November 06 2024

Formal specification and testing of QUIC

Name : Raghav Gade
Roll no. : 20CS02003
Supervisor : Dr. Srinivas Pinisetty
School of Electrical and Computer Sciences

Contents

Introduction

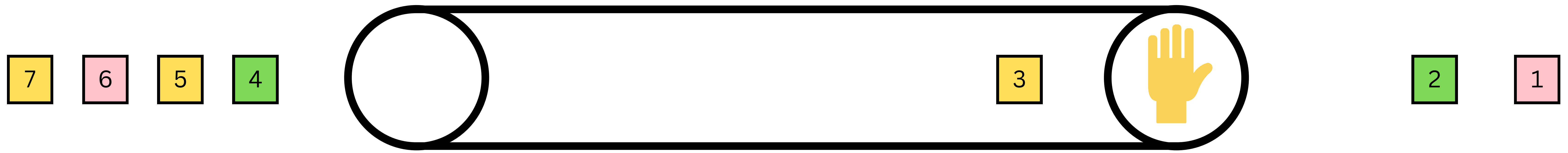
- **QUIC is a Internet secure transport protocol introduced by Google in 2013.**
- **Google's QUIC protocol implements TCP-like properties at the application layer atop a UDP transport.**
- **As of 2023, 8.4% of all websites use QUIC, while majority of traffic is browser clients accessing Google services. (around 98%).**
- **The QUIC standard is being developed in the form of an RFC: an English-language document. IETF officially published QUIC as RFC 9000 in 2021 with many impovements and extensions succeeding it.**
- **It is in the under review for standardization, thus there are many implementations based on one's interpretation of RFC.**

Problem with TCP/TLS

- Connection Overhead.
- TCP Head of line Blocking.
- Server load spikes.
- Connection Migration Limitations.
- Performance Bottlenecks in High Packet Loss Scenarios.
- Inefficiency in Multiplexing

