QUIC @ Microsoft

EPIQ 2021, by Nick Banks

Lots of uses for QUIC

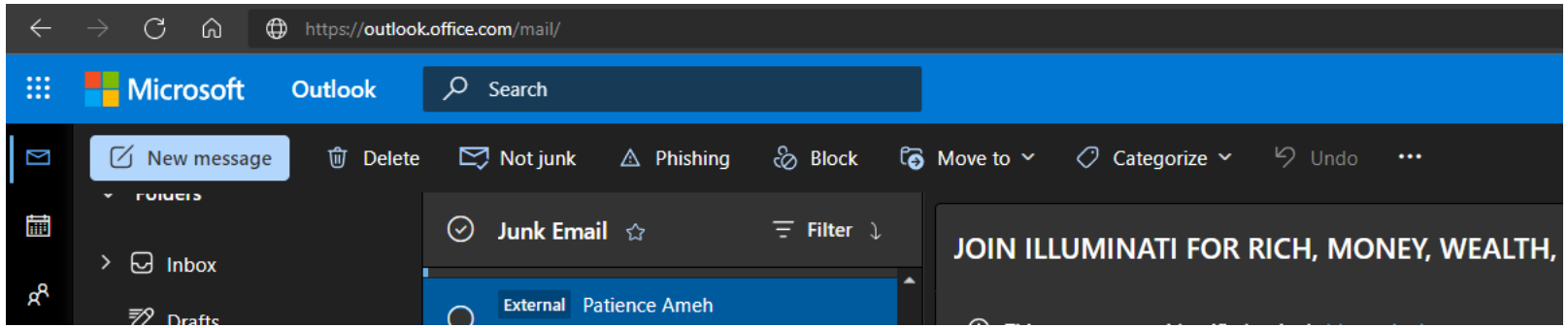
Products

- MsQuic (cross platform, FOSS)
- Windows Server 2022
  - HTTP/3 – Server
  - SMB over QUIC – Server (in Azure Edition)
- Windows 11
  - SMB over QUIC – Client
- .NET 6 (Windows & Linux)
  - HTTP/3
- Xbox Game Core
- External Partners
  - NDI, Visuality, DreamWorks and other large media companies

Exchange Online

Data from HTTP/3 usage to https://outlook.office.com

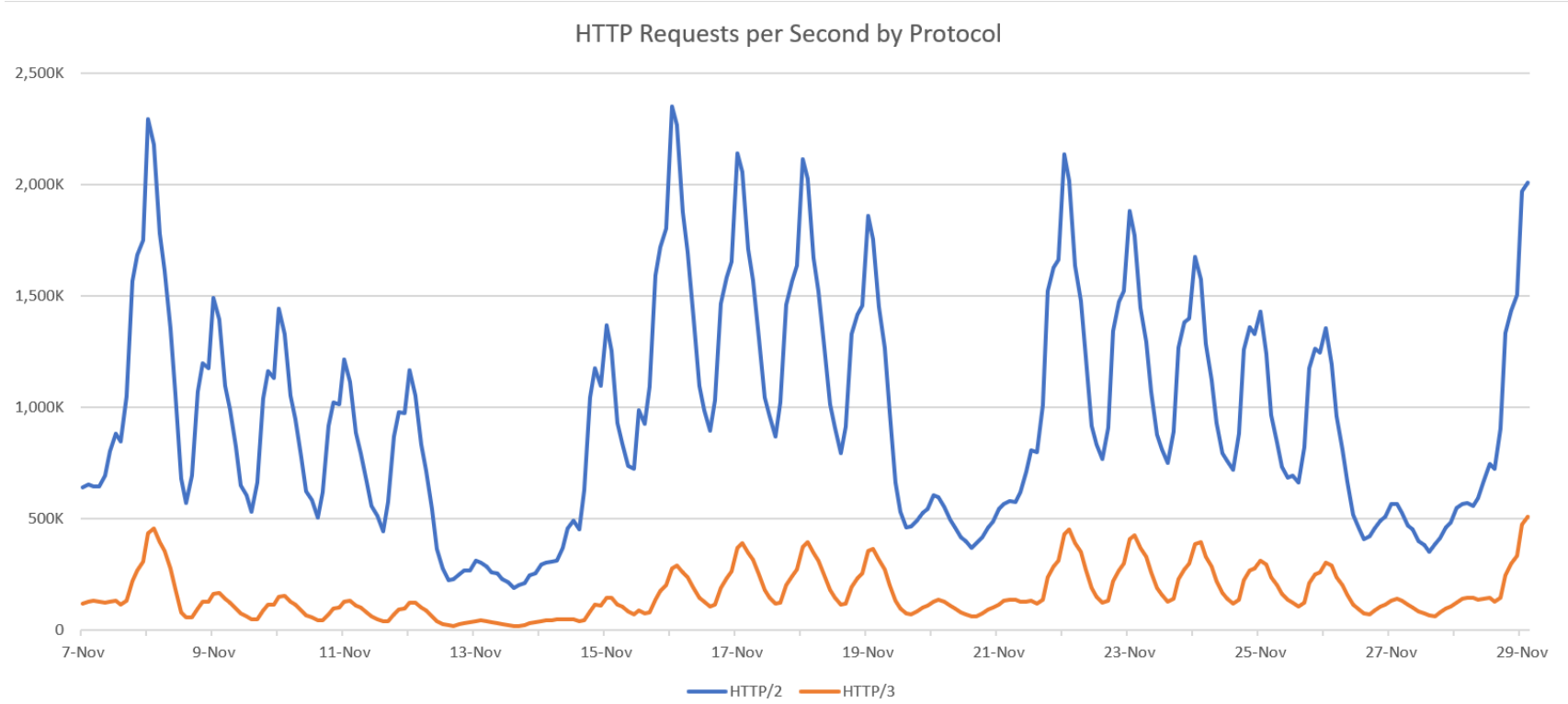
Exchange Online Scenario



- Mail application used by millions of corporate and individual users every day
- Network scenario:
  - Longer lived connections
  - Smallish requests (~8 KB)
  - Variable sized responses (most common 2 KB to 32 KB)
- We will be comparing HTTP/2 to HTTP/3

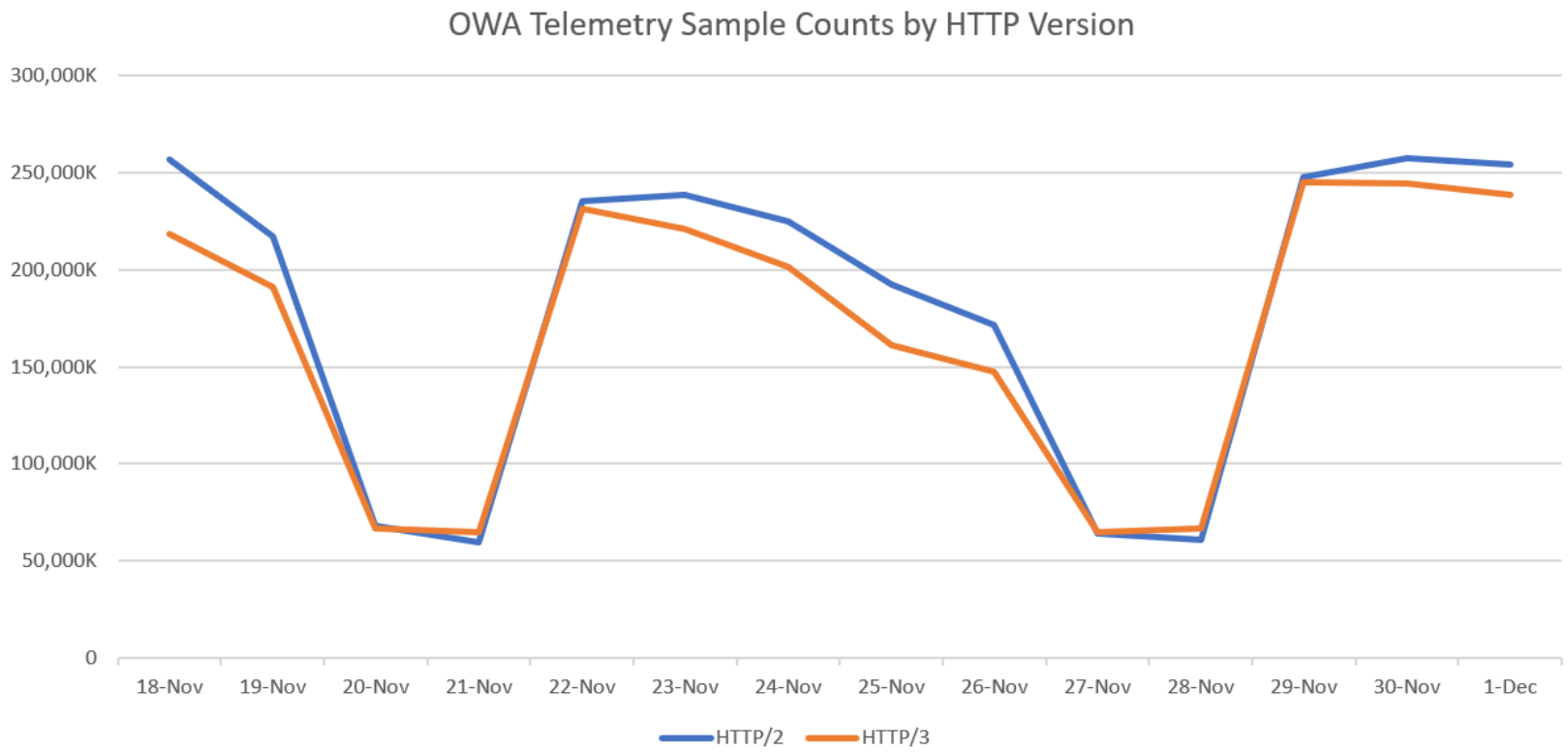
Exchange Online Usage

- 70% of worldwide front-end servers deployed latest Windows Server with HTTP/3 support
- Chart below shows **all** EXO H2/H3 usage; including browser, mobile and desktop clients

