**Journal of Applied Informatics and Computing (JAIC)**
Vol.10, No.2, April 2026, pp. 1291~1303
e-ISSN: 2548-6861


# HTTP/3 vs HTTP/2: A Comparative Analysis of Latency and Throughput in RESTful API Transactions Under Varying Concurrency, Payload, and Degraded Network Conditions

**Vidya Az Zahra [1]*, Rizka Ardiansyah [2]***
* Informatics Engineering, Faculty of Engineering, Tadulako University, Indonesia
vidyaazzahra44@gmail.com [1], rizka@untad.ac.id [2]

**Article Info**

***Article history:***

Received 2025-11-19
Revised 2026-02-20
Accepted 2026-02-27

***Keyword:***

*HTTP/3,*
*HTTP/2,*
*Network Performance,*
*RESTful API.*

**ABSTRACT**

This research empirically compares the performance of RESTful APIs based on HTTP/3 (quic-go) and HTTP/2 (Go standard library with TLS 1.3) under varying workload and network conditions. A rigorous quantitative factorial experiment design $2 \times 2 \times 2 \times 2 \times 3$ was implemented in a controlled GCP Virtual Machine environment. The experiment tested two protocols, two REST methods, two payload sizes, two concurrency levels, and three network conditions (Baseline, Medium, and Poor). Network degradation was emulated using traffic control to simulate latency, bandwidth limitation, and packet loss. Performance was measured using Latency, Time to First Byte (TTFB), and Throughput. Non-parametric statistics (Mann-Whitney U Test, Kruskal-Wallis H Test, and $\in_{rb}$ Effect Size) were used for hypothesis testing. Results show that HTTP/3's performance benefit is highly context-dependent. Under severely degraded network conditions, HTTP/3 exhibits a statistically and practically significant advantage, particularly for high-concurrency GET operations, achieving up to 31.2% lower latency and 45.4% higher throughput compared to HTTP/2. Across nearly all scenarios, HTTP/3 consistently delivers superior TTFB performance, averaging a 35.5% reduction, reflecting its efficient connection establishment and stream-level loss recovery. Conversely, under ideal network conditions with high concurrency and medium payload sizes, HTTP/2 outperforms HTTP/3, recording up to 29.3% lower latency and 22.7% higher throughput, suggesting implementation-level processing trade-offs in QUIC-based systems. These findings indicate that protocol selection should be guided by workload characteristics and network conditions rather than assumed protocol superiority. The results are implementation-dependent and derived from a controlled synthetic workload, which limits direct generalization to production environments.



## I. INTRODUCTION

The evolution of the Hypertext Transfer Protocol (HTTP) from version 1.1 to HTTP/2 introduced significant enhancements in web communication efficiency through the implementation of multiplexing and header compression [1]. Nevertheless, HTTP/2's foundation on the Transmission Control Protocol (TCP) transport layer leaves a fundamental vulnerability known as Head-of-Line Blocking (HoLB) [2]. This limitation means that the loss of a single packet at the TCP layer can effectively halt the processing of all data streams running on the same connection, making it suboptimal in networks characterized by instability, high latency, or substantial packet loss. As a fundamental solution to the HoLB problem and the inefficiencies of the TCP/TLS handshake, HTTP/3 was introduced, utilizing the QUIC (Quick UDP Internet Connections) protocol over the User Datagram Protocol (UDP) [3]. QUIC addresses HoLB by enabling independent data streams (per-stream loss recovery); a packet loss only affects the individual stream without impeding the processing of others [3][4].

Furthermore, HTTP/3 natively integrates Transport Layer

Security (TLS) 1.3 and supports connection resumption, a key advantage for reducing connection initiation latency [5].

Despite the claimed theoretical superiority of HTTP/3 in network resilience, in-depth empirical evaluation focused on RESTful API transactions, which form the backbone of microservices architecture and modern applications, remains limited. Prior studies tend to compare HTTP/3 with older protocols like HTTP/1.1 [6], or only focus on narrow aspects such as web page loading time [7]. Other research has compared HTTP/3 client implementations [8], focusing on client-side HTTP/3 performance using ngtcp2 and quiche with cURL GET requests, without delving into more complex RESTful API transactions or utilizing quic-go as the implementation. Conversely, [9] examined general concurrent web server performance but did not specifically analyze RESTful API performance using POST and GET methods in depth, and utilized a different HTTP/3 implementation than the focus of this study. Meanwhile, [2] compared HTTP/3 and HTTP/2 in a proxy environment, but this was generally limited to file transfer and web page loading, rather than specific client-server RESTful API transactions. Therefore, this research aims to fill this gap by providing a rigorous comparative performance analysis between the two protocols (HTTP/3 vs. HTTP/2) specifically within the context of RESTful API transaction.

Therefore, this research aims to fill this gap by providing a rigorous comparative performance analysis between the two protocols (HTTP/3 vs. HTTP/2) specifically within the context of RESTful API transaction. To the best of our knowledge, few studies have conducted a rigorous comparative analysis using the quic-go implementation within a comprehensive factorial design on RESTful API transactions. We acknowledge that the workload used is a synthetic microbenchmark replicating common API patterns, not a live production application.

This performance comparison requires strict variable isolation. Consequently, both protocols are tested using the Transport Layer Security (TLS) 1.3 security protocol. This configuration is absolutely necessary because HTTP/3 inherently integrates TLS 1.3 within QUIC [5]. By standardizing on TLS 1.3 for the HTTP/2 implementation, any observed performance difference can be purely attributed to the fundamental differences in the transport layer (QUIC/UDP vs. TCP).

The HTTP/3 implementation utilizes the quic-go library within a Golang environment. The selection of Golang and quic-go is based on their dominance in developing high-performance backend services, microservices, and API gateways that require granular control over networking and concurrency [10][11]. Testing was conducted in a controlled Virtual Machine (VM) environment on Google Cloud Platform (GCP), with data accessed via Google Cloud Storage (GCS). This isolated cloud environment ensures consistent replication.

This research simultaneously analyzes the interaction of five critical factors that replicate API workload:

1) Transaction Method (POST /upload and GET /download): These two methods represent the fundamental read and write operations in RESTful APIs, mirroring core activities like media sharing in modern cloud-based services [12][13]. Testing them observe how each protocol handles bidirectional data flow: POST primarily assesses client-side payload sending efficiency, while GET assesses server-side response payload delivery.
2) Payload Size (2 MB image and 10 MB video): Data volume is a critical determining factor in throughput performance [14]. The small payload (2 MB), represented by images, and the medium payload (10 MB), represented by video, are chosen because they represent the most common and relevant types of binary data transfer [15] in modern RESTful API interactions, especially in scenarios involving cloud storage services and media sharing platforms. This variation is designed to evaluate protocol efficiency, specifically addressing prior concerns that HTTP/3 implementations may be less efficient with large payloads [8].
3) Concurrency Level (100 and 500 Total Requests): The level of concurrency is a key determinant of server scalability and is a condition that stresses the multiplexing mechanisms of both protocols. The selection of these values is based on a study by [9] which suggests that HTTP/3 with QUIC can handle concurrent connections more efficiently because it does not suffer from head-of-line blocking. However, that testing was based on general server performance, not specifically on RESTful API workloads, and used a different HTTP/3 implementation. This study performed at two concurrency levels to evaluate simultaneous connection handling: 5 concurrent users making 20 requests each (totaling 100 requests), and 20 concurrent users making 25 requests each (totaling 500 requests).
4) Simulated Network Conditions (Baseline, Medium, dan Poor): This metric evaluates the protocol's resilience against suboptimal network conditions, which are often encountered in mobile or wireless networks [2]. Tolerance to packet loss is a key theoretical advantage of the QUIC protocol underlying HTTP/3 due to its per-stream loss recovery mechanism [16]. To test this resilience, this study uses three network condition scenarios: Baseline Network (no traffic control), Medium Network (Latency 30ms, Bandwidth 20Mbps, Packet Loss 0.5%), dan Poor Network (Latency 50ms, Bandwidth 10Mbps, Packet Loss 2%).

The observed dependent variables include Latency (Total Response Time), which reflects the overall end-user experience [7]; Time to First Byte (TTFB) which serves as an indicator of early connection and response initiation efficiency [7]; and Throughput (MB/s), which evaluates data

transfer efficiency under varying workload and network conditions [14][17].