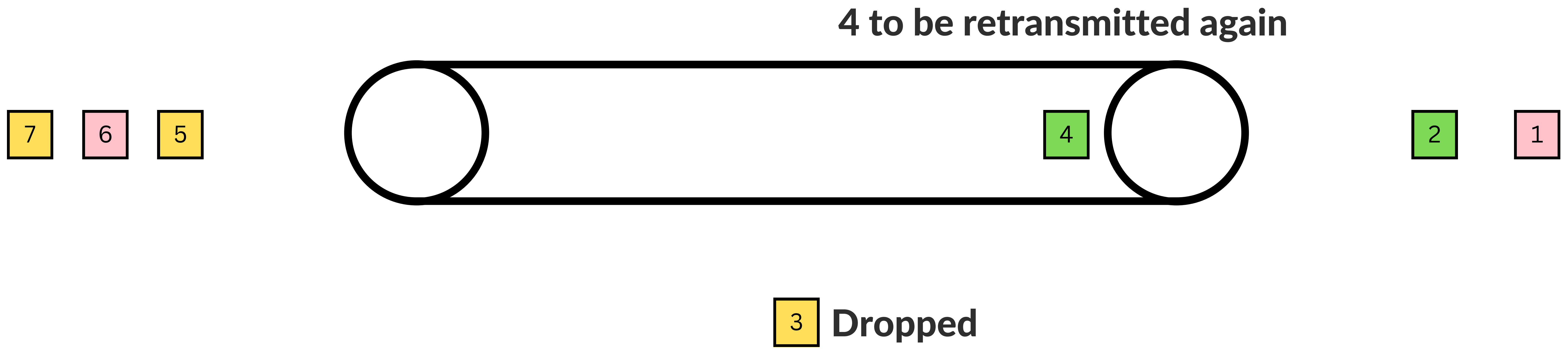
TCP Head of line blocking

🟥 HTML
🟨 CSS
🟩 JS



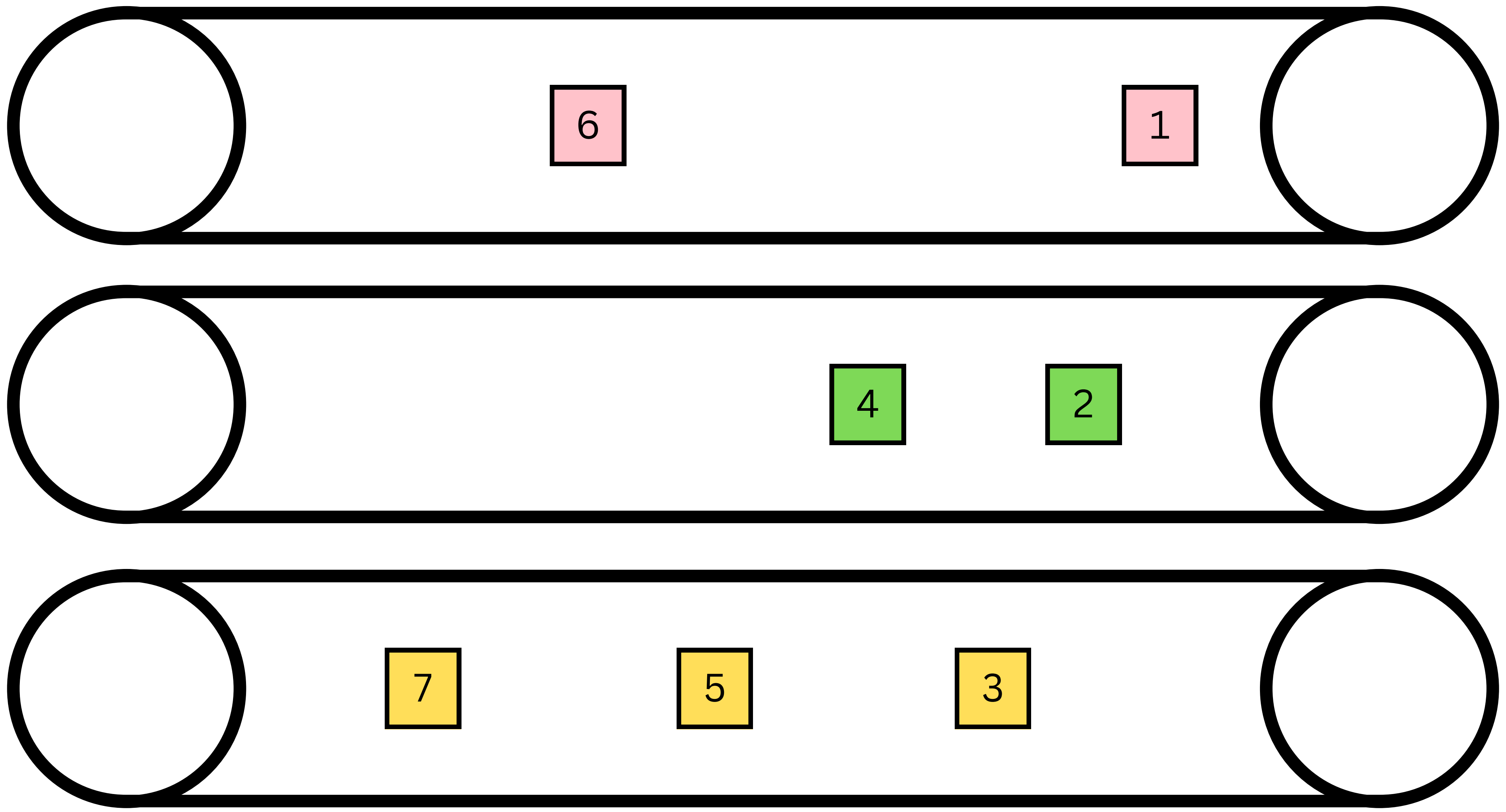
TCP Head of line blocking

🟥 HTML