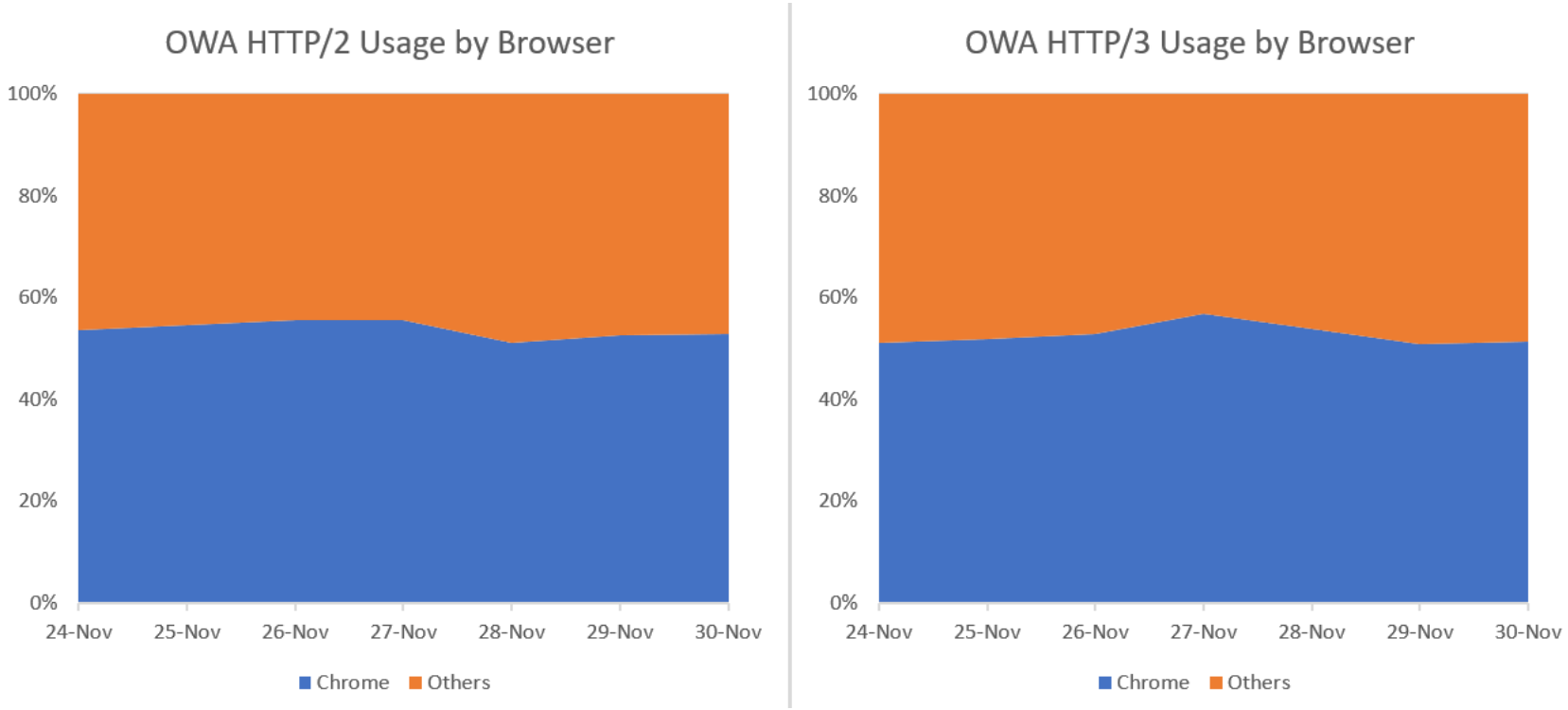


OWA Usage

- Outlook Web App data comes exclusively from **browser clients** visiting URLs such as outlook.office.com
- Chart below is OWA telemetry from latest deployment machines.
- It shows an **even usage** between HTTP/2 and HTTP/3.

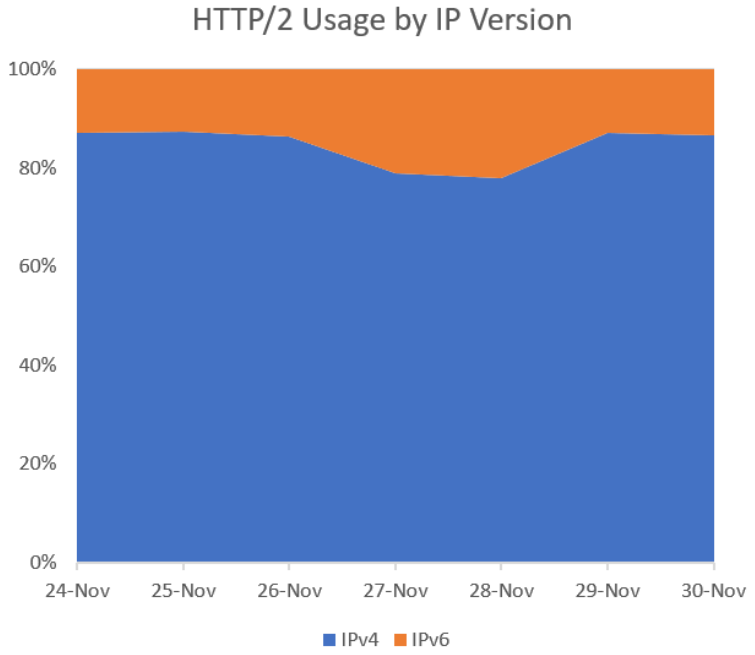


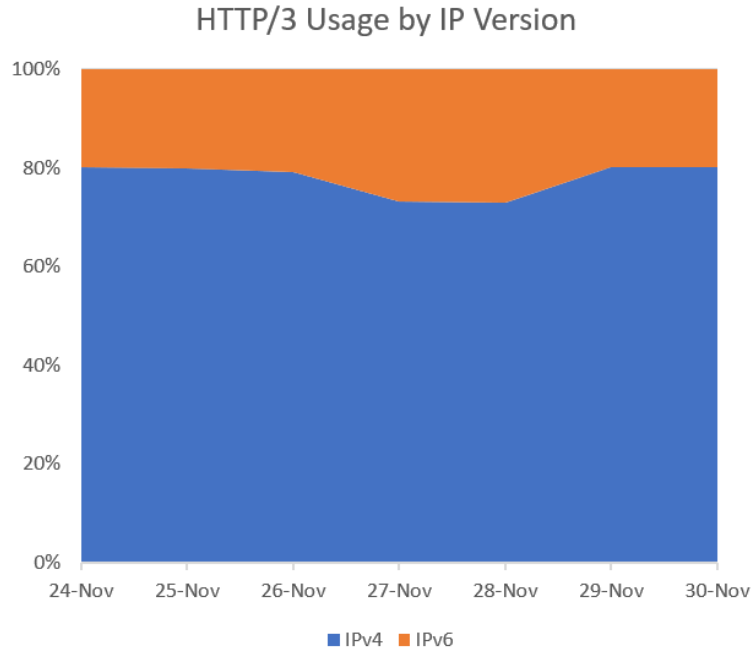
OWA Usage (Browsers)

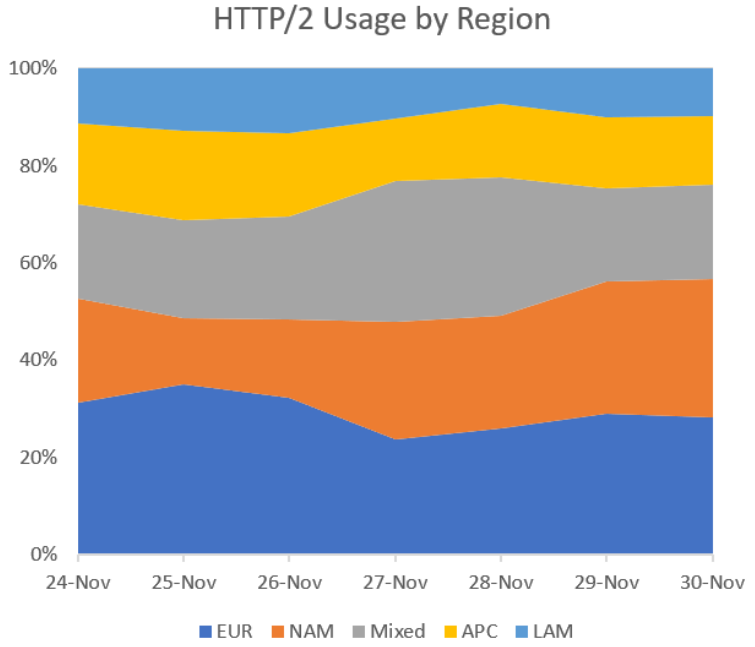


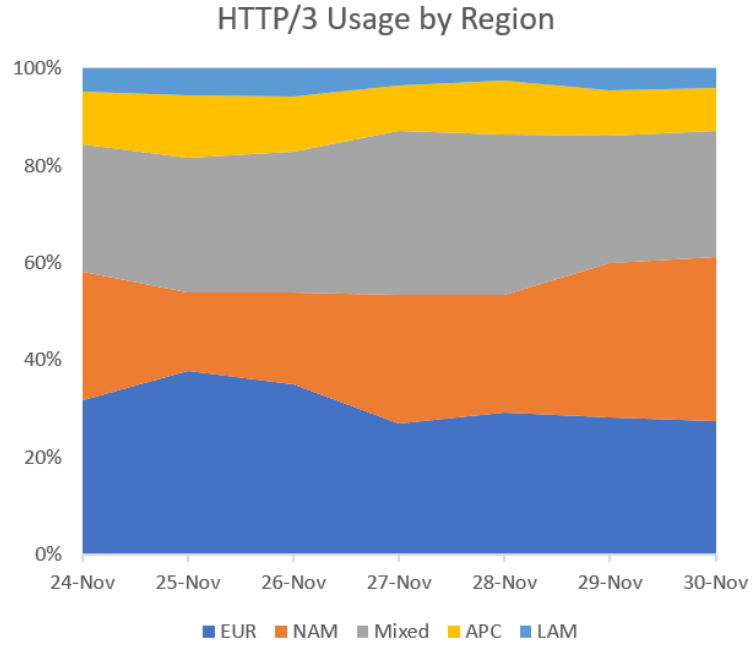
- Similar usages across browsers

OWA Usage (IP & Region)









