QUIC

Quick UDP Internet Connections

Multiplexed Stream Transport over UDP

IETF-88 TSV Area Presentation
2013-11-7

Presentation by
Jim Roskind <jar@>
Google Corp

What is QUIC?

Effectively replaces TLS and TCP out from under SPDY (predecessor of HTTP/2.0)

Provides multiplexed in-order reliable stream transport (especially HTTP) over UDP

Protocol is pushed into application space (unlike TCP which is handled in kernel)

Overview

- Why aren’t current protocols enough?
  - e.g., SPDY multiplexes streams, doesn't it?
- What could make QUIC valuable?
  - What could make it better(?) than SPDY?
- Status of efforts?

Why is SPDY fast?

- It is all about *latency (time till response)*
- SPDY multiplexes requests over one TCP connection
- SPDY compresses headers
- What is the problem?

Why isn't SPDY Enough?

- SPDY runs over TCP
  - Lose one SPDY packet: all the streams wait
    - HOL blocking
  - Lose one SPDY packet, bandwidth shrinks
    - Sharded connections have an advantage!!!
- SPDY may be slow to connect
  - TCP connect may cost 1-Round-Trip-Time (RTT)
  - TLS connect costs at least another RTT
- TLS and TCP are slow to evolve
  - More importantly, they are very slow to deploy
    - (at both ends, and in middle boxes!)

QUIC Goals

1. Deploy in today's internet
2. **Low latency** (connect, and responses)
    a. It is *ALL* about the latency
3. Reliable-stream support (like SPDY)
    a. Reduce Head Of Line (HOL) blocking due to packet loss
4. Better congestion avoidance than TCP
    a. Iterate and experiment
5. Privacy and Security comparable to TLS
6. Mobile interface migration
7. Improve on quality of sliced bread

QUIC Success Criteria

The Internet is faster and more pleasant to use

Two paths:

a) QUIC makes headway reducing latency

b) TCP and TLS steam ahead, and perhaps use techniques advocated for QUIC

Either way: The users will win.

