Improving the network stack: progress on QUIC and SMB3.1.1 for Linux

Presented by Steve French
Principal Software Engineer
Microsoft Azure Storage
- This work represents the views of the author(s) and does not necessarily reflect the views of Microsoft Corporation.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
Who am I?

- Steve French smfrench@gmail.com
- Author and maintainer of Linux cifs vfs (for accessing Samba, Azure, Windows and various SMB3/CIFS based NAS appliances)
- Co-maintainer of the kernel server (ksmbd)
- Member of the Samba team (co-creator of the “net” utility)
- Coauthor of SNIA CIFS Technical Reference, former SNIA CIFS Working Group chair
- Principal Software Engineer, Azure Storage: Microsoft
Outline

- Overview of QUIC
- Review of Xin Long’s status update on QUIC in Linux kernel
- Discussion of cifs.ko changes needed
- Coming soon … what to look forward to
- Testing discussion
Why is QUIC important?
We could ask ChatGPT ...

ChatGPT

QUIC (Quick UDP Internet Connections) offers several advantages over TCP as the transport protocol for SMB3.11 (Server Message Block version 3.1.1), particularly in terms of performance, security, and reliability.

1. **Reduced Latency**: QUIC is designed to minimize latency by reducing the number of round trips required for establishing a connection. It achieves this by combining the initial handshake and encryption setup into a single round trip, resulting in faster connection establishment compared to TCP.

2. **Multiplexing**: QUIC supports multiplexing of multiple streams within a single connection. This enables concurrent transmission of multiple data streams over a single connection, reducing the overall latency and improving throughput, especially in scenarios where multiple files are being transferred simultaneously in SMB.

3. **Connection Migration**: QUIC allows for seamless connection migration between different network interfaces or IP addresses without disrupting ongoing data transmission. This is particularly advantageous for mobile devices or scenarios where network conditions change frequently, ensuring continuous connectivity and minimizing downtime.
4. **Improved Security:** QUIC incorporates modern encryption protocols such as TLS 1.3 by default, providing robust encryption for data transmission and enhancing security compared to older versions of TLS used with TCP. Additionally, QUIC mitigates certain vulnerabilities associated with TCP, such as SYN flooding attacks, by utilizing UDP encapsulation.

5. **Adaptive Congestion Control:** QUIC includes built-in mechanisms for adaptive congestion control, which dynamically adjusts transmission parameters based on network conditions such as packet loss and latency. This helps optimize network utilization and ensures efficient use of available bandwidth, leading to improved performance and reliability.

6. **Faster Handshake:** QUIC's connection establishment process is optimized for efficiency, requiring fewer round trips compared to TCP's three-way handshake. This results in quicker initiation of data transfer sessions, reducing overall latency and improving user experience, especially for short-lived connections such as those typical in SMB file transfers.

By leveraging these advantages, using QUIC as the transport protocol for SMB3.1 can significantly enhance the performance, security, and reliability of file transfer operations, particularly in modern network environments where low latency, multiplexing, and adaptive congestion control are crucial requirements.
But it misses some of the obvious …

- It solves the “port 445” problem (e.g. it can be hard for me to demo SMB3.1.1 when Spectrum blocks port 445). SMB3.1.1 mounts to anywhere … save your files more easily into the cloud …

- It also could allow a mix of machine credentials (for QUIC encryption) and user credentials … which may be helpful for some workloads

- If QUIC is also supported in unencrypted mode (as some QUIC drivers allow for testing) it will likely be faster than TCP (if encrypting it is still a good alternative to requiring SMB3.1.1 GCM encryption)

- Will there be cases where “double encrypting” (at QUIC layer and also for SMB3.1.1 layer) will be valuable?
But it still would be awesome due to ...

- No “head of line blocking”
- Better congestion control
- Faster session establishment
- Reduced retransmissions (and Forward Error Correction)
- Large percentage of internet traffic is already over QUIC (and growing)
So why don’t we have QUIC yet on Linux?

• Well … we do in userspace, multiple drivers
  - e.g. https://github.com/microsoft/msquic is well tested and works on Linux

• But unlike Windows, we don’t have a kernel driver

• And many of the open source QUIC drivers wouldn’t port well to Linux kernel (even if only porting minimal required function) and don’t follow kernel coding style

• Last year (June 24th) – Xin Long started a kernel project which is looking VERY promising
Is this just for Windows?

