Reproducing "Evaluating QUIC Performance Over Web, Cloud Storage, and Video Workloads"

Dharmik Patel
Purdue University
West Lafayette
pate2146@purdue.edu

Jevin Modi
Purdue University
West Lafayette
modij@purdue.edu

Arnav Gupta
Purdue University
West Lafayette
gupta565@purdue.edu

Ajinkya Nawarkar
Purdue University
West Lafayette
anawarka@purdue.edu

Soham Sharma
Purdue University
West Lafayette
sharm442@purdue.edu

Nishtha Aggarwal
Purdue University
West Lafayette
aggarwn@purdue.edu

ABSTRACT

The QUIC protocol, designed by Google, promises improved performance and security compared to traditional TLS/TCP protocols. This project uses experimental methodologies and comparative analysis to evaluate QUIC's performance across diverse workloads, including web browsing and video streaming. By reproducing the findings of the paper "Evaluating QUIC Performance Over Web, Cloud Storage, and Video Workloads," we investigate whether QUIC achieves its goals of reduced latency and enhanced throughput.

The experiments utilize a controlled environment with scripts and tools to measure key performance metrics such as latency, data transfer rates, and throughput. Comparative studies between QUIC and TLS/TCP protocols highlight performance gains and potential drawbacks in real-world scenarios. Results are contextualized to reflect workload-specific implications, offering insights into the adoption of QUIC in various networked applications.

KEYWORDS

QUIC, TLS, TCP, video workloads, web workloads, performance evaluation

ACM Reference Format:
Dharmik Patel, Jevin Modi, Arnav Gupta, Ajinkya Nawarkar, Soham Sharma, and Nishtha Aggarwal. 2024. Reproducing "Evaluating QUIC Performance Over Web, Cloud Storage, and Video Workloads". In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Problem Domain and Goals

The proliferation of latency-sensitive applications, such as web browsing and video streaming, has exposed the limitations of traditional TCP-based transport protocols. Efforts to enhance TCP's performance through extensions like TCP Fast Open and Multipath TCP have faced deployment challenges due to middlebox ossification. QUIC, initially introduced by Google and now standardized by the IETF, addresses these challenges by adopting UDP as its substrate and integrating encryption and transport features in user space. This innovative protocol promises reduced connection establishment times, improved resilience to packet loss, and support for multiplexed streams without head-of-line blocking.

The original paper aimed to evaluate QUIC's performance in web, cloud storage, and video workloads through tests performed across diverse network settings, including high-bandwidth, low-RTT links in Germany and low-bandwidth, high-RTT links in India. The study provided a comprehensive performance comparison with TLS over TCP, reporting significant gains in connection times, throughput for small files, and video streaming quality with QUIC. These findings highlighted the protocol's potential to reshape internet communication and content delivery.

1.2 Differentiation of our approach vs the papers

Our approach shares the goal of evaluating QUIC's performance across web and video workloads but introduces several key methodological differences for enhanced control and customization:

(1) **Environment and Tools:**
   - **Paper:** Relied on diverse physical vantage points, including educational networks and residential setups, with bespoke tools for TLS and QUIC measurements.
   - **Our Work:** Employs *ContainerNet* to create isolated, reproducible network environments simulating two distinct setups: high-bandwidth, low-RTT and low-bandwidth, high-RTT. These virtualized testbeds provide consistent conditions for evaluating QUIC and TLS/TCP.

(2) **Containerized Architecture:**
   - **Paper:** Used physical machines, such as VMs and Raspberry Pis, for data collection.
   - **Our Work:** Implements a fully containerized setup via ContainerNet deployed over Amazon AWS with separate Docker containers for server and client roles, ensuring modularity and ease of deployment across different environments.

(3) **Metrics Collected:**

Conference'17, July 2017, Washington, DC, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- **Paper:** Expanded the granularity of measurements to include metrics like total download time, CPU utilization, and detailed throughput variations. These additional metrics enable a nuanced analysis of resource trade-offs and network behavior.
- **Our Work:** Focuses on foundational metrics like *connection time*, *time to first byte* (TTFB), and *download time*. These metrics are targeted at capturing QUIC's core performance benefits in a controlled environment and allow for direct comparison with the findings of the paper.

