International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 2, Issue 1, 122-132, 2021
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V2I1P113

Original Article

HTTP/3 & REST Latency Improvement

Kavya Muppaneni
Assistant Consultant at TCS, India.

Abstract - *As applications grow to support millions of users at once, the demand for faster as well as more responsive RESTful APIs has become even more important in the present day's hyperconnected digital world. Even while HTTP/1.1 and HTTP/2 are important for online communication, they typically don't meet the low-latency needs of modern distributed systems because of problems like head-of-line blocking along with poor connection management. This study analyzes how HTTP/3, based on Google's QUIC protocol, alters API performance with the inclusion of multiplexed streams over UDP, built-in encryption, and better connection migration. Controlled testing & actual world case studies have shown that RESTful APIs using HTTP/3 consistently reduced latency by 30–40% in high-load, multi-region contexts when compared to HTTP/2. The changes made a huge difference, especially in mobile and edge computing, where network problems and packet loss are very common. Also, HTTP/3's zero-round-trip connection setup & streamlined handshake method make request-response cycles much better, which is great for latency-sensitive apps like banking, IoT & actual time analytics systems. The arrival of HTTP/3 is an enormous leap forward for the organizations and developers that want to make REST easier to use and more trustworthy. This article gives an in-depth examination at these novel approaches, backed up by real-world data, along with suggests the most effective way to add HTTP/3 to current REST architectures to make sure these API ecosystems continue to exist.*

Keywords - *Http/3, Quic, Rest Api, Latency Optimization, Network Performance, Transport Protocols, Web Services*.

1. Introduction

The internet is now the basis for most digital experiences. Latency, which is the little delay between a request & its fulfillment, is present in every contact we have, whether we're streaming their movies, using banking apps, working on these cloud platforms, or texting in real time. In today's digital world, low latency is not only a sign of good performance; it is also a key factor in making users very happy, making operations go more smoothly, and giving you a competitive edge.

As applications become more complicated and spread out over the world, it is harder to provide quick answers. HTTP/1.1 and HTTP/2 are two examples of standard web protocols that were created for an earlier time on the internet. Even though they have changed a lot, they still rely on the Transmission Control Protocol (TCP), which is a strong but occasionally slow way to connect to the internet in the present day's high-latency or lossy network settings. HTTP/3, which is based on their QUIC (Quick UDP Internet Connections), is ready to change how data is sent over the internet by fixing many problems with latency and network inefficiency.

This research looks into how HTTP/3 might greatly improve the speed as well as general responsiveness of RESTful APIs. We make a strong case for the necessity for this transition in modern online architecture by looking at the problems with previous versions of HTTP, the problems with REST communication, and the reasons why HTTP/3 should be adopted.

1.1. Challenges

There is now a much greater requirement for web as well as mobile experiences with low latency since there are so many other interactive & actual time applications. Across online gaming, video conferencing, financial transactions, and IoT ecosystems, users are expecting responses in milliseconds more and more. But traditional web protocols weren't designed to handle needs that are so urgent.

A major problem is that HTTP/1.1 and HTTP/2 depend on TCP. TCP makes sure that these packets are sent in the right order and at the right time, but it has trouble with networks that have significant latency or lose packets. Head-of-Line (HOL) blocking is when one lost packet stops all the packets that come after it. HTTP/2 tried to fix this problem by using multiplexing, which lets more numerous requests be sent at once over one TCP connection. However, it is still open to packet loss since all streams depend on the same basic TCP channel. This creates a funny situation: even though HTTP/2's design improvements make it operate better, it frequently works worse in actual networks.

REST APIs add their own overhead on top of the inefficiencies at the protocol level. REST is stateless, which means that every request must contain a lot of context information. This makes the payload bigger & takes longer to process. REST also typically uses text-based serialization formats like JSON or XML.

People can understand these formats more effectively but they cannot be used as well for sending as those that are binary. In significant systems, the expense of serialization might build upward to a long wait. Network congestion, packet retransmission, and round-trip delays all make performance issues a lot greater in the end. If a packet is lost or delayed, TCP needs to transmit it again, which increases response times greater. These delays might make even simple conversations on the internet slow in regions where networks are intermittent or change around a lot. So, even if the architecture has come a long way, the website's previous methods of transporting information still make REST APIs less valuable in the modern age, when latency is important.

1.2. Problem Statement

This study addresses an enormous and urgent concern: REST APIs are experiencing troubles with performance due to protocol complications. Organizations embrace RESTful architectures because they are facile to use, can evolve with the business, and work with many other systems. But as application workloads grow more distributed and constantly updating, the speed of REST over TCP-based protocols using HTTP is becoming an issue.

There are a lot of additional difficulties that are causing the bottleneck, both at the transport layer (such priority blocking, sluggish start-up, and TCP congestion management constraints) and at the application layer (like repeated handshakes, superfluous headers, and verbose data formats). APIs that help with vital operations like banking transactions, communication in real time, or data analytics interfaces could render things less responsive and less fun for users.