🟨 CSS
🟩 JS

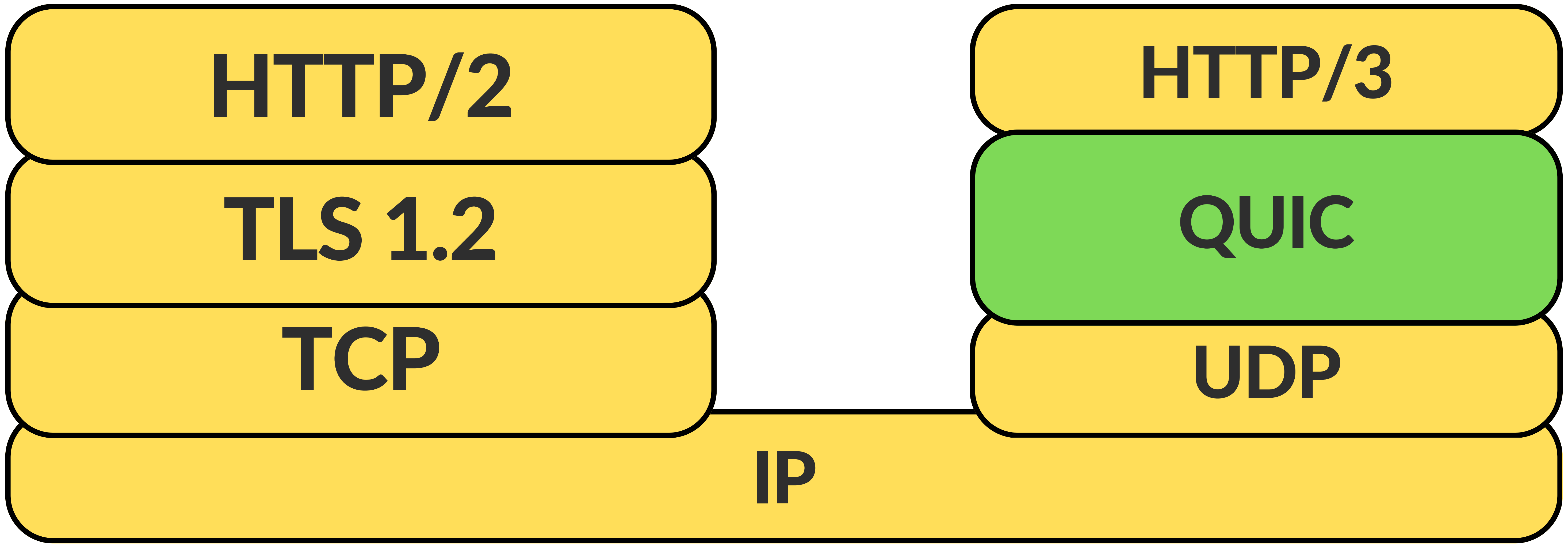


Solution by QUIC

🟥 HTML
🟨 CSS
🟩 JS

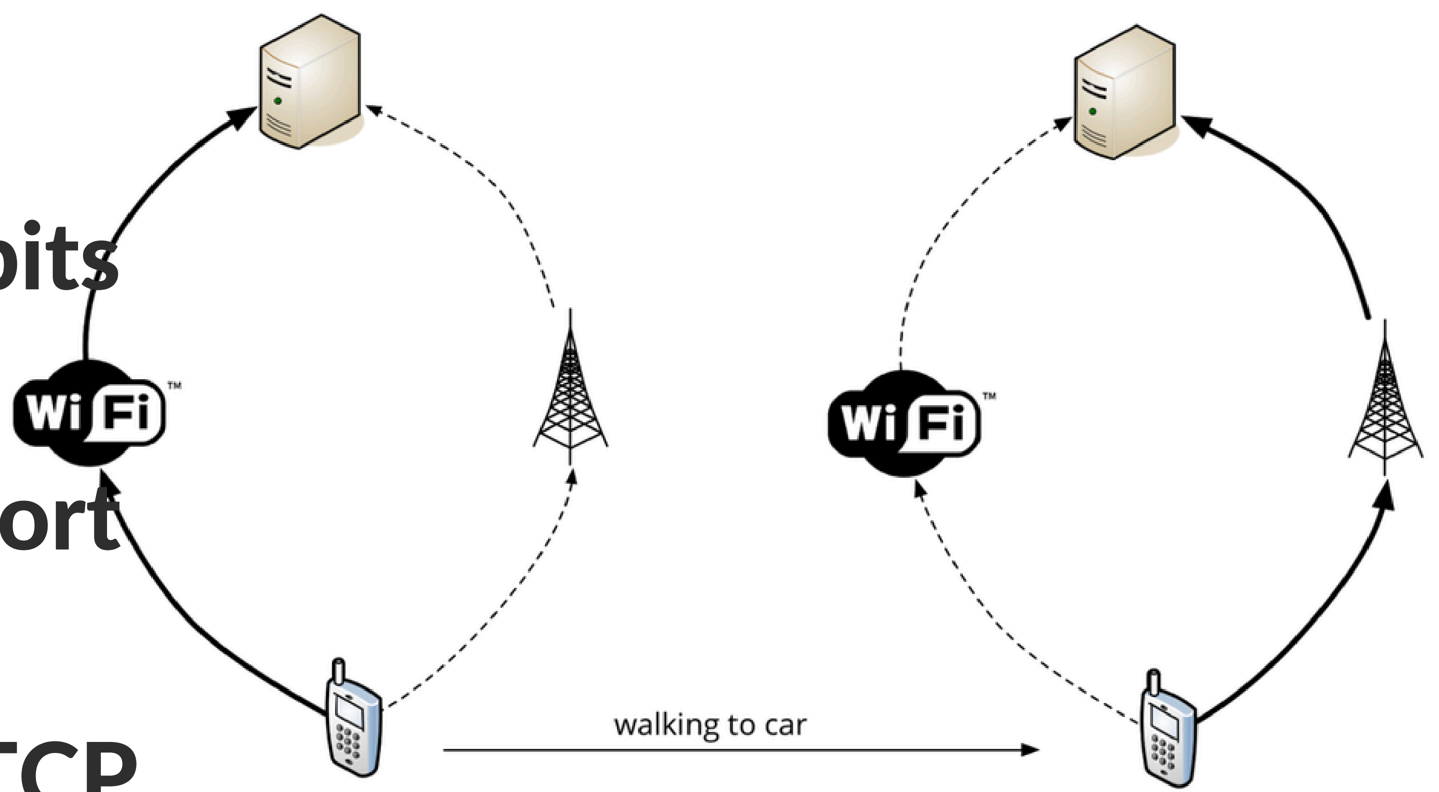


Protocol stack (Comparison)