Field Data, Plus Application Needs Drive Development

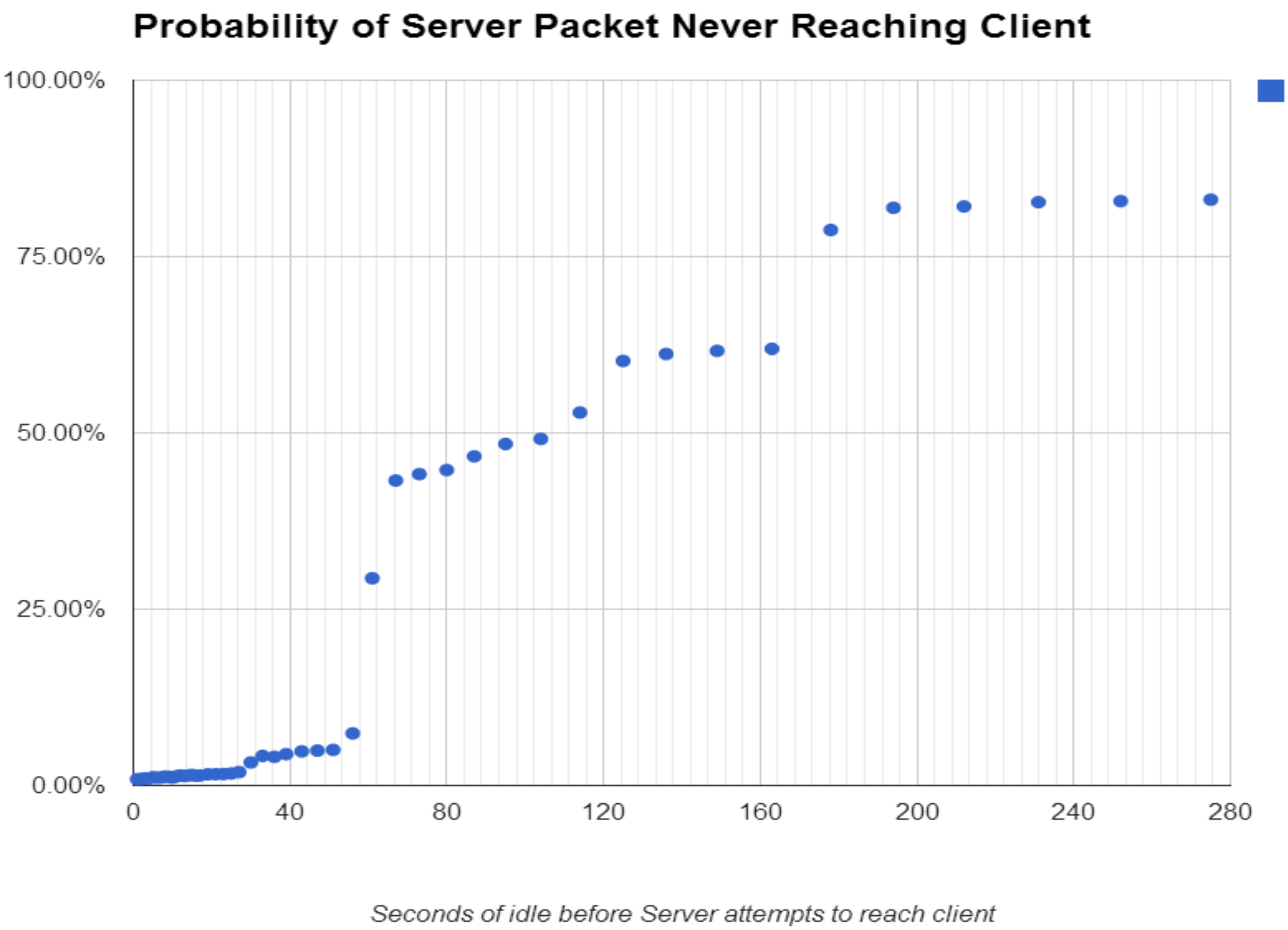
- Client side data from Chrome
  - Real users; Real user machines; Real cross traffic; Real ISPs
  - Aggregate data, and perform A/B experiments
- Server side instrumentation evaluates application impact
  - User happiness drives everything

Can we really Deploy a UDP Protocol in Today's Internet?

- UDP works for gamers and VOIP
  - They really care about latency
- 91-94% of users can make outbound UDP connections to Google
  - Tested for users that had TCP connectivity to Google

- UDP is plausible to build a transport in today's internet
  - See NAT Unbinding data in supporting slides

NAT Unbinding:
How much idle until unbinding?



How does QUIC achieve 0-RTT Connection Cost?

- Speculate that the server's public key is unchanged since last contact
  - Propose session encryption key in first packet
- Include GET request(s) immediately after
  - Upgrade to Perfect Forward Secrecy ASAP

- Similar speculative techniques tried/developed in TLS and TCP
  - See crypto doc for fancy details
  - See supporting slides for some highlights

Congestion Avoidance via Packet Pacing

- QUIC has pluggable congestion avoidance
  - TCP Cubic is baseline
  - Working on Pacing *plus* TCP Cubic
  - Working on bandwidth estimation to drive pacing
- QUIC monitors inter-packet spacing
  - Monitors one-way packet transit times
  - Spacing can be used to estimate bandwidth
  - Pacing reduces packet loss

Does Packet Pacing really reduce Packet Loss?

- Yes!!! Pacing seems to help a lot
- Experiments show notable loss when rapidly sending (unpaced) packets
- Example: Look at 21st rapidly sent packet
  - 8-13% lost when unpaced
  - 1% lost with pacing
- See supporting slides on “Relative Packet ACK probabilities” for some details

How might QUIC connection survive a Mobile Network Change?