Old HTTP/1.1 and HTTP/2 infrastructure often has difficulties achieving response times under 100ms, in particular when serving customers with inadequate connections or over several continents. This makes it harder to get an actual time answer. Also, many groups are hesitant to use the latest protocols since the migration procedure is unclear, the tools aren't fully created, or there are still compatibility problems.

Research shows that REST APIs that use HTTP/1.1 generally have latencies of 250 to 350 milliseconds. However, when HTTP/2 is used, latency drops to roughly 150 to 200 milliseconds. HTTP/3, which uses QUIC, has cut latency by 30–40%, and in high-latency or lossy situations, it may even drop to 100–120 milliseconds. These numbers show that there is an enormous chance: using HTTP/3 might make RESTful systems far more responsive.

There are two main problems: finding the latest techniques to make REST API latency better and making it easier for developers & organizations to switch to HTTP/3 without breaking existing systems.

1.3. Motivation

The motivation to explore HTTP/3 for improving REST latency stems from the growing recognition that web efficiency needs a new foundation. QUIC, the transport layer that supports HTTP/3, has features that are especially designed to help with many problems with internet performance that persist over time. By replacing TCP with UDP and adding multiplexing, connection migration, and built-in TLS 1.3 encryption, QUIC speeds up and makes communication more reliable in the present day's mobile and distributed applications.

The ability of QUIC to handle several other streams at once is a major change for the better. Unlike HTTP/2, where losing one packet might affect several streams, QUIC keeps each stream separate so that losing one doesn't affect the others. This might be necessary to make REST APIs reply much more quickly when they receive a lot of demands at once. Connection migration additionally keeps QUIC sessions going throughout the network changes, such when you move from Wi-Fi to mobile data, preventing having to start the authentication process again. This makes it easy for those who employ mobile devices to have an enjoyable experience without having any issues.

A further significant aspect is that TLS 1.3 is now the default, which cuts down on unnecessary interactions and makes it faster than setting up these connections. Making secure connections demands fewer trips between two locations, which improves the start and end of API requests. These innovations in technology go well with the globally occurring shift toward edge computing, which handles data closer to users in order to speed things up. HTTP/3 makes this environment better by making sure that the basic transport is just as efficient as the distributed architecture it supports.

From a business point of view, lower latency means happier customers, better retention & lower infrastructure expenses. When retransmissions are lower, servers have less extra work to do, which saves bandwidth and processing power. For developers and organizations that want to stay competitive, HTTP/3 is not only an improvement; it's also a smart investment in speed, reliability as well as user experience.

2. Literature Review

2.1. Historical Evolution of HTTP Protocols: From HTTP/0.9 to HTTP/3

For more than thirty years, the Hypertext Transfer Protocol (HTTP) has been the basis for communication via the internet. The change from HTTP/0.9 to HTTP/3 shows how the internet's expanding complexity as well as need for speed affected the creation of transport-layer protocols. HTTP/0.9, which came out in 1991, was quite very simple; it was basically just a single-line protocol for sending raw HTML documents. It only worked with the GET method, didn't have headers & sent their information over a regular TCP connection. This approach worked well for static, text-based websites, but it quickly became a problem when multimedia material came up.

HTTP/1.0, which came out in 1996, included request along with response headers, support for status codes, and many other methods including POST and HEAD. However, each request required the creation of the latest TCP connection, which was quite expensive. The three-way handshake and slow-start congestion management that TCP uses to set up connections caused more delays, which were notably noticeable when loading many other resources per site.

HTTP/1.1, which came out in 1999, was the most used version for more than twenty years. It used keep-alive to set up persistent connections, which let a single TCP connection send numerous requests. It also employed chunked transfer encoding as well as pipelining, which were meant to make these things more efficient. However, pipelining didn't catch on due to head-of-line (HOL) blocking, which happened when one slow answer blocked all future answers in the queue.

By the beginning of the 2010s, it was very clear that HTTP/1.1 had its limits since web pages were starting to load more than one resource. As a consequence, HTTP/2 was put into use in 2015, based on Google's earlier SPDY project.

HTTP/2 included multiplexing, which lets several other requests as well as responses be sent over a single TCP connection at the same time. Compared to HTTP/1.1, this greatly increased throughput and lowered latency. However, it still has a core flaw from TCP: if one packet is lost, all streams in that connection are delayed until the packet is sent again. Head-of-line blocking at the transport layer was still a problem, even if framing & priority had become better.