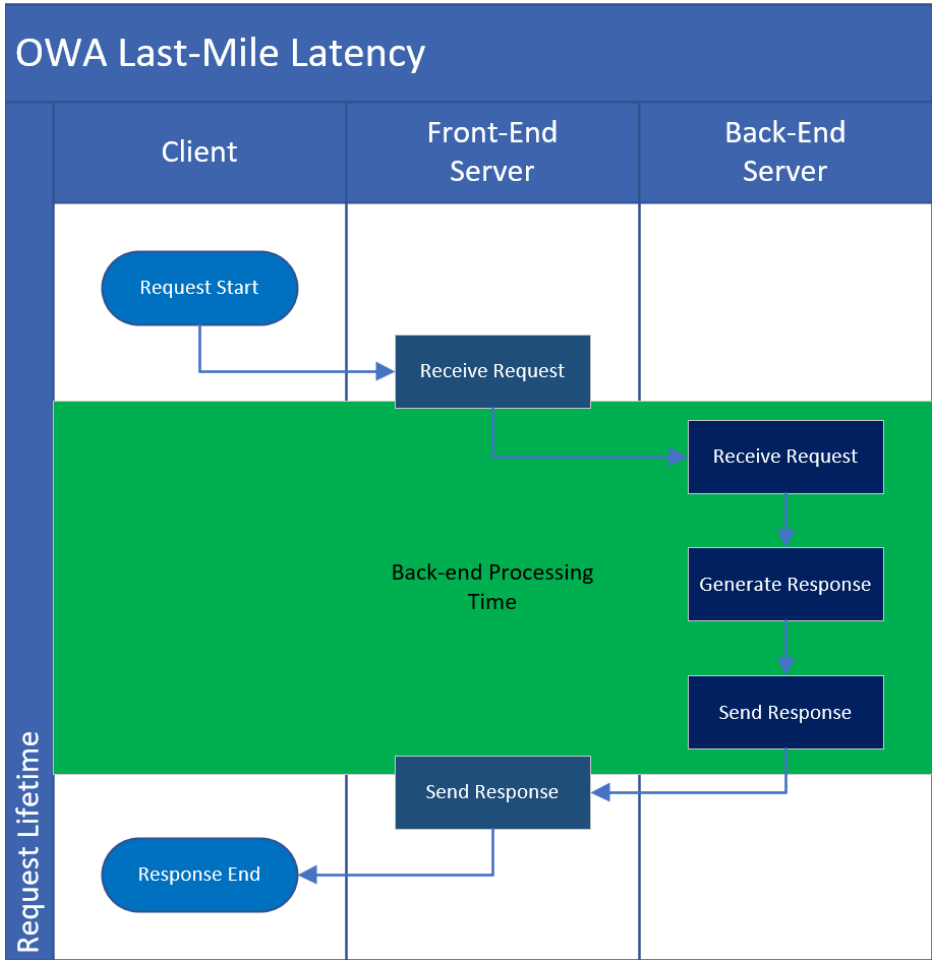
OWA Usage

- Summary
  - Evenly split HTTP/2 and HTTP/3 usage overall
  - Similar usage across browsers, IP protocol versions, and regions
  - Similar usage across request size and response size (not shown)

- Conclusions
  - Almost perfectly even split of usages across the two protocols gives us a great opportunity to do A:B comparisons between the two
  - HTTP/3 browser-based users exist in abundance already
    - Any HTTP/2-based services that add HTTP/3 support should expect significant usage immediately.

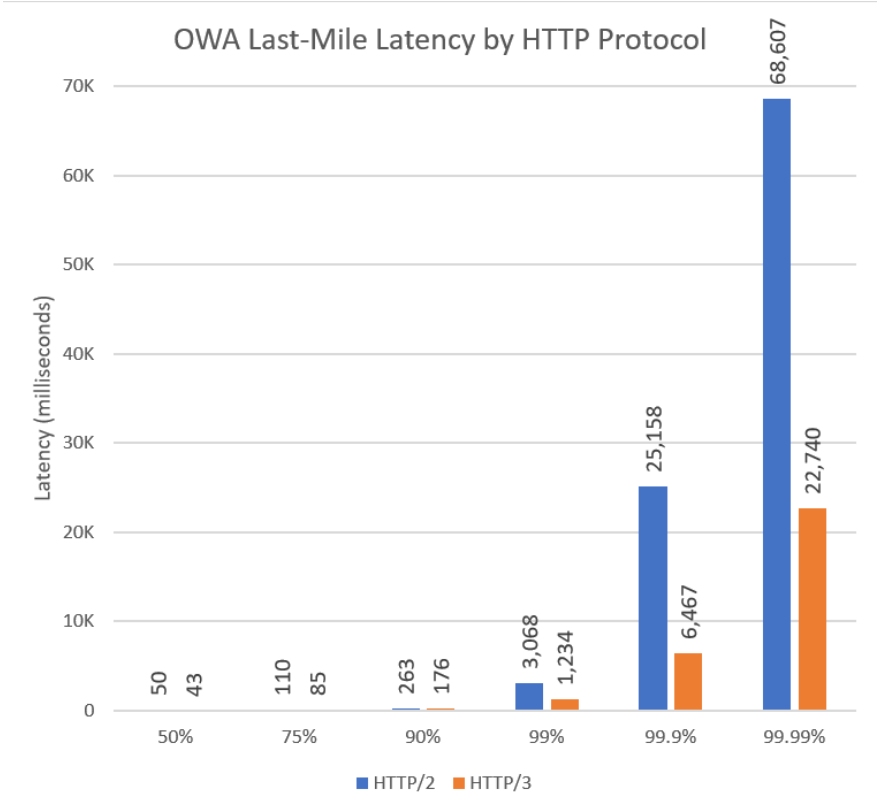
OWA Request Latency Definition

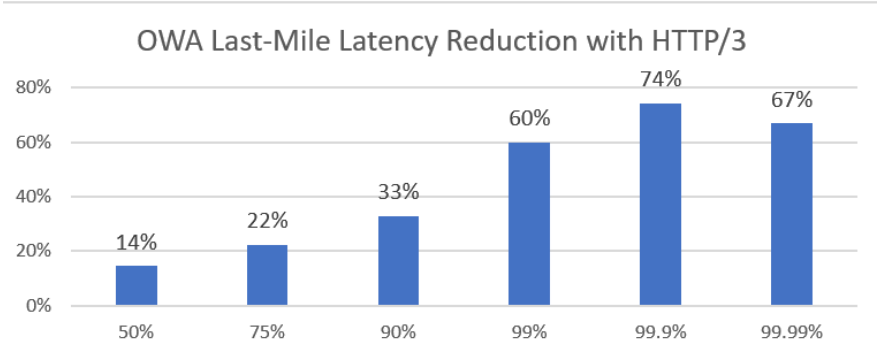
- "Last mile latency" is the total latency from request start to response end, minus the back-end server processing time
- OWA latency data primarily looks at last mile latency to analyze the latency costs of the network



OWA Request Latency by HTTP Protocol

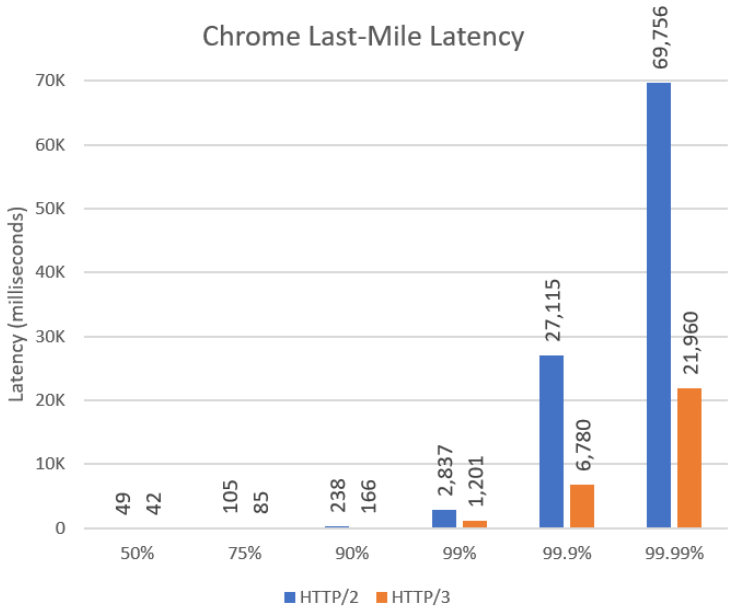
- These charts compare the last week's last-mile request latencies for HTTP/2 and HTTP/3
- Reductions can be seen across the board, but are increasingly significant at the higher percentiles