- Ned’s slides and learn.microsoft.com explain more about Windows SMB3.1.1 over QUIC, but Visuality systems also supports SMB3.1.1 over QUIC
- QUIC will be a great addition to cifs.ko, ksmbd.ko and Samba server and tools
Windows QUIC SMB3.1.1 config
Example Windows mount from Ned’s post

```
PS C:\> net use \WIN-ORCV2RI0P4C.corp.contoso.com\c$ /p:n /transport:QUIC
System error 67 has occurred.
The network name cannot be found.

PS C:\> New-SmbServerCertificateMapping -Name WIN-ORCV2RI0P4C.corp.contoso.com -Thumbprint de8730b472adc3cd2d832387b8ff0f84891e09cd -StoreName My

<table>
<thead>
<tr>
<th>Name</th>
<th>Subject</th>
<th>Thumbprint</th>
<th>DisplayName</th>
<th>StoreName Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN-ORCV2RI0P4C.corp.contoso.com</td>
<td>CN=quiceverywhere</td>
<td>de8730b472adc3cd2d832387b8ff0f84891e09cd</td>
<td>quiceverywhere</td>
<td>My</td>
</tr>
</tbody>
</table>

PS C:\> net use \WIN-ORCV2RI0P4C.corp.contoso.com\c$ /p:n /transport:QUIC
Drive Z: is now connected to \WIN-ORCV2RI0P4C.corp.contoso.com\c$.
The command completed successfully.
PS C:\>
```
Diagram showing SMB3.1.1 over QUIC (for Windows)

- (Thanks to Obaid)

### SMB/QUIC: Client

1. Client opens `\ServerName\Share\foo.tst`
2. Client resolves `ServerName` using DNS
3. Client attempts TCP and QUIC simultaneously*
4. Client will start using whichever connects first
5. Client’s multichannel will negotiate interfaces with server and will select most optimal protocols
6. Client sends SMB messages
The New Kernel QUIC driver

QUIC in Linux Kernel

Overview

As mentioned in [https://github.com/bin/quic](https://github.com/bin/quic): "some people may argue that TLS handshake should stay in user space and use syscall to user space in kernel to complete the handshake. This repo is to..."
New Kernel QUIC driver

• What I am impressed with so far:
  – Kernel style and code looks good
  – Very easy to build the kernel driver and run their functional tests
  – Reasonable size (20.4KLOC about ½ of which is kernel code)
  – Good set of functional tests included (it passed all of them when I tried)
  – Maintainers have been responsive to emails and questions
New Kernel QUIC driver

- Easy to build and install (see quic/README.md)
Overview of the proposed kernel QUIC driver

Next 15 slides courtesy of Xin Long <lucien.xin@gmail.com> See https://github.com/lxin/quic
Thanks to Xin and also for contributors to the code e.g. Pengtao He <hepengtao@xiomi.com> and Tyler Fanelli <tfanelli@redhat.com>
What we’ll discuss today

▪ Background
▪ Implementation
▪ Usage
▪ Next Step
Background

- What is QUIC
- Why QUIC in Kernel is Needed
- In Kernel QUIC Evolution
What is QUIC

RFC9000 - A UDP-Based Multiplexed and Secure Transport

- UDP based
  - Connection Migration
- Multistreaming
- Secured by TLS 1.3
  - Rekeying/Session Resumption/0-RTT
- Transport Protocol
  - Flow Control/Congestion Control

Other RFCs:
- RFC8999 - Version Negotiation
- RFC9001 - Using TLS to Secure
- RFC9002 - Congestion Control
- RFC9221 - Unreliable Datagram
- RFC9287 - Greasing the QUIC Bit
- RFC9368 - Version Negotiation 2
- RFC9369 - QUIC Version 2
Why QUIC in Kernel is Needed

- **Kernel Consumers**
  - SMB, NFS ...

- **Socket APIs**
  - listen/accept/connect/send/recv/close/get/setsockopt ...

