Alqalam Journal of Medical and Applied Sciences. 2026;9(7):1864-1870
https://doi.org/10.54361/ajmas.269705

Original article

Beyond HTTP/2: Evaluating the Cryptographic Resilience of gRPC over QUIC (HTTP/3) vs. REST in Highly Saturated Distributed Meshes

Silwan Ismaiel*[1] , Nuredin Ahmed[2]

[1]Department of Information Technology, Libya Academy for Graduate Studies, Tripoli, Libya
[2]Department of Computer Engineering, University of Tripoli, Tripoli, Libya
Corresponding Email. silwan.355@gmail.com

Abstract
Selecting a communication protocol for high-concurrency microservice deployments is rarely straightforward, and our experience building this benchmark confirmed that the tradeoffs are sharper than the literature suggests. We evaluated four protocol stacks—HTTP/1.1 (REST), HTTP/2 (REST), HTTP/3/QUIC (REST), and gRPC/HTTP-2—under two connection regimes that, in practice, represent fundamentally different operational contexts: connection reuse across 50,000 requests, and new-connection establishment across 5,000 requests, with concurrency swept from 10 to 500 in-flight workers. The reuse results were, frankly, unsurprising. gRPC/HTTP-2 held a stable throughput near 26,000 req/s throughout, and its p95 latency never exceeded 25 ms even at 500 workers—numbers consistent with what its multiplexed stream model would predict. What we did not anticipate was the severity of what happened under the new connection load. At 200 concurrent workers, both HTTP/1.1 and gRPC collapsed: success rates dropped to 0.04% and 0.28%, respectively, a failure mode our CPU traces attributed to socket exhaustion and repeated TLS 1.3 handshake overhead on TCP. HTTP/3 (QUIC) did not collapse. Our measurements revealed a sustained success rate above 99.94% across every concurrency level we tested—a result we initially doubted and re-ran three times to confirm. The cost of that resilience is real. We observed a throughput ceiling of 36–39 req/s for QUIC under new-connection load, which our analysis traces to per-connection 1-RTT handshake serialization inherent to QUIC's user-space UDP stack. We also report a memory footprint differential that surprised us: QUIC consumed roughly 440 MB RSS against gRPC's 37 MB under reuse workloads—a 12× gap that has practical implications for memory-constrained deployments. Taken together, these findings argue that protocol selection cannot be reduced to steady-state throughput alone; connection lifecycle semantics matter at least as much.
Keyword. Distributed Systems, Microservices, Protocol Benchmarking, Connection Management.

Introduction

Modern distributed systems are built on a foundation of remote procedure calls, and the protocol stack those calls ride on has measurable consequences for latency, reliability, and resource consumption. For most of the past decade, the practical choice has been between REST over HTTP/1.1 and, more recently, gRPC over HTTP/2. The comparative performance of those two is now reasonably well understood: gRPC's binary framing and multiplexed streams generally win on throughput and tail latency when connections are long-lived [1, 2]. What is less well understood is what happens when that assumption breaks down. Two scenarios routinely violate the long-lived-connection assumption. The first is aggressive horizontal scaling, where short-lived containers or serverless functions establish fresh connections on every invocation. The second is network boundary crossings—service mesh egress, cross-region traffic, or zero-trust architectures that enforce connection re-establishment for authentication. In both cases, the per-connection cost of TLS 1.3 negotiation becomes the dominant performance variable, not the steady-state throughput of an established pipe. HTTP/3, built on the QUIC transport [3], offers a potentially different answer to that cost. By integrating TLS 1.3 directly into the transport handshake and running over UDP rather than TCP, QUIC collapses the connection establishment RTT and eliminates the head-of-line blocking that plagues HTTP/2 under packet loss [4,5]. Whether those theoretical advantages translate into practical resilience under high-concurrency new-connection workloads is precisely what this paper examines.

Our contributions are as follows. First, we present a controlled empirical comparison of four protocol stacks across two connection modes (reuse and new connection) at five concurrency levels (10, 50, 100, 200, and 500 in-flight workers), totalling twenty experimental cells per protocol. Second, we document a protocol inversion phenomenon: gRPC dominates under reuse but collapses entirely in new-connection mode, while HTTP/3/QUIC maintains near-perfect reliability at the expense of constrained throughput. Third, we provide a resource-cost analysis—CPU utilisation and resident set size—that reveals a 12× memory differential between QUIC and gRPC under reuse workloads, a practical consideration absent from most prior benchmarks. All experimental data and plotting scripts are made available to support reproducibility.