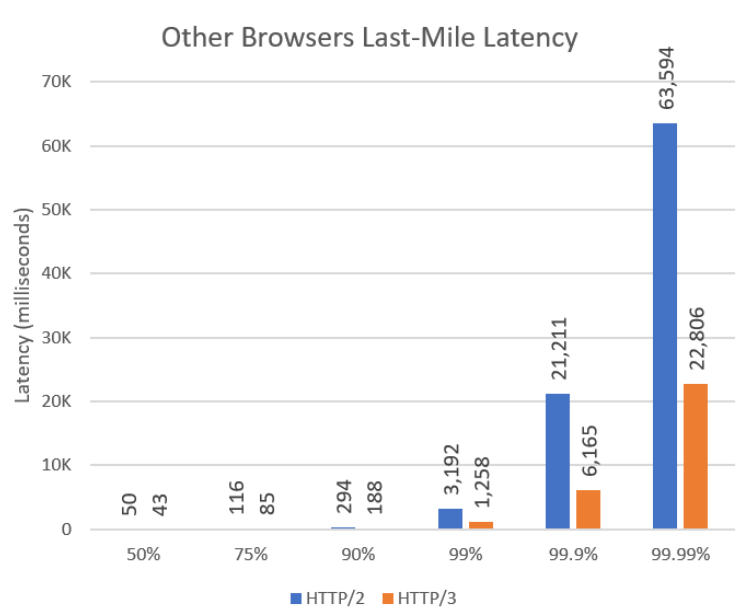


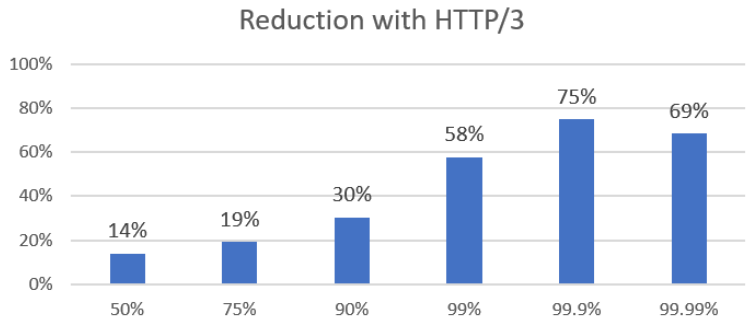


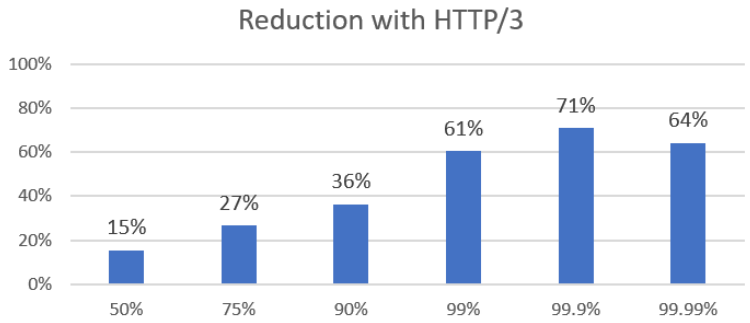
	50%	75%	90%	99%	99.9%	99.99%	Samples
HTTP/2	49.73	109.69	262.87	3,068.03	25,158.40	68,607.30	501,635,219
HTTP/3	42.52	85.2	176.43	1,234.44	6,466.83	22,740.38	482,402,673

OWA Request Latency by Browser



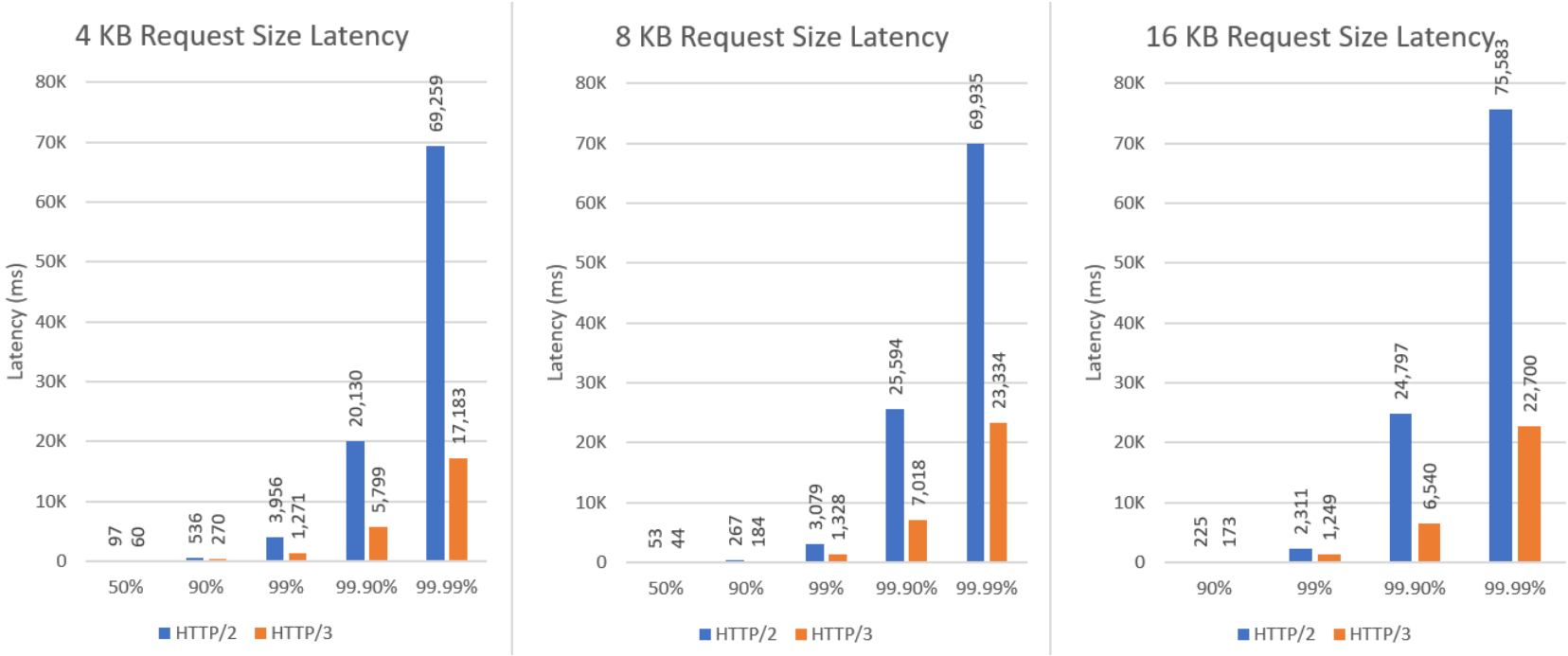






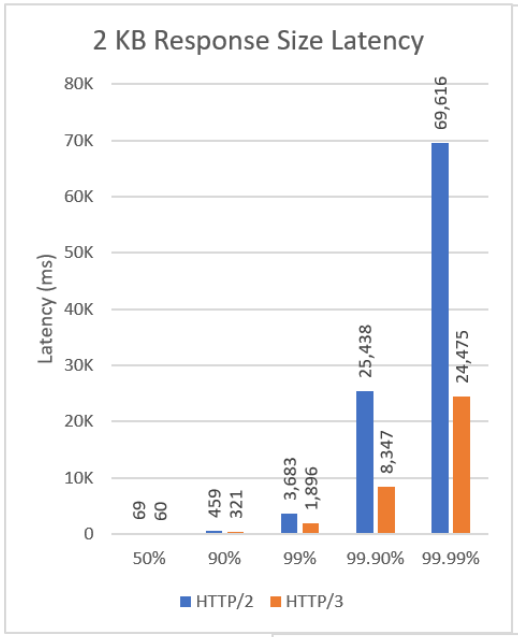
	50%	75%	90%	99%	99.9%	99.99%	Samples
H2|Chrome	49.33	104.97	238.22	2,836.60	27,114.71	69,755.79	395,103,842
H3|Chrome	42.44	85.01	166.2	1,201.37	6,779.66	21,960.00	362,982,034
H2|Others	50.33	115.83	293.88	3,191.54	21,210.89	63,594.05	347,007,385
H3|Others	42.62	85.13	187.51	1,258.21	6,164.79	22,805.70	345,919,549

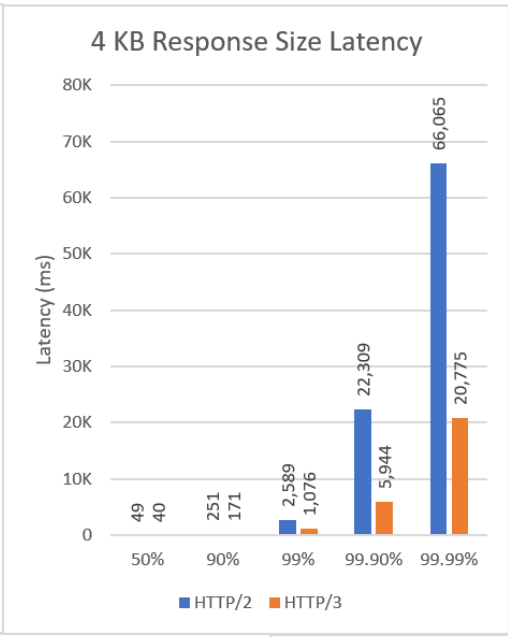
OWA Request Latency by Request Size

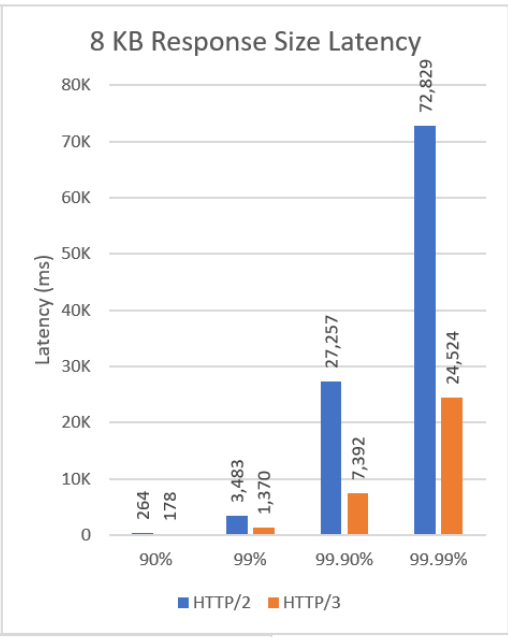


- Split by the most common request sizes
- HTTP/3 is still a clear winner.