(4) **Implementation Tools:**
- **Paper:** Developed custom tools leveraging libraries like `libcurl` and `lsquic`.
- **Our Work:** Utilizes off-the-shelf tools combined with Alpine-based CURL implementations of HTTP/3 and Dockerized web/video servers, simplifying adoption and reproducibility.

(5) **Content Delivery:**
- **Paper:** Evaluated real-world websites and YouTube videos, including geographic diversity in measurements.
- **Our Work:** Simulates and hosts diverse workloads of varying sizes and types for both web and video applications, enabling controlled experimentation. By employing a reverse proxy setup via a Caddy server implementation, our approach ensures precise control over network conditions and workload characteristics, facilitating a systematic analysis of QUIC's performance across different scenarios.

2 APPROACH

2.1 Network Topology

Since the paper uses real switches, routers, Networks and QUIC servers for their tests, we have tried creating the similar environment using the virtual network. For this purpose we have used Mininet which is an excellent Network Virtualization tools. The networks topology is defined in the subsequent sections and highlighted via diagram in Figure 1.

We have used Containernet which is an extension of Mininet. It allows us to leaverage docker containers running on the host machine as host in the virtual network environment.

Below are some of the components in our virtual network.

2.1.1 *Client Network*.

- Since the original paper uses two different networks(India and Germany) for tests, the bandwidth and latency parameters will change accordingly in this network.
- For India Network, since it is home network, we have used bandwidth as 20 Mbps and latency of 80ms.
- For Germany Network, since it is a research network, we have used bandwidth as 500 Mbps and latency of 6ms.
- There are two client hosts in our client network
- h1:
  - Docker container
  - This host is used as a client to send request to server.
  - It runs script(`curl_request.sh`) that listens on the client interface for packets using tcpdump, sends 100 different requests each to web and video servers and stores the packets in the form of pcap files.
- h2:
  - Docker container
  - This host is used to create Network and application traffic into our network to simulate the real world networks.
  - Running load-testing tools (e.g., `artillery`) or other benchmarking scripts.

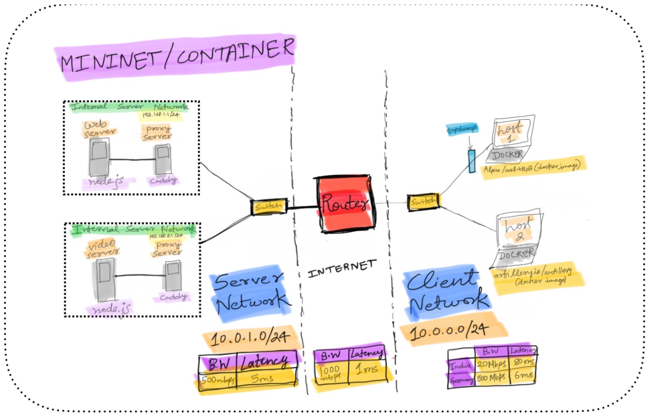

Figure 1: Network Topology

2.1.2 *Router*.

- The router (`r1`) is a Linux-based host with IP forwarding and routing enabled.
- Gateway between the clients and proxy servers for web and video. It can act as an internet.
- Latency is introduced to replicate the real world networks and multiple hops in the internet.
- The router routes traffic from clients (varying bandwidth and latency, depending on the network to test) to proxy servers and return traffic is properly routed back.

2.1.3 *Switches*.

- Three switches in our topology: `s1`, `s2`, and `s3`.
- s1:
  - Connects the client hosts (`h1`, `h2`) to the router (`r1`).
  - Ensures that all outbound/inbound traffic from clients pass through the router for routing and policy enforcement.
- s2:
  - Connects the router (`r1`) to the public interfaces of the web and video server proxies (`srv1-proxy`, `srv2-proxy`).
  - Forms the core network segment on the server side, with 1000 Mbps bandwidth and 1ms latency to simulate a stable backbone connection.
- s3:
  - Internal switch dedicated to Server Group 1.
  - Connects `srv1-proxy` to its internal `srv1-web` and `srv1-video` servers.
  - This segregation allows for simulating internal service traffic distinct from the public-facing traffic.

