Level 4 Project Report Template

Ivan Nikitin
October 28, 2021

Abstract

Every abstract follows a similar pattern. Motivate; set aims; describe work; explain results.

"XYZ is bad. This project investigated ABC to determine if it was better. ABC used XXX and YYY to implement ZZZ. This is particularly interesting as XXX and YYY have never been used together. It was found that ABC was 20% better than XYZ, though it caused rabies in half of subjects."

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Ivan Nikitin   Date: 25 March 2022

Contents

1 | Introduction

2 │ Background

2.1   The Internet of Things

## 2.2   The evolution of the transport layer

In the following section we provide a background on transport layer protocols applicable to the body of work conducted. Together, the protocols described comprise most of the traffic on the Internet. Hence, smaller use-case protocols that do exist were not seen as applicable.

First described by Cerf and Kahn (1974), the Transmission Control Protocol (TCP) has been the main protocol of the Internet suite since its initial implementation. TCP provides a *reliable* and *ordered* delivery of bytes. That is, TCP ensures that data is not lost, altered or duplicated and is delivered in the same order that it was sent. This is achieved by assigning a sequence number to each transmitted packet and requiring an *acknowledgment* (commonly referred to as ACK) from the receiving side. If an ACK is not received, the data is re-transmitted. On the receiving side, the sequence numbers can also be used to order packets as intended by the sender.

As TCP is a connection based protocol, connection establishment must take place before any data can be transmitted. The receiving side (the server) must bind to and listen on a network port and the sender (the client) must initiate the connection using the process of a *three-way handshake* as shown in Figure 2.1. In the first step of the handshake, the client sends a segment with a *synchronise sequence number* (SYN) that indicates the start of the communication and the sequence number that the segment starts with. The server responds with an acknowledgment - ACK, and the sequence number it will start its segment with - SYN. Hence, this step is often referred to as the SYN-ACK. In the third and final step, the client must acknowledge the response. At this point, the connection on which data can be transferred is established.

In order to achieve *secure communication*, TLS (Rescorla 2018) is often used in the TCP stack. In order to do this a separate TLS handshake has to occur in order to specify the version of TLS to use, decide on the cipher suites, authenticate the server via its public key and certificate authority's signature, and generate a session key that can be used for symmetric encryption during communication. In the TLS handshake the first step is for the client to send a *ClientHello* message that specifies the highest version of TLS that the client supports, a list of suggested cipher suites, compression methods and a random number. The server responds with a *ServerHello* message that contains the selected TLS version, cipher suite, compression method and its own random number. The server then sends its certificate and *ServerKeyExchange* message along with the *ServerHelloDone* message indicating that it has completed its part of the negotiation process. The client will respond with the *ClientKeyExchange* message (CKE) which, depending on the chosen cipher suite, may contain a public key. This is followed by the client sending a *ChangeCipherSpec* message indicating that all communication from this point is authenticated and encrypted along with a finished message. The server responds with the same message and the TLS connection is established.

Due to the establishment of communication and properties guaranteed by TCP, such as reliability through retransmission, an inherent trade-off is created, and latency is lengthened. Hence, in use-cases where reliability and connection state is not required, the User Datagram Protocol
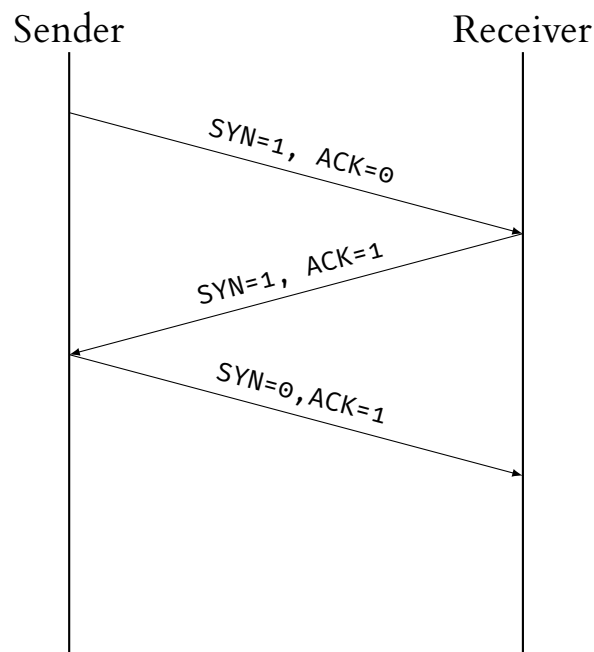
Figure 2.1: The TCP handshake needed for connection establishment. The values of the SYN and ACK fields being set indicate the kind of segment sent. For example, for the SYN-ACK stage both the SYN and ACK fields are set.

(UDP) (Postel 1980) is preferred. UDP uses a connectionless communication model that aims to have a minimal number of semantics. The only mechanisms provided by UDP are port numbers and checksums in order to ensure data integrity. This is preferable for real-time systems as using TCP for these would cause an overhead to latency and retransmission of packets that are no longer needed by the application.

QUIC is a relatively new general-purpose transport layer protocol originally designed by Roskind (2012) at Google as part of the Chromium web engine. In 2015 the first draft of the QUIC protocol was submitted to the IETF and was later standardised (Iyengar and Thomson 2021).

The aim of QUIC is to improve upon, and eventually make obsolete, TCP by using the concept of multiplexing, which is a method of combining several signals or channels of communication over one shared medium. QUIC establishes multiplexed connections between the communicating endpoints using UDP.

