

optimizing Firefox's HTTP3 IO stack

Max Inden

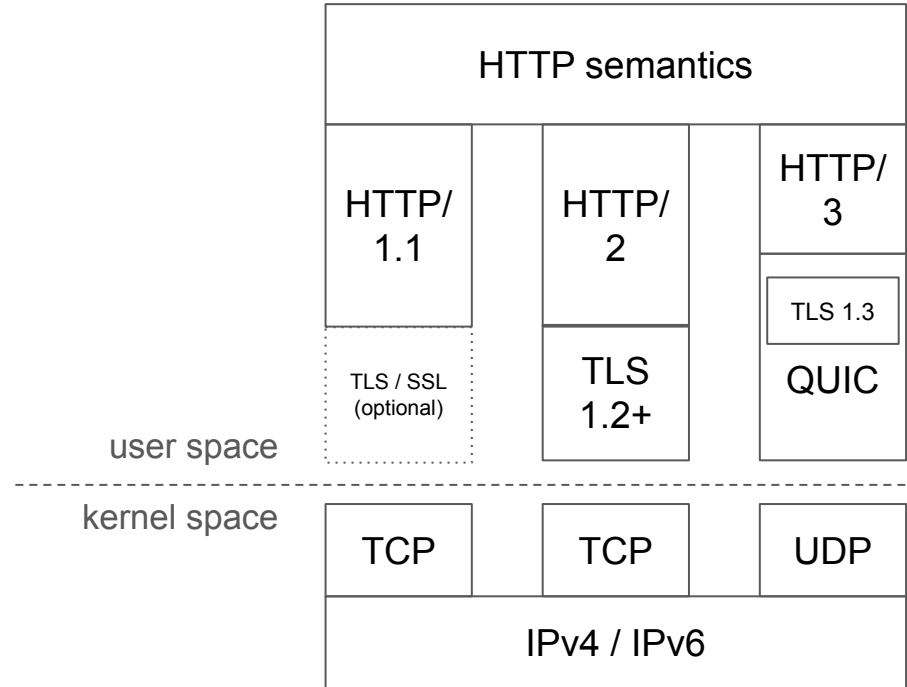
- Software engineer at Mozilla
- Working on HTTP3 and QUIC in Firefox
- mail@max-inden.de
- @mxinden
- Previously p2p, Kubernetes and Prometheus



What is Firefox?

What is QUIC?

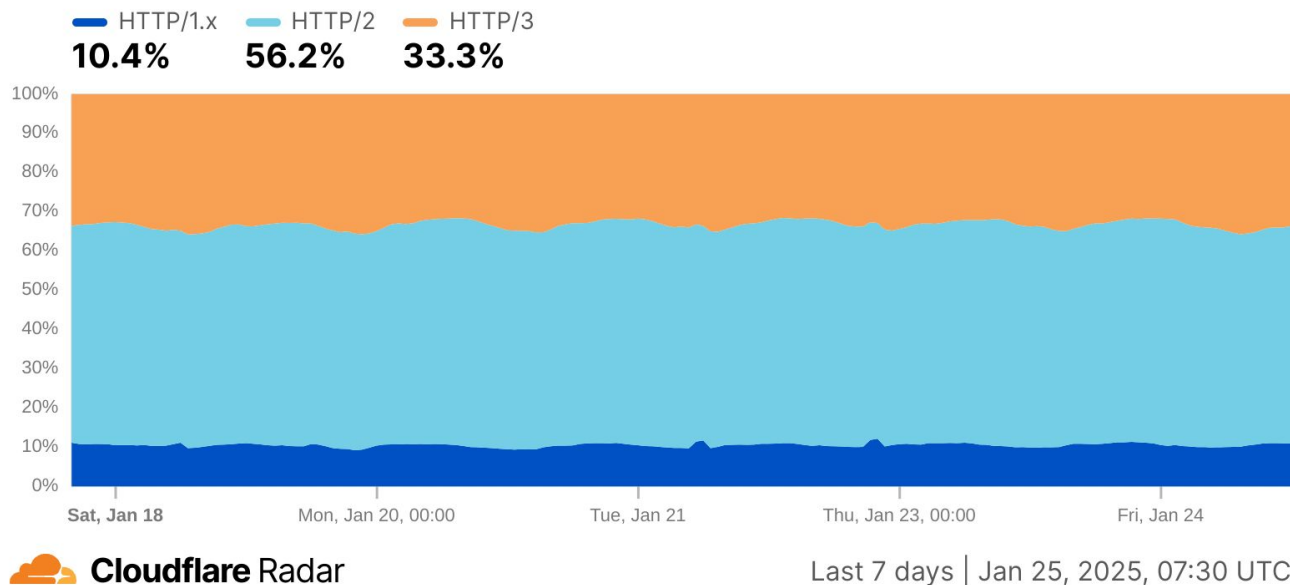
- general purpose transport protocol
- on top of UDP
- encrypted (meta) data
- 1 RTT connection establishment
- 0 RTT on consecutive connections
- stream based
- no head-of-line blocking
- connection migration
- easy to evolve
- often in user space
- ...