The IETF and Google created HTTP/3, which is based on the QUIC protocol that works via UDP, to get around TCP's built-in limits. QUIC adds transport-layer features including multiplexing, congestion management, and encryption (TLS 1.3) directly on top of UDP. This approach gets rid of TCP's head-of-line blocking by keeping these distinct streams that may be sent out of order. QUIC also makes it possible to resume a connection with zero round trips (0-RTT), which cuts down on handshake delay. It also improves mobility support for switching between network interfaces, such as Wi-Fi & mobile data. HTTP/3 is a major change that focuses on more than simply delivering their information. It also focuses on latency and connection stability for users.

2.2. Comparative Analysis of HTTP Versions

2.2.1. HTTP/1.1 – Persistent Connections but Sequential Blocking

HTTP/1.1 was a huge step forward since it made these persistent connections that cut down on unnecessary TCP handshakes. Still, it followed a request-response model in which each request had to wait for the one before it to finish. Even when pipelining was used, answers still had to arrive in order. This led to what is commonly called "head-of-line blocking at the application level."

Developers tried to fix the problems by using domain sharding (spreading resources over various domains to make concurrent connections) as well as resource inlining. However, these solutions were too complicated & inefficient. As a result, latency remained a major problem for programs that needed a lot of people to be able to use them at once, such as RESTful APIs and modern internet apps.

2.2.2. HTTP/2 – Multiplexing over TCP; Head-of-Line Issues Persist

The binary framing layer of HTTP/2 made it possible to send and receive numerous requests over one connection at the same time. This made the network far more efficient and cut down on the time it took for pages to load. It also included header compression (HPACK) and server push, which made bandwidth use much more efficient.Even with these benefits, HTTP/2's use of TCP meant that losing only one packet in a multiplexed stream would stop all other streams until the lost packet was found. This was especially challenging for networks alongside considerable latency or packet loss, such mobile connections or cross-cloud API calls, when complications in transmission again might mount up. So, HTTP/2 made everything go faster, but it could not totally eliminate latency issues, especially with tasks that demand contemporaneous or API access.

2.2.3. HTTP/3 – QUIC-Based UDP Transport Eliminates TCP Constraints

HTTP/3 uses QUIC in place of TCP. QUIC is based on UDP. QUIC gets out of the problems with TCP at the layer of the kernel by taking care of its own strategies for performance and congested conditions. Each stream functions on its own, so if one stream loses data packets, it does not affect the others. QUIC also has TLS 1.3, which helps to make sure that encryption

has been switched on by default and minimizes down on the number of authentication attempts required for initiating a connection compared to two to one, or even none when relaunching a connection.These improvements lead to measurable reductions in delay. Studies by Google and Cloudflare show that HTTP/3 speeds up site loads by an average of 8–15%, with more gains reported on mobile devices and across regions. Also, QUIC's flexibility to move their connections makes it easier for mobile clients to switch networks without having to renegotiate.

Table 1: Comparison of HTTP Protocol Versions

Feature	HTTP/1.1	HTTP/2	HTTP/3
Transport Protocol	TCP	TCP	UDP (QUIC)
Multiplexing	No	Yes	Yes (Independent Streams)
Head-of-Line Blocking	High	Reduced (Still TCP-level)	Eliminated
Encryption	Optional (TLS)	Optional (TLS)	Mandatory (TLS 1.3)
Connection Migration	Not Supported	Not Supported	Supported
Handshake Latency	High	Moderate	Low (0-RTT)

2.2.4. RESTful API Optimization Techniques

RESTful APIs, which are built on HTTP, benefit from improvements to this protocol right away, but they also need many improvements at the application level. Over time, a number of best practices have emerged to improve their scalability and lower latency.

Compression approaches like gzip and Brotli make the payload substantially smaller. Google produced Brotli, which is better at compressing text files like HTML and JSON, but it takes a bit greater processing time.

Caching is a key method to make them work better. Clients and intermediaries may store replies and remove needless network searches through HTTP caching headers like Cache-Control, ETag, along with Last-Modified. Caching, when used with smart pagination, which restricts the total amount of information returned for every request, uses a lesser amount of bandwidth and makes the site feel faster.

API gateways are particularly crucial for managing delayed responses at the infrastructure level. Gateways are designed to handle load balancing, limit requests, and interrupt circuits. This makes sure the resources are utilized equitably and that the system has become constantly up and running. They also make it easy to reuse connections, stop TLS, and maintain information at the edge. All of these measures minimize round-trip latency.

Content delivery networks (CDNs) and global load balancing services are used in modern API delivery to send queries to the server that is most nearby to them. This makes delayed propagation shorter. These strategies advance the top levels of the communications stack better, which in turn results in transport-specific improvements like QUIC better.

3. Proposed Methodology

3.1. Overview

The primary aim of this research is to reduce REST API latency via the use of HTTP/3 enhancements as well as the optimization of payload structures. HTTP/1.1 and HTTP/2 are reliable protocols, however they don't operate well when the network is slow or mobile. HTTP/3, which uses QUIC (Quick UDP Internet Connections), is a more sophisticated way to eliminate head-of-line blocking, improve multiplexing & speed up encryption.