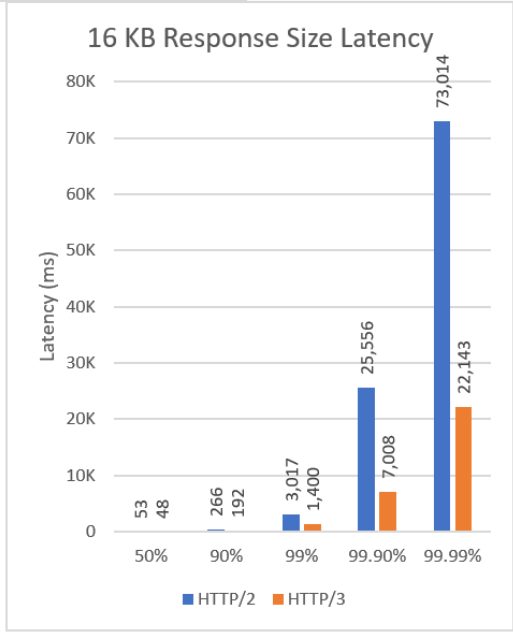
OWA Request Latency by Response Size

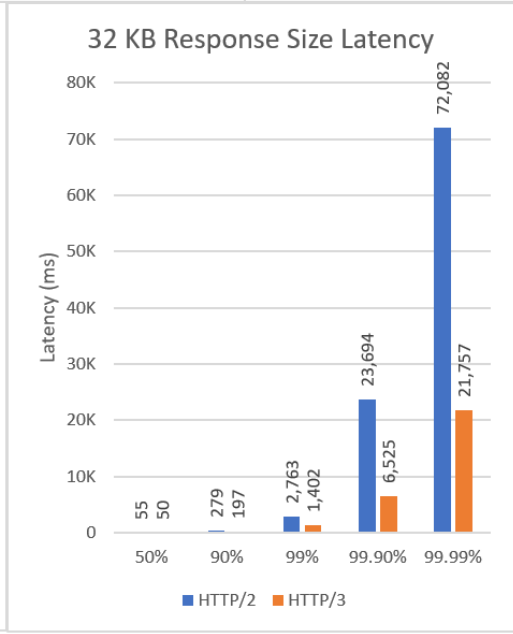






- Latency for most common response sizes
- HTTP/3 is a clear winner independent of the size

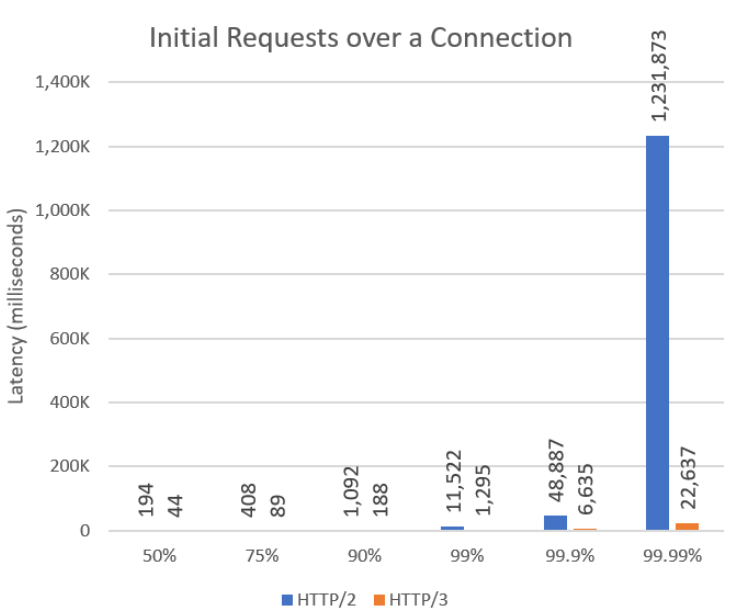


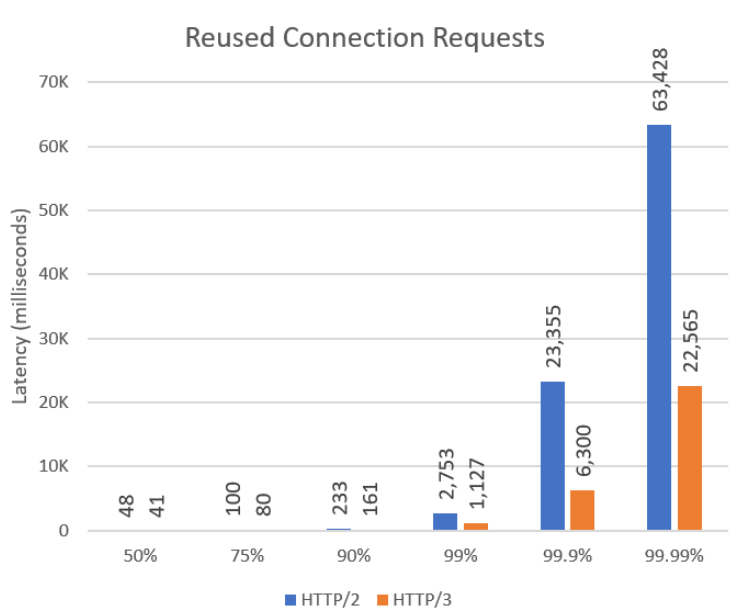


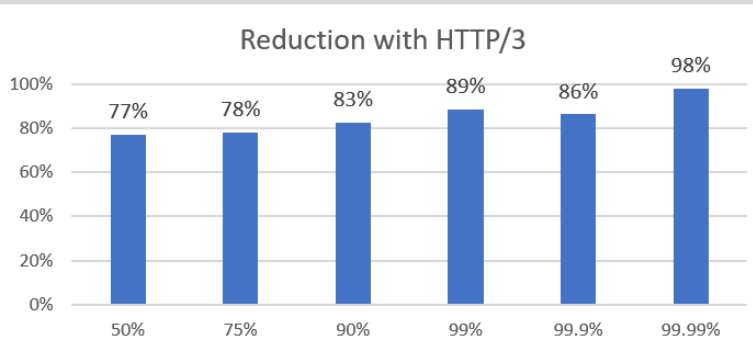
OWA Request Latency

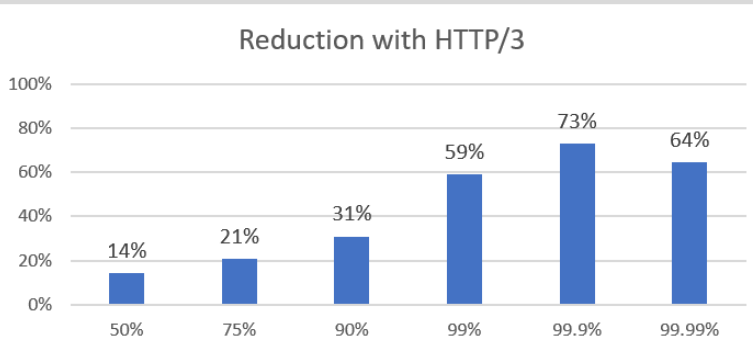
- Conclusions
  - HTTP/3 is a clear winner for OWA
  - No known data bias might be skewing things "unfairly" in HTTP/3 favor.
  - New HTTP/3 deployments should expect significant reductions in last-mile latency
  - Strongly recommend the upgrade where/when possible

OWA Request Latency by Connection Reuse





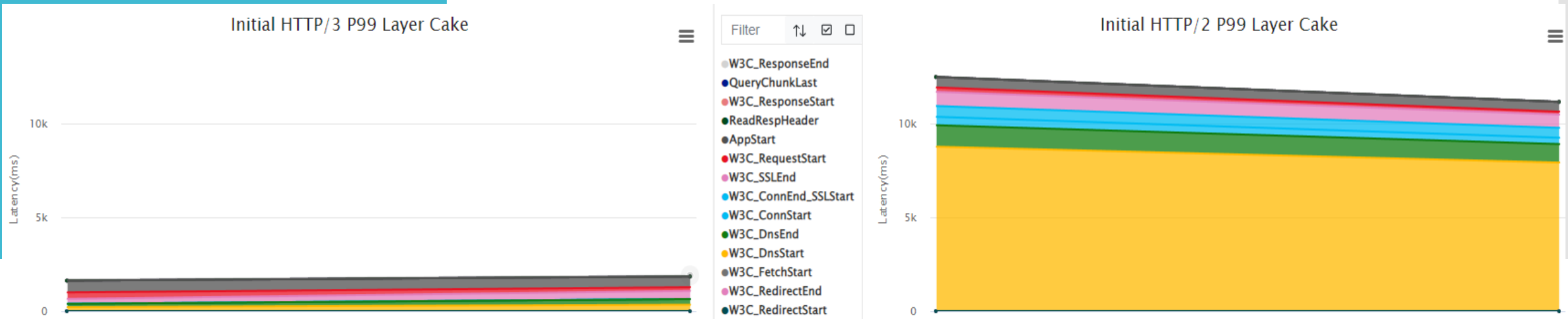




	50%	75%	90%	99%	99.9%	99.99%
H2|Initial	193.79	408.37	1,092.45	11,521.90	48,887.22	1,231,872.56
H3|Initial	43.9	89.49	187.56	1,294.85	6,634.68	22,637.20
H2|Reuse	47.82	100.48	232.75	2,752.53	23,355.08	63,427.71
H3|Reuse	41.07	79.83	161.14	1,127.04	6,299.94	22,565.32