QUIC on the Internet today

HTTP/1.x vs. HTTP/2 vs. HTTP/3 Worldwide

Distribution of human traffic by HTTP version

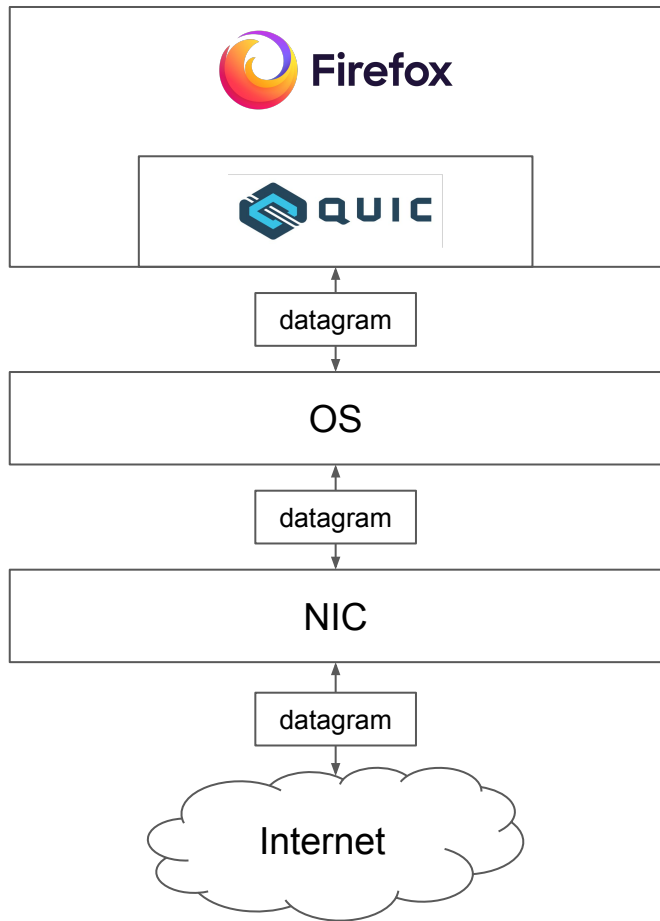


QUIC in userspace

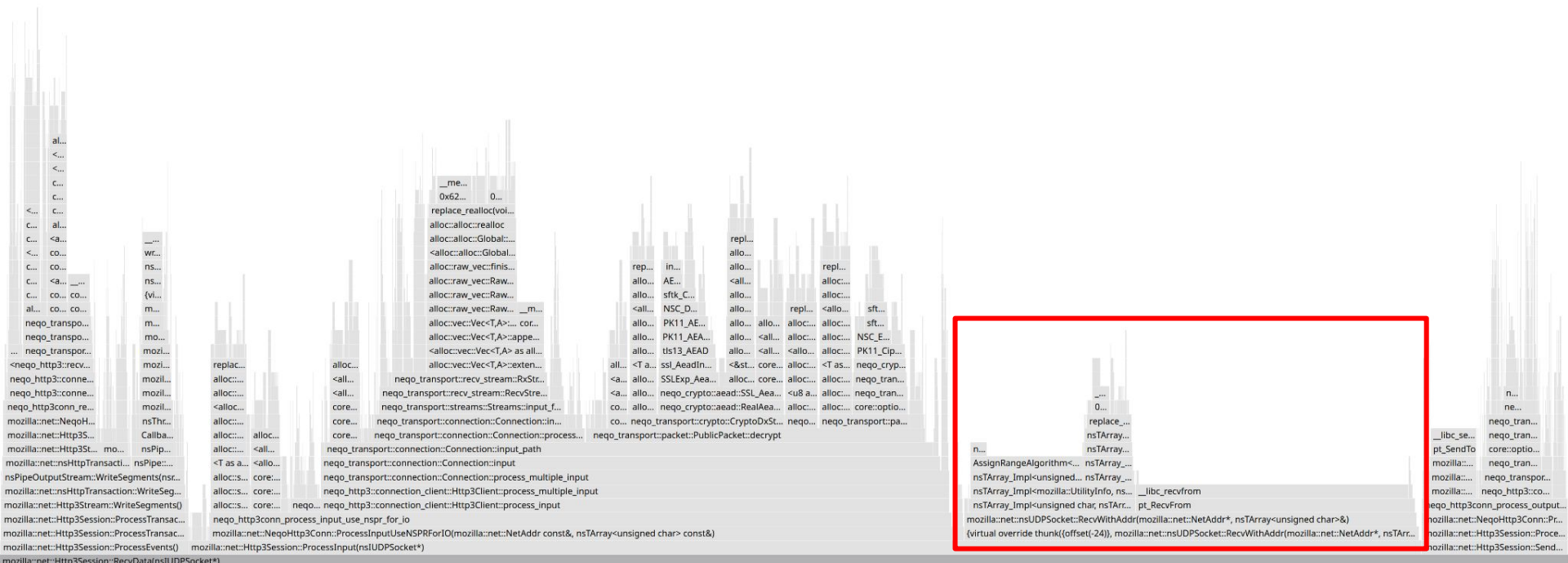
Unsurprisingly an un-optimized userspace transport protocol on top of UDP is not as fast as heavily optimized TCP in kernel space.