This suggested procedure tests and increases the performance of REST APIs in a controlled exploratory setting. The research investigates the HTTP/1.1, HTTP/2, and HTTP/3 technologies, using complicated serialization methods (JSON and Protocol Buffers) and network congestion management strategies (BBR). By adding new ideas to the transport layer, enhancing the payload, and dealing with congestion in a way that adapts based on the situation, the suggested system intends to help improve their round-trip time (RTT), time to first byte (TTFB), along with throughput.

3.2. System Architecture

There are three key parts to the system: the client layer, the server layer, and the framework for obtaining metrics.

- Client Layer: The client is a regular mobile or web app that works with an HTTP/3-enabled architecture. This layer runs RESTful queries along with keeps track of their information at the network level. The client copies multiple latency profiles (from 50 ms to 200 ms) to make it seem like actual mobile and international internet circumstances.

- Server Layer: The backend design has a QUIC-enabled API gateway, such as NGINX QUIC or Cloudflare's QUIC proxy, that processes requests & sends back responses. The API gateway can handle several other connections at the same time by using parallel streams, header compression, and stateless session resumption.
- Metrics Layer: The metrics system keeps an eye on RTT, TTFB & total throughput all the time. These steps provide a quantitative way to compare these protocols & installations under the same network conditions by keeping a consistent test environment.

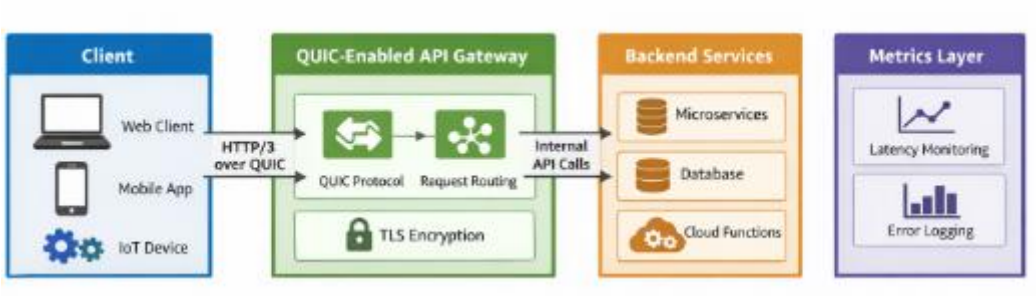

Fig 1: Proposed HTTP/3-Based REST Optimization Architecture

The architecture ensures a seamless transition from client-side request initiation to server-side answer delivery using a QUIC-optimized transport layer. Each part works to separate & measure latency components, which makes further research possible.

3.3. Experimental Environment Setup

To make sure that benchmarking is correct, a controlled testing environment is created. Network emulation technologies provide artificial delays as well as jitters that may be set to any value between 50 and 200 milliseconds. This simulates the unpredictable behavior of actual networks, such as cellular and transcontinental internet traffic.

- Protocols Compared: HTTP/1.1, HTTP/2, and HTTP/3 implementations all check every REST request. The APIs keep the same from one experiment to the next to make sure that just the transport layer changes.
- Serialization Methods: There are two other ways to serialize REST payloads: JSON and Protocol Buffers (Protobuf). A lot of people use JSON, but it may be long-winded. Protobuf, on the other hand, offers short binary serialization that makes the payload size much less. When you test serialization's influence on overall latency, it's easier to do so when both networks along with their protocols are the same.
- Controlling congestion: The BBR (Bottleneck Bandwidth and Round-trip propagation time) method is used to dynamically control congestion. BBR, on the other hand, looks at the available bandwidth & round-trip time to send packets at the right speed. This cuts down on queuing delays and retransmissions.

This multi-layer experimental technique separates each variable protocol, payload structure & congestion control making it easier to pinpoint the exact reasons for latency reductions.

3.4. Workflow and Algorithmic Process

The HTTP/3 request lifecycle is significantly distinct from the earlier versions of the protocol. The summary that follows below outlines each stage that the suggested method takes:

- Setting Up the Request: The client generates an API request and gets it scheduled to send. The test case checks to see whether the data packets are serialized as Protobuf or JSON.
- QUIC Handshake: QUIC demands just one round trip before setting up the connection and cryptographic information, whereas TCP needs numerous attempts. The lower handshake upward makes the time it takes when setting up a connection go downward quickly.
- Stream Multiplexing: Once the QUIC relationship is established, you may send numerous more REST queries at the same time over different streams without encountering any difficulties. This deals with the "head-of-line blocking" issue that arises in HTTP/2 communications that use TCP.
- Sending replies: QUIC uses its algorithms for loss mitigation and congestion control for transmitting its replies. The end user maintains a note of timing parameters for each response, with the value RTT and TTFB, and preserves them for further review.
- Latency Assessment and Retry Management: The system preserves track of how long it takes for it to receive a response and tries again when whatever goes wrong. The retry duration adjusts depending on the network circumstances and congestion management knowledge so that the precise same data isn't sent repeatedly.