- **Avoiding Data Copies**
  - zero-copy via sendfile()

- **Offloading in NICs**
  - Similar to TLS/IPsec offloading

- **Less Interoperability Issues**
  - Too many Userland QUIC implementations
In Kernel QUIC Evolution

In-Kernel QUIC ≠ Crypto Offloading for Userland QUIC

- net: support QUIC crypto (a patchset posted on 2022)
- Offloading Encryption to QUIC Enabled NICs (LPC2023)
- TLS 1.3 Handshake in kernel/lib
- TLS Handshake netlink in kernel/net (upstream kernel)
- Long/Handshake Packets processed by libngtcp2
Implementation

- Idea
- Handshake Architecture
- User Data Architecture
- Socket Process
Idea

In-kernel QUIC with Userspace Handshake

- Userspace: Only raw TLS Handshake Messages creating and processing
- Kernel: Create IPPROTO_QUIC type socket running over UDP tunnels
  - No protocol number needed from IANA, like IPPROTO_MPTCP
  - Reasons why ULP layer is not used:
    - Connection Migration
    - Common Socket APIs
    - Transport Protocol
- Kernel consumers use handshake netlink to request a handshake
Handshake Architecture

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>APP1</td>
<td>APP2</td>
</tr>
<tr>
<td>ktls-utils: libquic:</td>
<td>quic_handshake_server/client/param()</td>
</tr>
<tr>
<td>send/recvmsg()</td>
<td>set/getsockopt()</td>
</tr>
<tr>
<td>CMSG handshake_info</td>
<td>SOCKOPT_CRYPTO_SECRET/TRANSPORT_PARAM_EXT</td>
</tr>
<tr>
<td>set/getsockopt()</td>
<td>set/getsockopt()</td>
</tr>
<tr>
<td>SOCKOPT_CRYPTO_SECRET/TRANSPORT_PARAM_EXT</td>
<td>set/getsockopt()</td>
</tr>
<tr>
<td>Userspace</td>
<td>Kernel</td>
</tr>
<tr>
<td>Handshake Netlink APIs</td>
<td>Handshake Netlink APIs</td>
</tr>
<tr>
<td>ktls-utils: tlshd</td>
<td>ktls-utils: tlshd</td>
</tr>
<tr>
<td>smb</td>
<td>nfs</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

socket (IPPROTO_QUIC) | Protocol
stream | connection_id | cong | path | timer
packet | frame | crypto | pnamp
input | output
UDP tunnels
User Data Architecture

<table>
<thead>
<tr>
<th>socket (IPPROTO_QUIC)</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>connection_id</td>
</tr>
<tr>
<td>packet</td>
<td>frame</td>
</tr>
<tr>
<td>input</td>
<td>output</td>
</tr>
</tbody>
</table>

APP1

APP2

... | set/getsockopt() | SOCKOPT_KEY_UPDATE/CONNECTION_MIGRATION/STREAM_OPEN/RESET/STOP ...

Userspace

send/recvmsg()

CMSG stream_info

Kernel

send/recvmsg()

set/getsockopt()

SMB

NFS

...
Socket Process

Client:

- SK Hashtable
- Conn ID Hashtable

Client sends:
- Initial
- Handshake
- User Data

Server:

- listen sk
- reqsk
- SK Hashtable
- Conn ID Hashtable

Server receives:
- sk

Server action:
- = accept()
Usage

- From Userspace
- From Kernel
- Advanced APIs
### From Userspace

#### Client

```c
sockfd = socket(IPPROTO_QUIC)
bind(sockfd)
connect(sockfd)
quic_client_handshake(sockfd)
```

```c
sendmsg(sockfd)
close(sockfd)
```

#### Server

```c
listenfd = socket(IPPROTO_QUIC)
bind(listenfd)
listen(listenfd)
```

```c
sockfd = accept(listenfd)
quic_server_handshake(sockfd, cert)
```

```c
recvmsg(sockfd)
close(sockfd)
close(listenfd)
```

---

Sample: [https://github.com/lxin/quic/blob/main/tests/sample_test.c](https://github.com/lxin/quic/blob/main/tests/sample_test.c)
From Kernel

- **Client**
  
  ```c
  __sock_create(IPPROTO_QUIC, &sock)
  kernel_bind(sock)
  
  kernel_connect(sock)
  tls_client_hello_x509(args:{sock})
  
  kernel_sendmsg(sock)
  sock_release(sock)
  ```

- **Server**
  
  ```c
  __sock_create(IPPROTO_QUIC, &sock)
  kernel_bind(sock)
  
  kernel_listen(sock)
  
  kernel_accept(sock, &newsock)
  tls_server_hello_x509(args:{newsock})
  
  kernel_recvmsg(newsock)
  sock_release(newsock)
  sock_release(sock)
  ```

Sample: [https://github.com/lxin/quic/blob/main/net/quic/sample_test.c](https://github.com/lxin/quic/blob/main/net/quic/sample_test.c)
Advanced APIs