  - Web and video servers will always connect via proxy server. They are not publicly routable.

2.1.4 *Servers (Web and Video).*

- The network includes two sets of servers, each consisting of a proxy server, a web server and a video server.
- Proxy Servers (`srv1-proxy`, `srv2-proxy`):
  - Publicly accessible entry point into the server networks
  - Run Caddy-based proxy configurations listening on secure ports (e.g. 443) and can be mapped externally (e.g., to 8443 and 9443).
- Web Servers (`srv1-web`, `srv2-web`):
  - Node.js-based servers.
  - Expose a web application on port 8000.
- Video Servers (`srv1-video`, `srv2-video`):
  - Node.js-based servers that simulate video content delivery.
  - Expose services on port 9000.
- Both the web and video servers support network diagnostic tools (e.g., `tcpdump`, `ping`, `iperf3`) for collecting and analyzing network metrics, similar to the client hosts.

2.2 Server Side Implementation

The server-side implementation of this project is a containerized architecture utilizing Docker, Caddy, and Node.js to serve static and dynamic content. It consists of the following components:

- **Caddy Reverse Proxy:** The **Caddy reverse proxy** serves as the gateway for all incoming traffic, listening on port `8443` and handling secure HTTPS communication using internal TLS. It routes requests based on URL paths:
  - Requests with `/web*` are forwarded to the **web service** on port `8000`.
  - Requests with `/video*` are routed to the **video service** on port `9000`.
- **Web Service:** The **web service** is a Node.js application powered by Express.js, designed to handle diverse content:
  - **Static Content:** Hosted in the `public` directory, including:
    - **Images:** `sample.png`, `sample.jpg`.
    - **Documents:** `sample.pdf`, `sample.csv`.
    - **JSON API Responses:** `sample.json`.
    - **Static page:** `index.html`.
  - **Dynamic Routing:** Requests under `/web/*` are dynamically resolved to files within subdirectories of `public`.
- **Video Service:** The **video service**, also built with Node.js and Express.js, specializes in delivering video content in various sizes:
  - **Available Videos:** MP4 files in sizes of **1 MB**, **10 MB**, **50 MB**, and **100 MB**.

  It provides two delivery modes:
  - **Static File Delivery:** Videos served directly under /video/:filename.
  - **Streaming:** Implements HTTP range requests, allowing partial content delivery for efficient playback.

  This setup facilitates the testing of QUIC and HTTP/3 under diverse streaming scenarios.
- **Inter-Service Networking:** All services are interconnected within a shared Docker network (`server-internal-net`) using Docker's bridge driver. The reverse proxy communicates with backend services over private IPs and ports, ensuring isolation and secure inter-service data flow.

The server-side implementation integrates the reverse proxy, web service, and video service into a scalable, secure, and modern infrastructure.

2.3 Client Side Implementation

The client-side implementation uses Docker containers based on the Alpine image to automate the process of capturing and analyzing network traffic for QUIC and TCP connections during simulated web and video workloads. This setup ensures a lightweight and portable environment for executing curl requests and capturing network data. We generate packet captures (.pcap files) and logs, allowing for performance evaluation and protocol comparison. This automation can include multiple iterations, ensuring repeatability and reliability of the tests.

For web workloads, we handle HTTP/3 (QUIC) and HTTP/2 (or below) (TCP) requests separately. It begins by defining *.pcap* filenames that include iteration numbers and placeholders for content type. Unique ports are dynamically selected in each iteration to avoid conflicts. For each request, tcpdump is launched in the background to capture network packets on the designated port. The script then executes a curl command with the $http3$ flag to perform QUIC requests, while standard HTTP requests omit this flag. The results are logged separately for QUIC and TCP workloads in the respective logs. After each request, the tcpdump process is terminated to ensure that the captures are complete and do not overlap. Sleep intervals are strategically used to maintain sequential execution and avoid race conditions.