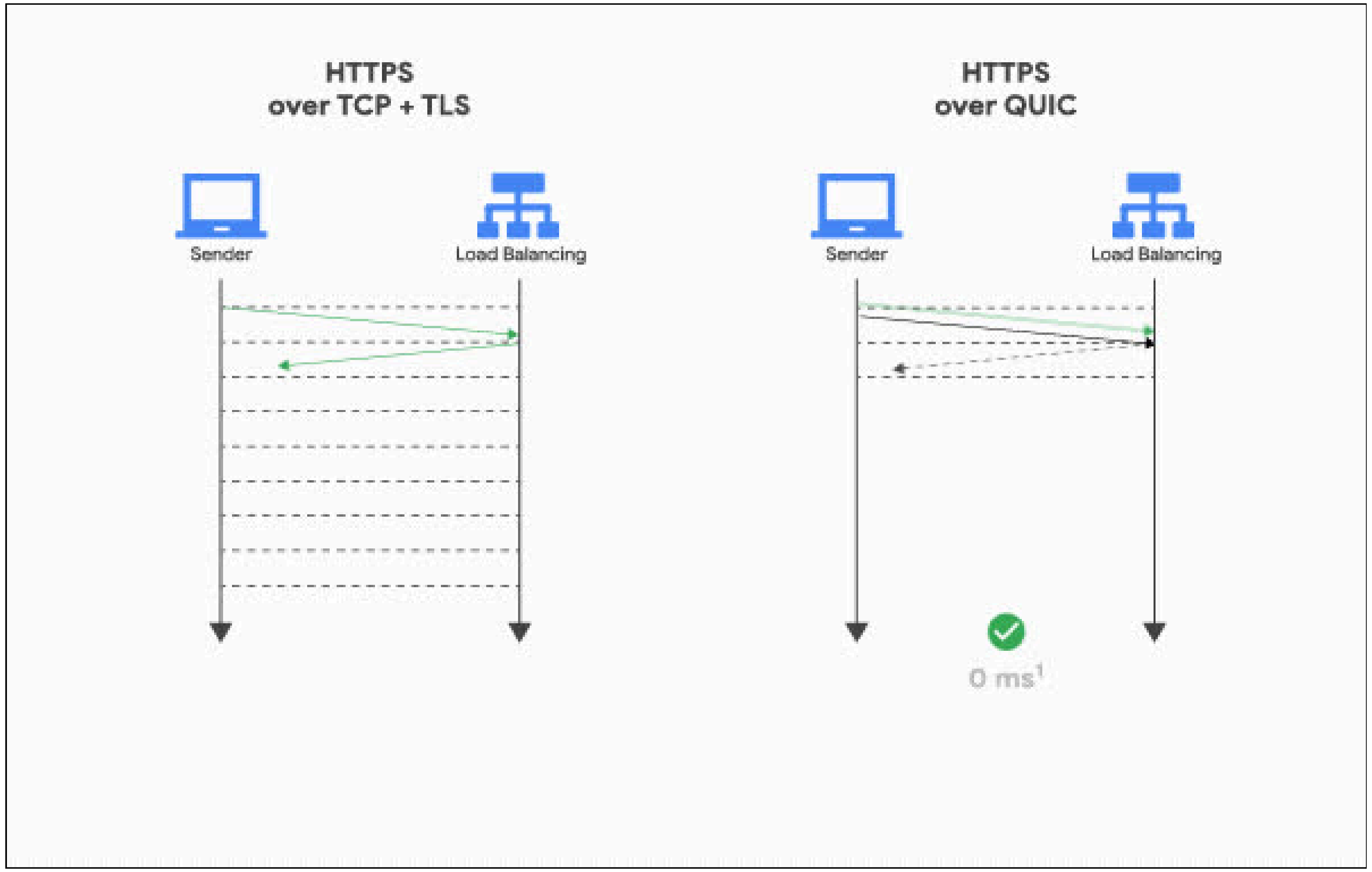


Features of QUIC

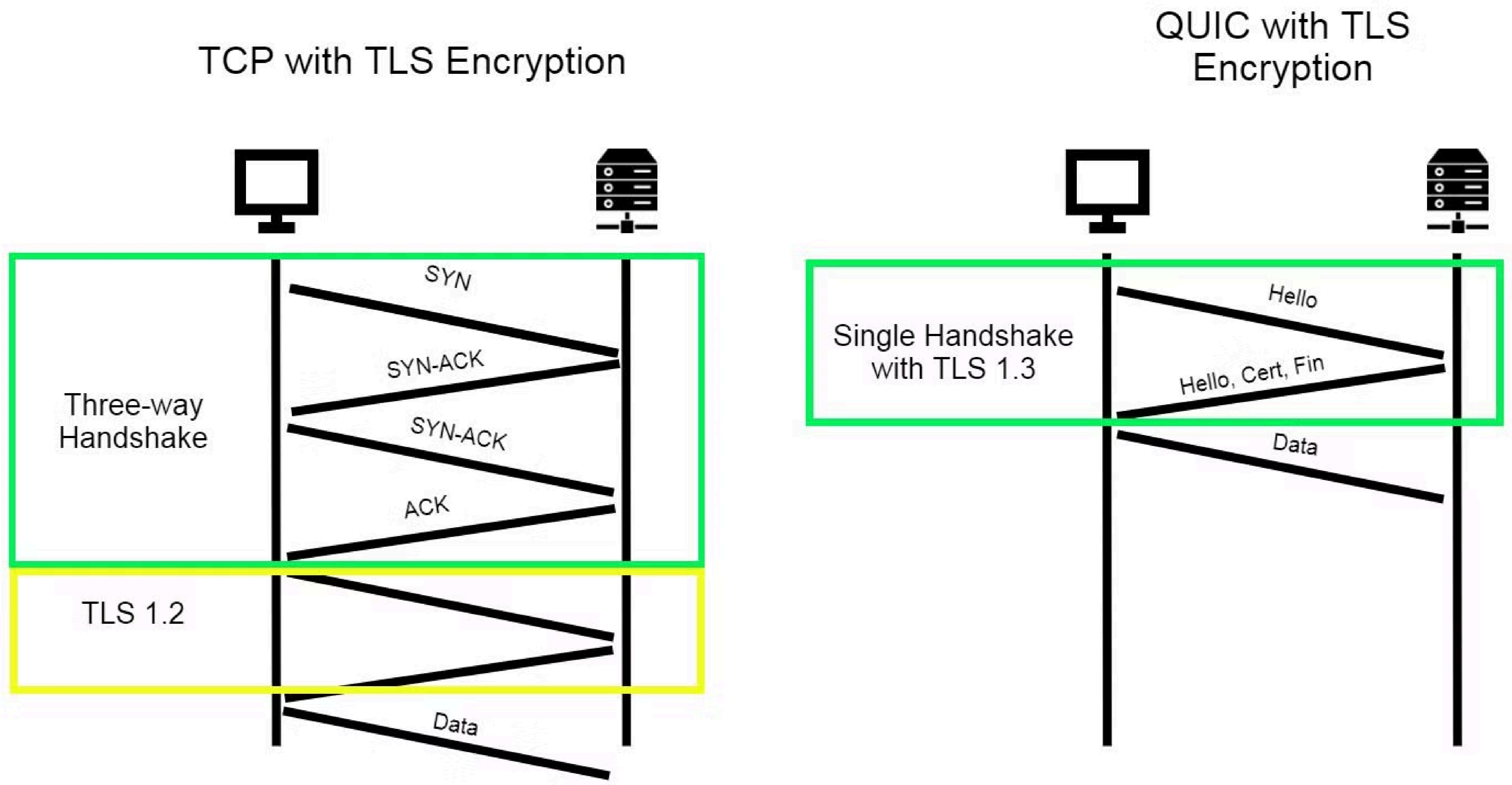
- Connection identified by Connection ID
    - As opposed to common 5-tuple and 64 bits
    - Chosen randomly by the client
    - Enables connection mobility across IP, port
- Runs in user-space
- Faster connection establishment than TLS/TCP
- Four encryption types
    - initial, handshake,
    - 0-RTT (for early data)
    - 1-RTT (for normal data with forward secrecy)
- In addition to CID's, QUIC packets contain unique sequence numbers that are used to detect packet loss.
- Anti-ossification properties