The primary goal is to empirically compare HTTP/3 and HTTP/2 across varying load and network conditions to establish data-driven thresholds and guidelines on where adopting HTTP/3 offers a statistically and practically significant performance advantage. Specifically, this study addresses the following research question: Under what network and workload conditions does HTTP/3 outperform HTTP/2 for RESTful API transactions in terms of latency, TTFB, and throughput?

The hypothesis formulated for this study is based on the main architectural claims of QUIC/HTTP/3, namely the mitigation of Head-of-Line Blocking (HoLB) and handshake efficiency [5][18], which theoretically yield significant benefits in networks with high latency and packet loss [2]:

- Null Hypothesis (H0): There is no significant difference in REST API performance between the HTTP/3 and HTTP/2 protocols across various network conditions.
- Alternative Hypothesis (H1**)**: The HTTP/3 protocol demonstrates significantly better performance compared to HTTP/2 under degraded network conditions.

## II. METHODE

The research method employed is a purely quantitative comparative study utilizing a factorial experimental design (2 × 2 × 2 × 2 × 3). This design aims to analyze the main effects of five independent variables and scenario-specific behaviour on the performance of REST API transactions based on the HTTP/3 and HTTP/2 protocols. The overall research flow is presented in Figure 1.

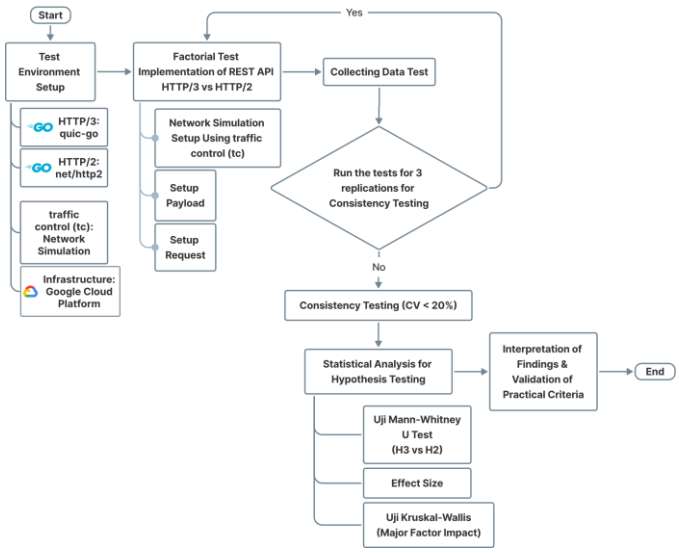


Figure 1. Flowchart Design for Experiment Testing

### A. Test Environment Setup

The infrastructure was designed as a controlled environment using the Google Cloud Platform (GCP) services. All testing components (HTTP/3 & HTTP/2 Client and Server) were executed within a single Virtual Machine (VM) inside the same Private Network on GCP. This infrastructure selection ensures consistent Round-Trip Time (RTT) and isolates the testing from public internet fluctuations, ensuring that the measured performance variations are solely caused by the experimental variables, not external factors. The main specifications of the infrastructure and software are listed in Table 1:

TABLE I
HARDWARE AND SOFTWARE SPECIFICATIONS

| Virtual Machine (GCP) | Protocol Ports | Key Software Tools |
|---|---|---|
| Instance Type: e2-medium (2 vCPUs, 4 GB Memory) | HTTP/2: Port 4442 | Network Emulation: tc (traffic control) |
| Operating System: Ubuntu 24.04 LTS | HTTP/3: Port 4443 | Protocol Validation: curl |
| Storage: 200GB Balanced persistent disk | | Network Validation: iperf3 |
| Region: asia-southeast2-a | | HTTP/3 Implementation: quic-go (Golang) |
| | | HTTP/2 Implementation: net/http2 (Golang) |

Prior to commencing the factorial testing, server functionality validation was conducted using curl to ensure that both protocols were running on the correct port and transport layer, thereby confirming that the servers were indeed operating under their respective protocols by checking for a 200 response code.

### B. Factorial Test Implementation

The design of performance comparison between HTTP/3 and HTTP/2 in RESTful API transactions, which illustrates the architecture, is shown in Figure 2. This architecture shows the HTTP/3 client (quic-go) and HTTP/2 client (net/http2) sending POST and GET requests to the server integrated with Google Cloud Storage (GCS). This systematic design results in 144 scenarios (2 Protocols × 2 Methods × 2 Payloads × 2 Concurrency × 3 Networks, multiplied by 3 replications per scenario). The rigorous testing sequence initiates with the HTTP/3 protocol using the POST method. This begins by testing the small payload under a light load concurrency within the Baseline Network condition. Performance metrics are meticulously recorded for this specific scenario. After that, the GET process is executed. This entire process is subsequently repeated for the high load concurrency. Following this, the payload is increased to the medium payload, and the sequence of both light and high load concurrency is executed again. This entire testing process will be repeated for the Medium Network and Poor Network. After that, the process will be tested for HTTP/2.

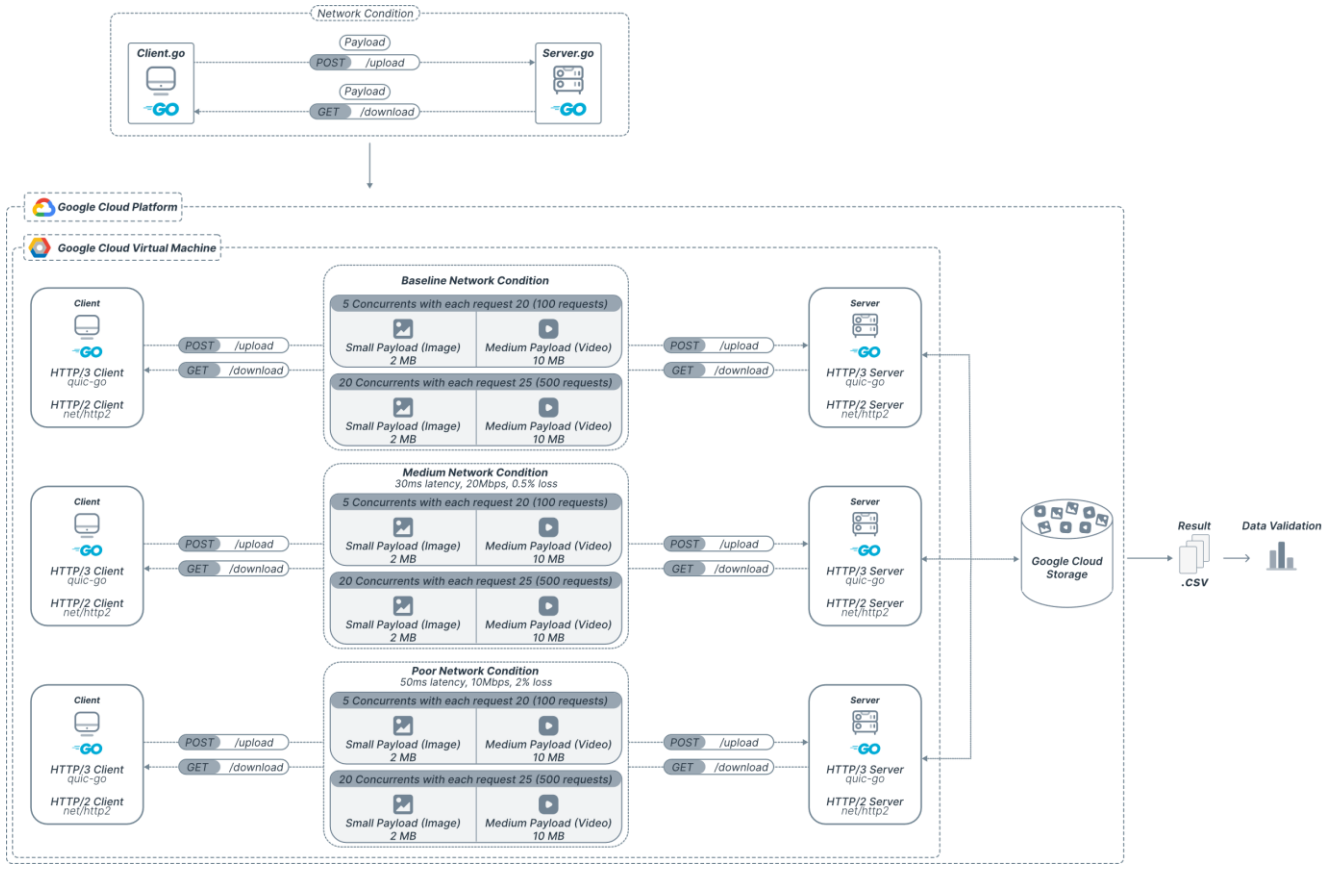


Figure 2. Design of the Testing Process Scenario

The above scenario design will be implemented to analyze the main effects and complex interactions among five independent variables on key performance metrics. The five independent variables influencing this research are as follows:

TABLE II
INDEPENDENT VARIABLES (FACTORS)

| Factor | Variants | Description |
|---|---|---|
| Protocol (2) | HTTP/3, HTTP/2 | The core variable under comparison. |
| Method (2) | POST (Upload), GET (Download) | Representing fundamental REST API operations. |
| Payload Size (2) | Small (2 MB), Medium (10 MB) | Testing protocol efficiency against data volume. |
| Concurrency Level (2) | Low (100 requests total), High (500 requests total) | Testing scalability and connection handling under load. |
| Network Condition (3) | Baseline, Medium, Poor | Testing resilience against simulated network degradation. |

To evaluate protocol resilience, three network condition scenarios were simulated separately using the tc (traffic control) tool on the testing Google Virtual Machine. This simulation replicates common challenges encountered in mobile environments or unstable networks.

TABLE III
TABLE NETWORK SIMULATION

| Condition | Latency Delay | Bandwidth Limitation | Packet Loss |
|---|---|---|---|
| Baseline Network | 0 ms | No Limit | 0% |
| Medium Network | 30 ms | 20 Mbps | 0.5% |
| Poor Network | 50 ms | 10 Mbps | 2.0% |

Network condition validation was conducted to ensure that the simulated network environments (Baseline, Medium, and Poor) were successfully applied using tc (traffic control) and verified through iperf3.

To comprehensively understand the recorded performance differences, an insight into the unique handshake, transmission, and multiplexing processes of each protocol is essential. The fundamental distinctions lie in the transport layer utilized and how each protocol simultaneously handles data streams.

For HTTP/2, communication begins with the HTTP/2 client establishing a TCP connection to the HTTP/2 server on port 4442, followed by a TLS 1.3 handshake to secure the communication channel. The use of TLS 1.3 in the HTTP/2 configuration was intentionally chosen to ensure a fair comparison with HTTP/3, which natively integrates TLS 1.3 within the QUIC protocol. Once the basic connection is established and secured, the client sends the HTTP/2 Connection Preface, and the server responds with a Settings Frame. At this point, the client can begin creating HTTP/2 streams for each request. The server processes these requests, interacts with Google Cloud Storage (GCS) for upload and download operations, and returns the responses through the corresponding streams. The server also tracks connection status and the number of requests being handled.

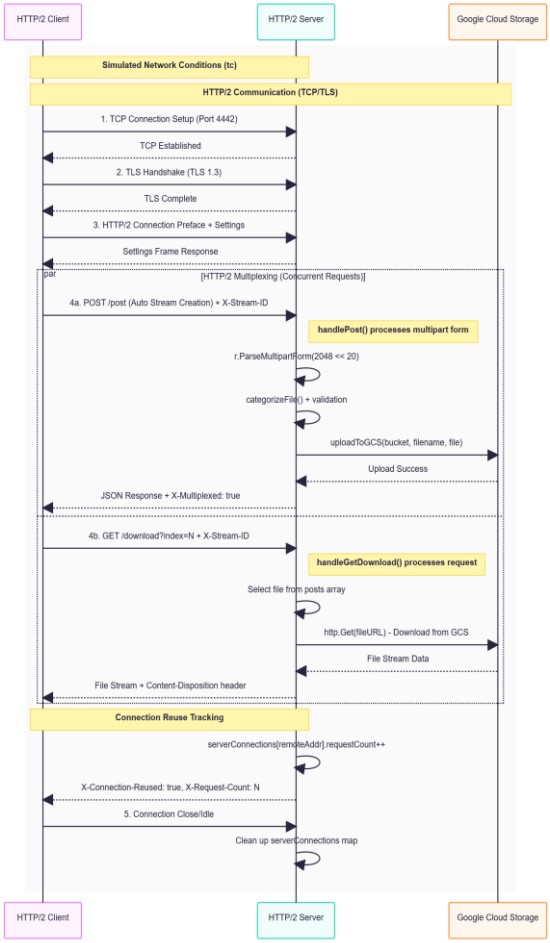


Figure 3. HTTP/2 Client-Server Flow

For HTTP/3 communication, the QUIC protocol, which operates over UDP, is utilized. The HTTP/3 client initiates a UDP connection to the HTTP/3 server on port 4443. The key distinction lies in the QUIC handshake, which inherently integrates TLS 1.3 for secure and efficient connection establishment. Once the QUIC handshake is completed, the client can immediately create HTTP/3 streams. The server processes each stream separately, interacts with Google Cloud Storage (GCS) as needed, and sends back the corresponding responses. The server also monitors and manages all active streams. Ultimately, both HTTP/2 and HTTP/3 connections are closed once all requests have been completed or after a specified timeout period.

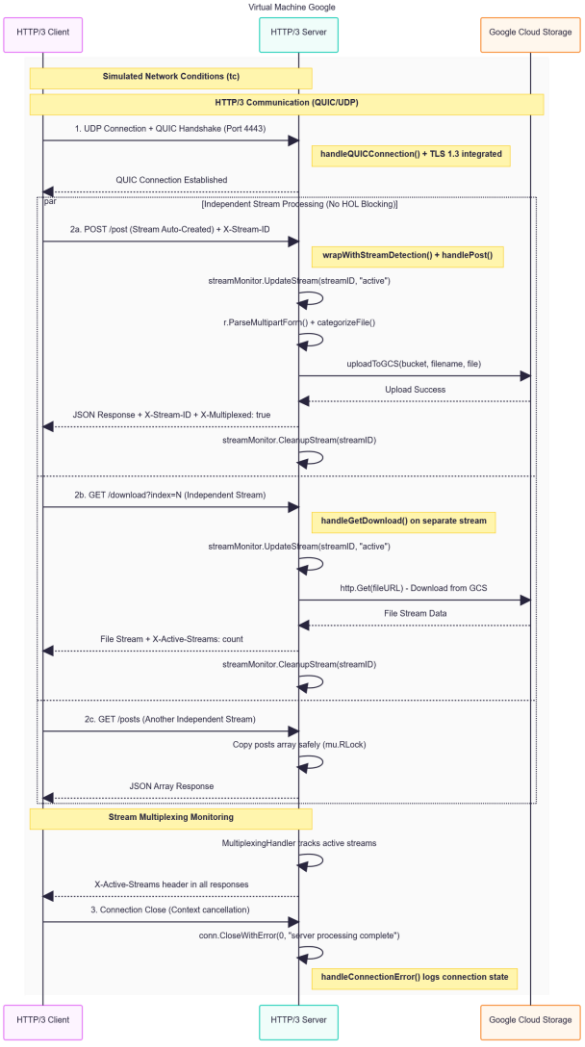


Figure 4. HTTP/3 Client-Server Flow

Each performance metric measured from the communication flow was automatically recorded, processed, and averaged. The final results of these metric calculations were then exported and stored in Comma-Separated Values (CSV) format to facilitate further data analysis.