We also include a separate workload for video downloads, where we test varying file sizes: $1MB, 10MB, 50MB, and 100MB$. For each file size and iteration, *.pcap* files are generated, named with details about the video size, iteration, and region. Similar to the web workload, ports are selected dynamically and tcpdump is employed to capture network traffic. The curl requests are adapted to download video files over HTTP/3 (QUIC) and HTTP/2 (or below) (TCP). Again, the results are logged separately for QUIC and TCP workloads in the respective logs. Each request and its associated capture follow the same cleanup and process management steps as the web workload.

2.4 Data Collection

2.4.1 *Extracting metrics.*

- Python script uses `pyshark.FileCapture` to read packets from a given pcap file.
- Various metric calculations required for QUIC analysis
  - QUIC packets are identified based on their packet type:
    - **Long header packets:** Typically correspond to handshake stages (e.g., Initial, Handshake).
    - **Short header packets:** Correspond to actual application data.
  - **Total Time:** The total duration from the beginning of the QUIC connection attempt to the end of data reception.

  - **Time to First Byte (TTFB):** Time elapsed from when the connection attempt began until the arrival of the first byte of application data.
  - **Download Time:** Time taken to receive all subsequent data after the first byte has arrived.
  - **Connection Time:** Time required to complete the QUIC handshake and fully establish the connection before data transfer begins.
- Various metric calculations required for TCP analysis
  - **Connection SYN Time:** Timestamp of the first TCP SYN packet, indicating the start of the TCP three-way handshake.
  - **Connection ACK Time:** Timestamp of the TCP ACK packet meaning the three-way handshake is completed and fully established TCP connection.
  - **TLS Time:** Difference in the timestamp of the Client Hello message (`handshake_type = 1`) and Server Hello message, indicaiting TLS handshake.
  - **Download Time:** Difference in the timestamp of the first TLS application data packet (e.g., HTTP response) and final TLS data packet prior to session teardown (e.g., FIN-ACK), indicating data transfer phase.
  - Additional metadata such as the TLS version and the identified workload type etc. are also recorded for in detail analysis.
- By distinguishing handshake packets from payload packets, the script can separate the connection establishment phase from the data transfer phase.
- These metrics provide insight into various aspects of QUIC performance, including handshake efficiency, initial response times, and overall data delivery speed.

3 EVALUATION

We have evaluated web workloads with different file types such as pfd, png, html, css, js etc by sending 100 requests of each type and have captured the packets for analysis. For video workloads, we have sent about 35 http request of different file sizes such as 1mb, 10mb, 50mb and 100mb to the video server and captured traffic for the same.
The web and video workloads tests are done for both India network(20 Mbps BW and 80ms latency) and Germany(500 Mbps BW and 6ms latency).

3.1 Measuring Network Performance Using `iperf`

To evaluate the network performance between the client and server hosts, we use `iperf3`, an open source tool for measuring bandwidth, latency, and other network-related metrics.

- Bandwidth Measurement: Provides detailed information about the bandwidth achieved during the test, measured in Mbps or Gbps. Results include metrics such as transfer size, bandwidth, and retransmission rates.
- Latency and Jitter
- Connection Stability - Consistent bandwidth and low retransmission rates indicate a stable and reliable network connection between the client and server

3.2 Network Traffic Generation Tools

To comprehensively evaluate the performance of our network topology, we employ a suite of open-source traffic generation tools. These tools enable us to simulate various traffic patterns, protocols and user behaviors which provides a realistic assessment of network performance under different conditions. The primary tools utilized in our study are `iperf3`, `D-ITG` (Distributed Internet Traffic Generator), and `Tsung`. Below, we discuss each tool in detail, highlighting their features and limitations.

3.2.1 `iperf3`. `iperf3` is a widely-adopted tool for measuring maximum achievable bandwidth on IP networks. It supports both TCP and UDP protocols and provides detailed metrics on throughput, packet loss, and jitter.

Limitations: Cannot simulate real-world internet traffic as it generates uniform traffic.

3.2.2 `D-ITG` *(Distributed Internet Traffic Generator).* `D-ITG` - tool designed to generate realistic traffic patterns by simulating various network protocols and user behaviors. It supports a wide range of protocols, including HTTP, FTP, and custom protocols.

3.2.3 `Tsung`. `Tsung` - high-performance benchmarking tool designed for testing the scalability and performance of various network services, including HTTP, WebDAV, SOAP, and PostgreSQL. It is particularly well-suited for simulating multiple users and complex interaction patterns with server applications.

