Expectations

What counts as “profound” feedback?

***Abstract*—Although the QUIC protocol is specified in-depth in the `RFC 9000`, the technical implementations differ significantly. The design choice of a suitable QUIC library is crucial for the available functionality and performance of a launching project. This paper analyses the utilisation and internal architecture of Cloudflare's QUIC implementation quiche based on the C4 model, its relation to the host application and quiche-specific features that are not explicitly defined in the `RFC 9000`. Subsequently, our findings are added to the QUIC Explorer, a project intended for developers to facilitate the decision of a suitable QUIC library.**

***Index Terms*—QUIC, QUIC explorer, quiche, C4 model, architecture, Congestion Control, stream prioritisation, BoringSSL**

1. Introduction

Since the Internet became increasingly important in our daily lives, TCP was one of the most used protocols for data transmission, but it had to face some challenges like higher throughput, lower latency and better security. In 2016, the first Internet-Draft of Quick UDP Internet Connections (QUIC) was published by Google's developer team on the IETF forum [1]. The main goal of QUIC was to merge the advantages of TCP and UDP in one protocol with a focus on security, reliability and performance. In 2021, after several iterations and improvements, the IETF published the `RFC 9000` that defines the QUIC protocol [2]. Already in October 2019, Cloudflare released version 0.1.0 of their QUIC implementation called quiche on GitHub [3]. Since then, the library has been continuously developed and improved.

According to Alessandro Ghedini, one of the leading developers of quiche, it was a major design goal to provide most functionality of QUIC to the host application but implemented in a minimal and intuitive Application Programming Interface (API). However, the field of application of quiche should not be restricted by any assumptions taken during the development process [3].

In this paper, we will analyse quiche's architecture and internal relations from an abstract point of view. Mainly, two abstraction levels (container & component) of the C4 model by Simon Brown will be considered [4]. We will deliberately not analyse and discuss the code level, as it goes beyond our scope and could be outdated shortly due to the quick development. In Section 2, the communication of quiche with its environment and inner elements will be analysed. Section 3 some features of quiche that are not strictly defined in the `RFC 9000` will be discussed. Furthermore, for easy access to the analysis results, they will be added to the `QUIC Explorer` [5]. This information pool is designed for developers to gain a quick overview of the capabilities of different QUIC implementations. It is intended to facilitate the team's design choices and to provide a comprehensive overview of the QUIC ecosystem.

2. C4 Model-Based Analysis

To strike a balance between an abstract view of the integration of quiche into a software project and a detailed analysis of the internal structure, we chose the C4 model to visualize and describe the architectural elements of the library. Unlike the prevalent Unified Modeling Language (UML), the C4 model is less formal and more lightweight, which makes it easier to understand, especially for people without a deep knowledge of software architecture. Nevertheless, it can represent such a system, which is as complex as the architecture of quiche in a simple but precise way [6]. In total, the C4 model consists of four abstraction levels: system context, container, component and code. Each level focuses on a certain degree of abstraction to address different stakeholders [7, p. 920].

Since we want to focus on a structural analysis of quiche, we start in Section 2.1 at the container level to point out the interaction of the library with the software environment. Afterwards, we will dive deeper into the component level (Section 2.2) to analyze the library's architecture. To clarify the terms *container* and *component* in the context of the C4 model: A component is a cohesive set of functionalities that are not separately deployable. It functions as a facade with a well-defined interface for its underlying elements (e.g. structs, classes, instances) one level deeper. In contrast, a container is composed of several logically separate components that operate as a single deployable unit [4].

2.1. Container Level

quiche is designed as a multifunctional user-space library that enables rapid, iterative development and the possibility to be integrated into various services of different purposes [8]. On the one hand, Cloudflare developed quiche for their application in their Content Delivery Network (CDN) backbone, where it needs to be highly performant and reliable [3], [9]. On the other hand, quiche was in its initial phase released as an open-source project to be co-developed by the community and integrated into a wide range of applications outside of Cloudflare. As

***Abstract*—Whenever a new software project starts many essential and extensive decisions must be taken by the developers. Theses design choices are crutial for the course of the project and should well-considered. This paper outlines the general architecture of Cloudflare's QUIC implementation Quiche based on the C4 model by Simon Brown to faciliate such tremendous decisions. We analyze the library's internal structure, its relation to the host application and Quiche-specific features that are not explicitly defined in the `RFC 9000`.**

***Index Terms*—QUIC, QUIC explorer, Quiche, C4 model, architecture, Congestion Control, stream prioritization, BoringSSL**

1. Introduction

Since the Internet became more and more important in our daily live, TCP was one of the most used protocols for data transmission, but it had to face some challenges like higher throughput, lower latency and better security. In 2016 the first Internet-Draft (I-D) of Quick UDP Internet Connections (QUIC) was published by some Google developer on the IETF forum [1]. The main goal of QUIC was to combine the advantages of TCP and UDP in one protocol with a focus on security, reliability and performance. In 2021, after several iterations and improvements, the IETF published the `RFC 9000` that defines the QUIC protocol [2].

Already in October 2019 Cloudflare released version 0.1.0 of their QUIC implementation called Quiche [3] on GitHub. Since then, the library has been continuously developed and improved. Allessandro Ghedini, one of the main developers of Quiche, summerized it as followed:

> “The main design principle that guided quiche's initial development was exposing most of the QUIC complexity to applications through a minimal and intuitive API, but without making too many assumptions about the application itself, in order to allow us to reuse the same library in different contexts” [3].

In this paper we will analyze the software architecture and internal relations of Quiche in an abstract manner. Mainly two abstraction levels (container & component) of the C4 model by Simon Brown [4] will be considered, not to become outdated during the next few commits. In Section 2 the communication of Quiche with its environment and innere elements will be analyzed. In Section 3 some features of Quiche that are not strictly defined in the `RFC 9000` will be discussed. Furthermore, for an easy access of the analysis results, they will be added to the `QUIC Explorer` [5]. This information pool is designed for developers to gain a quick oberview of the capabilities of different QUIC implementations. It is intended to faciliate the team's design choices and to provide a comprehensive overview of the QUIC ecosystem.

2. C4 Model-Based Analysis

To strike a balance between an abstract view on the integration of Quiche into a software project and a detailed analysis of the internal structure, we chose the C4 model by Simon Brown to visualize and describe the architectural elements of the code. To answer the question why UML is not used: It is useful to document the current state of the project code, but if the code changes the UML diagrams have to be updated as well. But we aim for a precise and understandable abstraction of the entire codebase that does not go beyond our scope [6, minute 19]. In total the C4 model consists of four abstraction levels: system context, container, component and code. Each level focus on a certain degree of abstraction in order to address different stakeholders [7, p. 920]. Since we want to focus on a structural analysis of quiche, we start in section 2.1 at the container level to point out the interaction of the library with the software environment. Afterwards, we will dive deeper into the component level (section 2.2) to analyze the software architecture of the library. To clarify the terms `container` and `component` in the context of the C4 model: “A component is a grouping of related functionality encapsulated behind a well-defined interface, [which] are not separately deployable units” [4, section Component]. In contrast, a container is composed of several logically separate components that operate as a single deployable unit.

2.1. Container Level

Quiche is designed as a multifunctional user-space library. This “facilitated its deployment as part of various applications and enabled iterative changes to occur at application update timescales” [8]. On the one hand, Cloudflare developed Quiche for their own application in the thier Content Delivery Network (CDN) backbone, where it needs to be highly performant and reliable [3], [9]. On the other hand, Quiche was in its initial phase released as an open-source project to be co-developed