- TCP relies on src/dest IP/port pairs
  - Mobile client (changing network/IP) means broken TCP connection :-(
  - Broken TCP connection means big reconnect latency
- QUIC relies on a 64 bit GUID in all packets
  - Client source IP is used only to respond to the mobile client
- ...and of course with QUIC, if we lose....
  - ...then fast 0-RTT reconnect is a fallback

How can a Forward Error Correction (FEC) Packet help?

- Trade increased bandwidth for decreased latency

- QUIC has packet level Error Correction
  - Keep a running-XOR of (some) packets
    - Send XOR as an Error Correction packet

- ...but is packet loss bursty?
  - XOR Error Correction won't work if we have several consecutive losses!

Will Error Correction Coding really help?

- Packet loss is not that bursty :-)

Example:

- 20 packets with about 1200 Bytes each
  - Retransmit needed 18% of the time
- 20 packets plus FEC Packet
  - Retransmit needed 10% of the time

- 5% extra bandwidth ==> -8% retransmits

See support slide on Retransmit Probabilities for 1200B payloads for experimental data

Status of Efforts (11/2013)

- Currently landing, limping, and evolving
  - In Chrome and in some Google servers
  - Trying to work as well as TCP Cubic
  - FEC built in… but not turned on
  - 0-RTT works when same server is hit
- Try prototype Chrome canary
  - about:settings Enable QUIC :-) (must restart)
  - about:net-internals to look at activity
- Try test-server in chromium codebase
- Longer road to Crypto PCI compliance for handling credit cards :-(

How can I contribute?

News group: proto-quic@chromium.org

https://groups.google.com/a/chromium.org/d/forum/proto-quic

Contribute to Chromium source tree!

Evolving wire spec tries to record state-of-the-Chromium-tree for landed code

...but debugging often drives changes

Design document has motivations and justifications

FAQ For Geeks addresses some questions

Backup Slide areas:
Other Nice Stuff in QUIC Design

1. Defend against "Optimistic ACK Attack"
   a. Defend against Amplification Attacks
2. Handle header compression (like SPDY)
   a. ...despite out-of-order arrival of packet (context)!
3. Support TCP Congestion Avoidance
   a. Baseline: prevents Internet Congestion Collapse
4. 0-RTT Server-side redirects
   a. Hand off service to other server without an RTT, while getting good crypto in that new server's stream