By leveraging these tools, we can comprehensively evaluate the network performance of our topology under various traffic conditions, ensuring robust and reliable network behavior in real-world scenarios.

3.3 Application Traffic Generation

In addition to transport-layer traffic generation tools, it is essential to simulate application-layer traffic to simulate real-world applications perform under various network conditions. We utilized Artillery, a modern, powerful, and easy-to-use load testing toolkit. Artillery allows us to test the performance and scalability of web applications, APIs, and real-time services by simulating realistic user interactions and traffic patterns.

3.3.1 Protocols Supported. Artillery supports multiple protocols like http/https, websocket, gRPC, etc.

3.3.2 Interpreting Artillery Results. After executing the load test, Artillery provides a detailed report containing various performance metrics such as response times, requests per second, latency, errors, latency, visualization etc.

4 EVALUATION RESULTS

4.1 India

4.1.1 Video Workloads.

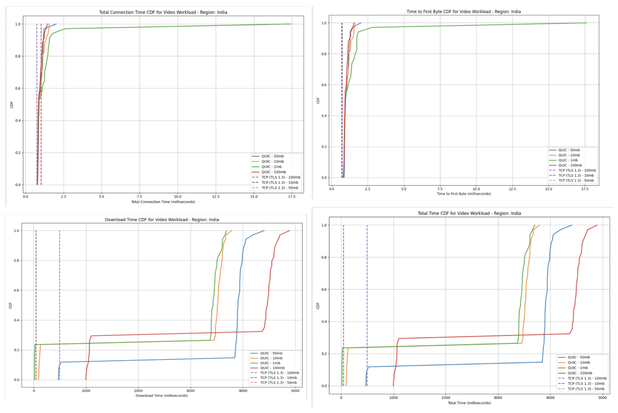

Figure 2: Video Workloads - India Region: Total Connection Time, Time to First Byte, Download Time, and Total Time CDFs.

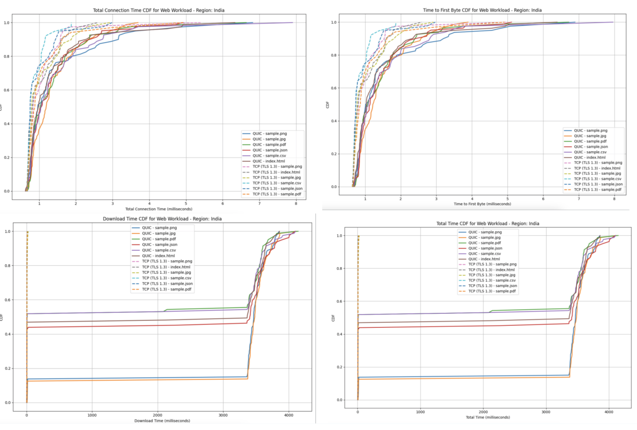

Figure 3: Web Workloads - India Region: Total Connection Time, Time to First Byte, Download Time, and Total Time CDFs.

Total Connection Time CDF for Video Workload - Region: India. The total connection time CDF reveals that TCP consistently achieves faster and more stable connection times compared to QUIC. For all file sizes (1MB, 10 MB, 50 MB, and 100 MB), TCP shows tight clustering around a median connection time of under 2 milliseconds. In contrast, QUIC demonstrates much greater variation, with some connections taking significantly longer than the median.

Time to First Byte CDF for Video Workload - Region: India. The time-to-first-byte (TTFB) CDF further underscores TCP's superior performance. Across all video file sizes, TCP maintains consistently lower TTFB values, with minimal variation. QUIC, however, shows significant variability, particularly for larger file sizes, where the tail of the distribution indicates much longer TTFB for some connections.

Download Time CDF for Video Workload - Region: India. For download times, TCP outperforms QUIC across all file sizes, with faster and more predictable performance. The CDF curves for TCP are tightly grouped, indicating low variability. In contrast, QUIC shows a wider spread in download times, reflecting its inconsistent performance.