1-RTT connection



1-RTT connection



Implementations of QUIC

Implementation	Language	Description
Chromium	C++	This is the source code of the Chrome web browser and the reference gQUIC implementation. It contains a standalone gQUIC and QUIC client and server programs that can be used for testing.
ms-quic	C	A cross platform QUIC implementation from Microsoft designed to be a general purpose QUIC library
mvfst	C++	Implementation of IETF QUIC protocol in C++ by Facebook.
kwik	Java	Client and server implementations of the QUIC protocol (RFC 9000) in 100% Java. Supports HTTP3 (RFC 9114)
aioquic	Python	This library features an I/O-free API suitable for embedding in both clients and servers.
picoquic	C	A minimal implementation of QUIC aligned with the IETF specifications. One of the best performers in interoperability testing
neqo	Rust	Implementation by Mozilla

Formal specifications of QUIC

- QUIC state machine using Synoptic by Northeastern University to check performance of various implementations in large number of environments and its comparison with TCP.
- Security analysis of the QUIC handshake protocol based on symbolic model checking with tools like ProVerif and Verifpal
- Formal Specification of QUIC using Ivy and compositional based testing by Microsoft Research.

Testing of QUIC

Performance testing

Performance in a variety of network scenarios, including bottleneck bandwidth, RTT, and policed rate. It can also test multiple flows on multiple machines, and multiple congestion control algorithms in a single test

Interoperability testing

How different implementations of QUIC are compatibile with each other. Check how different types of encryption levels behave with each other. For eg, Handshake messages with data, 1-RTT, 0-RTT

Compositional based testing

Specify a closed system of communicating agents, the same specification is used to generate tests for both client and server roles.
Does not model QUIC as an FSM.

Fuzz testing

Execute or simulate an existing implementation, and mutate the behaviors of some nodes, for example, by delaying, duplicating, or modifying messages. Nothing about conformance to the protocol standard, and does not result in a formal specification of the protocol.