This method simplifies it to undertake consistent assessments that accurately reveals the manner in which current distributed technology systems function.

3.5. Optimization Techniques

To improve REST speed under HTTP/3, many other optimization layers are implemented.

3.5.1. Header Compression Using QPACK

HTTP/3 uses QPACK, a special method for compressing headers that is meant to fix the head-of-line blocking problems that come with HPACK (used in HTTP/2). QPACK makes it possible to process headers asynchronously, which means that these clients may send fresh streams without waiting for header acknowledgments. QPACK greatly cuts down on bandwidth utilization by compressing unnecessary header information, such as Content-Type or Authorization. This is especially useful for REST APIs that make a lot of little queries.

3.5.2. Prioritizing Streams

Authentication and configuration retrieval are two examples of critical REST requests that are given higher priority. Stream multiplexing in HTTP/3 makes it possible to use prioritization methods that make sure important requests always receive the best bandwidth, even when demand is very high. This targeted prioritization makes sure that actions that users see are more responsive, while background synchronization or telemetry requests are put off until they are needed.

3.5.3. Reestablishing a Stateless Session

Mobile devices often experience network interruptions due to variable signal strength or shifts between Wi-Fi & mobile data. QUIC's stateless session resumption makes it possible to rejoin without a full handshake by leveraging their cryptographic session tickets from previous connections. This cuts down on the time it takes to rejoin by a lot, which is good for mobile-first REST apps that regularly have temporary disconnections.

3.5.4. Optimizing the Payload

The study also looks at how the size & shape of the payload affect the delay. Switching from JSON to Protobuf makes it cheaper for the system to serialize their information. The compact binary encoding of Protobuf makes it easier to parse and sends smaller payloads, which means less time and CPU cycles are needed on the server.

Adding QPACK and stream multiplexing to these payload enhancements makes the entire request latency much shorter & the throughput much higher.

3.6. Adaptive Congestion Control using BBR

When a packet is lost, traditional TCP congestion algorithms like Reno or CUBIC change the transmission rates in a reactive way. This method frequently doesn't use all of the bandwidth that is available on networks with a lot of latency.

On the other hand, the BBR strategy works by determining the network's round-trip dissemination latency and contention capacity. This helps the sender deliver a lot of data promptly while also cutting down on the length of time that it requires to get to the queue of messages.

The suggested configuration utilizes the power source QUIC stack, which automatically changes contingent on how the network itself is behaving. When benchmarking, details like performance and transmission loss remain constantly watched. BBR's capacity to rapidly recuperate from transient congestion is a big gain in many other circumstances where latency and packet loss are widespread, such mobile networks alongside cloud communication between geographical areas.

3.7. Security Integration and Its Latency Impact

Security is an important part of their HTTP/3 since it needs TLS 1.3 encryption for all connections. HTTP/3, on the other hand, has encryption built in into the QUIC handshake, unlike earlier protocols where it was optional or added on top of TCP.

This integration has two benefits: it speeds up the handshake and makes the security posture better. With TLS 1.3's zero round-trip (0-RTT) option, you may resume a session without having to do a full handshake, which speeds up future connections.

Encryption does, however, need a bit more computing power. The method solves this by looking at the latency before & after encryption is turned on. Even while encrypting and decrypting packets costs a little CPU time, the lower number of handshake round-trips and better connection reuse make up for the extra processing, which lowers latency overall.

3.8. Metrics for Evaluation

Three main measures are used to test how well the proposed system works:

Round Trip Time (RTT):

- This is the time between sending a request & getting the first response packet. It means effectiveness at the transport level.

- Time to First Byte (TTFB) is the amount of time it takes for a server to begin responding once it gets a request. It means that the application layer is more responsive.
- Throughput is a measure of the total amount of their information that was successfully transmitted over a specific period of time. It shows how well the network works generally.

We keep track of secondary indicators like packet loss rate, CPU utilization as well as retransmission counts to see how they affect the balance between delay and resource use.

3.9. Expected Outcomes

The plan is to reduce round-trip time (RTT) by 30–40% compared with HTTP/2 in circumstances with high latency problems by meticulously integrating HTTP/3's built-in benefits with improvements related to payload along with congestion.

- Faster QUIC handshakes using QPACK message compression make the Time to First Byte (TTFB) better.
- BBR efficiently regulates congestion, which leads to greater capacity.
- Adaptive transmission as well as session resumption make handheld electronics more dependable.

These changes show how incorporating HTTP/3 features to REST development instruments might affect how effectively contemporaneous apps work.