Total Time CDF for Video Workload - Region: India. The total time CDF combines the effects of connection time, TTFB, and download time. TCP maintains its lead with consistently lower and less variable total times across all file sizes. QUIC, in comparison, exhibits substantial variation, especially for larger file sizes, resulting in longer total times for a significant portion of the connections.

4.1.2 *Web Workloads.*

Total Connection Time CDF for Web Workload - Region: India. For web workloads, TCP once again achieves faster connection times than QUIC. However, TCP shows slightly more variation in connection times compared to video workloads, with the curves for different file types (e.g., PNG, JSON, and HTML) slightly diverging. QUIC, on the other hand, maintains its trend of higher variability, with some connections taking considerably longer than the median.

Time to First Byte CDF for Web Workload - Region: India. The time-to-first-byte (TTFB) results follow a similar pattern. TCP achieves consistently lower TTFB values with relatively minimal variation across file types. QUIC shows higher variability, with a wider spread in TTFB values. This variability is particularly pronounced for smaller file types, such as PNG and JSON.

Download Time CDF for Web Workload - Region: India. The download time CDF indicates that TCP achieves faster download times than QUIC for most file types. TCP's performance remains predictable with tight CDF curves, while QUIC displays significant variability. The gap between TCP and QUIC is less pronounced for smaller files, but QUIC's variability increases with larger file types such as PDF and HTML.

Total Time CDF for Web Workload - Region: India. The total time CDF consolidates the trends observed in connection time, TTFB, and download time. TCP consistently outperforms QUIC across all file types, with lower total times and less variability. While TCP exhibits slightly more variation than it does for video workloads, QUIC remains the less consistent protocol, with a significant fraction of connections taking much longer to complete.

4.2 Germany

4.2.1 *Video Workloads.*

Total Connection Time CDF for Video Workload - Region: Germany. TCP achieves faster connection times across all file sizes with minimal variability. Even for larger file sizes, such as 100MB, TCP's connection times remain consistent. In contrast, QUIC exhibits wider variability in connection times, particularly for smaller file sizes (e.g., 1MB and 10MB). This reinforces TCP's reliability and efficiency in connection establishment.

Time to First Byte CDF for Video Workload - Region: Germany. For TTFB, TCP consistently outperforms QUIC with faster and more predictable performance. TCP's distribution remains tightly clustered, demonstrating low variability across all file sizes. QUIC,

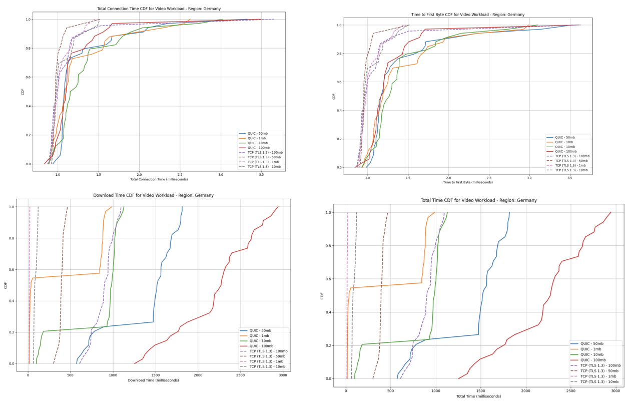

Figure 4: Video Workloads - Germany Region: Total Connection Time, Time to First Byte, Download Time, and Total Time CDFs.

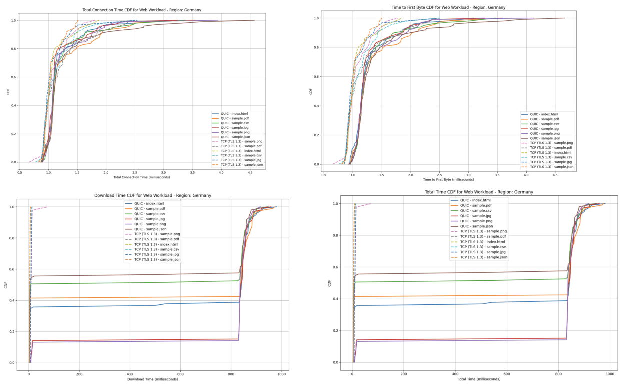

Figure 5: Web Workloads - Germany Region: Total Connection Time, Time to First Byte, Download Time, and Total Time CDFs.

on the other hand, shows significant dispersion in its TTFB values, particularly for larger files like 100MB.