QUIC facilitates data exchange on the UDP connection through the concept of *streams*. Streams are an ordered byte-stream abstraction used by the application to send data of any length. Streams are created by either the sending or receiving side and can be both unidirectional and bidirectional. Each side can send data concurrently on the stream and can open any number of streams (specifically, a field for the maximum number of streams is set during the connection). Hence, subject to the constraints imposed by flow control, QUIC allows an arbitrary number of streams to send arbitrary amounts of data on the UDP connection.

By doing so, QUIC also achieves secondary goal of lifting congestion control algorithms from the kernel space to the user space. Hence, congestion control algorithms can evolve without having to be tied down to kernel level semantics and constraints.

Compared to TCP/TLS, QUIC combines the transport and cryptographic handshakes in order to minimise the time needed for connection establishment. A comparison of the handshakes can

(a) *TCP/TLS* **(b)** *QUIC*
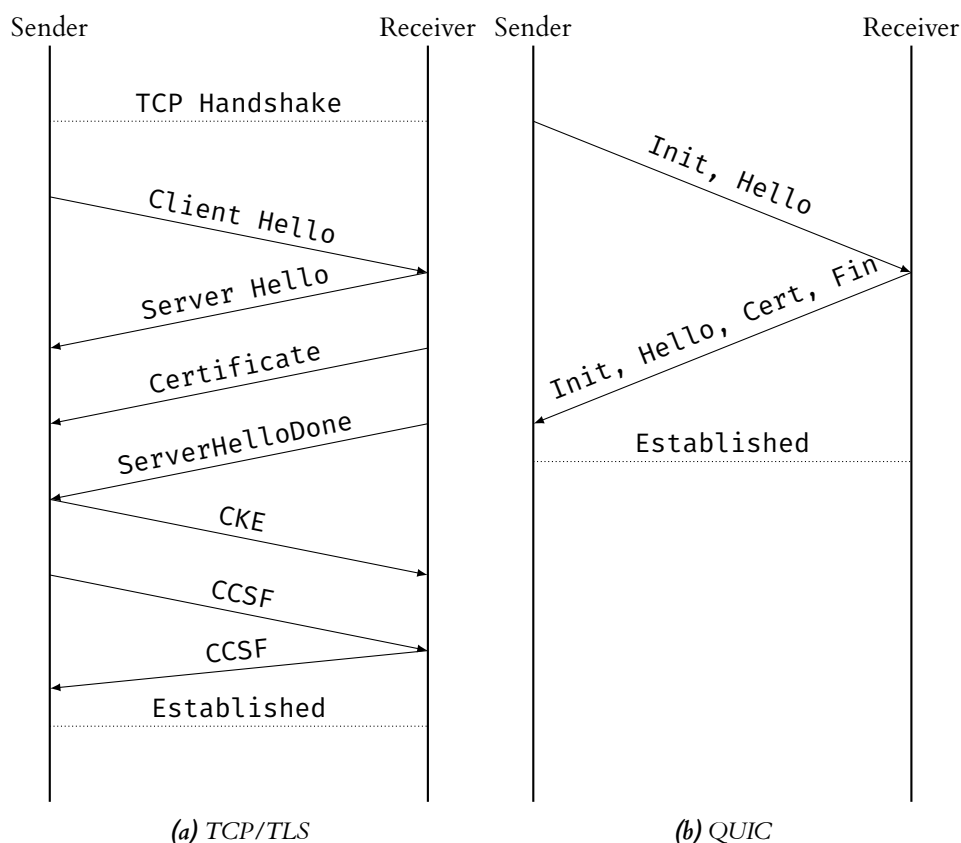
Figure 2.2: *Handshakes required to establish secure data transmission in the TCP/TLS stack (a) and the QUIC stack (b). In the case of TCP/TLS we can see that the handshake is substantially more complex and that the TLS handshake requires the full TCP handshake before it can proceed. In both cases these handshakes can be made quicker. In the case of TLS, version 1.3 allows for one less round trip before data can be sent, and in the case of QUIC 0-RTT connection re-establishment may be used in come cases, allowing to send data in the first packet.*

be seen in Figure 2.2. TLS is still used to secure QUIC as described by Thomson and Turner (2021) unless a different cryptographic protocol is specified. The initial QUIC handshake keeps the same handshake messages as TLS, however it uses its own framing format, replacing the TLS record layer. This ensures that the connection is always authenticated and encrypted, unlike in TLS where the initial handshake is still vulnerable. The combination also means that QUIC typically starts sending data after just one round-trip.

2.3 The Rust programming language

3 | Methodology

4 | Evaluation

4.1 Performance

4.2 Binary sizes

5 | Conclusion

A | Data

5 | Bibliography

V. Cerf and R. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, May 1974. ISSN 1558-0857. doi: 10.1109/TCOM.1974. 1092259.

J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Request for Comments RFC 9000, Internet Engineering Task Force, May 2021. URL `https://datatracker.ietf.org/doc/rfc9000`. Num Pages: 151.

J. Postel. User Datagram Protocol. Request for Comments RFC 768, Internet Engineering Task Force, Aug. 1980. URL `https://datatracker.ietf.org/doc/rfc768`.

E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. Request for Comments RFC 8446, Internet Engineering Task Force, Aug. 2018. URL `https://datatracker.ietf.org/doc/rfc8446`. Num Pages: 160.

J. Roskind. QUIC: Design Document and Specification Rationale - Google Docs, 2012. URL `https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit`.

M. Thomson and S. Turner. Using TLS to Secure QUIC. Request for Comments RFC 9001, Internet Engineering Task Force, May 2021. URL `https://datatracker.ietf.org/doc/rfc9001`. Num Pages: 52.