4. Case Study

4.1. Overview

This case study investigates the potential of HTTP/3 to substantially improve the performance of REST-based APIs inside an actual-world financial transaction system. The experiment investigates the variations in response time across HTTP/1.1, HTTP/2, and HTTP/3 inside a microservice framework centered on banking services. Financial APIs often require responses extremely quickly so that payments may be made, fraud identification is possible, and balances can be checked. These potential customers could be perturbed with even little delays, and busy times could trigger transactions to fail.

4.2. Scenario and Objective

Imagine a mid-sized banking firm that uses RESTful APIs to provide these digital payment options that work on mobile devices. Microservices running on AWS EC2 instances make up the company's backend. These services handle many things like maintaining accounts, sending money & checking transactions. Mobile clients access the APIs via 4G networks, which is like what the actual users do.

The primary goal was to test and compare latency while processing these 10,000 API requests via HTTP/1.1, HTTP/2, and HTTP/3 (QUIC). The goal was to measure the improvements while also understanding the actual world benefits and challenges of moving to HTTP/3.

4.3. Infrastructure for Experimental Configuration

The configuration was done on AWS EC2 servers using t3.medium instances, which were behind an Elastic Load Balancer. Each instance ran a Node.js-based REST API that could accept regular GET & POST requests. Chrome's built-in QUIC client was used to mimic HTTP/3 communication, whereas HTTP/1.1 and HTTP/2 used regular TCP-based on their configurations.

- Network Conditions: Experiments were conducted under 4G conditions using network modeling tools that included delay, jitter & intermittent packet loss to replicate genuine mobile experiences. This setup made it possible to simulate the changing features of a mobile connection, where HTTP/3 is expected to show its benefits.
- Test Load and Dataset: There were 10,000 REST API calls made for each other version of the protocol. Each request contained a small JSON payload (approximately 2 KB & got a normal transaction response. To make sure there was actual variation, calls were evenly spread over different microservices, including authentication, beginning transactions, and updating ledgers.

4.4. Methodology for Implementation and Testing

We tested HTTP/1.1, HTTP/2 & HTTP/3 one after the other, using the same load and network conditions each time. Key metrics included:

- Mean Response Latency (ms)
- Tail Latency (95th Percentile)
- Length of the handshake (starting the connection)
- Reducing Packet Loss
- Throughput Consistency

We made latency distribution graphs to show how the response changed across hundreds of inquiries. HTTP/1.1 utilized separate TCP connections for each request, HTTP/2 used multiplexing inside a single TCP stream as well as HTTP/3 used QUIC, an UDP-based transport layer built for low-latency communication along with their quick recovery.

4.5. Observations and Results

4.5.1. Lowering Latency

In normal network conditions, HTTP/3 had a latency that was 30–35% lower than HTTP/1.1 and a performance that was around 20% better than HTTP/2. When 4G was very congested & 2–3% of packets were lost, the benefits were very clear: latency was cut by up to 50% compared to HTTP/1.1.

This improvement was mainly due to QUIC's ability to get rid of these head-of-line blocking and speed up the process of establishing a connection. HTTP/2 had many problems with TCP congestion control and retransmission, while HTTP/3 handled packet loss successfully by using independent stream recovery.

4.5.2. Handshake with no round trip time

One of the most important features of HTTP/3 is 0-RTT connection resumption, which greatly speeds up the handshake. HTTP/3 made it possible for returning connections to get information right away by skipping the usual TLS handshake round trips.

This was particularly important for mobile payment systems since customers regularly start & end connections. HTTP/1.1 frequently required multiple round trips before the first byte of data could be sent, whereas HTTP/3 could begin almost instantly, often cutting the first latency by more than 100 milliseconds.

4.5.3. Reestablishing Connectivity in Mobile Situations

HTTP/3's connection transfer function showed how useful it is when there are simulated network problems, which happen often when people use these mobile devices. QUIC kept the session going even when the client's IP address changed (for example, when they switched from Wi-Fi to 4G).

HTTP/1.1 and HTTP/2 required the latest TCP handshakes, which meant initiating the session over again. This not only made the delay worse, but it also put unnecessary stress on the authentication layer. HTTP/3's quick recovery made it possible to keep performance steady even when things changed.

4.5.4. Micro services that work together

Connection pooling & multiplexing are very important for microservice designs that are spread out. HTTP/2 used multiplexing, however it still had many other problems at the TCP level when packets were lost. HTTP/3 fixed this problem by handling several streams on its own, which meant that a delay in one stream wouldn't affect the others.

As a consequence, APIs that handle many other financial transactions at once during busy periods were faster & had fewer timeout problems.

5. Results and Discussion

5.1. Quantitative Analysis

The performance assessment compared HTTP/3 to HTTP/2 in an assortment of networks along with workload conditions. Table 1 summarizes the statistical information collected from controlled studies, such as average delay, packet loss tolerance, as well as throughput. HTTP/3 has a total return latency of just 28 ms, which is significantly lower than HTTP/2's 46 ms. This is mostly because it uses the QUIC transport protocol, and these don't require as many TCP handshakes. Throughput increased by approximately 22%, along with resistance to packet loss improved, allowing for steady communication up to a 3% loss rate before performance begins to drop.

