Linux QUIC

Bringing a Modern Secure Transport into the Kernel

Xin Long

Red Hat

Netdev 0x1a
July 13, 2026

Overview

Motivation

QUIC Essentials

A UDP-Based Multiplexed and Secure Transport

- **Faster Connection Setup:** Combined transport and TLS handshake
- **No Head-of-Line Blocking:** Multiplexed Streams
- **Connection Migration:** Connection identified by Connection ID, not IP/port
- **No Middle-box Issue:** UDP-based

Advantages in Kernel

For Kernel Subsystems

Enables SMB/NFS to use QUIC via net/handshake APIs with minimal changes.

ALPN-Based Dispatch

Routes QUIC connections in-kernel based on ALPN for different services or processes.

Standard Socket API

Supports listen(), accept(), connect(), sendmsg()/recvmsg(), and common socket options.

Performance

Reduces syscall overhead, minimizes data movement, and prepares for crypto offload.

Standards Coverage

Core RFCs

- RFC9000 - QUIC: A UDP-Based Multiplexed and Secure Transport
- RFC9001 - Using TLS to Secure QUIC
- RFC9002 - QUIC Loss Detection and Congestion Control

Extension RFCs

- RFC9221 - An Unreliable Datagram Extension to QUIC
- RFC9287 - Greasing the QUIC Bit
- RFC9368 - Compatible Version Negotiation for QUIC
- RFC9369 - QUIC Version 2

Extension I-Ds

- Managing multiple paths for a QUIC connection
- QUIC Stream Resets with Partial Delivery
- QUIC Acknowledgment Frequency
- QUIC Address Discovery
- QUIC-LB: Generating Routable QUIC Connection IDs
- Extended Key Update for QUIC
- QUIC Extended Acknowledgement for Reporting Packet Receive Timestamps