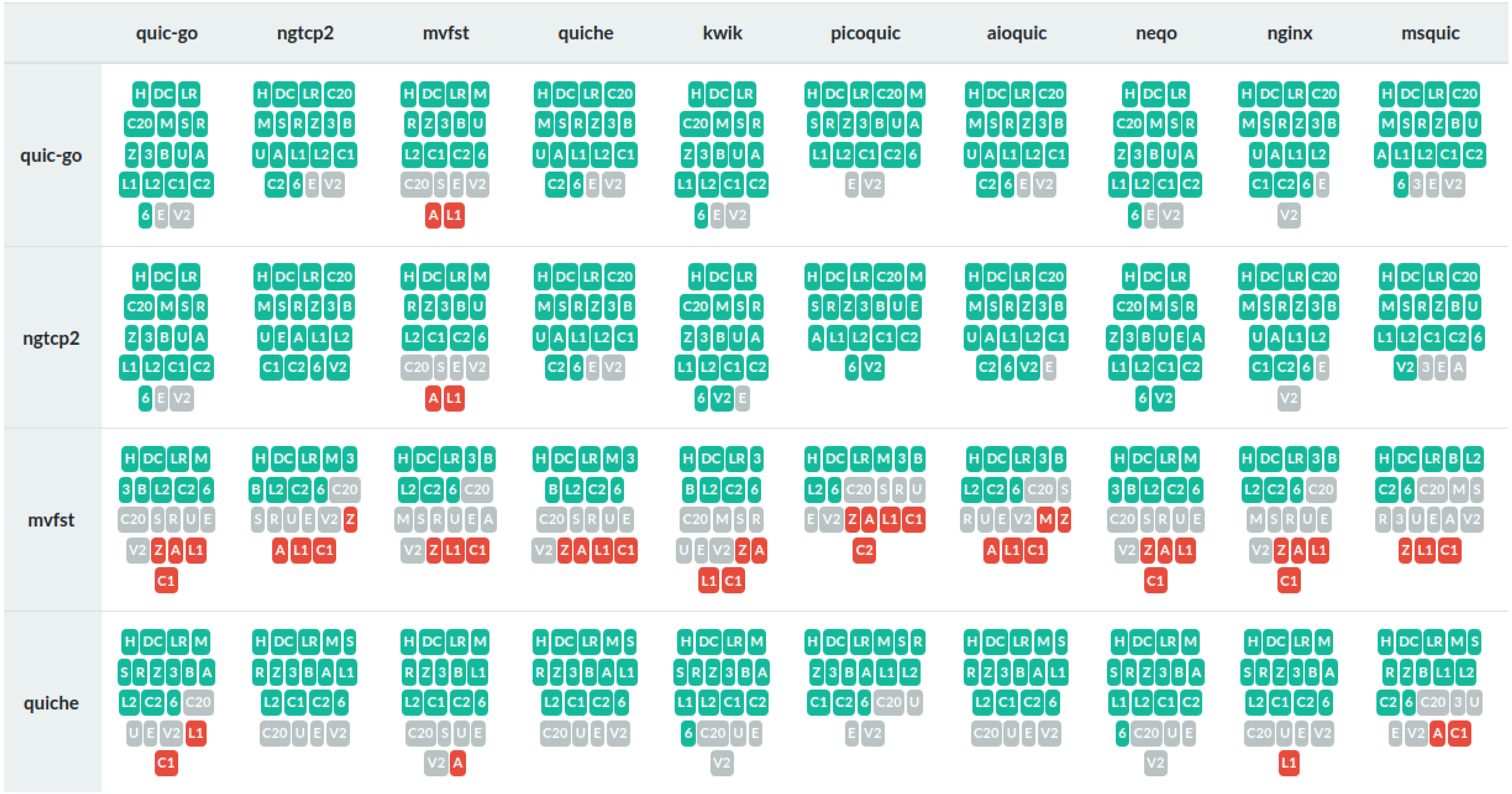
Performance Testing

- The only large-scale performance results for QUIC in production come from Google.
- Google claims that QUIC yields a 3% improvement in mean page load time on Google Search when compared to TCP, and that the slowest 1% of connections load one second faster when using QUIC.
- QUIC achieves better quality of experience for video streaming, but only for high-resolution video.
- In mobile environment, due to QUIC's implementation in userspace as opposed to TCP's implementation kernal space, resource contention might negatively impact performance independent of the protocol's optimizations for transport efficiency.
- QUIC outperforms TCP greatly on variable bandwidth networks.

Interoperability testing

- Primary method used to date to validate QUIC.
- As QUIC standard exists as an English document, it is ambiguous and broadly open to interpretation. As a result, there are multiple independent implementations.
- These implementations represent a kind of commentary on the standard document, providing concrete interpretations where the language may be vague, unclear or contradictory.
- Interoperability is insufficient to guarantee that current implementations will be interoperable with future ones.

Interoperability testing results



	quic-go	ngtcp2	mvfst	quiche	kwik	picoquic	aioquic	neqo	nginx	msquic
quic-go	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR M R Z 3 B U L2 C1 C2 6 C20 S E V2 A L1	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z B U A L1 L2 C1 C2 6 3 E V2
ngtcp2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2	H DC LR M R Z 3 B U L2 C1 C2 6 C20 S E V2 A L1	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 V2 E	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 V2 E	H DC LR C20 M S R Z 3 B U E A L1 L2 C1 C2 6 V2	H DC LR C20 M S R Z 3 B U A L1 L2 C1 C2 6 E V2	H DC LR C20 M S R Z B U L1 L2 C1 C2 6 V2 3 E A
mvfst	H DC LR M 3 B L2 C2 6 C20 S R U E V2 Z A L1 C1	H DC LR M 3 B L2 C2 6 C20 S R U E V2 Z A L1 C1	H DC LR 3 B L2 C2 6 C20 M S R U E A V2 Z L1 C1	H DC LR M 3 B L2 C2 6 C20 S R U E V2 Z A L1 C1	H DC LR 3 B L2 C2 6 C20 M S R U E V2 Z A L1 C1	H DC LR M 3 B L2 6 C20 S R U E V2 Z A L1 C1 C2	H DC LR 3 B L2 C2 6 C20 S R U E V2 M Z A L1 C1	H DC LR M 3 B L2 C2 6 C20 S R U E V2 Z A L1 C1	H DC LR 3 B L2 C2 6 C20 M S R U E V2 Z A L1 C1	H DC LR B L2 C2 6 C20 M S R 3 U E A V2 Z L1 C1
quiche	H DC LR M S R Z 3 B A L2 C2 6 C20 U E V2 L1 C1	H DC LR M S R Z 3 B A L1 L2 C1 C2 6 C20 U E V2	H DC LR M R Z 3 B L1 L2 C1 C2 6 C20 S U E V2 A	H DC LR M S R Z 3 B A L1 L2 C1 C2 6 C20 U E V2	H DC LR M S R Z 3 B A L1 L2 C1 C2 6 C20 U E V2	H DC LR M S R Z 3 B A L1 L2 C1 C2 6 C20 U E V2	H DC LR M S R Z 3 B A L1 L2 C1 C2 6 C20 U E V2	H DC LR M S R Z 3 B A L1 L2 C1 C2 6 C20 U E V2	H DC LR M S R Z 3 B A L2 C1 C2 6 C20 U E V2 L1	H DC LR M S R Z B L1 L2 C2 6 C20 3 U E V2 A C1

H represents Handshake messages, DC represents transfer & Z represents 0-RTT messages, etc