### C. Collecting Data Test

Performance data collection was conducted by measuring three main performance metrics (Dependent Variables) to assess the efficiency and scalability of protocol communication. These metrics are highly relevant for modern protocols like HTTP/3 in the context of user experience and network resilience:

TABLE IV
PERFORMANCE METRICS (DEPENDENT VARIABLES)

| Metric | Variants |
|---|---|
| Latency (Total Response Time) | The time required from sending the first byte of the request until the final byte of the complete response is received at the client (measured in milliseconds, ms). |
| Time to First Byte (TTFB) | The elapsed time between request dispatch and the reception of the first byte of the response, representing an end-to-end observable approximation of connection and stream initiation latency. (measured in milliseconds, ms). |
| Throughput | The effective per-request application throughput, calculated as the ratio of transferred payload size to the end-to-end request duration. This metric reflects protocol efficiency under concurrent workloads rather than aggregate link saturation. (measured in MegaBytes per second, MB/sec). |

### D. Consistency Testing

The Consistency Testing phase utilized the Coefficient of Variation (CV) to evaluate the reliability and stability of the performance measurements across the three replications conducted for each experimental scenario. This test is crucial for network performance research, as it minimizes the impact of unexpected network fluctuations and ensures data repeatability. Adopting a conservative approach to data precision, the research categorized consistency based on the CV value as follows measurements were classified as highly consistent if CV ≤ 10%, consistent if 10% < CV ≤ 15%, and sufficiently consistent if 15% < CV ≤ 20%. Measurements that meet the "Sufficiently Consistent" threshold are deemed reliable enough to proceed to the main statistical hypothesis testing. In line with system and network performance testing practices, a CV ≤ 20% limit is commonly used as an acceptable precision threshold [19].

### E. Statistical Analysis for Hypothesis Testing

The selection of non-parametric statistical tests, namely the Mann-Whitney U test and the Kruskal-Wallis H test, is justified by the inherent nature of network performance data and the experimental constraints. Preliminary examination, including the Shapiro–Wilk test, indicated that the distribution of latency and throughput measurements across many test scenarios did not satisfy the assumption of normality ($p > 0.05$ failed to reject H0). Furthermore, the constraint of a small sample size (n=3 replications per scenario) makes the variance assumption required for robust parametric methods (such as ANOVA) unstable. Therefore, non-parametric rank tests were adopted because they are robust against non-normal distributions and variances, ensuring reliable comparison of medians for hypothesis testing. This approach is also supported by empirical findings which consistently report that network timeseries data tends to be highly skewed or heavy-tailed [7], [18]

The Mann-Whitney U Test was used to compare the medians of the two independent protocol groups (HTTP/3 vs. HTTP/2) for each test scenario (H1 was accepted if the p-value < 0.05). To analyze the aggregate impact of factors with more than two levels, the Kruskal-Wallis H Test was applied. Crucially, to quantify the magnitude or practical significance of the observed difference, the Effect Size was determined using the Rank-Biserial Correlation ($\in_{rb}$). The criteria for interpreting the magnitude of the Effect Size and the required threshold for the Relative Median Difference are detailed in the table below:

TABLE V
CRITERIA FOR PRACTICAL SIGNIFICANCE

| Quantitative Criteria | Threshold Value | | Practical Interpretation |
|---|---|---|---|
| | Effect Size Category | Notes | |
| Absolute Effect Size | $0.1 \leq \lvert\in_{rb}\rvert < 0.3$ | Small | Minor difference, may not justify migration. |
| | $0.3 \leq \lvert\in_{rb}\rvert < 0.5$ | Moderate | Substantial difference, supports an engineering/migration decision. |
| | $0.5 \leq \lvert\in_{rb}\rvert \leq 1.0$ | Large | Very strong difference, migration is highly recommended. |
| Relative Median Difference (Percentage Difference) | Percentage Difference ≥ 10% | | Clear and technically relevant performance difference. |

The use of Effect Size ($\in_{rb}$) is highly appropriate for the non-parametric Mann-Whitney U test. The threshold of $\lvert\in_{rb}\rvert \geq 0.3$ was selected to classify the difference as Moderate or Large, which is an adaptation of Cohen's conventions for engineering contexts [20]. A difference reaching the Moderate level is deemed practically significant, indicating that the observed performance change is substantial enough to inform engineering or deployment decisions, depending on workload and system requirements.

Furthermore, a Relative Median Difference ≥ 10% was used as the relevant technical threshold. In network and web performance studies, a percentage difference of 10% or more is considered to signal a clear and technically relevant performance improvement or degradation for the user experience or for development decisions, as implied in performance analyses like that by [7].

### *F. Interpretation of Findings & Validation of Practical Criteria*

The final interpretation of findings required that both statistical significance (p-value) and practical significance (effect size $\in_{rb}$) criteria be simultaneously satisfied. This approach ensures that the reported performance differences reflect meaningful technical relevance rather than isolated statistical artifacts.

From a practical perspective, observed Moderate to Large effect sizes ($|\in_{rb}| \geq 0.3$), when combined with relative median differences exceeding 10%, indicate performance variations that are likely to be noticeable in production environments. For latency-sensitive RESTful APIs, such differences may influence Service Level Agreement (SLA) compliance, particularly for user-facing operations such as authentication, configuration retrieval, and content delivery.

This interpretation aligns with the methodological perspective emphasized by [21], which highlights the importance of jointly considering statistical and practical significance when translating quantitative performance results into actionable engineering decisions. In performance-sensitive systems such as API services, effect sizes and median-based differences provide a more operationally meaningful indicator of system impact than p-values alone.

### *G. Limitations and Threats to Validity*

To ensure a balanced and objective interpretation of the findings, several limitations and potential threats to validity must be acknowledged.

This study focuses on three primary performance metrics—Latency, Time to First Byte (TTFB), and Throughput—and does not explicitly measure resource utilization such as CPU or memory consumption. Consequently, references to "QUIC processing overhead" are inferential, derived from observable time-based performance patterns rather than direct quantification of computational cost.

The experimental results are specific to the quic-go implementation of HTTP/3 and the Go standard net/http2 library. Differences in implementation maturity, optimization strategies, or programming language ecosystems (e.g., Cloudflare quiche or ngtcp2) may yield different performance characteristics in other environments [21][22]. In addition, all experiments were conducted on a single Google Cloud Platform virtual machine. Cloud-specific factors such as virtualized network interfaces, CPU scheduling, and region-dependent latency may influence transport-layer behavior and absolute performance values [7][18], limiting direct generalization to other deployment contexts.

The workload generation relied on a synthetic microbenchmark designed to approximate typical RESTful API behavior. While this approach enables controlled and reproducible experimentation, it does not fully capture real-world traffic dynamics such as bursty request patterns, mixed endpoint workloads, or diurnal load variations.

From a statistical perspective, the analysis involved multiple pairwise comparisons using the Mann–Whitney U test across scenarios. The absence of explicit correction for multiple comparisons slightly increases the risk of Type I error, while the limited number of replications (n = 3 per scenario) reduces statistical power for detecting small effects.

Network degradation was emulated using deterministic traffic control (tc) parameters. Real-world network impairments are often dynamic, asymmetric, and bursty, which may produce different performance outcomes. Furthermore, the experiments intentionally employed QUIC 1-RTT without 0-RTT session resumption to ensure reproducibility under controlled conditions; therefore, the reported TTFB improvements should be interpreted as conservative estimates of HTTP/3 performance.

Although this study frequently discusses QUIC processing overhead when interpreting performance trade-offs, such overhead is not directly measured. Instead, the interpretation is aligned with prior literature that reports increased CPU utilization and userspace packet processing costs in QUIC implementations under high-throughput conditions [23][24]. Direct measurement of resource utilization and protocol stack profiling is thus positioned as a validation step for future research rather than a missing requirement of the present analysis. Finally, while tail latency and resource-level metrics are important for production systems, this study intentionally emphasizes median-based latency and throughput to isolate protocol-level behavior, consistent with prior controlled transport-layer evaluations [7][16].

