A QUIC future in Rust

Dirkjan Ochtman
RustFest Paris, May 2018

https://github.com/djc/quinn

About me

 **@djc**  **@djco**

- https://dirkjan.ochtman.nl/

- Rust projects: Askama, tokio-imap & imap-proto

- Substantial contributions: *ring*, rustls, cargo, planetrs, Gentoo Linux maintainer for rust/cargo ebuilds

- Currently in between jobs. Would love to help you or your employer fill out gaps in the Rust ecosystem!

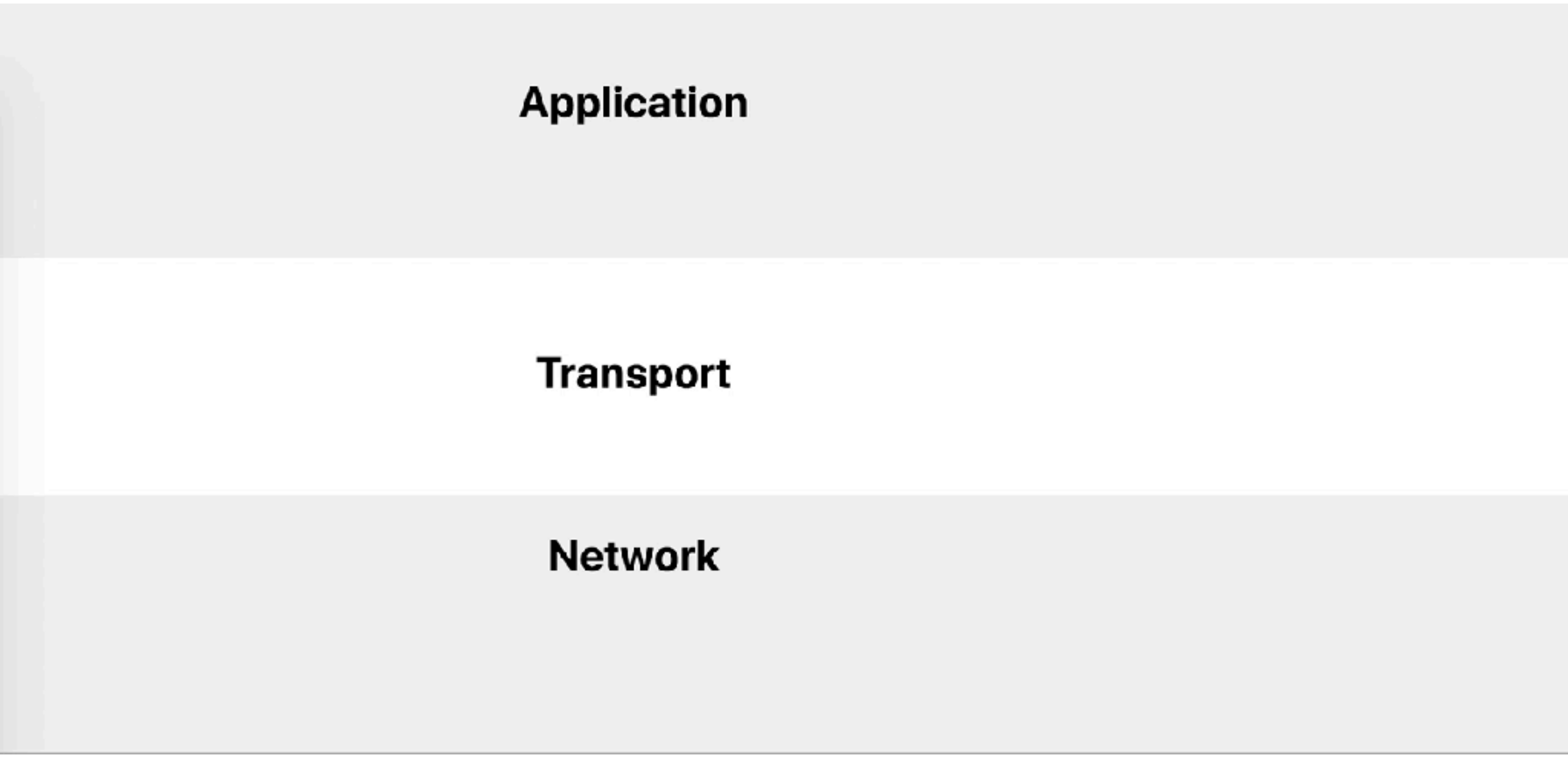
What is QUIC?

- Quick UDP Internet Connections
- Re-envision TCP after ~40 years of use
- Especially geared towards use as an HTTP (2) transport
- Started at Google in 2012, IETF working group established in 2016, planning for final drafts in Nov 2018
- (SPDY announced 2009; first draft of HTTP 2 in 2012; RFC 7540 published May 2015; SPDY removed in 2016.)