Download Time CDF for Video Workload - Region: Germany. TCP demonstrates significantly faster and more consistent download times across all file sizes. QUIC, however, exhibits considerable variation, especially for medium to large file sizes such as 10MB and 100MB. This variability underscores TCP's superiority in efficiently handling video data transfers.

Total Time CDF for Video Workload - Region: Germany. For total time (sum of connection time, TTFB, and download time), TCP once again outperforms QUIC by achieving consistently lower total times with minimal variability across all file sizes. In contrast, QUIC displays substantial variation, particularly for larger files, highlighting its slower and less predictable performance.

4.2.2 Web Workloads.

Total Connection Time CDF for Web Workload - Region: Germany. For web workloads, TCP once again achieves faster connection times than QUIC. However, TCP shows slightly more variation in connection times compared to video workloads, with the curves for different file types (e.g., PNG, JSON, and HTML) slightly diverging. QUIC, on the other hand, maintains its trend of higher variability, with some connections taking considerably longer than the median.

Time to First Byte CDF for Web Workload - Region: Germany. TCP demonstrates faster and more predictable TTFB values for all file types. While there is slightly more variability in TCP's TTFB compared to its video workload performance, its performance remains significantly more stable and efficient than QUIC's. QUIC exhibits much broader variation in TTFB, indicating less reliable performance.

Download Time CDF for Web Workload - Region: Germany. TCP achieves faster and more consistent download times than QUIC for all file types. QUIC's download times vary significantly, particularly for larger file types such as PDFs, while TCP maintains tightly clustered performance. This highlights TCP's efficiency in handling web data transfers.

Total Time CDF for Web Workload - Region: Germany. For total time, TCP continues to outperform QUIC with consistently lower values. QUIC's performance is characterized by significant variation across all file types, while TCP's total times remain tightly clustered and predictable, further emphasizing its reliability and efficiency.

5 CHALLENGES FACED

- **Server Implementation Issues:** Integrating a containerized architecture for server implementation posed challenges in configuring inter-container networking. Specifically, setting up the Caddy server as a reverse proxy required troubleshooting for proper routing of requests to the web and video services. Debugging these network flows within Docker's shared network was time-consuming but essential for achieving seamless protocol compatibility.
- **Client Implementation Issues:** Measuring performance metrics like Time to First Byte (TTFB) and download times using `depot_tools` that is used by the paper for QUIC analysis was challenging. The tools lacked sufficient support for HTTP/3. To address this, we replaced `depot_tools` with the Alpine-based implementation of `curl`, ensuring reliable support for QUIC requests. This adjustment streamlined data collection but required additional setup and testing to validate results.

6 CONCLUSION

In this project, we successfully implemented a containerized architecture to evaluate the performance of the QUIC protocol across web and video workloads. By leveraging tools such as Docker, Caddy, and Alpine-based `curl` implementations, we ensured a reproducible and modular testing environment.

Our evaluation revealed that TCP consistently outperformed QUIC in connection times, Time to First Byte (TTFB), and download times across both web and video workloads. This result diverges from the findings in the original paper, which highlighted QUIC's superior performance in reducing connection establishment time and improving throughput. The discrepancies can be attributed to several factors:

- **Hardware and Software Implementations:** Our implementation relied on containerized, Alpine-based environments, which may not have been as optimized for QUIC as the setups in the original paper.
- **Packet Analysis Challenges:** QUIC's use of long and short headers posed difficulties in analyzing packet flows and pinpointing the start of data packets in packet captures (pcaps), potentially affecting the accuracy of key metrics.
- **Controlled vs. Real-World Environments:** Unlike the real-world network setups used in the paper, our controlled environment may not have showcased QUIC's advantages, such as resilience to packet loss and head-of-line blocking.
- **Protocol Optimization:** The original study may have leveraged finely tuned QUIC implementations, whereas we used off-the-shelf tools with minimal optimizations, limiting QUIC's potential.

Overall, our findings highlight the importance of experimental context and implementation specifics in performance evaluations. While QUIC shows promise as a modern transport protocol, its adoption and effectiveness may depend heavily on network conditions, workload types, and the degree of protocol optimization.