Userspace QUIC might do:

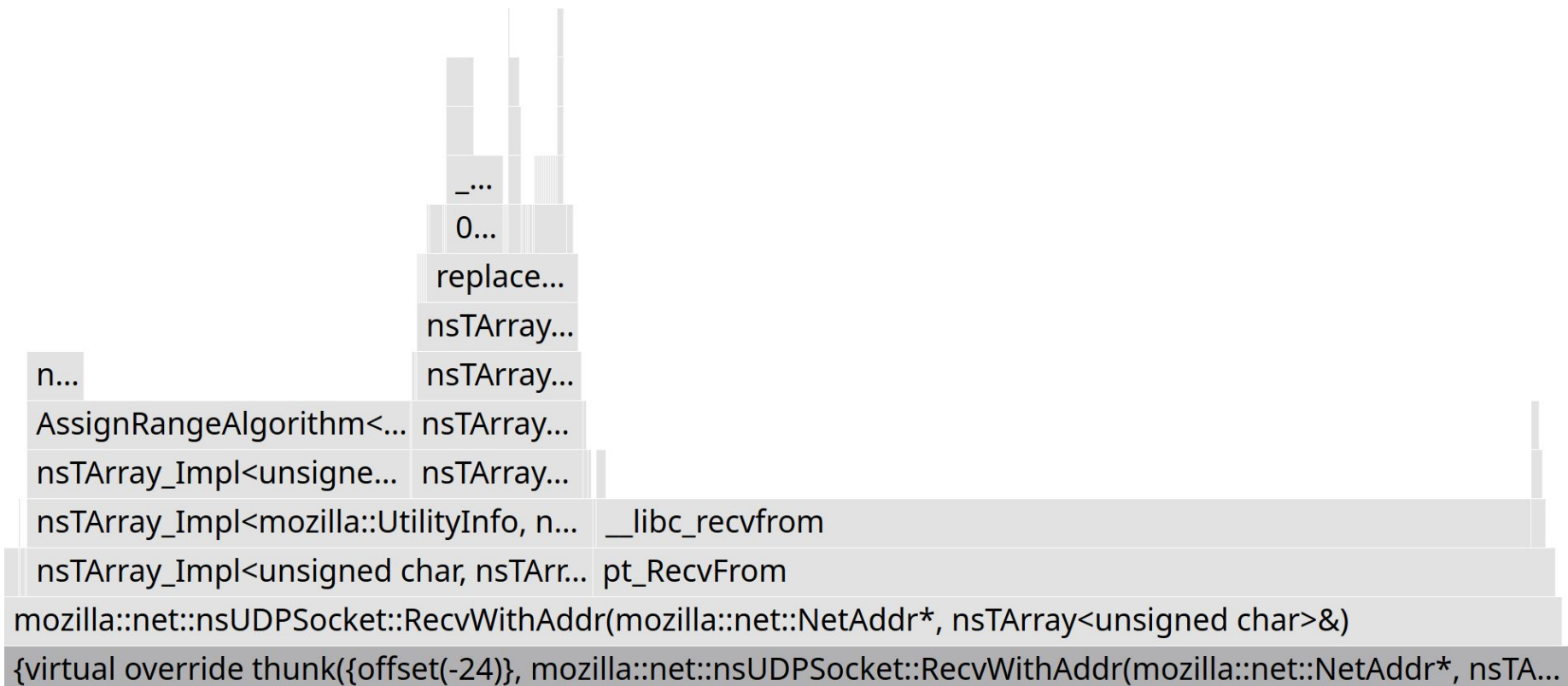
- one syscall per UDP datagram
- < 1500 byte MTU for Internet traffic
- ACK of up to every second packet
- [de Bruijn et al.](#) “3.5x the CPU cycles per byte”



