QUIC Geek FAQ (for folks that know about UDP, TCP, SPDY, and stuff like that)

What is QUIC? QUIC is the name for a new experimental protocol, and it stands for Quick UDP Internet Connection. The protocol supports a set multiplexed connections over UDP, and was designed to provide security protection equivalent to TLS/SSL, along with reduced connection and transport latency. An experimental implementation is being put in place in Chrome by a team of engineers at Google.

What are some of the distinctive techniques being tested in QUIC? QUIC will employ bandwidth estimation in each direction into congestion avoidance, and then pace packet transmissions evenly to reduce packet loss. It will also use packet-level error correction codes to reduce the need to retransmit lost packet data. QUIC aligns cryptographic block boundaries with packet boundaries, so that packet loss impact is further contained.

Doesn't SPDY already provide multiplexed connections over SSL? Yes, but SPDY currently runs across TCP, and that induces some undesirable latency costs (even though SPDY is already producing lower latency results than traditional HTTP over TCP).

Why isn't SPDY over TCP good enough? A single lost packet in an underlying TCP connection stalls all of the multiplexed SPDY streams over that connection. By comparison, a single lost packet for X parallel HTTP connections will only stall 1 out of X connections. With UDP, QUIC can support out-of-order delivery, so that a lost packet will typically impact (stall) at most one stream. TCP's congestion avoidance via a single congestion window also puts SPDY at a disadvantage over TCP when compared to several HTTP connections, each with a separate congestion window. Separate congestion windows are not impacted as much by a packet loss, and we hope that QUIC will be able to more equitably handle congestion for a set of multiplexed connections.

Are there any other reasons why TCP isn't good enough? TCP, and TLS/SSL, routinely require one or more round trip times (RTTs) during connection establishment. We're hopeful that QUIC can commonly reduce connection costs towards zero RTTs. (i.e., send hello, and then send data request without waiting).

Why can't you just evolve and improve TCP under SPDY? That is our goal. TCP support is built into the kernel of operating systems. Considering how slowly users around the world upgrade their OS, it is unlikely to see significant adoption of client-side TCP changes in less than 5-15 years. QUIC allows us to test and experiment with new ideas, and to get results sooner. We are hopeful that QUIC features will migrate into TCP and TLS if they prove effective.

Why didn't you build a whole new protocol, rather than using UDP? Middle boxes on the Internet today will generally block traffic unless it is TCP or UDP traffic. Since we couldn't significantly modify TCP, we had to use UDP. UDP is used today by many game systems, as well as VOIP and streaming video, so its use seems plausible.

Why does QUIC always require encryption of the entire channel? As we learned with SPDY and other protocols, if we don't encrypt the traffic, then middle boxes are guaranteed to (wittingly, or unwittingly) corrupt the transmissions when they try to "helpfully" filter or "improve" the traffic.

UDP doesn't have congestion control, so won't QUIC cause Internet collapse if widely adopted? QUIC employs congestion controls, just as it employs automatic retransmission to support reliable transport. QUIC will attempt to be fair with competing TCP traffic. For instance, when conveying Q multiplexed flows, and sharing bandwidth with T concurrent TCP flows, we will try to use resources in the range of Q / (Q+T) bandwidth (i.e., "a fair share" for Q additional flows).

Why didn't you use existing standards such as SCTP over DTLS? QUIC incorporates many techniques in an effort to reduce latency. SCTP and DTLS were not designed to minimize latency, and this is significantly apparent even during the connection establishment phases. Several of the techniques that QUIC is experimenting with would be difficult technically to incorporate into existing standards. As an example, each of these other protocols require several round trips to establish a connection, which is at odds with our target of 0-RTT connectivity overhead.

How much do QUIC's techniques reduce latency? This is exactly the question we are investigating at the moment, and why we are experimenting with various features and techniques in Chromium. It is too early to share any preliminary results - stay tuned.

Is there any way to disable QUIC, if I really want to avoid running it on my Chromium browser? Yes. You can visit `about:flags` and set the "Experimental QUIC protocol" to "Disabled."

Where can I learn more about QUIC? If you want a lot of background, and need material to help you sleep, you can look at the QUIC Design Document and Specification Rationale. For cryptographers that wonder how well the i's are dotted, and t's crossed, there is a QUIC Crypto Specification. If you'd rather see client code, you can take a look at the Chromium source directory. If you're wondering about what a server might have to do, there is some prototype server code. Finally, if you just want to think about the bits on the wire, and how this might look, there is an evolving wire specification.

Is there a news group for discussing QUIC? Yes. proto-quic@chromium.org a.k.a., https://groups.google.com/a/chromium.org/d/forum/proto-quic