Table of contents

- **QUIC: problem statement, concepts and process**
- Rust ecosystem: rustls, *ring*, futures, tokio, h2, bytes
- Quinn: design, implementation status, future

Head-of-Line blocking



QUIC goals

- Minimize latency
- Provide multiplexing
- Changes limited to path endpoints
- Enable multipath and forward error correction
- Provide always-secure transport
- Expect rapid distributed development and testing

QUIC documents

- Invariants: things that cannot change in future versions
- Transport: core protocol logic
- Recovery: loss detection and congestion control
- TLS: how to secure the transport payloads
- QPACK: header compression (static & dynamic)
- HTTP: mapping of HTTP (2) semantics to QUIC

Recovery

- Differences with TCP
  - Monotonically increasing packet numbers
  - No reneging (reduces memory pressure on sender)
  - More ACK ranges (less reliant on timeouts)
  - Explicit correction for delayed ACKs

QUIC TLS

- TLS 1.3 handshake with a custom extension
  - Extension for announcing transport parameters
- After handshake, encrypt payloads with negotiated secret
  - No more wrapping in TLS messages
- Ongoing discussion about more TLS simplification

QUIC HTTP

- ALPN protocol "hq" negotiated during TLS handshake
- Builds on HTTP 2, differs in order to leverage QUIC
  - Different header
  - Different framing

Interoperability matrix

client ↓ | server →	Minq	Mozquic	Quicly	quant	ngtcp2	mvfst	picoQUIC	winquic	f5	ATS	Pandora	AppleQUIC	ngx_quic	lgquic	quic-go	quicker	quicr	Quinn
Minq																		
Mozquic																		
Quicly																		
quant		VHDCRZS		VHDCRZS	VHDCRZ		VHDCRZS	VHDCRZS	VHDC	VHDCRZ			V			VHDCRZ	VHDCRZS	
ngtcp2		VHDCRS		VHDCRZS	VHDCRZ		VHDCRZS	VHDCRZS	VHDC	VHDCRZ			V			VHDCRZ	VHDCS	
mvfst																		
picoQUIC		VHDCRZS		VHDCRZS	VHDCRZ		VHDCRZS	VHDCRZS	VHDC	VHD			HDCS			VHDCRZ	VHDCRS	
winquic	-	VHDCRZS	-	VHDCRS	VHDCRZ	-	VHDCRZS	VHDCRZS	VHDC	VHDCR	-	-	HDC	-	-	VHDCR	VHDCS	
f5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ATS		VHD		VHD	VHD					VHD								
Pandora																		
AppleQUIC																		
ngx_quic	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
lgquic	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-
quic-go	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
quicker	-	VHDCR	-		VHDCRZ	-	VHDCS	VHDCRZS			-	-	VH	-	-	VHDCRZ	VHDCR	
quicr		VHDCRS		VHDCS	VHDCR		VHDCS	VHDCS	V	VHDCR			VHDC			VHDC	VHDCRS	VH
Quinn																		VHD

Version Negotiation — Handshake — Stream Data — Connection Close — Resumption — 0-RTT — Stateless Retry

Table of contents

- QUIC: problem statement, concepts and process
- **Rust ecosystem: rustls, *ring*, futures, tokio, h2, bytes**
- Quinn: design, implementation status, future

rustls

- Easy to work with/hack on
- QUIC transport parameters support:
    - Extension traits to add methods to Session types
        - Add `new_quic()` constructors to pass in local params
        - Add `get_quic_transport_parameters()` to fetch peer's
- Collateral contributions:
    - Added optional support for SSLKEYLOGFILE

ring

- Secure crypto, based on Google's BoringSSL
- rustls and webpki
- Easy access from rustls cipher suite to *ring* algorithms

futures, tokio, h2

- Worked with basic futures in tokio-imap
- Looked for things to steal from the experts
  - How to create a good futures-based API?
  - How does the API layer interact with library internals?

