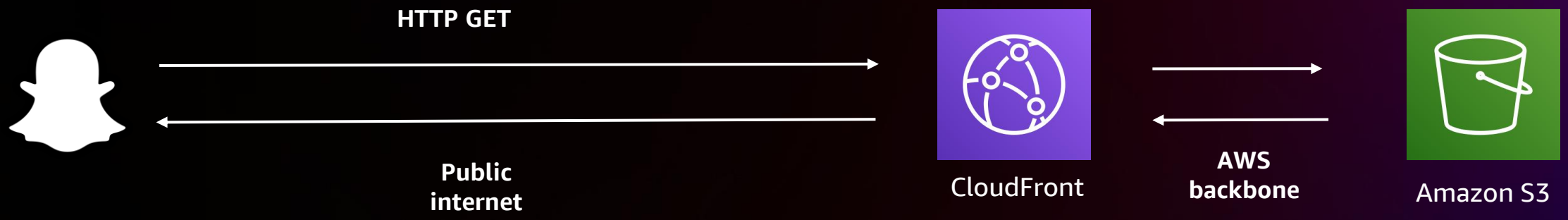
Snap deployment

- Snap content delivery
- Snap and QUIC
- Testing at scale
- Challenges
- Results
- Snap lessons learned and next steps

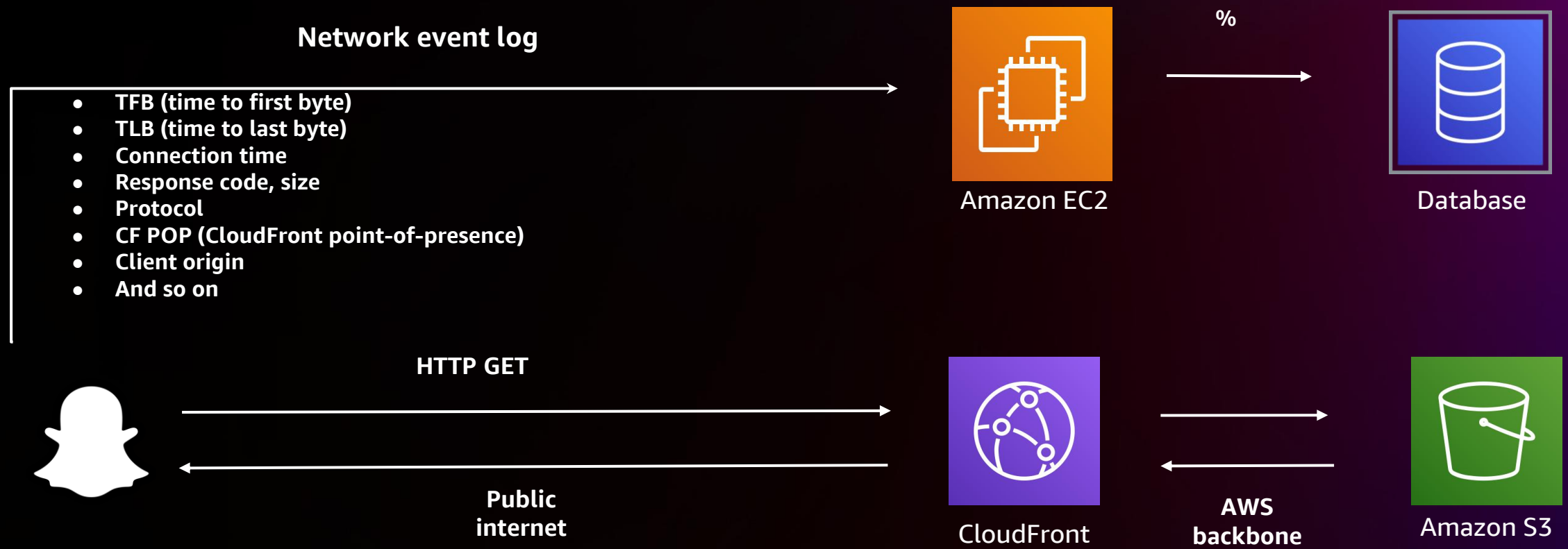
Snap background

- Snapchat aspires to be the fastest way to communicate
- Snap has a long term investment in QUIC
- Amazon CloudFront is a core infrastructure to Snap
- Amazon CloudFront HTTP/3?
  - Yes, please!
- 330M+ use Snapchat every day
- New technology needs validation