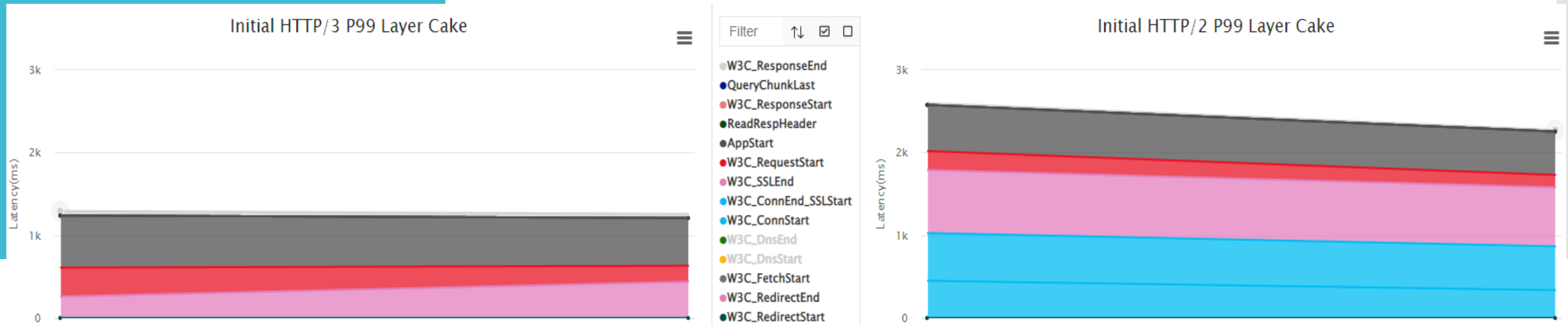
Initial Connection Request Latency

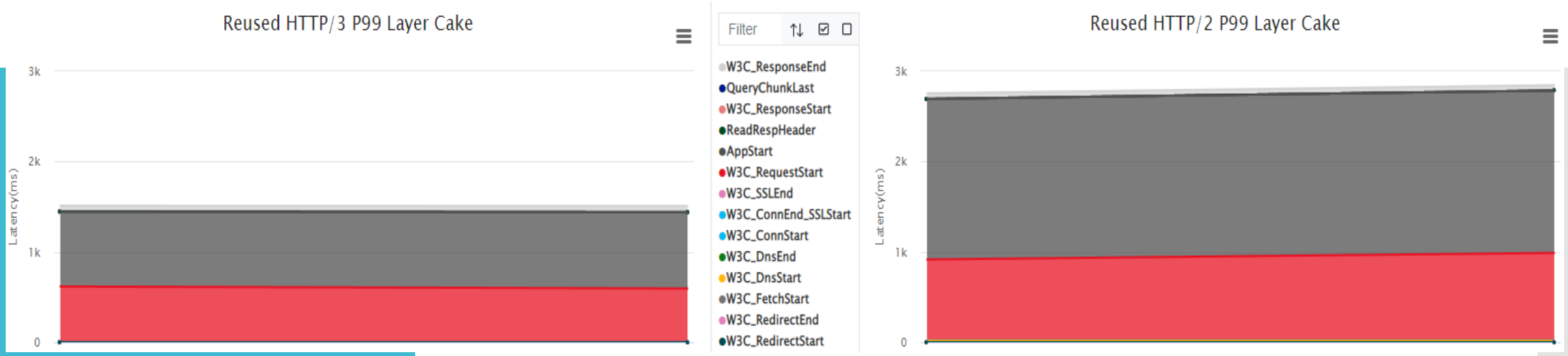
- Several factors make HTTP/2 look worse than HTTP/3 at high percentiles
  - Since HTTP/3 generally happens after HTTP/2, the cost of things like DNS and other start up logic get (unfairly?) attributed to HTTP/2 and not HTTP/3
  - Beyond that, QUIC has improved handshake, so the Conn layers are eliminated and the SSL layer smaller for HTTP/3.



Initial Connection Request Latency

- Even when we exclude the DnsStart and DnsEnd layers, HTTP/3 is still a clear win, but thins are a lot closer.
- Because of these factors that make H2 and H3 data not an "apples to apples" comparison, and since generally most requests go over reused connections anyways, we don't focus too much on initial requests





Reused Connection Request Latency

- 2 major factors for latency when reusing an existing connection:
  - **RequestStart** – The time it takes to get a connection and be allowed to send the request on it. Can generally be thought of as HoL blocking.
  - **AppStart** – An estimate of the time from when the client sends the request to when the server is aware of it. Usually considered a catch-all for network latency and everything else.

Reused Connection Request Latency

- Improvements in QUIC and HTTP/3 (compared to HTTP/2)
  - Less HoL blocking from:
    - Independent stream data
    - FIFO stream scheduling
    - Per-packet encryption
  - Better loss recovery
    - Unique packet numbers
    - Improved probe timeout logic
    - Much larger SACK limit
  - Pacing
- No data that indicates how much each of these help reduce latency for HTTP/3
  - Open area of investigation on how to get this information

Other Thoughts

- Exchange Online does not have 0-RTT because it is not currently supported by Windows TLS
- EXO has QUIC-based load balancing and MsQuic supports client migration, though it's a largely unexplored area so far
- MsQuic supports DPLPMTUD for MTU discovery, but we don't have any data around common MTU sizes yet
- We have just started the journey of collecting and analyzing telemetry for HTTP/3

Questions so far?

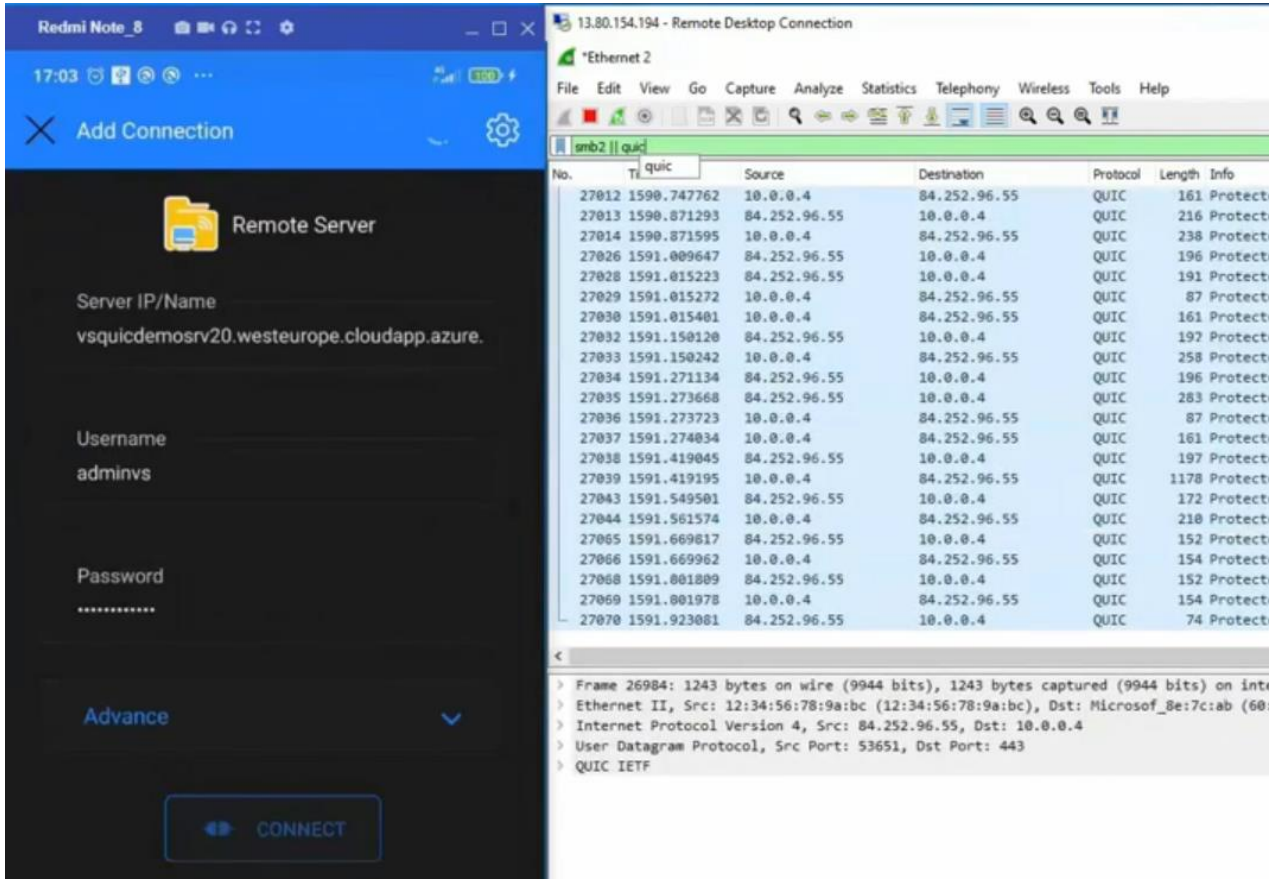
More still to come!

Other Products

Built on top of MsQuic

SMB over QUIC

- Server support in Windows Server 2022 Azure Edition
- Client support in Windows 11
- 3rd party support in Visuality Android app



.NET Core and Kestrel Web Server

- .NET 6 recently release
  - Windows and Linux support
  - Preview HTTP/3 support in Kestrel, HttpClient and YARP
  - HTTP/3 support in IIS and http.sys
  - .NET is the first platform with gRPC over HTTP/3

Misc

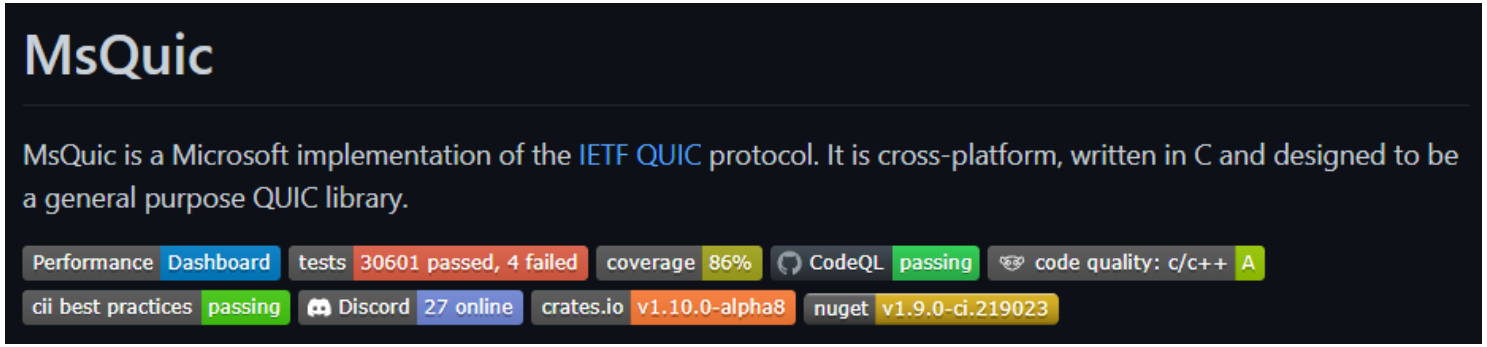
- Xbox Game Core support for MsQuic for game studios
- NDI 5.0 move to MsQuic for network communication
- DreamWorks production studios

MsQuic

Open-source, cross-platform library on https://github.com/microsoft/msquic

Statistics

- Code is 4+ years old, with 2 years of that on GitHub
- 1484 commits since open sourcing
- Supports Windows kernel mode, Windows user mode, Xbox, Linux, Android, and macOS.
- Written in C, with wrappers exposed for C++, C# and Rust.
- 200k+ lines across all source 553 files
  - 100k lines of production code
  - 52k line of platform-independent 'core' code
- Automated CI runs for every PR and commit
  - 30k+ test cases across various platforms
  - 87 Azure Pipeline jobs; 77 GitHub Action jobs
  - Functional tests, stress, interop, CPU perf, WAN perf, and more