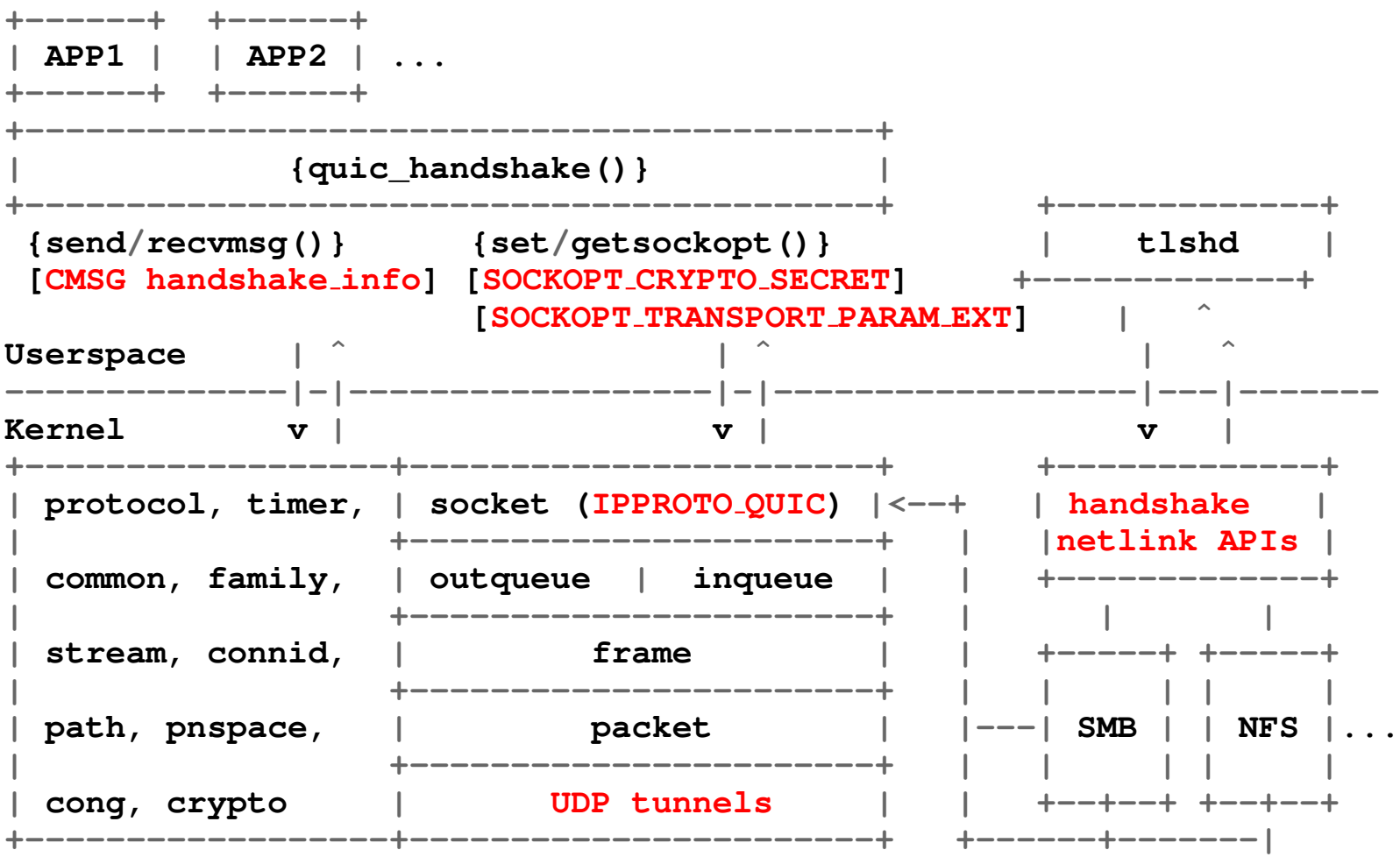
Implementation

Core Design

1. Userspace Handshake
   - Raw TLS Handshake Messages are processed in userspace by a TLS library like GnuTLS.
   - These messages are exchanged with kernel via sendmsg() and recvmsg(), with cryptographic details in control messages.
2. Kernel Transport
   - The entire QUIC protocol, aside from the TLS Handshake Messages processing and creation, is managed in the kernel.
   - IPPROTO_QUIC (similar to IPPROTO_MPTCP) instead of ULP (unlike kTLS), operating over UDP tunnels.
3. Kernel Consumer
   - Initiate a handshake request from the kernel to userspace using the existing net/handshake netlink.
   - The userspace component, such as tlshd service, then manages the processing of the QUIC handshake request.