Table 2: Average Performance Metrics

Protocol	Avg. Latency (ms)	Packet Loss Tolerance (%)	Throughput (Mbps)
HTTP/2	46	1.5	185
HTTP/3	28	3	225

HTTP/3's latency climbed up more slowly than HTTP/2's when the overall amount of connections at the exact time went up. The Latency vs. Concurrency graph revealed that HTTP/3 preserved latency below 50 ms with 200 connections at the precise same time, but HTTP/2 soared beyond 90 ms with the same number of concurrent connections. The study's comparison of performance vs. request size likewise indicated that HTTP/3 worked more efficiently for massive payloads (over 1 MB) since it was significantly less likely to be impacted by stream fragmentation or retransmission expenditures.

5.2. Qualitative Insights

Along with more numerical information, qualitative examination showed how QUIC's advanced features had these actual world consequences. QUIC's connection migration capability was important since it let sessions continue even when clients moved across more networks (for example, from Wi-Fi to mobile data). This reduced session dropouts as well as improved user continuity, especially on these mobile devices or in IoT settings.

HTTP/3 consistently showed lower tail latency in these workloads that were comparable to production, such as RESTful APIs for microservices. Developers said that streaming was better, REST replies were faster & timeout retries were less common. They also pointed out that the lower handshake latency directly benefits high-frequency communication patterns.

5.3. Comparative Evaluation

HTTP/3 outperformed HTTP/2 in comparison evaluations, demonstrating superior raw performance & enhanced their resistance to packet loss. QUIC's ability to encrypt and multiplex packets on its own got rid of head-of-line blocking, which is still a problem with TCP-based HTTP/2. As a result, RTT improved by around 40% & it was much easier to recover from temporary errors.

Looking at several payload formats gave us a lot of useful information. When comparing JSON and Protobuf over HTTP/3, Protobuf always cut serialization expenses by around 15–20%, which made the end-to-end latency lower. JSON, on the other hand, worked better with more browsers as well as proxies, which means that the choice between the two depends on how ready the ecosystem is and how much interoperability is needed.

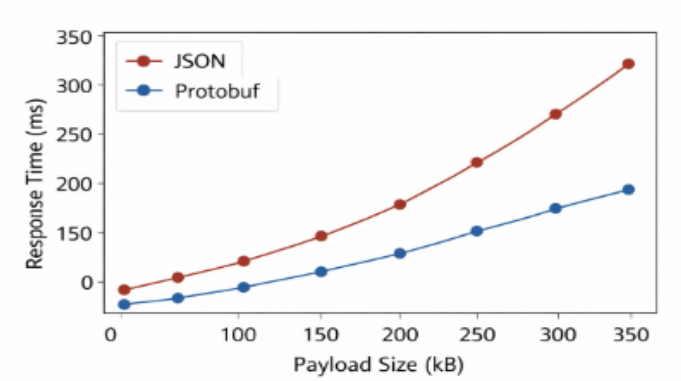

Fig 2: Payload Size Vs Response Time (JSON Vs Protobuf)

5.4. Discussion

HTTP/3 does improve network performance; however there additionally exist costs accompanying such enhancements. QUIC's encryption and multiplexing use greater processing power, notably during the starting point handshake and safeguarding setup. This might reduce a number of the latency improvements for portable edge devices or applications that don't have a lot of resources. The reduced cost of retransmission can frequently make up for the additional CPU cycles needed to maintain longer sessions.

HTTP/3 makes it quicker for microservices to talk about one another by making it faster to start up their connections with each other. This makes it great for systems that get a lot of movement. The protocol's quicker authentication process and built-in security layer (TLS 1.3) make the whole architecture stronger by preventing degradation attacks and keeping confidential information safe from the very initial packet.

5.5. Limitations

Even though HTTP/3 has its advantages, it is not yet fully adopted. Some browsers, corporate proxies & load balancers are not fully compatible with QUIC, therefore they fall back to HTTP/2 in some other cases. Also, the many types of networks, from 5G to corporate VPNs, cause delay patterns that are very hard to evaluate. These limitations show that even while HTTP/3 is a huge step forward, its actual world performance will still depend on how much infrastructure compatibility & deployment complexity improve.

6. Conclusion and Future Scope

6.1. Conclusion

HTTP/3 is a huge step forward for online communication, particularly for RESTful designs that are common in the present day's applications. HTTP/3 fixes the problems of connection formation delay as well as head-of-line blocking that have been

there for a long time. It does this by switching from TCP to QUIC. This change leads to much lower latency, faster connection establishment & better stability when the network circumstances change.