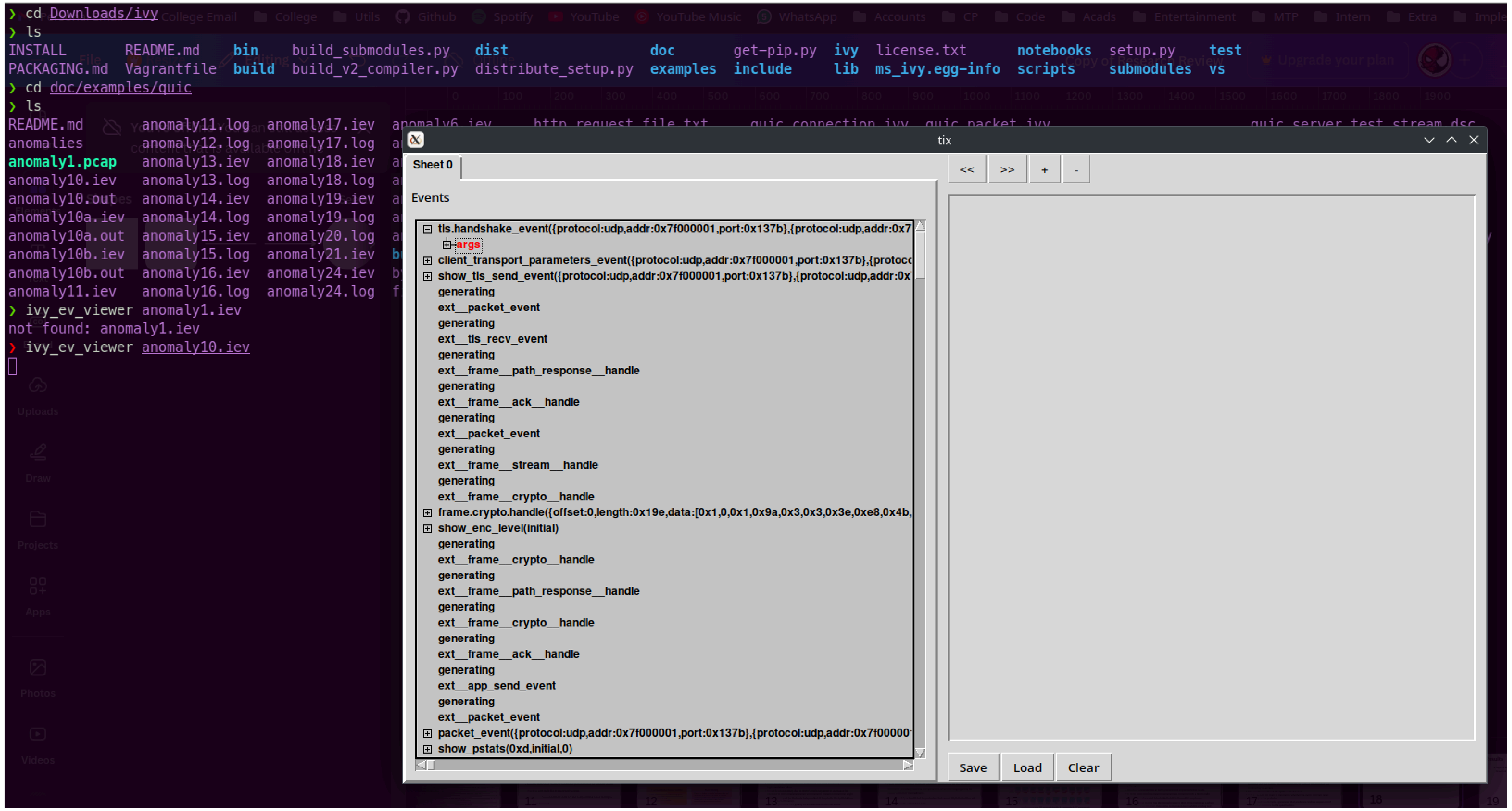
Compositional Based Testing

- Use formal specification of the wire protocol to generate automated randomized testers for implementations of QUIC.
- The testers effectively take one role of the QUIC protocol, interacting with the other role to generate full protocol executions.
- Generates significantly more diverse stimuli and stronger correctness criteria.
- Discover vulnerabilities at both protocol and implementation levels.
- A finite-state machine cannot communicate correctly with a QUIC node because of unbounded non-determinism in the node’s responses.
- Compositional testing has the formal property that, if the system as a whole violates its specification, then some component must exhibit a failing test.

Writing formal specification using Ivy

- QUIC wire protocol is specified using an abstract machine that monitors protocol events.
- This specification is coded is language called Ivy, given by Microsoft Research which is archived since 2021.
- A protocol state is stored in a collection of mathematical functions and relations and the events associated with QUIC such as transmission are modeled by an action.
- This machine tracks all the QUIC-related events.
- It first consults its state to determine whether the event is in fact legal according to the protocol.
- The Ivy tool can compile a generator that produces random sequences of events that conform to the specification. Uses SMT solver.

Images of runtime monitors and results



```
❯ cd Downloads/ivy
❯ ls
INSTALL       README.md   bin   build_submodules.py   dist                 doc       get-pip.py  ivy  license.txt     notebooks  setup.py    test
PACKAGING.md  Vagrantfile build build_v2_compiler.py  distribute_setup.py  examples  include     lib  ms_ivy.egg-info scripts    submodules  vs
❯ cd doc/examples/quic
❯ ls
README.md        anomaly11.log  anomaly17.iev  anomaly6.iev    http_request_file.txt    quic_connection.ivy  quic_packet.ivy        quic_server_test_stream.dsc
anomalies        anomaly12.log  anomaly17.log  a
anomaly1.pcap    anomaly13.iev  anomaly18.iev  a
anomaly10.iev    anomaly13.log  anomaly18.log  a
anomaly10.out    anomaly14.iev  anomaly19.iev  a
anomaly10a.iev   anomaly14.log  anomaly19.log  a
anomaly10a.out   anomaly15.iev  anomaly20.log  a
anomaly10b.iev   anomaly15.log  anomaly21.iev  b
anomaly10b.out   anomaly16.iev  anomaly24.iev  b
anomaly11.iev    anomaly16.log  anomaly24.log  f
❯ ivy_ev_viewer anomaly1.iev
not found: anomaly1.iev
❯ ivy_ev_viewer anomaly10.iev
```