7 FUTURE WORK

This project provided valuable insights into the performance of QUIC and TCP, but several areas remain for future exploration:

- **Incorporating Real-World Network Variability:** Future work could expand the evaluation to include real-world network scenarios, such as high-latency or lossy networks, to better understand QUIC's resilience to packet loss and head-of-line blocking.
- **Analyzing Resource Utilization:** Measuring CPU and memory usage for QUIC and TCP would help evaluate their computational trade-offs, especially in resource-constrained environments.

By addressing these areas, future research can provide a deeper understanding of QUIC's potential and its applicability in diverse networking environments.

REFERENCES

[1] T. Shreedhar, R. Panda, S. Podanev, and V. Bajpai, "Evaluating QUIC performance over web, cloud storage, and video workloads," *IEEE Trans. Netw. Serv. Manage.*, vol. 19, no. 2, pp. 1366-1380, Jun. 2022.

[2] Our project implementation repository. Available at: https://github.com/Dharmik1710/Networks---Analyzing-QUIC-protocol

APPENDIX: INDIVIDUAL CONTRIBUTIONS

Each member of our team made significant contributions not only to their assigned tasks but also to various other aspects of the project, including the proposal, project implementation, presentation, and final report. The collaborative effort ensured that every part of the project benefited from diverse perspectives and collective problem-solving. Despite this overlap in contributions, the following provides a more specific breakdown of the primary responsibilities assigned to each team member throughout the project.

Arnav Gupta

Arnav played a key role in the analysis and evaluation of results. He collaborated with team members collecting data to identify the most relevant metrics to capture for the study. Additionally, he processed the generated PCAP files by converting them into CSV format, creating meaningful graphs from the data, and analyzing these visualizations. His efforts ensured the results were clearly presented during the online presentation and elaborated on in greater detail in this report.

Ajinkya Nawarkar

Ajinkya's contribution to this project included implementing the Web Service and Video Service on the server side using Node.js and integrating them with the Caddy Reverse Proxy within a containerized Docker architecture. On the client side, he contributed to automating testing workflows, including capturing and analyzing QUIC and TCP network traffic using tools like tcpdump, and ensuring efficient performance evaluation through structured logging and dynamic resource management. This work enabled a robust, scalable, and repeatable framework for protocol analysis.

Nishtha Aggarwal

Nishtha contributed to enhancing the client-side implementation by scripting tools to automate the generation of packet captures (PCAPs) for both TCP and QUIC connections. To streamline analysis, she helped develop scripts to convert PCAP files into CSV formats, enabling structured and detailed examination of network metrics. Additionally, she designed and implemented scripts to create graphs that visualized key performance metrics, such as connection time and throughput, for comparative evaluation of TCP and QUIC.

Soham Sharma

Soham's contribution to this project included implementing the client-side video workload functionality and creating a containerized Docker architecture that enabled the client to send 100 curl requests. He designed and integrated logging mechanisms to capture and analyze metrics essential for evaluating results. This work facilitated efficient testing, ensured scalability, and provided a structured approach to performance analysis and evaluation.

Dharmik Patel

Dharmik's contribution in this project has been about designing and implementing the network topology. He has also worked on setting up the networks and routing, both on the client and server side and introduced latency and bandwidth restriction on all the network devices(server switches, client switches, router) to simulate real world networks. He has also tried introducing network level traffic using open source tools like iperf, D-ITG and Tsung. Artillery is used for generating application level traffic to efficiently analyse the TCP and QUIC performance.

Jevin Modi

Jevin developed a Python-based framework using pyshark.FileCapture to extract and analyze key metrics from pcap files, enabling a detailed evaluation of QUIC and TCP performance. This included parsing handshake and payload packets to calculate metrics such as connection time, time to first byte (TTFB), download time, and total connection time for QUIC, as well as SYN/ACK times, TLS handshake duration, and download time for TCP. The framework provided valuable insights into handshake efficiency, initial response times, and data delivery speed, forming the foundation for comparative performance analysis. In addition to this, Jevin also helped across all different parts of the project including testing and resolving issues within Network setup.