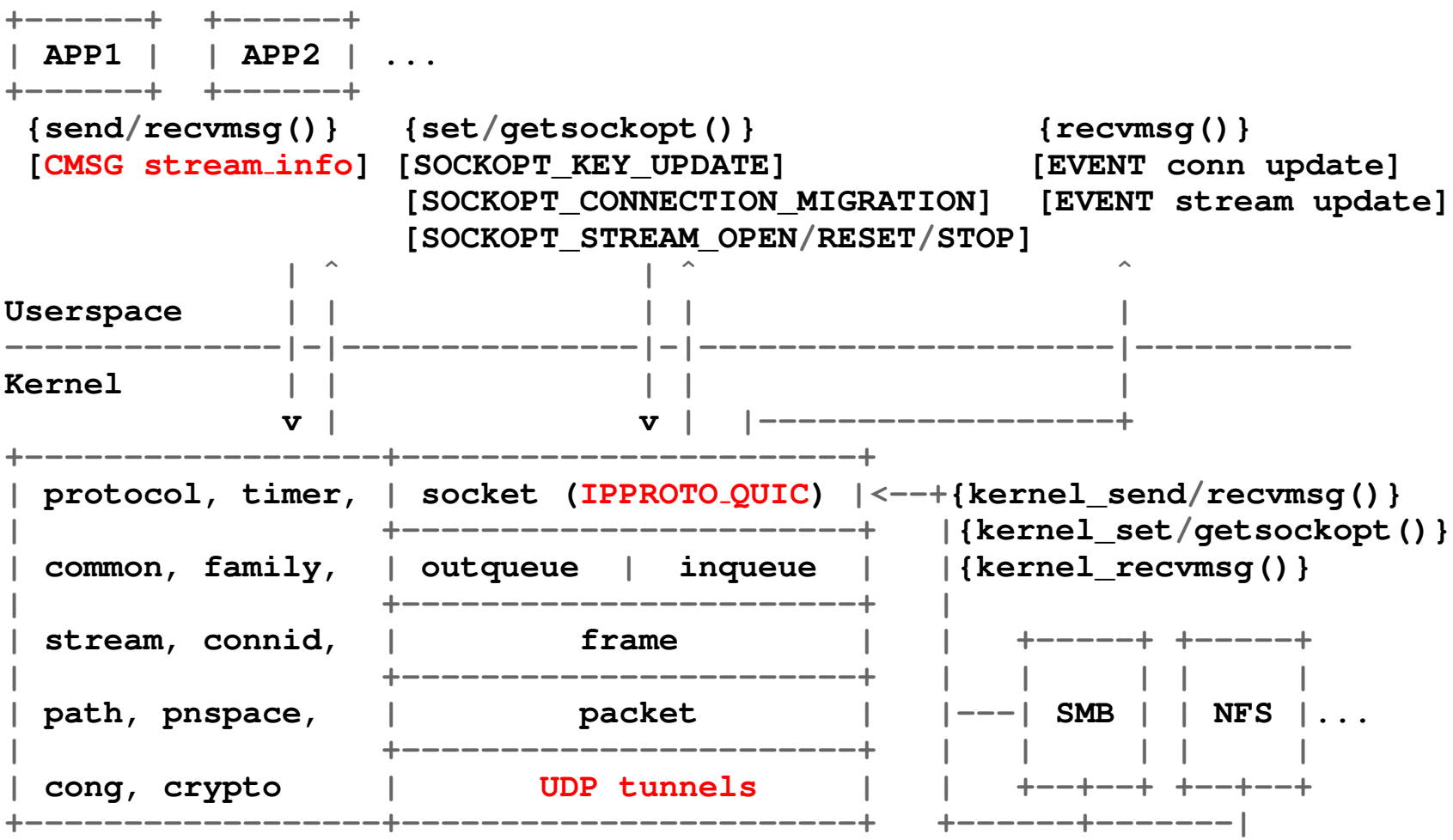
Architecture

Handshake



Architecture

Datapath



Socket Model

```
struct quic_hashinfo {
    struct quic_shash_table shash;     // 1 --- * quic_source_conn_id
    struct quic_shash_table lhash;     // 1 --- * sock (listen)
    struct quic_shash_table chash;     // 1 --- * sock (regular)
    struct quic_uhash_table uhash;     // 1 --- * quic_udp_sock
};


struct quic_sock {
    struct inet_sock          inet;    // 1 --- 1 sock
    struct list_head          reqs;    // 1 --- * quic_request_sock (listen)

    struct quic_data          ticket;
    struct quic_data          token;
    struct quic_data          alpn;

    struct quic_stream_table  streams; // 1 --- * quic_stream
    struct quic_conn_id_set   source;  // 1 --- * quic_source_conn_id
    struct quic_conn_id_set   dest;    // 1 --- * quic_dest_conn_id
    struct quic_path_group    paths;   // 1 --- 1 quic_path  1 --- 1 quic_udp_sock
    struct quic_cong          cong;
    struct quic_pnspace       space[];
    struct quic_crypto        crypto[];

    struct quic_outqueue      outq;    // 1 --- * quic_frame
    struct quic_inqueue       inq;     // 1 --- * quic_frame
    struct quic_packet        packet;
    struct quic_timer         timers[];
};
```

Interfaces

Workflow

Userspace Application

```
  Client                                  Server
-------------------------------------------------------------------
sockfd = socket(IPPROTO_QUIC)           listenfd = socket(IPPROTO_QUIC)
bind(sockfd)                            bind(listenfd)
                                        listen(listenfd)

connect(sockfd)
quic_handshake(session)
                                        sockfd = accept(listenfd)
                                        quic_handshake(session)

sendmsg(sockfd)                         recvmsg(sockfd)
close(sockfd)                           close(sockfd)
                                        close(listenfd)
```

- connect() only initializes the initial keys and connection IDs, and does routing.
- quic_handshake() initials a handshake, then peer returns a new socket and process the handshake.

Workflow

Kernel Consumer

Client	Server
`__sock_create(IPPROTO_QUIC, &sock)`	`__sock_create(IPPROTO_QUIC, &sock)`
`kernel_bind(sock)`	`kernel_bind(sock)`
	`kernel_listen(sock)`
`kernel_connect(sock)`	
`tls_client_hello_x509(args:sock)`	
	`kernel_accept(sock, &newsock)`
	`tls_server_hello_x509(args:newsock)`
`kernel_sendmsg(sock)`	`kernel_recvmsg(newsock)`
`sock_release(sock)`	`sock_release(newsock)`
	`sock_release(sock)`

- tlshd service (from ktils-utils) must be started in userspace.
- tls_client_hello_x509() and tls_server_hello_x509() are APIs from net/handshake.

Socket API

Control Messages

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

struct msghdr {
    ...
    void *msg_control;    /* ancillary data */
    socklen_t msg_controllen; /* ancillary data buffer length */
};

struct cmsghdr {
    socklen_t cmsg_len; /* # bytes, including this header */
    int cmsg_level;     /* originating protocol */
    int cmsg_type;      /* protocol-specific type */
                        /* followed by unsigned char cmsg_data[]; */
};
```

```
cmsg_level:
#define SOL_QUIC        288


cmsg_type:
enum quic_cmsg_type {
    QUIC_STREAM_INFO,
    QUIC_HANDSHAKE_INFO,
};
```

```
cmsg_data:
struct quic_stream_info {
        __s64    stream_id;
        __u32    stream_flags;
};

struct quic_handshake_info {
        __u8     crypto_level;
};
```

Socket API

Socket Options

```
During_Handshake:

#define QUIC_SOCKOPT_CRYPTO_SECRET              13
#define QUIC_SOCKOPT_TRANSPORT_PARAM_EXT        14

Prior_to_Handshake:

#define QUIC_SOCKOPT_TRANSPORT_PARAM            8
#define QUIC_SOCKOPT_CONFIG                     9
#define QUIC_SOCKOPT_TOKEN                      10
#define QUIC_SOCKOPT_ALPN                       11
#define QUIC_SOCKOPT_SESSION_TICKET             12