## III. RESULT AND DISCUSSION

### *A. Collecting Data*

Data collection was performed using a $2 \times 2 \times 2 \times 2 \times 3$ factorial experimental design. The Latency, TTFB, and Throughput metrics were recorded from three replications for each scenario, and the resulting averages were then visualized as follows:

1) Latency and Time-to-First-Byte (TTFB)

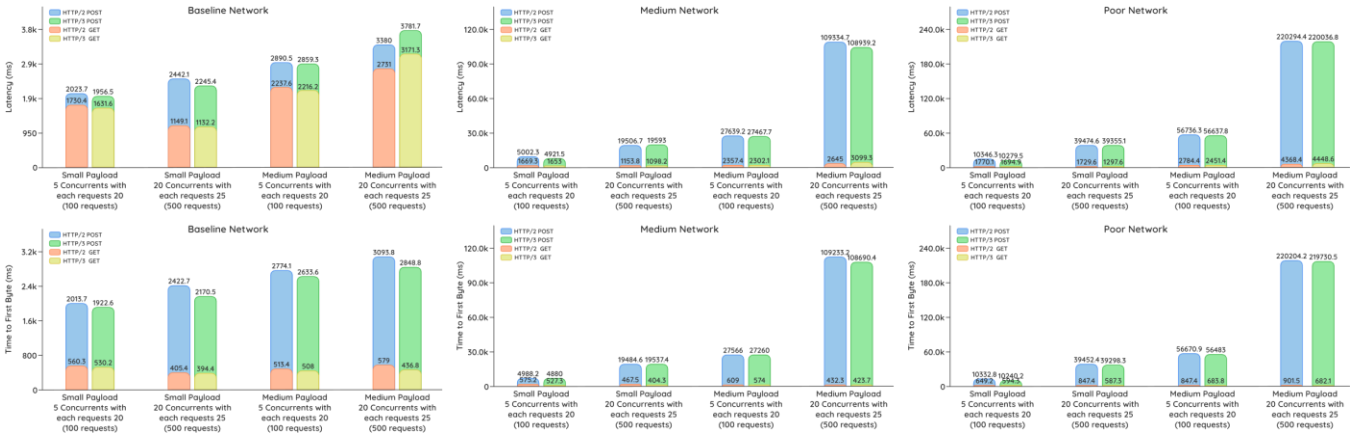


Figure 5. Latency and Time-to-First-Byte Comparison Across Network Conditions

Figure 5 presents a comparative analysis of Latency and Time to First Byte (TTFB) between HTTP/2 and HTTP/3 across Baseline, Medium, and Poor Network conditions. Under baseline conditions, HTTP/3 achieves lower latency for small payloads under high concurrency, for example recording 2245.4 ms compared to 2442.1 ms for HTTP/2. However, for medium payloads under high concurrency, HTTP/2 demonstrates lower latency, reaching 3380 ms for POST operations compared to 3781.7 ms for HTTP/3.

Across all Baseline scenarios, HTTP/3 consistently records lower TTFB values. For medium payload GET under high load, HTTP/3 achieves a TTFB of 2848.8 ms compared to 3093.8 ms for HTTP/2.

In the Medium Network condition (30 ms latency delay, 20 Mbps bandwidth, and 0.5% packet loss), latency differences become more workload-dependent. For small payload GET under high concurrency, HTTP/3 records a latency of 1098.2 ms, compared to 1153.8 ms for HTTP/2. Conversely, for medium payload GET under high load, HTTP/2 records a lower latency of 2645 ms compared to 3099.3 ms for HTTP/3.

Despite this variability, HTTP/3 maintains consistently lower TTFB values across nearly all scenarios.

Under Poor Network condition (50 ms latency delay, 10 Mbps bandwidth, and 2% packet loss), HTTP/3 consistently demonstrates lower latency and TTFB across most scenarios. For small payload GET under high concurrency, HTTP/3 records a latency of 1297.6 ms compared to 1729.6 ms for HTTP/2. Minor reversals are observed in isolated scenarios; however, HTTP/3 maintains consistently lower TTFB values across all tested conditions.

2) Throughput

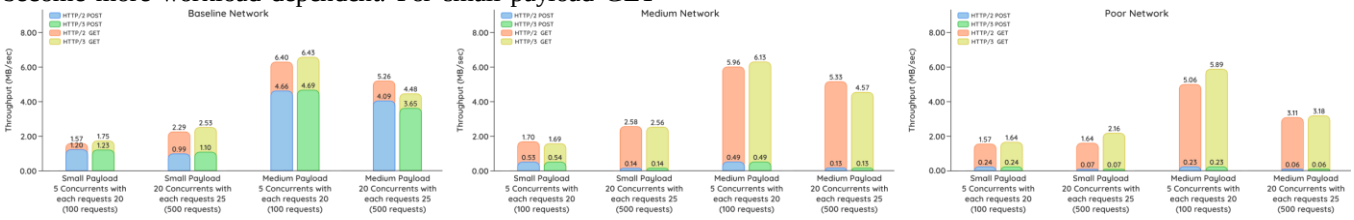


Figure 6. Per-request Throughput Under Increasing Concurrency Levels

Figure 6 illustrates the per-request throughput (MB/sec) of HTTP/2 and HTTP/3 under varying payload sizes, concurrency levels, and network conditions. Under Baseline Network conditions, throughput differences between the two protocols are workload-dependent. For small payload GET operations, HTTP/3 achieves slightly higher throughput under high concurrency (2.53 MB/sec vs. 2.29 MB/sec). However, for medium payloads combined with high concurrency, HTTP/2 consistently outperforms HTTP/3. In this scenario, HTTP/2 reaches 5.26 MB/sec for GET and 4.09 MB/sec for POST, while HTTP/3 records lower values of 4.48 MB/sec and 3.65 MB/sec, respectively.

Under the Medium Network condition, per-request throughput is largely comparable for upload (POST) operations across both protocols, with values converging around 0.13–0.54 MB/sec depending on payload size and concurrency. For download (GET) operations, HTTP/3 maintains similar throughput for small payloads but exhibits a noticeable decline for medium payloads under high concurrency, where HTTP/2 achieves 5.33 MB/sec compared to HTTP/3's 4.57 MB/sec.

In contrast, under the Poor Network condition, HTTP/3 demonstrates a clear throughput advantage in most GET scenarios. The most pronounced difference is observed for small payload GET operations under high concurrency, where HTTP/3 achieves 2.16 MB/sec compared to HTTP/2's 1.64 MB/sec. For medium payload GET requests under low concurrency, HTTP/3 also achieves higher throughput (5.89

MB/sec vs. 5.06 MB/sec). For POST operations, throughput remains uniformly low for both protocols due to bandwidth and packet loss constraints, with negligible differences across scenarios.

Although POST operations exhibit similarly low throughput for both protocols due to upload-side constraints, HTTP/3 consistently maintains a slight edge. From a production standpoint, these findings indicate that HTTP/3 significantly improves effective per-request throughput in unstable networks, making it more suitable for mobile and wireless environments where packet loss is common.

### B. *Consistency Testing*

The CV test results demonstrated that the majority of metrics (over 90% of scenarios) for both POST and GET operations across all network conditions (Baseline, Medium, Poor) fell into the Highly Consistent category (CV<10%). Some GET scenarios in the Medium and Poor Networks exhibited CV values between 10% and 20% (Categorized as Consistent or Moderately Consistent), which was attributed to the fluctuations of the simulated network. However, since all metrics met the acceptable threshold for consistency (CV<20%) the data were deemed valid and reliable to proceed with the main statistical analysis.

### C. *Statistical Analysis for Hypothesis Testing*

#### 1) Mann-Whitney U Test and Effect Size

The Mann-Whitney U Test confirmed the statistical significance of the performance differences, while the Effect Size ($\in_{rb}$) quantified the practical magnitude of these differences, revealing crucial insights into the protocol trade-offs:

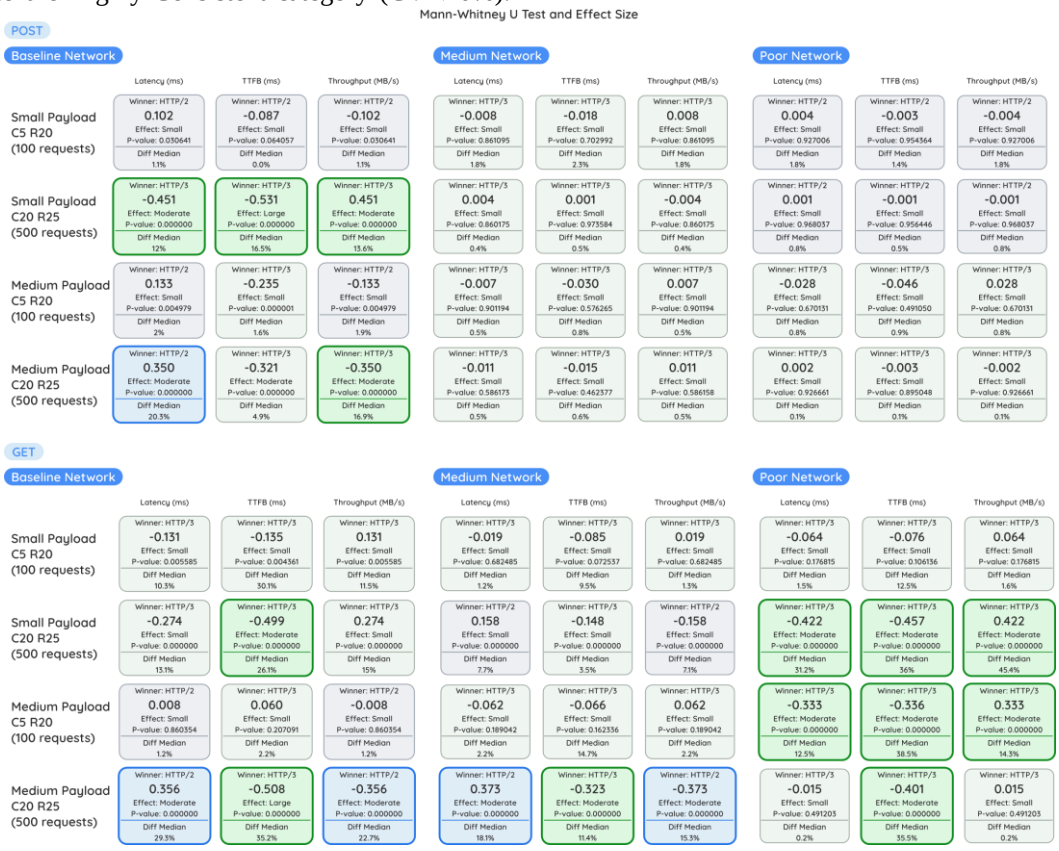


Figure 7. Statistical Results (Mann-Whitney U Test and Effect Size) for all Metrics Scenarios

The most impactful finding is the substantial Latency reduction achieved by HTTP/3 under Poor Network conditions, particularly for GET operations under high concurrency. In this scenario, HTTP/3 achieved a median Latency of 1028.6 ms compared to 1495.4 ms for HTTP/2, corresponding to a 31.2% reduction and a Moderate Effect Size ($\in_{rb} = -0.422$). This result is consistent with the ability of QUIC to mitigate the impact of packet loss through independent stream handling, especially under the simulated conditions of 2.0% packet loss and 50 ms additional delay. A similar behavior is observed for medium payloads under low concurrency in the Poor Network, where HTTP/3 reduced Latency by 12.5% ($\in_{rb} = -0.333$, Moderate). In contrast, under Baseline Network conditions, HTTP/2 consistently outperformed HTTP/3 when handling medium payloads with high concurrency. For GET operations, HTTP/2 achieved a median Latency of 2472.8 ms compared to 3197.8 ms for HTTP/3, representing a 29.3% advantage ($\in_{rb} = 0.356$, Moderate). A similar trend was observed for POST operations, with a 20.3% advantage for HTTP/2 ($\in_{rb} =$

0.350, Moderate). These results indicate that in the absence of packet loss and Head-of-Line Blocking (HoLB), protocol-level trade-offs may limit HTTP/3's scalability under high-volume concurrent workloads. Nevertheless, for small payloads in the Baseline Network, HTTP/3 still demonstrates statistically significant Latency reductions of approximately 10.3–13.1%, suggesting benefits during connection establishment phases.

The analysis of the Time to First Byte (TTFB) metric reveals a highly consistent advantage for HTTP/3 across nearly all tested scenarios. TTFB directly reflects connection establishment efficiency and early response delivery. The most pronounced gains were observed in GET operations under high concurrency. In the Baseline Network, HTTP/3 achieved a 35.2% lower TTFB for medium payloads (279.6 ms vs. 431.8 ms; $\in_{rb} = -0.508$, Large Effect Size) and a 26.1% reduction for small payloads ($\in_{rb} = -0.499$, Moderate). Under Poor Network conditions, HTTP/3 maintained this advantage, achieving TTFB reductions of 36.0% for small payloads ($\in_{rb} = -0.457$, Moderate) and 35.5% for medium payloads ($\in_{rb} = -0.401$, Moderate). Unlike Total Latency, TTFB appears less affected by the protocol-level trade-offs observed in high-concurrency scenarios, as it primarily captures the initial connection and response pipeline. These observations highlight the architectural advantage of HTTP/3 in minimizing initial response delay across varying network qualities.

The comparative analysis of Throughput (MB/s) reveals a polarized performance pattern that closely mirrors the Latency results. Network condition emerges as the dominant factor influencing protocol efficiency. Under Poor Network conditions, HTTP/3 demonstrates a clear and practically significant advantage. For GET operations with small payloads under high concurrency, HTTP/3 achieved a median throughput of 2.32 MB/s compared to 1.60 MB/s for HTTP/2, representing a 45.4% improvement and a Moderate Effect Size ($\in_{rb} = 0.422$). A similar benefit is observed for medium payloads under low load, with a 14.3% improvement ($\in_{rb} = 0.333$, Moderate). These findings indicate that HTTP/3's resilience under packet loss directly translates into higher effective data transfer efficiency per request.

Conversely, under Baseline Network conditions with medium payloads and high concurrency, HTTP/2 consistently achieves higher throughput. In this scenario, HTTP/2 outperformed HTTP/3 by 22.7% for GET operations ($\in_{rb} = -0.356$, Moderate) and 16.9% for POST operations ($\in_{rb} = -0.350$, Moderate). This suggests that when HoLB is not a limiting factor, HTTP/2 remains more efficient at sustaining high per-request throughput under stable, high-capacity network conditions.

2) Kruskal-Wallis H Test

To complement the comparative analysis and determine overall statistical significance, the Kruskal-Wallis H Test was applied. This test was used to analyze the aggregate impact of independent factors with more than two variants, namely Network Condition, Payload Size, and Concurrency Level, on the performance metrics of Latency, TTFB, and Throughput. The results of this test confirmed which variables most significantly influenced general REST API performance, irrespective of the specific comparison between HTTP/3 and HTTP/2.

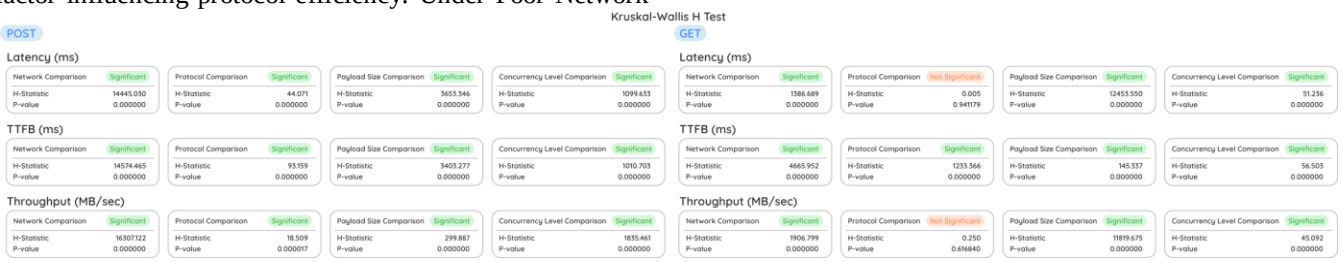


Figure 8. Kruskal-Wallis H Test Results for Group Comparison

The results confirmed that the majority of factors had a highly significant influence on performance, with the Null Hypothesis (H0) rejected in almost all tests (p-values approaching 0.000000). The Network Comparison and Payload Size Comparison factors were proven to be the most dominant variables significantly influencing all performance metrics for both POST and GET operations. The Protocol Comparison (HTTP/3 vs. HTTP/2) showed more complex findings: for the POST operation, HTTP/3 had a significantly lower (better) compared to HTTP/2, but conversely, for the GET operation, the total Latency and Throughput differences between the protocols were not statistically significant (H0 accepted). Crucially, for the GET operation, HTTP/3 still demonstrated significantly better performance in terms of TTFB (Time to First Byte). Further analysis indicated that scenarios with small payloads and a low concurrency level yielded the best overall time performance, while the use of medium payloads in the GET operation resulted in higher per-request Throughput compared to small payloads.