bytes

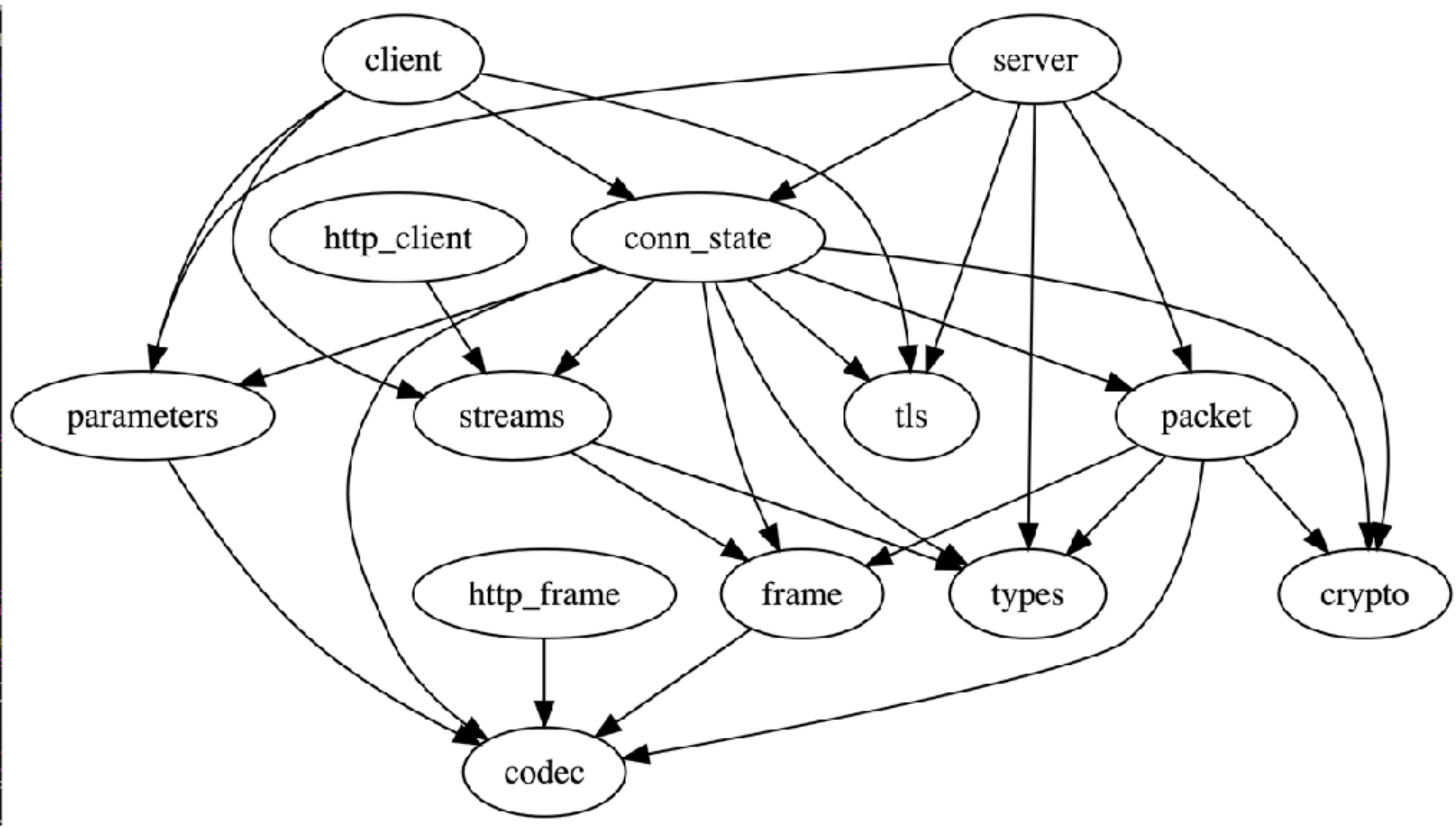
- Used `Buf` and `BufMut` traits to implement a `Codec` trait
- Based on the similar `Codec` trait seen in rustls
- Does all the low-level encoding & decoding

Table of contents

- QUIC: problem statement, concepts and process
- Rust ecosystem: rustls, *ring*, futures, tokio, h2, bytes
- **Quinn: design, implementation status, future**

Goals

- Implement the specifications faithfully
- Have clear abstractions & concepts
- Futures-based architecture — no polling
- Leverage the type system
- Good test coverage at different layers (currently ~85%)


client
server
http_client
conn_state
parameters
streams
tls
packet
http_frame
frame
types
crypto
codec

Implementation status

- Currently implements draft 11 w/ TLS 1.3 draft 28
- Can successfully handshake with other servers
- Some issues with other clients and Quinn server
- Most implementations have not implemented much HTTP
- Hope to keep this implementation moving forward

Big job

- Packetization
- Flow control
- Recovery and retransmission
- Connection migration
- Error handling

quicr

- https://github.com/Ralith/quicr (by Benjamin Saunders)
- Discovered after I started Quinn
- Uses OpenSSL instead of rustls
- Will consider merging implementations

Request for contributions

- https://github.com/djc/quinn
- Your contribution is much appreciated
- There are a few issues in the repository
- Happy to mentor anyone to help get started
- See you at the impl days?
- Or: contribute funding to help move the project forward