Post_Handshake:

#define QUIC_SOCKOPT_EVENT                      0
#define QUIC_SOCKOPT_STREAM_OPEN                1
#define QUIC_SOCKOPT_STREAM_RESET               2
#define QUIC_SOCKOPT_STREAM_STOP_SENDING        3
#define QUIC_SOCKOPT_CONNECTION_ID              4
#define QUIC_SOCKOPT_CONNECTION_CLOSE           5
#define QUIC_SOCKOPT_CONNECTION_MIGRATION       6
#define QUIC_SOCKOPT_KEY_UPDATE                 7
```

Socket API

Stream Peeloff

- Bind a stream to a separate socket (suggested by John Ericson).

```
#define QUIC_SOCKOPT_STREAM_PEELOFF                    xx
```

Example

```
/* Open an unidirectional stream */
optlen = sizeof(sinfo);
sinfo.stream_id = -1;
sinfo.stream_flags = MSG_STREAM_UNI;
ret = getsockopt(sockfd, SOL_QUIC, QUIC_SOCKOPT_STREAM_OPEN,
                 &sinfo, &optlen);
if (ret)
    return -1;

/* Peel off the stream from the socket */
optlen = sizeof(pinfo);
pinfo.stream_id = sinfo.stream_id;
strmfd = getsockopt(sockfd, SOL_QUIC, QUIC_SOCKOPT_STREAM_PEELOFF,
                    &pinfo, &optlen);
if (strmfd < 0)
    return -1;

/* Send messages without stream_info cmsg */
ret = send(strmfd, msg, strlen(msg), 0);
```

Socket API

Notifications

```
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

struct msghdr {
    ...
    int msg_flags;              /* flags on message */
};
```

msg_flags:

```
enum quic_msg_flags {
    ...
    MSG_QUIC_NOTIFICATION           = MSG_MORE,
};
```

msg_data:

```
enum quic_event_type {
    QUIC_EVENT_NONE,
    QUIC_EVENT_STREAM_UPDATE,
    QUIC_EVENT_STREAM_MAX_DATA,
    QUIC_EVENT_STREAM_MAX_STREAM,

    QUIC_EVENT_CONNECTION_ID,
    QUIC_EVENT_CONNECTION_CLOSE,
    QUIC_EVENT_CONNECTION_MIGRATION,

    QUIC_EVENT_KEY_UPDATE,
    QUIC_EVENT_NEW_TOKEN,
    QUIC_EVENT_NEW_SESSION_TICKET,
    QUIC_EVENT_MAX,
};
```

TLS API

```c
/**
 * gnutls_quic_handshake:
 * @session: a #gnutls_session_t initialized for QUIC transport.
 *
 * Perform a TLS 1.3 handshake over a QUIC transport.
 *
 * Returns: %GNUTLS_E_SUCCESS on success, otherwise a negative error code.
 **/
int gnutls_quic_handshake(gnutls_session_t session);
```

Example

```c
gnutls_certificate_credentials_t x509_cred;
gnutls_session_t session;

gnutls_certificate_allocate_credentials(&x509_cred);

gnutls_init(&session, GNUTLS_CLIENT); /* GNUTLS_SERVER for server side */
gnutls_handshake_set_timeout(session, 0);

gnutls_priority_set_direct(session, prio, NULL);

gnutls_credentials_set(session, GNUTLS_CRD_CERTIFICATE, x509_cred);

/* fd is QUIC socket after connect() for client or accept() for server. */
gnutls_transport_set_int(session, fd);