### *D. Performance Trade-offs*

The degradation of HTTP/3 throughput under ideal network conditions is consistent with prior reports of increased computational overhead in QUIC implementations [24][25]. As this study does not directly measure CPU or memory utilization, this interpretation remains observational and is inferred solely from time-based performance metrics rather than explicit resource profiling. These processing trade-offs should therefore be interpreted as implementation-

dependent costs associated with enhanced transport-layer resilience, rather than as a fundamental limitation of HTTP/3.

Despite its consistently superior Time to First Byte (TTFB), HTTP/3 exhibits reduced per-request throughput under stable, high-capacity network conditions. Similar trade-offs have been reported in studies using alternative QUIC stacks and implementations, suggesting that such behavior is not unique to the Go-based quic-go environment [8][18] [21][23].

Although this study employs a comprehensive factorial design ($2 \times 2 \times 2 \times 2 \times 3$), the analysis primarily focuses on main effects and scenario-level comparisons rather than explicit higher-order interaction modeling. Nevertheless, the observed performance reversals under high concurrency and medium payload sizes—particularly in stable network conditions—indicate a meaningful interaction between workload intensity, payload size, and network quality. Prior QUIC performance studies similarly report that increasing concurrency can amplify processing overhead under stable networks, while degraded conditions tend to shift the performance advantage toward QUIC-based protocols [21][23].

Overall, the throughput results highlight a clear resilience–efficiency trade-off between HTTP/2 and HTTP/3. Under degraded network conditions, HTTP/3 consistently sustains higher effective per-request throughput for GET operations, consistent with its ability to mitigate the impact of packet loss through independent stream handling. Conversely, under stable and low-loss environments, HTTP/2 demonstrates superior throughput efficiency for medium-sized payloads under high concurrency, where Head-of-Line Blocking is minimal and protocol-level processing costs become more influential. These findings indicate that protocol selection should be workload-aware rather than protocol-centric, depending on whether the API is latency-sensitive or throughput-oriented. While HTTP/3 is advantageous for APIs where early response delivery dominates user experience and SLA compliance, HTTP/2 remains preferable for throughput-oriented services operating under stable, high-capacity network conditions.

### *E. Conceptual Interpretation of TTFB Contributions*

Time to First Byte (TTFB) represents the elapsed time from request initiation until the first byte of the response is received, encompassing connection establishment, security handshake, and initial request processing [22]. Across nearly all evaluated scenarios, HTTP/3 consistently achieves lower TTFB than HTTP/2; however, this advantage should be interpreted in the context of the protocol features enabled during the experiment.

These consistent TTFB gains highlight the architectural advantage of QUIC in the initial response phase, particularly through its integrated TLS 1.3 handshake and transport-layer stream multiplexing. Unlike total latency, TTFB reflects only the early connection and request initiation pipeline. Therefore, the observed improvements indicate transport-level efficiency rather than immunity to protocol-level processing trade-offs.

In the HTTP/2 configuration, all requests reuse an established TCP connection secured with TLS 1.3. While connection reuse reduces repeated handshake overhead, HTTP/2 remains constrained by TCP's in-order delivery semantics. Under concurrent workloads or degraded network conditions, packet loss at the TCP layer can delay the delivery of response headers across all active streams, increasing TTFB.

By contrast, HTTP/3 operates over QUIC using a 1-RTT handshake with integrated TLS 1.3. The observed TTFB improvements are consistent with QUIC's ability to allow independent stream progression even in the presence of packet loss, mitigating blocking during the initial response phase and enabling faster delivery of the first response byte.

It is important to note that 0-RTT connection resumption was intentionally not enabled in this study to ensure controlled and reproducible experimental conditions. Consequently, the reported TTFB improvements should be interpreted as conservative estimates of HTTP/3's transport-level efficiency. In real-world deployments with stable session resumption and 0-RTT support, the initial response latency advantage of HTTP/3 may be further amplified.

### *F. GET and POST operations*

From a semantic perspective, the observed performance differences between GET and POST operations are consistent with their inherent REST characteristics. GET operations are idempotent and predominantly server-driven, where performance is strongly influenced by server-side response delivery and stream scheduling. Consequently, HTTP/3's per-stream loss recovery provides substantial benefits for GET requests under degraded network conditions.

In contrast, POST operations primarily involve client-to-server data transmission, making throughput more sensitive to payload size and concurrency rather than connection initiation efficiency. This explains why HTTP/3's TTFB advantage does not always translate into superior throughput for POST requests, particularly under stable network conditions where Head-of-Line Blocking is not a dominant factor.

### *G. Interaction-Oriented Interpretation of Factorial Results*

Although this study employs a comprehensive factorial design ($2 \times 2 \times 2 \times 2 \times 3$), the analysis intentionally focuses on main effects and scenario-level comparisons rather than explicit higher-order interaction modeling. Nevertheless, the empirical results reveal clear interaction patterns between workload characteristics and network conditions, indicating that protocol performance is not governed by a single dominant factor.

A notable interaction emerges between concurrency level and network condition. Under Poor Network conditions, increasing concurrency consistently amplifies the performance advantage of HTTP/3, particularly for GET

operations, where QUIC's stream-level loss recovery sustains lower latency and higher robustness as request parallelism increases. In contrast, under Baseline Network conditions, the same increase in concurrency produces a performance reversal for medium payload workloads, with HTTP/2 achieving lower latency and higher throughput. This suggests that when packet loss and network instability are minimal, the benefits of HoLB mitigation diminish, and protocol efficiency becomes more sensitive to processing overhead and data transfer efficiency.

These observations indicate a strong interaction between concurrency intensity, payload size, and network quality, rather than uniform protocol superiority across scenarios. Consequently, protocol selection for RESTful API systems should be informed by the combined influence of workload and network characteristics rather than isolated performance metrics.

### *H. Comparison with Related Work*

The quantitative findings of this study reveal a significant performance polarity between HTTP/3 and HTTP/2 that is highly dependent on network and workload conditions, establishing a clear trade-off between resilience and efficiency consistent with prior literature. The dominant performance gain for HTTP/3 observed in Poor Network conditions (up to 31.2% lower Latency) reinforces the findings of studies focused on lossy environments, such as that by [2], which demonstrated the effectiveness of QUIC's non-blocking stream model in overcoming HoLB, a vulnerability inherent to HTTP/2's single TCP connection. This consistency is further supported by [7], who concluded that HTTP/3 provides sizable performance benefits only in scenarios with high latency or poor bandwidth.

Similar observations have also been reported by [24] and [25], who highlight that QUIC implementations may experience CPU-bound limitations under high-throughput and low-loss network conditions.

In terms of connection efficiency, the finding that HTTP/3 consistently reduced Time to First Byte (TTFB) by an average of 35.5% across nearly all scenarios validates the architectural superiority of QUIC. This aligns with [6] and [8], who noted that HTTP/3 excels in the pre-transfer phase, primarily because of the native integration of TLS 1.3 within the QUIC handshake.

Conversely, the observation that HTTP/2 performed 29.3% faster in Latency and 22.7% higher in Throughput for medium payloads under high concurrency in the Baseline Network points to trade-offs that are consistent with prior studies reporting higher computational overhead in QUIC-based implementations [8][9], although such overhead is inferred rather than directly measured in this study.

Finally, this research specifically analyzed a comprehensive factorial design on RESTful API transactions using both POST and GET methods, providing targeted performance validation for the quic-go implementation. This fills a critical gap left by previous works that often focused solely on general web server performance, web page loading metrics, client-side implementation stability, or proxy environments, thus providing a targeted comparison necessary for modern microservices architecture.

### *I. Practical Implications for API Design*

Adopting HTTP/3 is strongly recommended for user-facing API services operating over unstable networks characterized by high latency and packet loss, such as mobile, wireless, or geographically distributed environments. Across degraded network conditions, HTTP/3 consistently demonstrates lower latency, significantly faster Time to First Byte (TTFB), and higher effective per-request throughput, making it particularly suitable for latency-sensitive endpoints such as authentication, configuration retrieval, and control-plane APIs where early response delivery directly impacts perceived user experience and SLA compliance. Conversely, HTTP/2 remains more efficient for backend-to-backend communication and intra–data center workloads under stable network conditions, especially for medium-sized payloads under high concurrency, where minimizing sustained transfer latency and maximizing throughput are more critical than mitigating Head-of-Line Blocking. These findings indicate that protocol selection should be workload-aware rather than protocol-centric. A pragmatic deployment strategy is to support both HTTP/2 and HTTP/3 using protocol negotiation, allowing clients to dynamically select the most appropriate protocol based on prevailing network conditions while accounting for potential implementation-level processing costs associated with QUIC under highly stable, high-throughput scenarios.