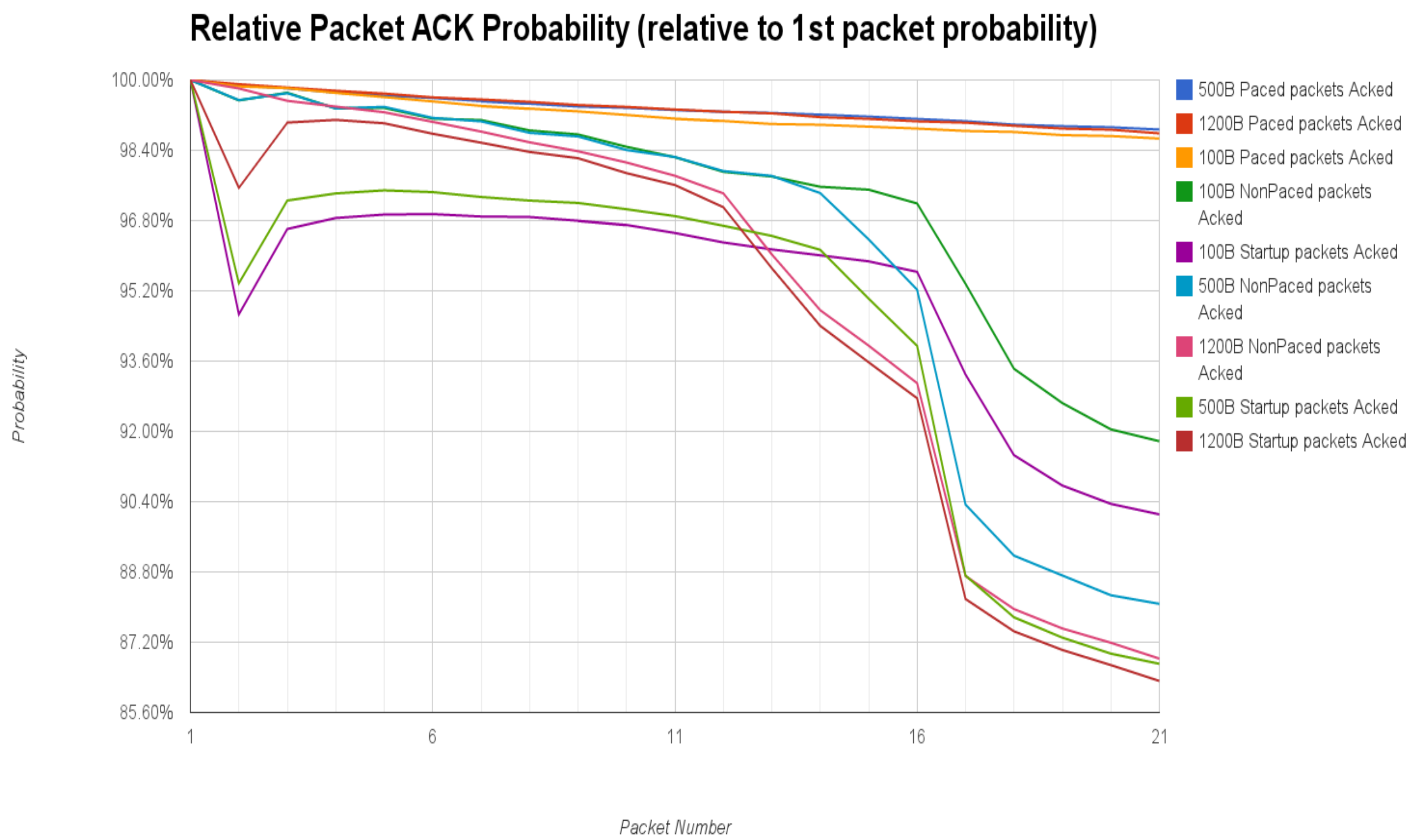
More Nice Crypto Things In QUIC

1. Encryption used for both UDP port 80 (HTTP) and for port 443 (HTTPS)
    a. … but port 80 does not authenticate server
2. Connections upgrade to Perfect Forward Secrecy asap
    a. After about 1 RTT
3. Packet padding to make traffic analysis a tad harder
4. FIN (like) and ACK packets authenticated
    a. No 3rd party teardown