Un-optimized Firefox QUIC UDP IO

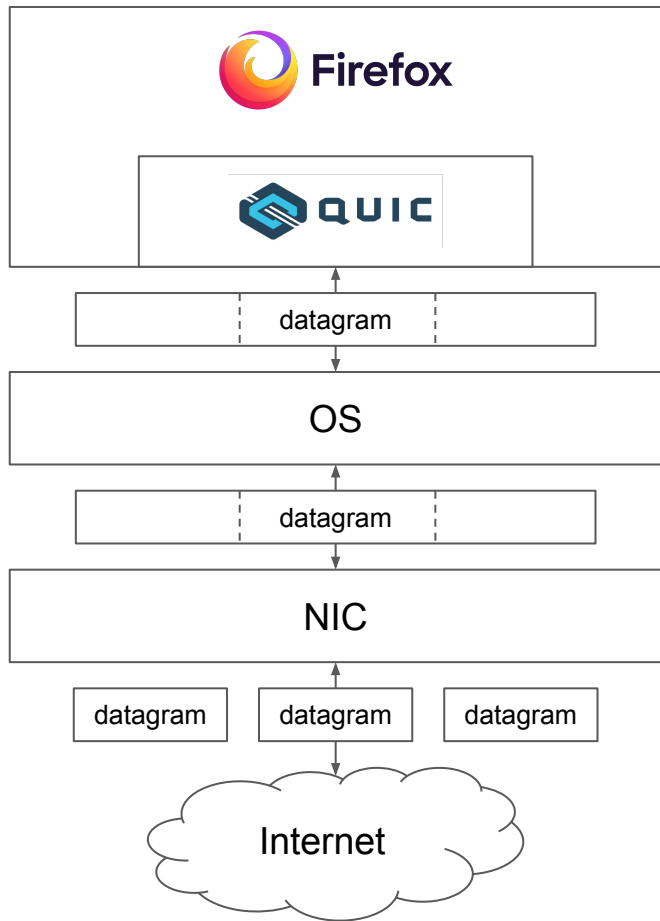


Un-optimized Firefox QUIC UDP IO



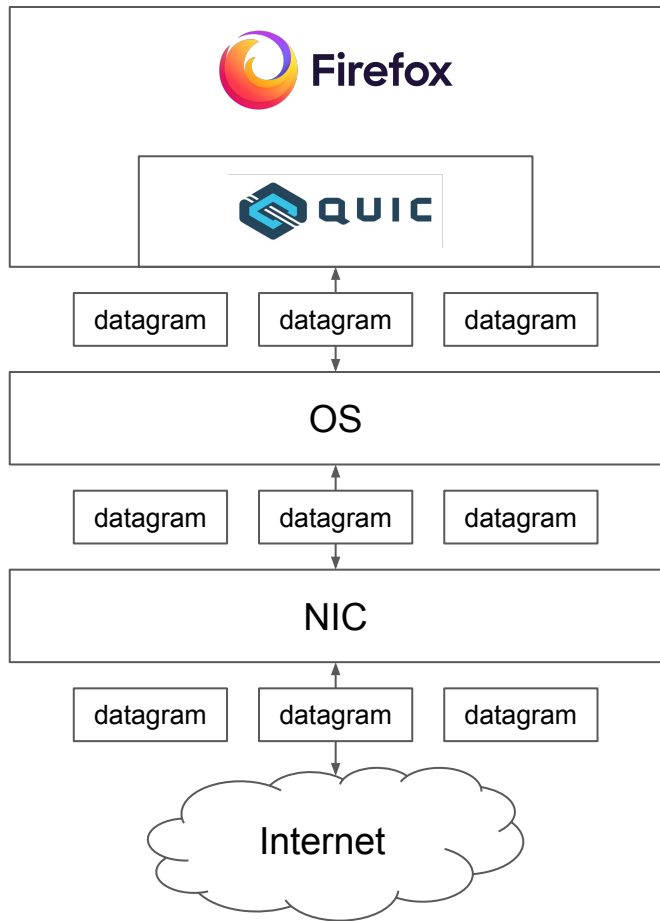
Segmentation offloading

- Linux's GSO/GRO
- Windows' USO/URO
- measurements on Firefox Nightly Linux GRO
 - 75th of read syscalls read [2 or more packets](#), 95th read 10 or more packets.
 - 75th of read syscalls read [2.4 KiB](#) total, 95th read 12 KiB total.
- close to the [1 Gbit/s on loopback](#) benchmark