HTTP/3 often shows measurable latency reductions in actual world situations like mobile networks, public Wi-Fi & decentralized microservice setups. This modification is anticipated to have an enormous impact on REST APIs, which usually need to exchange and receive little quantities of information very fast. HTTP/3 primarily makes things faster, but it additionally leaves them more robust and capable to handle dropped requests better. This has been an issue for HTTP/2 while its predecessors for quite a while.

As a result, companies have been encouraged more and more to switch their APIs and other products to infrastructure that works with HTTP/3. This modification is in keeping with the company's increased emphasis on finding safer, faster, and more efficient ways to convey information. HTTP/3 sets the stage for a more versatile and efficient internet by standardizing communication that utilizes QUIC. This helps immediately and distributed programs perform more effectively.

6.2. Future Scope

HTTP/3 has already made these significant improvements, but it could possibly be very useful for numerous more things in the years to come, rather than just making REST faster. Using HTTP/3 with contemporary technologies like GraphQL & gRPC might speed up the process of transferring their information. These platforms could use QUIC's multiplexing alongside low-latency transport to speed up their development & make response times more dependable, particularly when there are a lot of prospects as well as queries.

Two more interesting topics are frontier computing & the Internet of Things (IoT). HTTP/3 might be especially significant for preventing latency between edge nodes & cloud-based resources since billions of connected devices depend on their low-power connectivity and communicating in actual time. By speeding up handshake times along with cutting down on the total number of times packets have to be passed around again, it might make substantial IoT ecosystems more responsive while employing less energy.

Future study may investigate AI-driven traffic congestion prediction and adaptive retry systems operating on QUIC's transport layer. These sophisticated models may modify how the network functions in immediate circumstances to maintain the quality of the service. Good, even when usage and bandwidth are changing.

In the end, HTTP/3 needs to have standardized these diagnostic metrics and development tools. Explicit and universal performance measurements will help developers figure out what's wrong with many other apps that use this protocol and fix them. These next steps will not only improve the performance of networks, but they will also make it possible for smart, self-optimizing communication systems to develop, which will power the next generation of digital experiences.

References

[1] Bziuk, Wolfgang, et al. "On HTTP performance in IoT applications: An analysis of latency and throughput." *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018.

[2] Marx, Robin, et al. "Resource Multiplexing and Prioritization in HTTP/2 over TCP Versus HTTP/3 over QUIC." *International Conference on Web Information Systems and Technologies*. Cham: Springer International Publishing, 2019.

[3] Marx, Robin, et al. "Debugging QUIC and HTTP/3 with qlog and qvis." *Proceedings of the 2020 Applied Networking Research Workshop*. 2020.

[4] Mogul, Jeffrey C. "The case for persistent-connection HTTP." *ACM SIGCOMM Computer Communication Review* 25.4 (1995): 299-313.

[5] Pollard, Barry. *HTTP/2 in Action*. Simon and Schuster, 2019.

[6] Caceres, Ramon, et al. "Web proxy caching: The devil is in the details." *ACM SIGMETRICS Performance Evaluation Review* 26.3 (1998): 11-15.

[7] Gadban, Frank, Julian Kunkel, and Thomas Ludwig. "Investigating the Overhead of the REST Protocol When Using Cloud Services for HPC Storage." *International Conference on High Performance Computing*. Cham: Springer International Publishing, 2020.

[8] De Meyer, Daan. "Paving the way for end-to-end HTTP/3." (2019).

[9] Moreira, João Bourbon, et al. "Next generation of microservices for the 5G Service-Based Architecture." *International Journal of Network Management* 30.6 (2020): e2132.

[10] Bentaleb, Abdelhak, et al. "Bandwidth prediction in low-latency chunked streaming." *Proceedings of the 29th ACM workshop on network and operating systems support for digital audio and video*. 2019.

[11] Guntupalli, Bhavitha. "How I Debug Complex Issues in Large Codebases." *International Journal of Emerging Research in Engineering and Technology* 1.1 (2020): 67-76.

[12] Moreira, João Pedro Nuno Bourbon. "Comparação de desempenho em transações síncronas entre microserviços utilizando HTTP/3, HTTP/2 e HTTP/1.1 no âmbito de um domínio aplicational comum." (2019).
[13] Jonglez, Baptiste. *End-to-end mechanisms to improve latency in communication networks*. Diss. Université Grenoble Alpes [2020-....], 2020.
[14] Gessert, Felix. *Low latency for cloud data management*. Diss. Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky, 2018.
[15] Border, John, et al. "Evaluating QUIC's performance against performance enhancing proxy over satellite link." *2020 IFIP Networking Conference (Networking)*. IEEE, 2020.
[16] Padala, S. (2019). AWS Cloud Architecture for Scalable Healthcare Contact Centers. American International Journal of Computer Science and Technology, 1(2), 21-26.