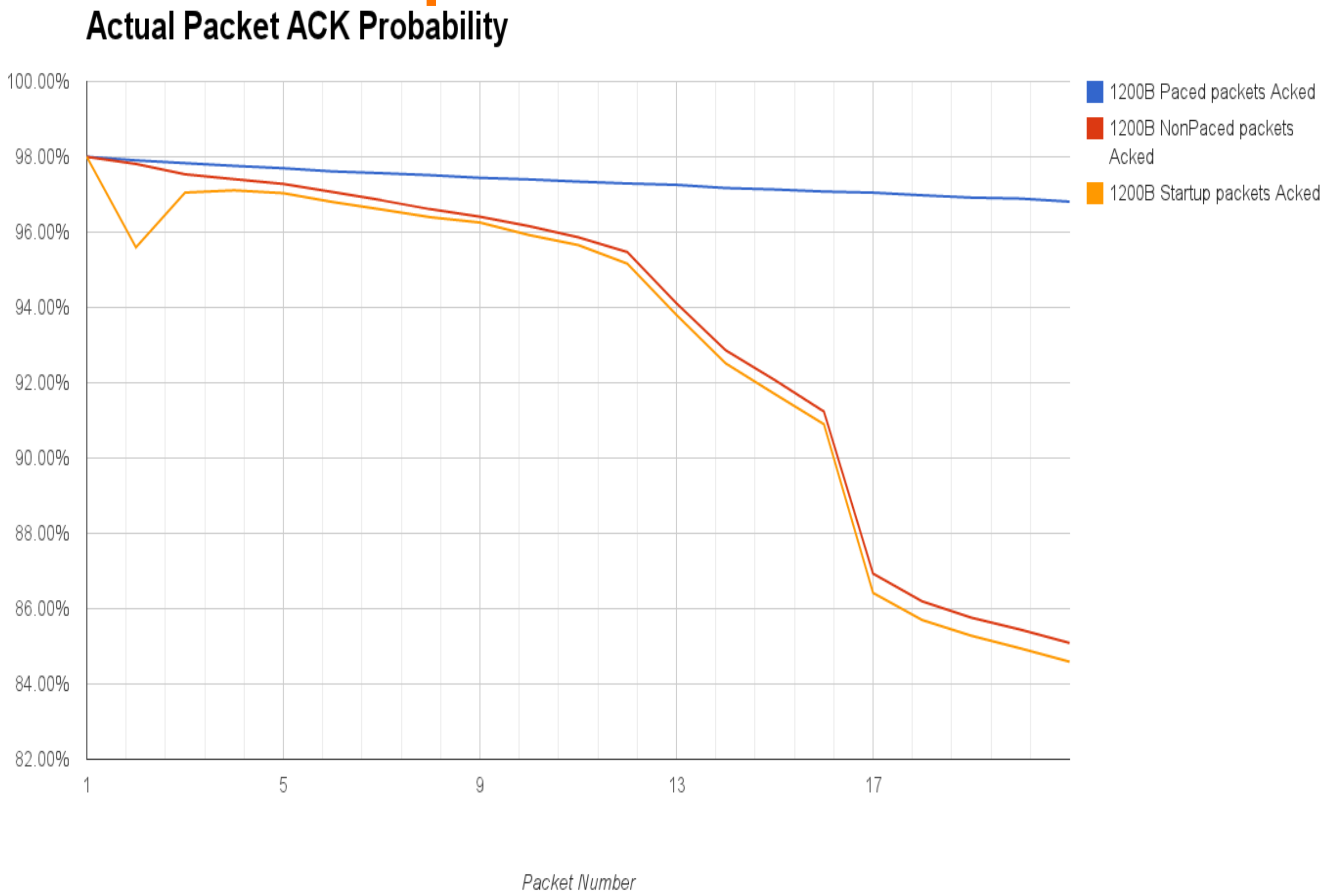
How is HOL Blocking Reduced?

- UDP is not in-order, like TCP
  - QUIC adds packet sequence numbers
- SSL crypto block depend(ed) on the previous block's decryption
  - QUIC uses packet sequence numbers as crypto-block Initialization Vector (IV) source
  - QUIC collapses and reuses protocol layers!
- SSL encrypted blocks don't match IP packet boundaries :-(
  - QUIC aligns encryption blocks with IP packets
  - One lost QUIC packet won't stop the next packet from being decrypted :-)