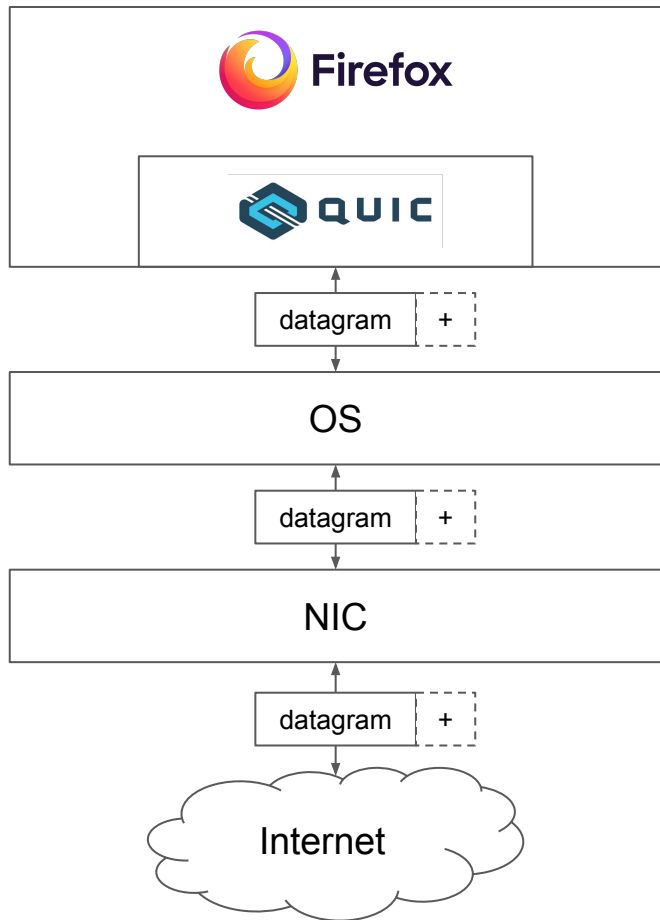
Multi-message syscalls

- MacOS sendmsg_x and recvmsg_x
- Similar to Linux's sendmmsg and recvmmsg
- [11% performance improvement](#) on loopback transfer



PLPMTUD

- Packetization Layer Path MTU Discovery for Datagram Transports ([RFC 8899](#))
- probe maximum MTU on path
- 1280 bytes vs 1472 bytes makes a difference
- See also
 - [Firefox QUIC implementation](#)
 - [Custura, A., Fairhurst, G., & Learmonth, I.: “Exploring usable Path MTU in the Internet”](#)



QUIC ACK frequency

- [QUIC Acknowledgment Frequency](#) IETF draft
- say a QUIC receiver ACKs every second packet on a 1 Gbit/s transfer
- $1 \text{ Gbit/s} / 8 / 1500 / 2 \approx 40\text{k ACK} / \text{s}$
- instead have the sender propose an ACK rate to the receiver

Additional wins

- QUIC UDP IO in Rust using [quinn-udp](#)
- Explicit Congestion Notification (ECN)
 - with modern syscalls, we can now read auxiliary IP metadata, e.g. ECN.
 - on Firefox Nightly we see
 - ~50% of paths being ECN capable
 - 75th percentile of QUIC connections see [>= 0.6% CE](#) marks on receive path.
- optimized memory management
 - we use a single long-lived 64k receive buffer for all QUIC connections
 - soundness check at compile time via Rust's borrow checker
 - [significant reduction](#) in CPU time on large transfers

What is next?

- rolling out
 - PMTUD
 - ECN
 - ACK frequency
- send-path optimizations
 - GSO, USO, sendmsg_x, ...
 - long lived send buffer

Get involved!

- try it yourself on Firefox Nightly, soon Beta and Release then after
- contribute to <https://github.com/mozilla/neqo>
 - Rust codebase
 - modern transport protocol
 - help make Firefox faster
- reach out:
 - <https://matrix.to/#/#neqo:mozilla.org>
 - mail@max-inden.de