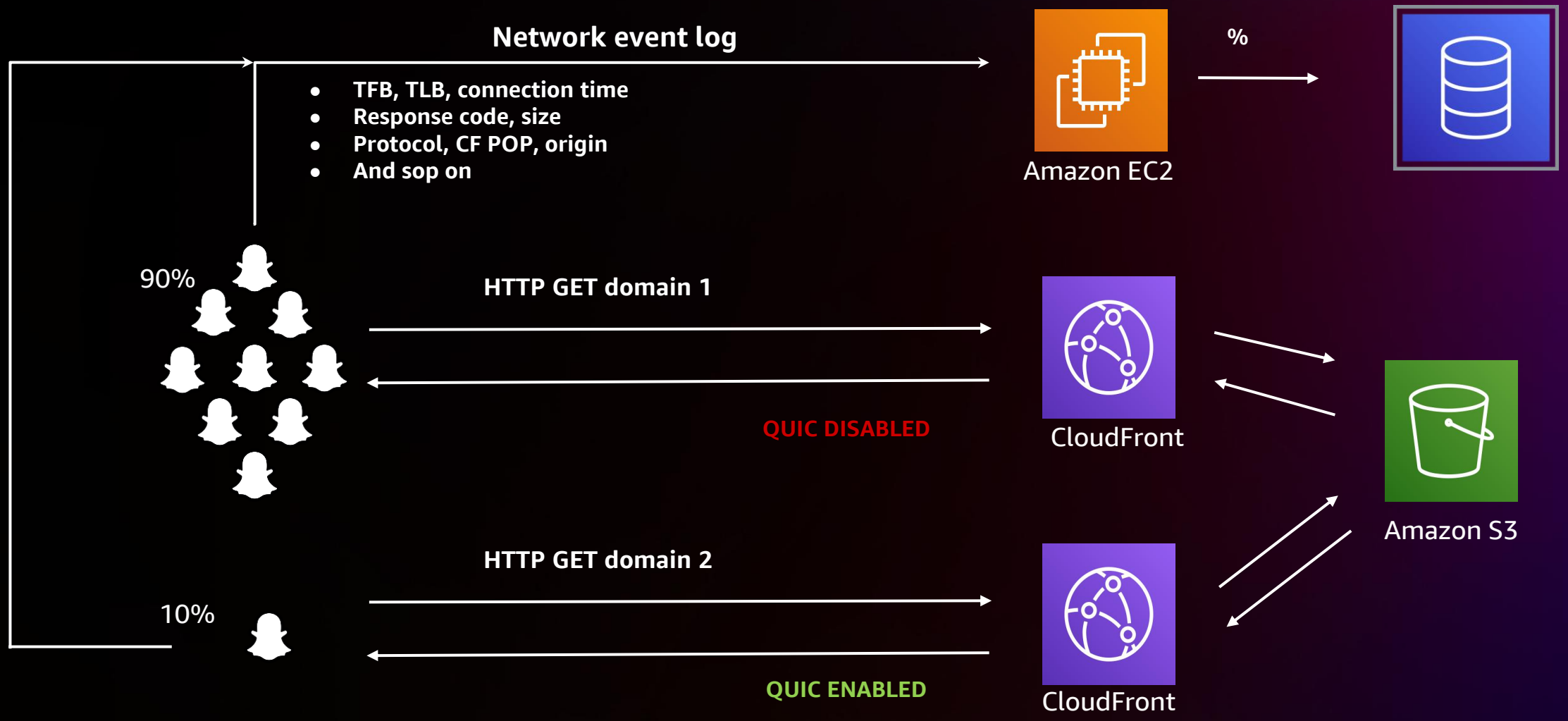
Snap media download



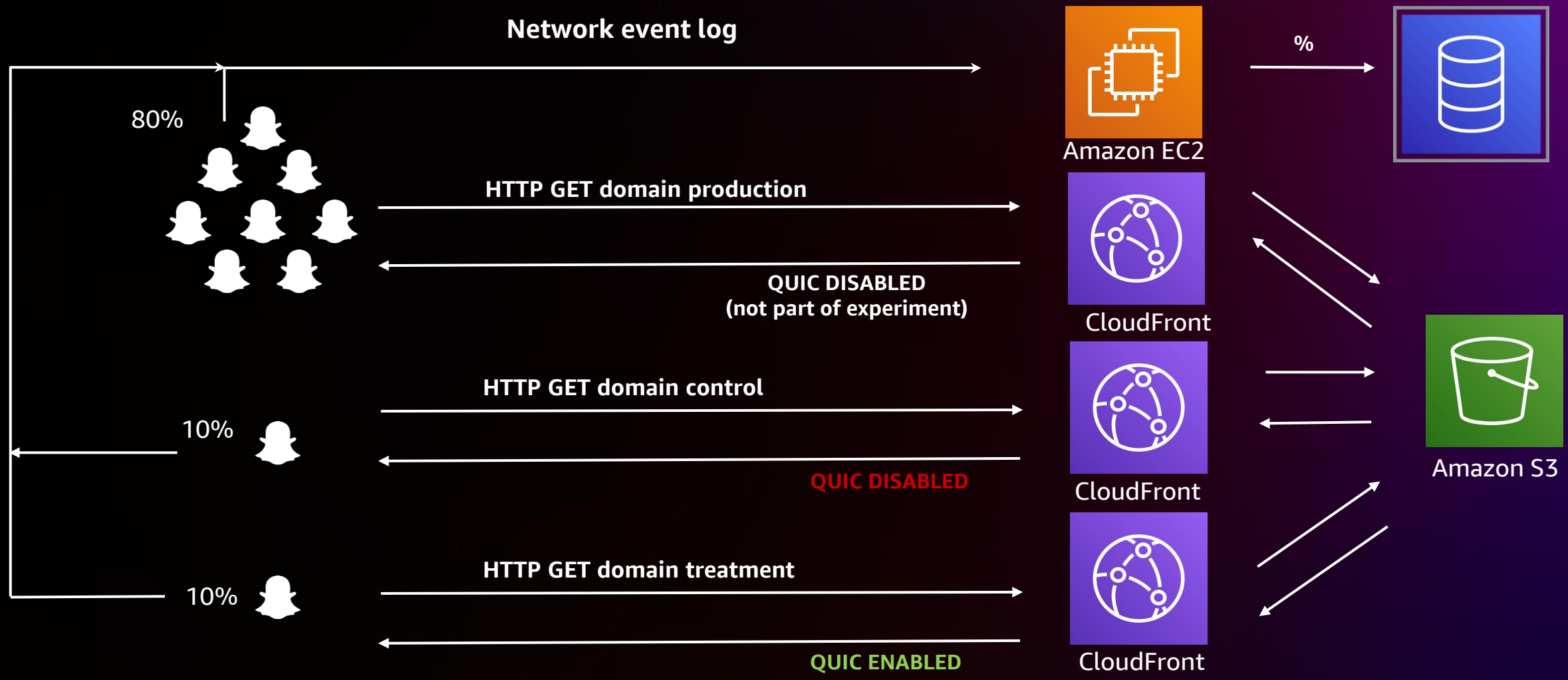
Snap media download



How to test QUIC (production mirroring)



How to test QUIC (counterfactual)



It’s not that simple!

- Autonomous service networks
  - ~10k ASNs
- Customer platforms
  - Android?
  - iPhone?
- Client libraries
  - NSURL?
  - Cronet?
  - OkHttp?

Results for Snap’s test

- Trimmed mean (99%) [average, excluding worst 1%, which is just noise]
  - Time to first byte 6–10% faster
  - Time to last byte is 6% faster

Lessons learned

- New technology is cool, but always be careful
- Slice and dice carefully
  - iOS versus Android?
  - High end versus low end?
  - ISP1 versus ISP2
  - Object size
- Representative sampling
- Iterate!

For more information

- HTTP/3 Support in Amazon CloudFront:
  - https://go.aws/3UJX0YX
- “QUIC: Design Document and Specification Rationale” by Jim Roskind
  - 70-page design spec, cut down to a mere 40 pages for publication in 2012
- Search for “IETF QUIC” and read latest RFC and standardization documents
- For more info on “trimmed mean” and better latency metrics to improve your latency – attend/view AMZ302 this Wednesday

Thank you!

Please complete the session survey in the **mobile app**