## IV. CONCLUSION

This research provided an empirical comparison of HTTP/3 (quic-go) and HTTP/2 (Go standard library) based RESTful API performance across various network and load conditions. The study accepted its hypothesis (H1), confirming that HTTP/3 offers significant performance benefits, particularly under degraded network conditions; however, these benefits were highly context-dependent.

Under severely degraded network conditions (Poor Network), HTTP/3 demonstrated a statistically and practically significant advantage. This superiority was most pronounced for high-load GET operations, achieving 31.2% lower Latency and 45.4% higher Throughput than HTTP/2, validating QUIC's effectiveness in mitigating Head-of-Line Blocking (HoLB). Furthermore, HTTP/3 consistently showed a dominant advantage in Time to First Byte (TTFB), being an average of 35.5% faster across almost all scenarios.

Conversely, under ideal, high-capacity conditions (Baseline Network), HTTP/2 proved superior for medium-payload transfers under high concurrency, recording 29.3% lower Latency and 22.7% higher Throughput. This reversal suggests that under ideal network conditions, HTTP/3 may face efficiency trade-offs that are consistent with previously reported QUIC processing costs at the implementation level,

although no direct resource utilization measurements were performed in this study.

The study concludes that HTTP/3 is strongly recommended for API services targeting unstable networks (high latency and packet loss), such as wireless or mobile connections, while HTTP/2 remains the more efficient choice for large-capacity transfers in ideal data center environments. For future research, it is advised to explicitly measure and quantify the QUIC Processing Overhead using server-side metrics (like CPU and memory usage) and compare these findings against other QUIC implementations (e.g., Cloudflare quiche).

It is important to emphasize that these recommendations are highly dependent on the evaluated implementation, workload characteristics, and network configuration. Different QUIC implementations, deployment environments, or real-world traffic patterns may yield different performance outcomes.

## REFERENCES


[1] J. Pendon and T. Warishe, "The Effects of HTTP Advances on Modern Page Loading," *Sci. Prepr.*, no. December, 2023, doi: 10.14293/PR2199.000570.v1.

[2] F. Liu, J. Dehart, J. Parwatikar, B. Farkiani, and P. Crowley, *Performance Comparison of HTTP/3 and HTTP/2: Proxy vs. Non-Proxy Environments*, vol. 1, no. 1. Association for Computing Machinery, 2024. [Online]. Available: http://arxiv.org/abs/2409.16267

[3] U. N. Hadianto, J. G. A. Ginting, D. Pranindito, and E. F. Cahyadi, "Throughput Assessment of HTTP/3 (QUIC)-Based Protocol on OpenLiteSpeed Web Server," pp. 179–184, 2025, doi: 10.1109/ies67184.2025.11160983.

[4] J. Koch, O. Falowo, and N. Elrod, "What We Know About HTTP/3 and Its Implementation: A Literature Review," 2024, doi: 10.1109/icmi60790.2024.10585883.

[5] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," 2020, [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-http

[6] P. Tripathi, M. H. Miraz, and S. Joshi, "Comparing Communication Efficiency: HTTP/3 versus HTTP/1.1 Latency in RESTful APIs," *Proc. - 2023 Int. Conf. Comput. Networking, Telecommun. Eng. Sci. Appl. CoNTESA 2023*, pp. 27–31, 2023, doi: 10.1109/CoNTESA61248.2023.10384951.

[7] M. Trevisan, D. Giordano, I. Drago, and A. S. Khatouni, "Measuring HTTP/3: Adoption and performance," *2021 19th Mediterr. Commun. Comput. Netw. Conf. MedComNet 2021*, pp. 1–8, 2021, doi: 10.1109/MEDCOMNET52149.2021.9501274.

[8] J. Dubec, J. Balazia, and P. Cicak, "Performance evaluation of the HTTP/3 client implementations," *2023 46th Int. Conf. Telecommun. Signal Process. TSP 2023*, pp. 260–263, 2023, doi: 10.1109/TSP59544.2023.10197834.

[9] X. Zak, J. Machaj, and L. Sevcik, "A Comparative Analysis of HTTP/2 and HTTP/3 Web Server Performance," *15th Int. Conf. Elektro 2024, ELEKTRO 2024 - Proc.*, pp. 1–6, 2024, doi: 10.1109/ELEKTRO60337.2024.10556831.

[10] M. Seemann, "Adding a QUIC API for Go's standard library TLS package," 2023, *libp2p Blog & News*. [Online]. Available: https://blog.libp2p.io/2023-09-13-quic-crypto-tls/

[11] Thierry Beigbeder, "Developing and deploying an HTTP/3 API in Go," 2025, [Online]. Available: https://blog.otvl.org/blog/http3-api-go-writing/

[12] T. Bressoud and D. White, "RESTful Application Programming Interfaces," Springer, Cham, 2020, pp. 715–755. doi: 10.1007/978-3-030-54371-6_23.

[13] S. Kemp, "Digital 2024: 5 Billion Social Media Users," *We Are Soc.*, 2024, [Online]. Available: https://wearesocial.com/us/blog/2024/01/digital-2024-5-billion-social-media-users/

[14] V. Thatikonda, "Building Scalable and Performant Apis: Best Practices and Patterns," *J. Pharm. Res. Reports*, pp. 1–3, Jan. 2022, doi: 10.47363/JPRSR/2022(3)146.

[15] P. Mesnard, "Data and Media Elements," Apress eBooks, 2022, pp. 79–98. doi: 10.1007/978-1-4842-8983-9_4.

[16] D. Saif, C. H. Lung, and A. Matrawy, "An Early Benchmark of Quality of Experience between HTTP/2 and HTTP/3 using Lighthouse," *IEEE Int. Conf. Commun.*, 2021, doi: 10.1109/ICC42927.2021.9500258.

[17] D. Phanekham and S. Nair, "Accurate Bulk Throughput Benchmarks Using Reduced Network Resources," *IEEE/IFIP Netw. Oper. Manag. Symp.*, pp. 1–5, 2023, doi: 10.1109/NOMS56928.2023.10154370.

[18] G. Perna, M. Trevisan, D. Giordano, and I. Drago, "A first look at HTTP/3 adoption and performance," *Comput. Commun.*, vol. 187, pp. 115–124, 2022, doi: https://doi.org/10.1016/j.comcom.2022.02.005.

[19] Robert Bourgault, "Understanding the Power of Coefficient of Variation in Software Performance Testing," 2023, [Online]. Available: https://dzone.com/articles/understanding-the-power-of-coefficient-of-variatio

[20] S. Sawilowsky, J. Sawilowsky, and R. Grissom, "Effect Size, The," 2025, pp. 791–796. doi: 10.1007/978-3-662-69359-9_689.

[21] H. Dźwigoł and M. Dźwigoł–Barosz, "Exploring the methodological framework and importance of statistical analysis in quantitative management research," *Zesz. Nauk.*, vol. 2025, no. 222, pp. 211–225, 2025, doi: 10.29119/1641-3466.2025.222.12.

[22] M. Zhang, Z. Yan, K. Dong, and H. Li, "Research on HTTP/3: Performance, Characteristics, Issues, and Improvement Mechanisms," pp. 399–404, 2025, doi: 10.1109/icetci64844.2025.11083994.

[23] R. Bonsu, K. O. Peprah, and W. Asiedu, "Analyzing the Impact of HTTP/3 on Web Application Performance," *J. Inf. Technol. Sci.*, vol. 10, no. 3, pp. 40–49, 2024, doi: 10.46610/joits.2024.v10i03.005.

[24] X. Zhang, A. Hassan, and Z. M. Mao, "QUIC is not Quick Enough over Fast Internet", doi: 10.1145/3589334.3645323.

[25] P. Bi, *LiteQUIC : Improving QoE of Video Streams by Reducing CPU Overhead of QUIC*, vol. 1, no. 1. Association for Computing Machinery. doi: 10.1145/3664647.3681670.