Priorities

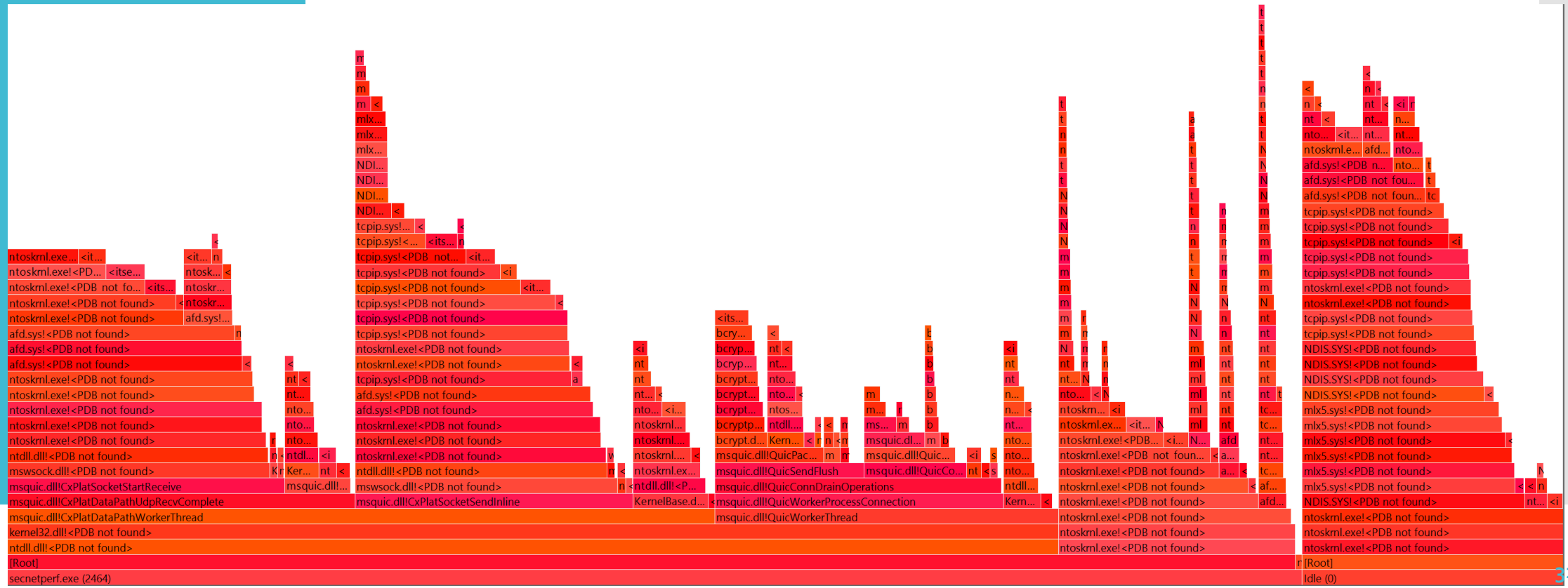
- High level requirements driven by primary dependents:
  - Windows HTTP/3
  - Windows SMB over QUIC
  - .NET Core HTTP/3
  - External GitHub consumers
- Results in the following priority (in order):
  - Quality
  - Interoperability
  - Performance
  - Additional features

Performance

- Beyond ensuring the core is solid, performance is the next biggest focus area
- Performance can mean lots of things though: CPU usage, memory usage, latency and even network utilization
- We care about all of these, but usually the following is our priority order:
  - CPU usage
  - Latency
  - Network utilization

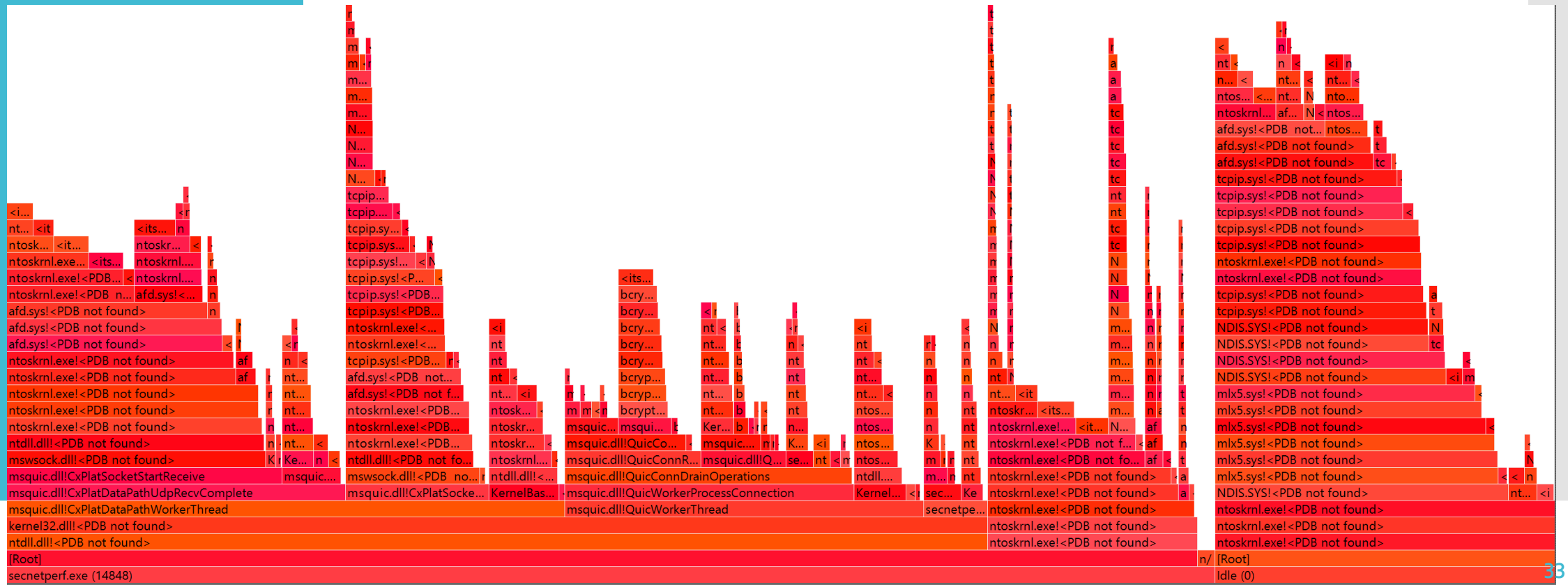
Performance: RPS (HTTP)

- Following data from Windows performance tests
- UDP continues to be the primary bottleneck for RPS (4KB)
- Server:



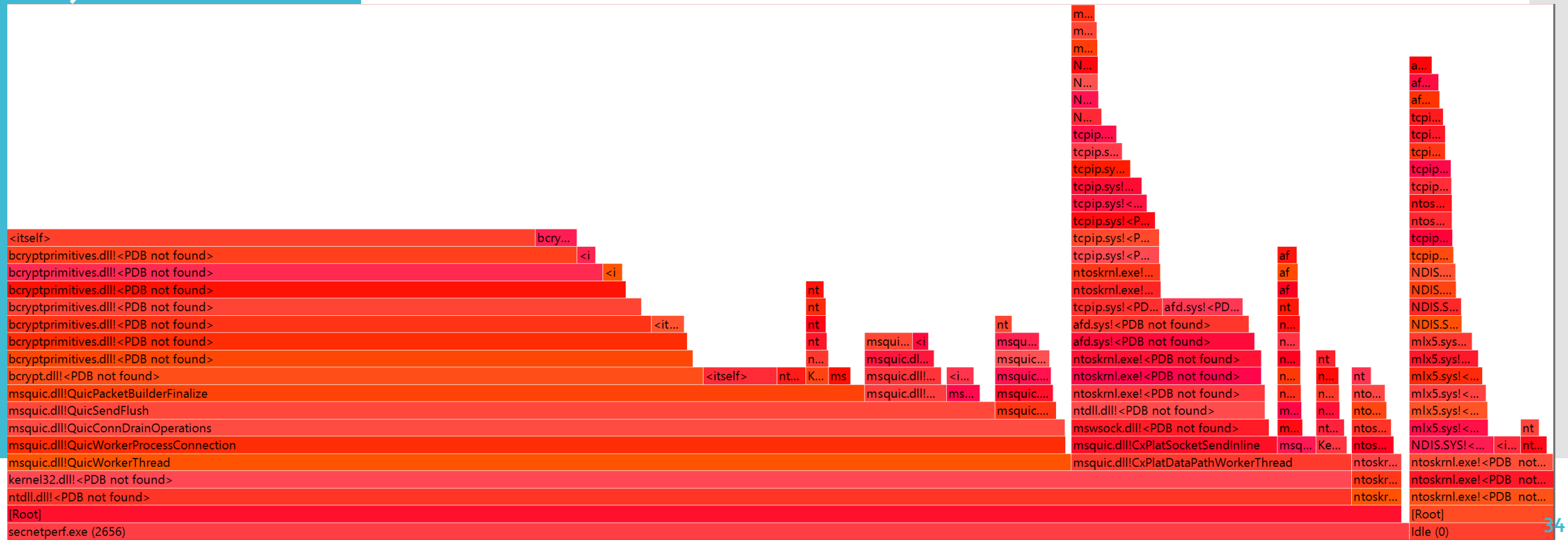
Performance: RPS (HTTP)

- Client:



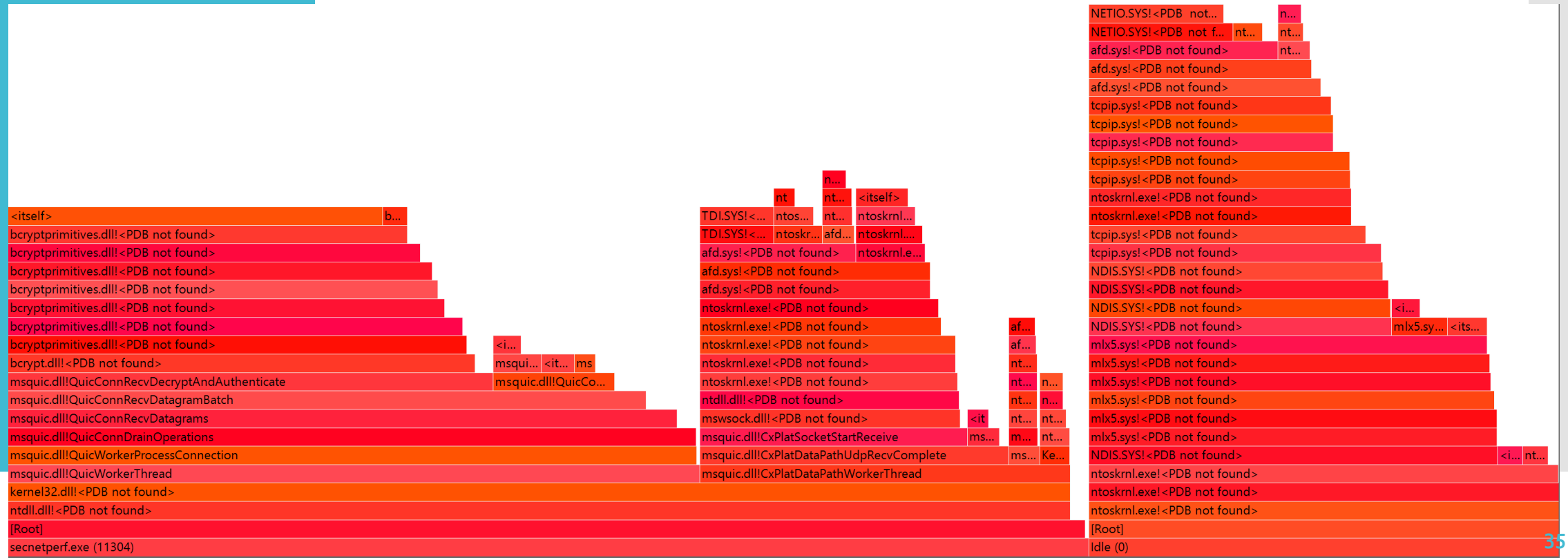
Performance: Throughput (SMB)

- Crypto is the primary bottleneck
- UDP usage is still significant, but batching has significantly reduced the costs there
- Server:

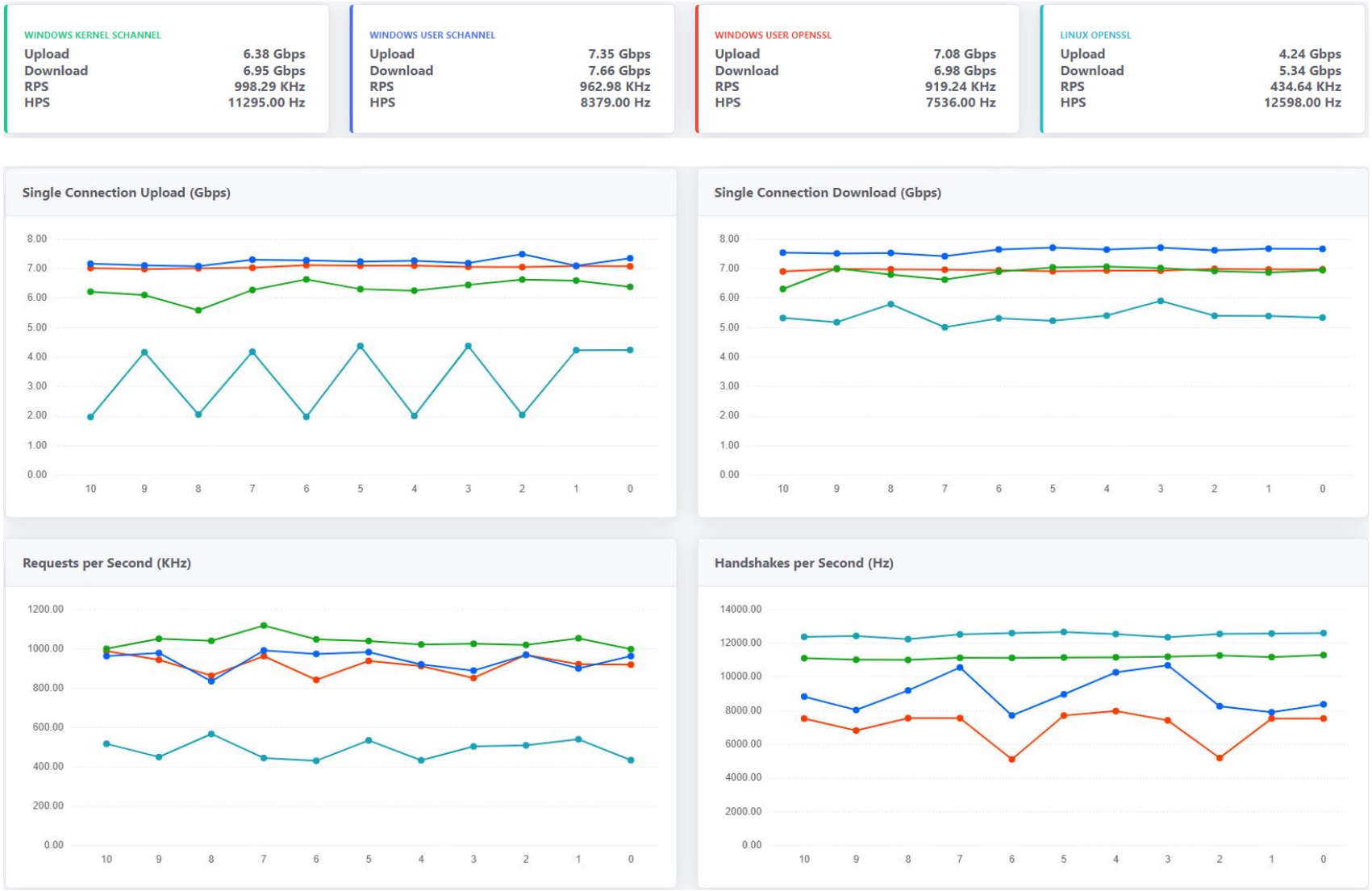


Performance: Throughput (SMB)

- Client:



Performance Dashboard



- https://microsoft.github.io/msquic/

WAN Performance

- We also actively measure simulated WAN performance
- Over 1000 different scenarios measured ever PR/commit
- Still a WIP to catch regressions

Usage %	NetMbps	RttMs	QueuePkts	Loss	Reorder	DelayMs	RateKbps
92.9	50	5	115	0	10000	5	46458
92.9	50	5	115	10000	10000	5	46447
92.9	50	5	115	10000	0	0	46439
92.9	100	5	229	0	10000	5	92863
92.8	5	5	11	10000	10000	5	4640
92.8	10	5	23	0	0	0	9280
92.8	5	5	11	1000	10000	10	4640
92.8	50	5	115	0	0	0	46395
92.8	10	5	23	1000	0	0	9278
92.8	10	5	23	10000	10000	10	9278
92.8	5	5	11	0	10000	10	4639
92.8	20	5	46	10000	0	0	18555
92.8	5	5	11	10000	0	0	4638
92.8	5	5	11	0	1000	5	4638
92.8	50	5	23	0	0	0	46378
92.8	10	5	23	0	10000	5	9275
92.7	5	5	11	0	0	0	4637
92.7	5	5	11	1000	10000	5	4637
92.7	5	5	11	1000	1000	5	4637
92.7	10	5	23	10000	1000	5	9274
92.7	5	5	11	1000	0	0	4637
92.7	10	5	23	10000	0	0	9274
92.7	200	5	458	0	0	0	185447
92.7	10	5	5	1000	0	0	9272
92.7	10	5	5	10000	0	0	9271
92.7	20	5	46	1000	0	0	18542
92.7	50	5	115	0	10000	10	46354
92.7	5	5	11	0	10000	5	4635
92.7	10	5	23	1000	10000	5	9270
92.7	100	5	229	0	0	0	92685
92.7	10	5	23	10000	10000	5	9268
92.7	10	5	23	0	10000	10	9267
92.7	5	5	11	10000	10000	10	4633
92.7	5	5	11	10000	1000	5	4633
92.7	50	5	23	10000	0	0	46329
92.6	10	5	23	0	1000	5	9264
92.6	5	5	11	1000	1000	10	4632
92.6	20	5	46	0	0	0	18522
92.6	10	5	23	1000	1000	5	9260
92.6	10	5	5	10000	10000	5	9259

Everything Else

- API Design
- Cross platform
- Open source
- Partners

github.com/microsoft/msquic/blob/main/src/tools/sample/sample.c

```
290  QUIC_STATUS
291  QUIC_API
292  ServerConnectionCallback(
293      _In_ HQUIC Connection,
294      _In_opt_ void* Context,
295      _Inout_ QUIC_CONNECTION_EVENT* Event
296      )
297  {
298      UNREFERENCED_PARAMETER(Context);
299      switch (Event->Type) {
300      case QUIC_CONNECTION_EVENT_CONNECTED:
301          //
302          // The handshake has completed for the connection.
303          //
304          printf("[conn][%p] Connected\n", Connection);
305          MsQuic->ConnectionSendResumptionTicket(Connection, QU
306          break;
307      case QUIC_CONNECTION_EVENT_SHUTDOWN_INITIATED_BY_TRANSPOR
308          //
309          // The connection has been shut down by the transport
310          // is the expected way for the connection to shut dow
311          // protocol, since we let idle timeout kill the conne
312          //
313          if (Event->SHUTDOWN_INITIATED_BY_TRANSPORT.Status ==
314              printf("[conn][%p] Successfully shut down on idle
315          } else {
316              printf("[conn][%p] Shut down by transport, 0x%x\n
317          }
318          break;
319      case QUIC_CONNECTION_EVENT_SHUTDOWN_INITIATED_BY_PEER:
320          //
321          // The connection was explicitly shut down by the pee
322          //
323          printf("[conn][%p] Shut down by peer, 0x%llu\n", Conn
324          break;
325      case QUIC_CONNECTION_EVENT_SHUTDOWN_COMPLETE:
326          //
```

Going Forward

- Just because QUIC is RFC and MsQuic has shipped version 1 doesn't mean we're done.
- There's still work to do to simply implement a few remaining features:
  - Client side of migration
  - Some QUIC extensions (Grease QUIC bit)
  - Upcoming version 2 (mostly a no-op)
- And there's more research-area work as well:
  - New congestion control algorithms (BBRv2)
  - ECN support
  - Additional stream/datagram scheduling logic
  - Telemetry integration and analysis (better qlog support)
  - More performance work (low latency scenarios)
- Feel free to contribute at https://github.com/microsoft/msquic!
  - Any code you contribute will be used by Windows Server, Windows Client, .NET, Office, Xbox, NDI, etc.

Questions?

All done!