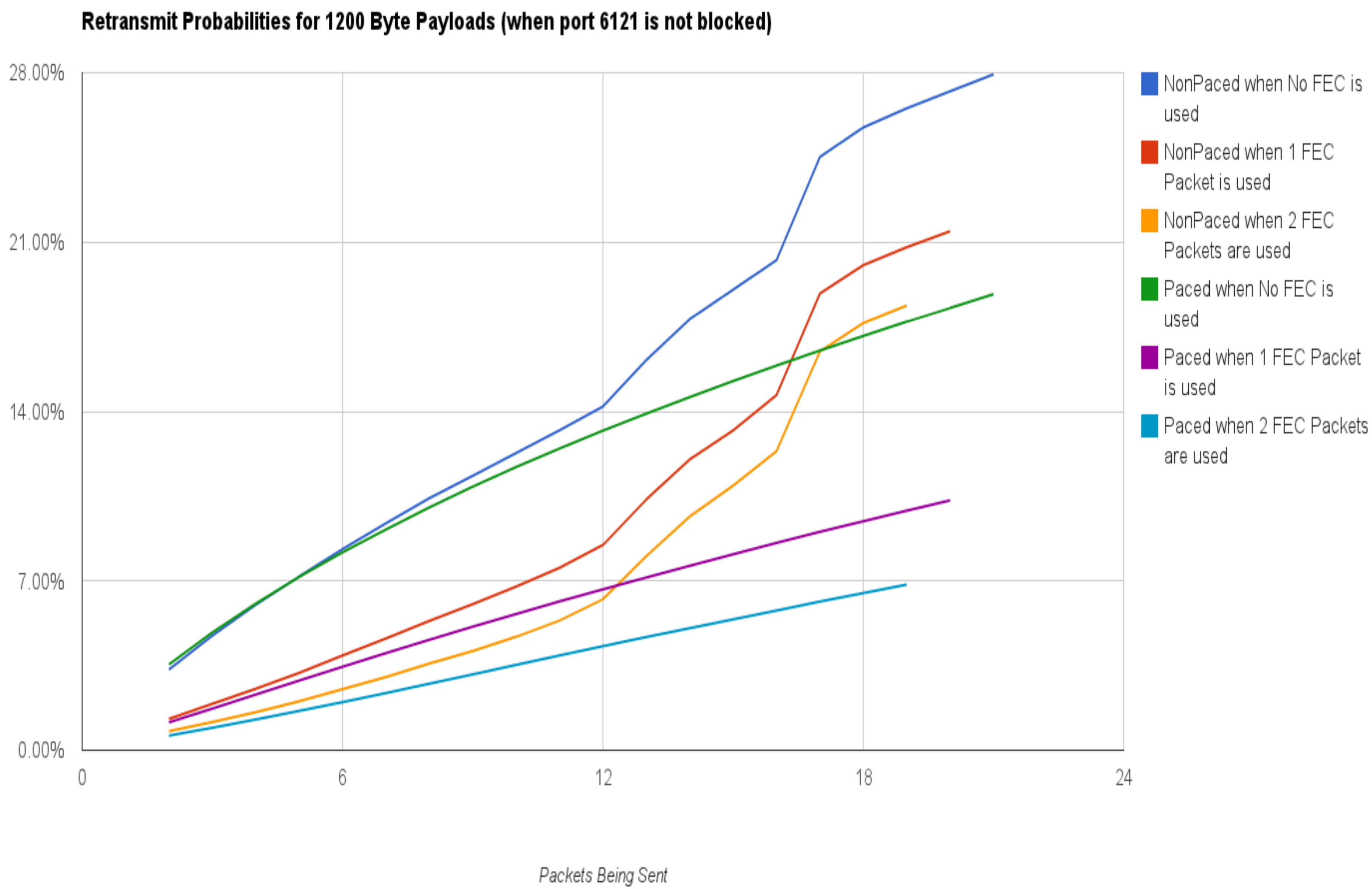
Client->Server->Client round trip:
21 packets: 1200B vs 500B vs 200B



Round trip ACK Probabilities



Retransmit-needed Probabilities



NAT Unbinding Results Caveats

1. Did not correct for 1% ambient packet loss
   a. Could have sent N packets after pause
2. Did not validate internet connectivity
   a. Users may have disconnected… so there is some disconnection conflation
3. Did not test to see if NAT was being used
   a. IPv6 *often* avoids NAT
   b. This could explain the 20% tail that “never”(?) unbinds