tix

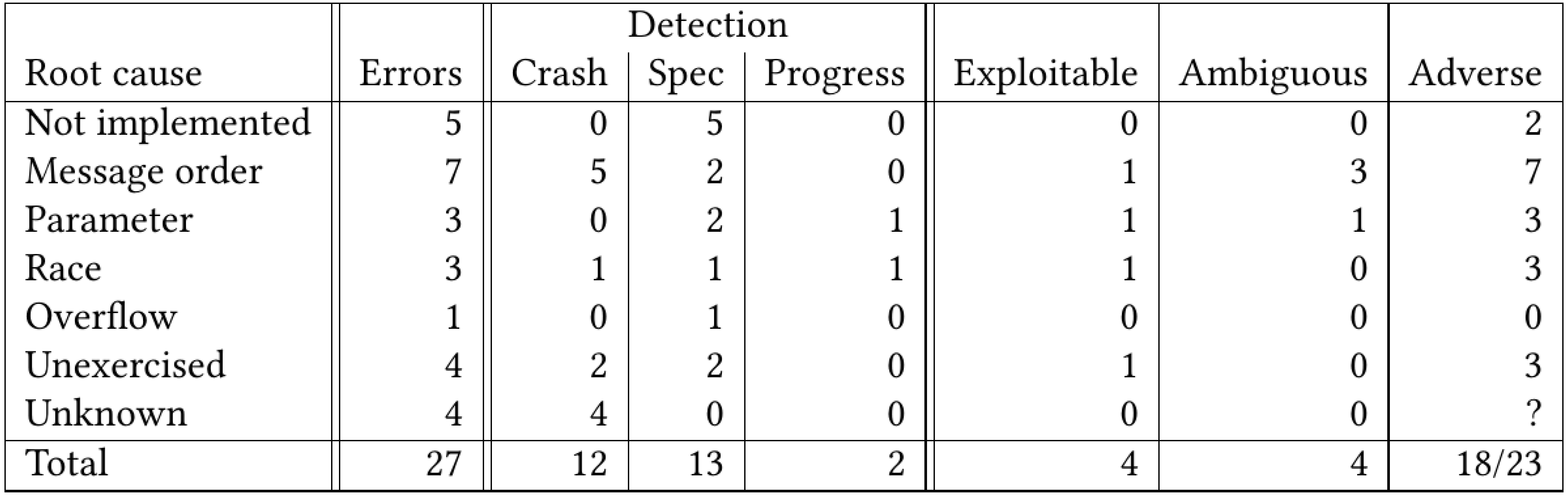
Sheet 0

Events

```
⊟ tls.handshake_event({protocol:udp,addr:0x7f000001,port:0x137b},{protocol:udp,addr:0x7
    ⊞ args
⊞ client_transport_parameters_event({protocol:udp,addr:0x7f000001,port:0x137b},{protoco
⊞ show_tls_send_event({protocol:udp,addr:0x7f000001,port:0x137b},{protocol:udp,addr:0x
  generating
  ext__packet_event
  generating
  ext__tls_recv_event
  generating
  ext__frame__path_response__handle
  generating
  ext__frame__ack__handle
  generating
  ext__packet_event
  generating
  ext__frame__stream__handle
  generating
  ext__frame__crypto__handle
⊞ frame.crypto.handle({offset:0,length:0x19e,data:[0x1,0,0x1,0x9a,0x3,0x3,0x3e,0xe8,0x4b,
⊞ show_enc_level(initial)
  generating
  ext__frame__crypto__handle
  generating
  ext__frame__path_response__handle
  generating
  ext__frame__crypto__handle
  generating
  ext__frame__ack__handle
  generating
  ext__app_send_event
  generating
  ext__packet_event
⊞ packet_event({protocol:udp,addr:0x7f000001,port:0x137b},{protocol:udp,addr:0x7f000000
⊞ show_pstats(0xd,initial,0)
```

Results

- Large fraction of draft standard could not be formalized.
- Any statement in the QUIC document on recovery and congestion control was not covered.
- Table below quantitively describes the errors occured during testing of different implementations of QUIC.



		Detection					
Root cause	Errors	Crash	Spec	Progress	Exploitable	Ambiguous	Adverse
Not implemented	5	0	5	0	0	0	2
Message order	7	5	2	0	1	3	7
Parameter	3	0	2	1	1	1	3
Race	3	1	1	1	1	0	3
Overflow	1	0	1	0	0	0	0
Unexercised	4	2	2	0	1	0	3
Unknown	4	4	0	0	0	0	?
Total	27	12	13	2	4	4	18/23

Conclusion

- **It is well recognized that Internet standards in the form of RFCs are ambiguous and difficult to interpret.**
- **Frustrating attempts to evaluate QUIC is the fact that the protocol is under rapid development, with extensive rewriting of the protocol occurring over the scale of months, making individual studies of the protocol obsolete before publication.**
- **QUIC has been selected as the basis of HTTP/3. Thus, after standardization, it is reasonable to expect that the protocol will carry a significantly larger portion of Internet traffic.**

References

1. Formal specification and testing of QUIC - https://dl.acm.org/doi/pdf/10.1145/3341302.3342087
2. Taking a long look at QUIC - https://github.com/arashmolavi/quic
3. Formal Analysis of QUIC Handshake protocol using Symbolic Model checking - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9328313
4. Ivy by Microsoft Research and QUIC - https://github.com/microsoft/ivy/tree/master/doc/examples/quic
5. Site to check QUIC interoperability results - https://interop.seemann.io/?run=2024-10-27T02:00
6. aioquic - An implementation of QUIC in python - https://github.com/aiortc/aioquic/tree/main
7. Usage of QUIC - https://w3techs.com/technologies/details/ce-quic
8. SIGCOMM 2020 Conference - https://www.youtube.com/watch?v=31J8PoLW9iM
9. HTTP/3 and QUIC blog - https://www.debugbear.com/blog/http3-quic-protocol-guide
10. Performance of QUIC article - https://www.researchgate.net/publication/353089470_The_Performance_and_Future_of_QUIC_Protocol_in_the_Modern_Internet
11. Talk by Author of Kwik – Java Implementation of QUIC - https://youtu.be/sULCOKfc87Y?si=ZyvWpSd61z_iiKsT

Thank You