QUIC for the kernel

By Jonathan CorbetJuly 22, 2025

The QUIC transport-layer network protocol is not exactly new; it was first covered here in 2013. Despite carrying a significant part of the traffic on the Internet, QUIC has been anything but quick when it comes to getting support into the Linux kernel. The pace might be picking up, though; Xin Long has posted the first set of patches intended to provide mainline support for this protocol.

QUIC was created to address a number of problems that have been observed with TCP on the modern Internet. The three-way handshake at the core of the TCP connection protocol adds latency to connections, causing the next cat video to be that much slower to arrive. TCP was not designed to support multiple simultaneous data streams; it suffers from head-of-line blocking, in which a dropped packet brings everything to a halt. All told, TCP does not perform as well as one might like for that all-important web-browsing use case.

TCP also transmits much of its connection metadata in the clear, where any party between the endpoints can read it. That can result in information leaks. But middleboxes on the Internet also make free use of connection information to filter out anything that does not match their idea of how a TCP connection should work. The result is protocol ossification — the inability to make any changes to the TCP protocol because the result will not survive transmission across the Internet. Attempts to improve TCP, such as multipath TCP, have to be carefully disguised as ordinary TCP to function at all. TCP has become almost impossible to improve.

QUIC is an attempt to address all of these problems. A streamlined connection-setup process eliminates the three-way handshake, making the establishment of connections faster. The protocol is built on top of UDP, and is designed with multiple streams in mind; the loss of one UDP packet will not affect any streams that did not have data in that packet. QUIC-specific transport data is contained within the UDP packets, and is always end-to-end encrypted, so middleboxes have no chance to inspect it. If UDP packets can get through, anything that QUIC does can get through as well.

The QUIC protocol is specified in RFC 9000, with some tweaks made in RFC 9369. The protocol is supported by a lot of software — particularly web browsers — found on a typical Linux system and is said to handle a majority of the connections to Google's servers, but the implementation is entirely in user space. This approach was taken to speed the development and distribution of QUIC; the people at Google who were pushing it did not want to have to wait until operating-system kernels with QUIC support were widely distributed. At this point, though, the evolution of the protocol has slowed, and minds are naturally turning toward kernel implementations, which hold the potential for better performance while making QUIC easily available to a wider range of applications.

The patch set aims to integrate QUIC as naturally as possible into the kernel. There is a new protocol type — `IPPROTO_QUIC` — that can be used with the `socket()` system call in the usual way. Calls to `bind()`, `connect()`, `listen()`, and `accept()` can be used to initiate and accept connections in much the same way as with TCP, but then things diverge a bit.

Within QUIC, TLS is used to manage authentication and encryption. Establishing a TLS session can involve a lot of complex, policy-oriented work involving certificate validation and more. As with the existing in-kernel TLS implementation, QUIC pushes that problem out to user space. Once a connection has been made, each side must handle the TLS handshake before the data can start flowing. The `sendmsg()` and `recvmsg()` system calls are used to carry out that setup; the libquic library and tlshd utility (from the ktls-utils project) can be used to handle that task. Once TLS setup is complete, data can flow normally between the endpoints.

It is worth noting that QUIC caches the results of the TLS negotiation on both sides of the connection. Once two systems have successfully connected, subsequent connections can skip most of the setup work, allowing data to be transmitted with the first packet.

QUIC is meant to be fast, but the benchmark results included with the patch series do not show the proposed in-kernel implementation living up to that. A comparison of in-kernel QUIC with in-kernel TLS shows the latter achieving nearly three times the throughput in some tests. A comparison between QUIC with encryption disabled and plain TCP is even worse, with TCP winning by more than a factor of four in some cases. Long offers some potential reasons for this difference, including the lack of segmentation offload support on the QUIC side, an extra data copy in transmission path, and the encryption required for the QUIC headers.

This performance gap will likely shrink over time. One of the motivations for getting QUIC into the kernel is to be able to take advantage of hardware-based protocol-offload functionality; that functionality does not really exist yet, but it seems clear that the network-interface vendors are interested in providing it. As QUIC gains the hardware support that TCP benefits from, and as the in-kernel implementation is further optimized, it should see some significant performance gains.

The current level of performance seemingly has not reduced interest in this implementation, though. There is an outstanding pull request [update: pulled on July 17] adding QUIC support to the Samba server and client implementations, and interest in adding that support for the in-kernel SMB and NFS filesystems. There is a repository adding kernel-based QUIC support to `curl`. Once QUIC is in the kernel, chances are that other applications will gain support as well.

That may take a little while, though. The posted series adds just over 9,000 lines to the kernel, and it is only the low-level support code; another series with the rest of the implementation is promised, but has not been posted as of this writing. The process of reviewing all that code has just begun, and can be expected to take some time; consider that the Homa protocol implementation remains unmerged after 11 revisions posted over nine months. QUIC may have a quicker experience than that, but one still should not expect to see it in the mainline before sometime in 2026 at best.