ret = gnutls_quic_handshake(session);
```

Use Cases

Samba Integration

- Linux QUIC has been integrated into Samba:
  `https://gitlab.com/samba-team/samba/-/merge_requests/4019`
- Stefan Metzmacher's slides:
  `https://www.samba.org/~metze/presentations/2025/SDC/StefanMetzmacher_SDC2025-transports-rev1-compact.pdf`

SMB and NFS Progress

- QUIC handshake is supported in ktls-utils:
  `https://github.com/oracle/ktls-utils`
- SMB in developing and should be ready after Linux QUIC is merged, Steve French's slides:
  `https://sambaxp.org/fileadmin/user_upload/sambaxp2024-Slides/sxp24-French-improving_network_progress.pdf`
- NFS over QUIC is in planning:
  `https://datatracker.ietf.org/doc/html/draft-cel-nfsv4-rpc-over-quicv1-05`

Curl Experiments

- Linux QUIC is being integrated into Curl:
  `https://github.com/moritzbuhl/curl/tree/linux_curl`
- Moritz Buhl's presentation:
  `https://www.youtube.com/watch?v=TX3ZLRXXopI`
- httpd deployed a http server based on httpd-portable over linux QUIC:
  `https://d.moritzbuhl.de/pub`

Testing

Interoperability

Functions

- Test Suite:
  `https://github.com/quic-interop/quic-interop-runner`

Interop Status

	ngtcp2	quiche	picoquic	neqo	msquic	linuxquic
ngtcp2	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2 CM BP BA	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R Z B U A L1 L2 C1 C2 6 V2 BP BA 3 E CM	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM
quiche	H DC LR M S R Z 3 B A L1 L2 C2 6 BP BA C20 U E V2 C1 CM	H DC LR M S R Z 3 B A L1 L2 C2 6 C20 U E V2 CM C1 BP BA	H DC LR M S R Z 3 B A L1 L2 C1 C2 6 BP BA C20 U E V2 CM	H DC LR M S R 3 B A L1 L2 C1 C2 6 BP BA C20 U E V2 Z CM	H DC LR M S R Z A L2 C2 6 BP BA C20 3 U E V2 CM B L1 C1	H DC LR R Z 3 B A L2 C2 6 BP BA C20 U E V2 M S L1 C1 CM
picoquic	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2 CM BP BA	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM Z	H DC LR C20 M S R Z B U A L1 L2 C2 6 V2 BP BA 3 E CM C1	H DC LR C20 M S R Z 3 B U E A L1 L2 C2 6 V2 BP BA CM C1
neqo	H DC LR C20 M S R Z 3 B U E L1 L2 C1 C2 6 V2 BP BA A CM	H DC LR C20 M S R Z 3 B U L2 C1 C2 6 E V2 CM A L1 BP BA	H DC LR C20 M S R Z 3 B U E L1 L2 C2 6 V2 BP BA CM A C1	H DC LR C20 M S R 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM Z	H DC LR C20 M S R Z B U L2 C2 6 V2 BP BA 3 E CM A L1 C1	H DC LR C20 M S R Z 3 B U E L1 L2 C2 6 V2 BP BA CM A C1
msquic	H DC LR C20 M S R Z B U A L1 L2 C1 C2 6 3 E V2 BP BA CM	H DC LR C20 M S R Z B U A L1 L2 C2 6 3 E V2 CM C1 BP BA	H DC LR C20 M S R Z B U A L1 L2 C1 C2 6 BP BA 3 E CM V2	H DC LR C20 M S R B U A L1 L2 C1 C2 6 V2 BP BA 3 E Z CM	H DC LR C20 M S R Z B U A L1 L2 C1 C2 6 V2 BP BA 3 E CM	H DC LR C20 M S R B U A L1 L2 C1 C2 6 BP BA 3 E Z V2 CM
linuxquic	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R Z 3 B U A L2 C1 C2 6 E V2 CM L1 BP BA	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM	H DC LR C20 M S R Z B U A L1 L2 C1 C2 6 V2 BP BA 3 E CM	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2 BP BA CM

Interoperability

Measurements

Measurement Results

	ngtcp2	quiche	picoquic	neqo	msquic	linuxquic
ngtcp2	G: 8852 (± 36) kbps C: 4925 (± 238) kbps	G: 9321 (± 17) kbps C: 3786 (± 190) kbps	G: 9208 (± 6) kbps C: 4554 (± 1547) kbps	G: 9423 (± 7) kbps C: 3808 (± 131) kbps	G: 9367 (± 8) kbps C: 8065 (± 300) kbps	G: 9125 (± 85) kbps C: 3502 (± 55) kbps
quiche	G: 8874 (± 40) kbps C: 4817 (± 163) kbps	G: 8570 (± 129) kbps C: 2576 (± 58) kbps	G: 9117 (± 8) kbps C: 6253 (± 1379) kbps	G: 9356 (± 3) kbps C: 4454 (± 256) kbps	G: 7529 (± 356) kbps C	G: 9219 (± 69) kbps C: 4421 (± 133) kbps
picoquic	G: 9011 (± 56) kbps C: 4978 (± 161) kbps	G: 9446 (± 20) kbps C: 7588 (± 217) kbps	G: 9260 (± 6) kbps C: 5566 (± 1440) kbps	G: 9498 (± 5) kbps C: 4984 (± 167) kbps	G: 9164 (± 96) kbps C: 8911 (± 209) kbps	G: 8627 (± 93) kbps C: 4416 (± 97) kbps
neqo	G: 8916 (± 64) kbps C: 4371 (± 134) kbps	G: 9299 (± 23) kbps C: 3078 (± 150) kbps	G: 9318 (± 6) kbps C: 5279 (± 1515) kbps	G: 9540 (± 13) kbps C: 3612 (± 361) kbps	G: 9488 (± 9) kbps C: 7327 (± 160) kbps	G: 9111 (± 191) kbps C: 4050 (± 214) kbps
msquic	G: 8901 (± 87) kbps C: 4357 (± 80) kbps	G: 9413 (± 5) kbps C	G: 9419 (± 48) kbps C	G: 9515 (± 11) kbps C	G: 9501 (± 8) kbps C: 7212 (± 653) kbps	G: 9130 (± 3) kbps C
linuxquic	G: 9127 (± 86) kbps C: 4249 (± 186) kbps	G: 9074 (± 105) kbps C: 4297 (± 152) kbps	G: 9403 (± 19) kbps C: 6015 (± 1797) kbps	G: 9282 (± 121) kbps C: 5482 (± 289) kbps	G: 8038 (± 938) kbps C: 7522 (± 187) kbps	G: 9316 (± 4) kbps C: 3811 (± 263) kbps

Syzkaller Fuzzing

- Test Case:
  `https://github.com/lxin/quic/tree/main/tests/syzkaller`
- Install test case into syzkaller:

```
# git clone https://github.com/google/syzkaller
# cd syzkaller/
# cp ${Test Case}/socket_inet_quic.txt* sys/linux/
# make all && cp bin/* /usr/local/bin/
```

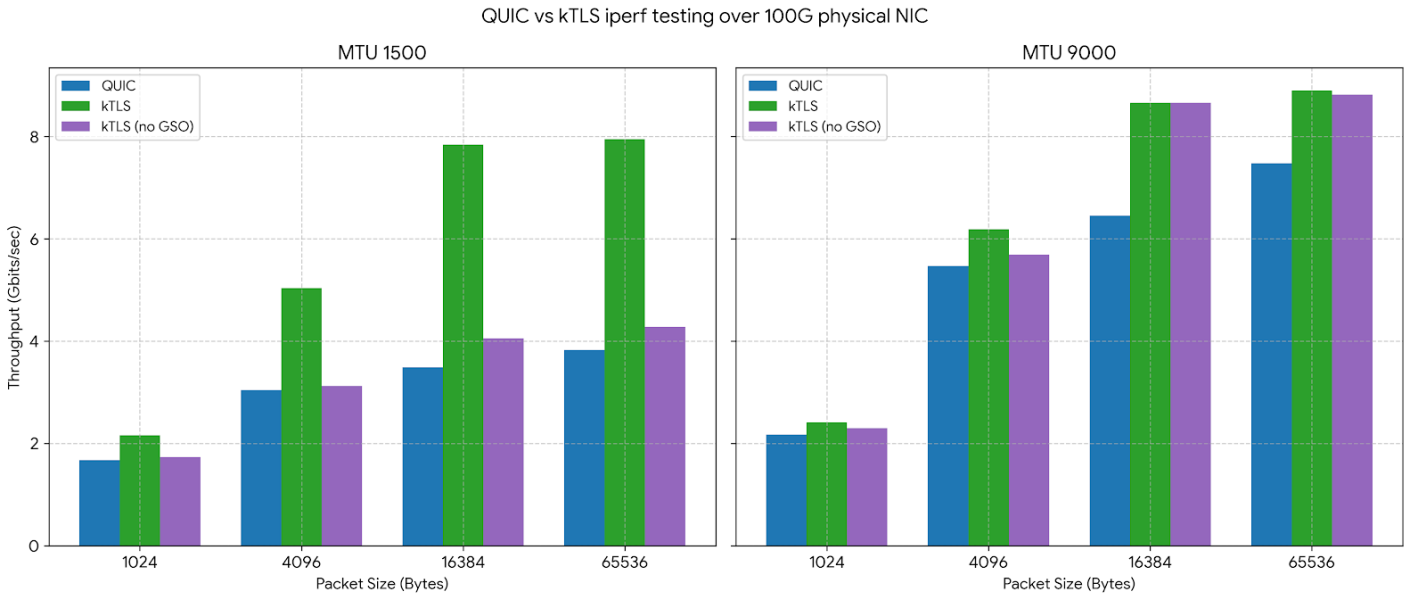
- Run test with enable_syscalls in syzkaller-test.cfg:

```
"enable_syscalls" : [
  "socket$inet_quic",
  "socket$inet6_quic",
  "getsockopt$inet_quic6_*",
  "getsockopt$inet_quic_*",
  "setsockopt$inet_quic6_*",
  "setsockopt$inet_quic_*",
  "ioctl$sock_inet6_quic_*",
  "ioctl$sock_inet_quic_*",
  "getsockname", "getpeername",
  "bind", "listen", "connect", "accept", "accept4", "close",
  "shutdown", "sendmsg", "sendmmsg", "sendto", "recvmsg",
  "recvmmsg", "recvfrom", "epoll_create", "epoll_create1",
  "epoll_ctl", "epoll_wait", "epoll_pwait", "epoll_pwait2",
  "mmap", "poll", "ppoll", "pread64", "preadv", "select",
  "pselect6", "sendfile"
],
```

Performance

Test Results vs kTLS

- Testbed and Test tools:
  - Test on 100G NIC with various MTUs and packet sizes
  - Linux QUIC tool: `https://github.com/lxin/iperf`
  - kTLS tool: `https://github.com/Mellanox/iperf_ssl`
- Test Results:



Performance

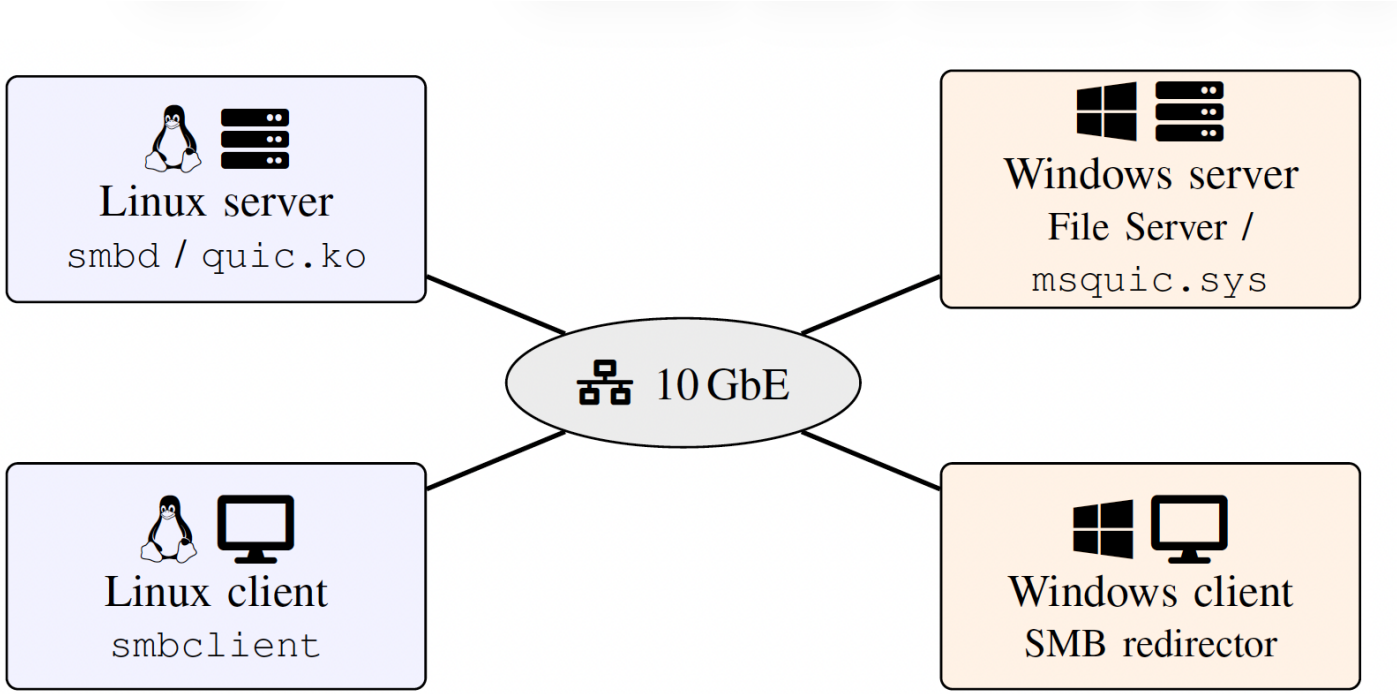
Gap Causes vs KTLS

- The absence of Generic Segmentation Offload (GSO) for QUIC.
- An additional data copy on the transmission (TX) path.
- Extra encryption required for header protection in QUIC.

Performance

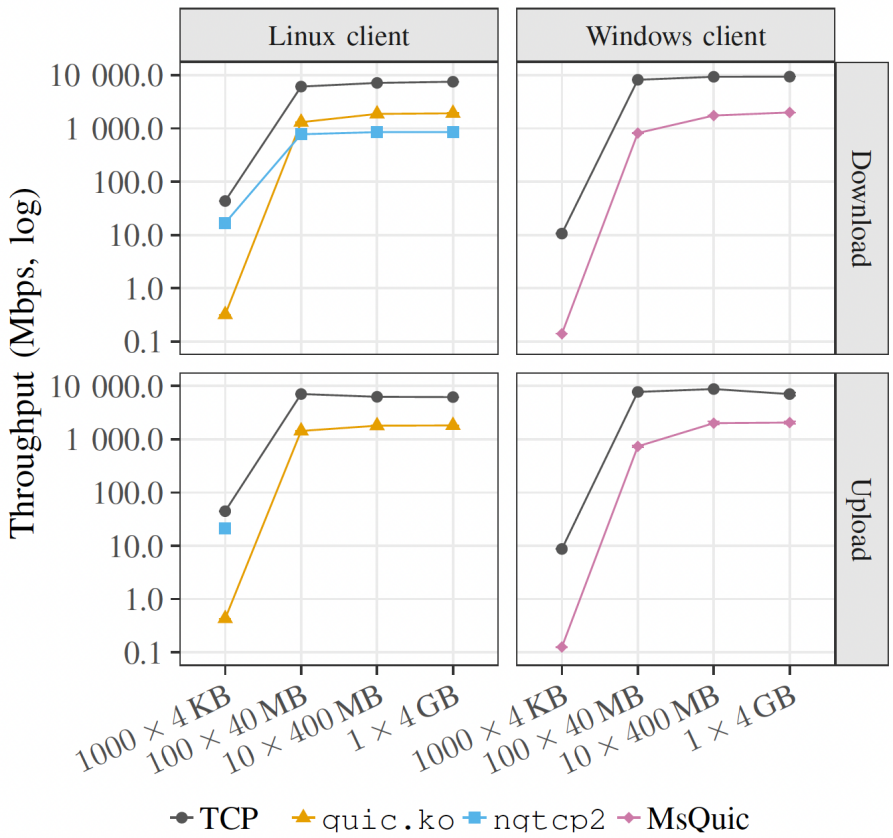
Testbed vs Userland QUICs

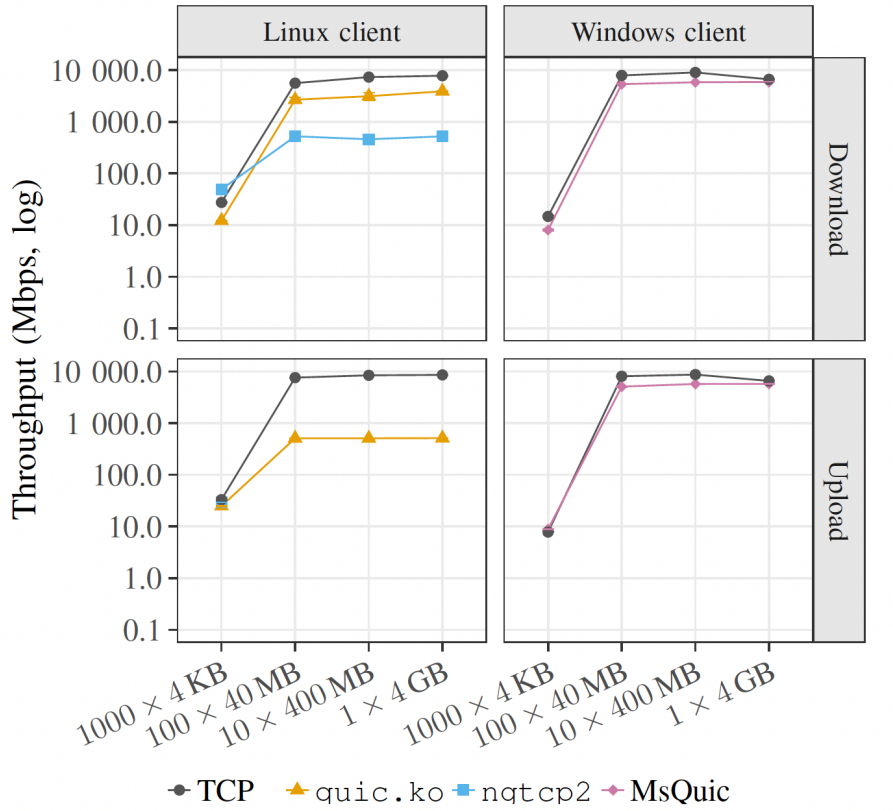
- Sebastian Rust's Testbed with Samba over QUIC:



Performance

Test Results vs Userland QUICs





Future Work

Upstreaming

1. Development:
   - Github Repo: `https://github.com/lxin/quic`
   - Mailing list: `quic@lists.linux.dev`
2. Kernel progress:
   - net: introduce QUIC infrastructure and core subcomponents: `https://lore.kernel.org/quic/`
   - net: implement QUIC data path, socket APIs, and documentation: `https://github.com/lxin/net-next/commits/quic/`
   - net: add sample tests and selftests for QUIC
3. Gnutls API draft:
   - gnutls_quic_handshake(): `https://github.com/lxin/gnutls/tree/quic-handshake`

NIC Offload

1. Framework: struct quicdev_ops (similar to xfrm and tls), we have some draft code already.
2. Marc Fiuczynski is building its own on AMD U200 fpga NIC, should be available and tested soon.
3. Broadcom, supports TX, and has planned RX flow offload. Provided dev_ioctl() interface for userland QUIC.
4. Intel, E810 QUIC DDP, not compatible with standardized QUIC.

API Evolution

1. Based on the use cases above, we have been proposing the socket APIs at an I-D:
   `https://datatracker.ietf.org/doc/html/draft-lxin-quic-socket-apis`
2. We're still exploring other new use cases, and would like to expand more socket APIs for your use cases.

Questions / Comments