- **Control Message**
  - `sendmsg()` with cmsg stream info to set stream_id and flag
  - `recvmsg()` with cmsg stream info to get stream_id and flag

- **Socket Options**
  - `SOCKOPT_KEY_UPDATE` for rekeying
  - `SOCKOPT_CONNECTION_MIGRATION` for connection migration
  - `SOCKOPT_STREAM_OPEN/RESET/STOP_SENDING` for stream management
  - `SOCKOPT_SESSION_TICKET` for session resumption and 0-RTT Data

- **Notification/Events**
  - Connection Update
  - Stream Update

**Man doc:** [https://github.com/lxin/quic/wiki/man](https://github.com/lxin/quic/wiki/man)

**Examples:** [https://github.com/lxin/quic/tree/main/tests](https://github.com/lxin/quic/tree/main/tests)
Next Step

- HW crypto offloading
- Internet Draft For QUIC Sockets API Extensions
Kernel QUIC specialities:

- Enables use for Kernel Consumers
- Common easy to understand Socket APIs, like listen/accept/connect/send/recv/close.
- Avoid Data Copies from Users to Kernel via sendfile() syscall.
- Easy to cooperate with offloading in NICs, no configuration needed from userspace
- Less Interoperability Issues to implement QUIC in OS
Recent WIP summary from Xin Long

What’s next for their driver (net/quic/quic.ko):

1. Improve the performance and complete the congestion control code (there's some testing data vs kTLS in github README)

2. Collaborate with Andy from Broadcom to implement the infrastructure for the Offloading inside HW NICs.

3. Work with another colleague to start an RFC doc standardizing the socket APIs for OS-level QUIC implementation.
What about the SMB3.1.1 pieces

- Changes on the SMB3.1.1 client side are small
- New socket type to support (see slide 26)
- ALPN (app layer protocol negotiation for TLS) Identification sequence used to identify the SMB2 protocol over QUIC is 0x73 0x6D 0x62 ("smb")
- See section 4.10 of MS-SMB2 and section 2.2.3.1.5 (SMB2_TRANSPORT_CAPABILITIES negotiate context which we will have to parse)
Next steps

- Continue following up with Dave and the network maintainers if we have updates on our SMB3.1.1 testing with it
- Continue building and testing Xin’s git tree until it is in linux-next or mainline (when we can move to using that version)
- Add SMB3.1.1 code to fs/smb/client/transport.c to read/write over the new QUIC driver
- Add SMB3.1.1 code to process the transport capabilities SMB3.1.1 negotiate context to fs/smb/client/smb2pdu.c
More Next steps

- What about user space tooling?
  - updates to cifs-utils and/or Samba tools could help make this easier for users
  - New mount option for cifs.ko to force use of QUIC if available
- What about certificate setup and exchange?
  - Could Samba tooling help?
- Encourage code contributions, review and testing ...
  - Would love some help on this exciting feature
Testing Improvements
Test ... test ... test ...
Additional tests are encouraged (quic or smb specific)

- See the tests subdirectory of the quic github tree
- The basic xfstests should be fine for testing SMB3.1.1 over QUIC, but we will need to come up with some reconnect tests (today mostly in the buildbot’s “DFS” test group) to exercise reconnect with the new code, and will need to test various encryption options (QUIC only, SMB3.1.1 only and both QUIC and SMB3.1.1 encryption)
quic function tests

- To run the functional tests
  - cd tests
  - make run
  - (runs 192 functional tests
  - then does various perf tests)
Xfstest automation

- Will also be important to add xfstest scenarios (e.g. local.config using a mount option that forces use of QUIC instead of TCP)
- Example here is over TCP and runs 100s of fs tests
Thank you for your time

- Future is very bright!
Additional Resources to Explore

https://github.com/lxin/quic

And an alternative open source implementation in userspace:
https://github.com/microsoft/msquic

https://techcommunity.microsoft.com/t5/storage-at-microsoft/smb-over-quic-is-ga-and-we-have-lots-of-news/ba-p/2928695
and there is also a newer post on SMB3.1.1 and QUIC by Ned

https://www.snia.org/sites/default/files/SDCEMEA/2021/snia-SMB-over-QUIC.pdf


Two research papers with interesting observations about areas where Linux QUIC drivers can improve perf:
https://publikationen.bibliothek.kit.edu/1000161904/152028985