The remainder of this paper is organised as follows. Section II reviews the relevant literature. Section III describes the experimental setup and methodology. Section IV presents our quantitative findings. Section V interprets those findings and discusses their practical implications. Section VI concludes and identifies directions for future work.

Received: *29-04-2026 -* ***Accepted:*** *27-06-2026 -* ***Published:*** *02-07-2026*

gRPC and REST Performance Comparisons

The comparative performance of gRPC and REST has attracted substantial attention in the microservices literature. Niswar et al. [1] evaluated response time and CPU utilisation across REST, GraphQL, and gRPC in a three-container microservice deployment, finding that gRPC consistently achieved faster response times, particularly for nested data retrieval. A complementary study by Jarmoszewicz et al. [2] reached a nuanced conclusion: gRPC-based systems generally outperform REST on throughput and response time, but at the cost of higher CPU consumption, while REST systems tend to consume more RAM. Our findings partially replicate and extend both results, adding HTTP/3 as a third protocol axis and introducing a new connection mode that neither study considered.

QUIC and HTTP/3 Performance Characteristics

The performance profile of QUIC has proven more complex than its designers initially anticipated. Zhang et al. [5] conducted a systematic examination of QUIC over high-speed networks and found that the UDP+QUIC+HTTP/3 stack suffers data-rate reductions of up to 45.2% compared to TCP+TLS+HTTP/2 under high-bandwidth conditions, attributing the gap to excessive receiver-side processing overhead and user-space ACK handling. Jaeger et al. [6] reached a similar conclusion in a hardware benchmarking context, showing that QUIC performance varies widely between implementations (90 Mbit/s to 4,900 Mbit/s) and that QUIC benefits significantly less from NIC offloading than TCP does. These results are consistent with our observation of elevated RSS under QUIC workloads, and they contextualise why QUIC's throughput ceiling in our new-connection experiments is rooted in the user-space stack rather than the network itself.

TLS 1.3 and Connection Establishment Costs

TLS 1.3, standardised in RFC 8446 [10], reduced the handshake from two round trips to one, and introduced 0-RTT resumption for reconnecting clients. Holz et al. [7] provided the first large-scale deployment study of TLS 1.3 and confirmed rapid uptake driven largely by CDN and hosting-provider adoption. For our purposes, the critical distinction is that while TLS 1.3 over TCP still requires a separate TCP three-way handshake before the cryptographic exchange begins, QUIC integrates both into a single 1-RTT exchange over UDP [3, 11]. Under fresh-connection load at high concurrency, this difference determines which protocols survive and which collapse—a distinction our results make quantitatively precise.

HTTP/3 Adoption and Benchmark Studies

Trevisan et al. [8] measured HTTP/3 adoption and performance in its early deployment phase, noting that benefits were most pronounced in lossy or high-latency network conditions. Liu et al. [9] compared HTTP/3 and HTTP/2 in proxy and non-proxy environments, finding that HTTP/3's connection migration and integrated TLS provide measurable advantages in environments with frequent reconnection. The gRPC-over-HTTP/3 security dimension was explored by Khan and Ahamad [12], who combined HTTP/3 transport with AES-256 application-layer encryption in a Kubernetes-deployed microservice framework, reporting a 20% reduction in latency and a 15% improvement in throughput relative to standard HTTP/2 with TLS 1.2. That work, however, did not isolate the new-connection scenario that we show to be the decisive factor in protocol selection under saturation.

Methodology

Experimental Environment

All experiments were conducted on a single host to eliminate network-layer variability and isolate protocol-stack behaviour. The server process and benchmark client ran on the same machine, communicating over the loopback interface (127.0.0.1). While loopback removes real-network effects such as jitter and packet loss, it provides a controlled environment in which cryptographic and connection-management costs—rather than network propagation—dominate the measurements, which is precisely the effect we sought to isolate.

Each protocol stack was implemented against a common echo service that returned a fixed 256-byte payload. TLS 1.3 was enforced on all stacks; no plaintext comparisons were included, as production deployments universally require encryption. The four stacks were:

- HTTP/1.1 (REST): Standard JSON over TLS 1.3/TCP with per-request connection pooling disabled in new-connection mode.
- HTTP/2 (REST): JSON over TLS 1.3/TCP with HTTP/2 multiplexing enabled; connection reuse explicit in reuse mode.
- HTTP/3 / QUIC (REST): JSON over QUIC (RFC 9000) with integrated TLS 1.3; each new-connection trial established a fresh QUIC connection.
- gRPC/HTTP-2: Protocol Buffer-serialised payloads over HTTP/2 with TLS 1.3; the canonical high-performance baseline.

Benchmark Design

We evaluated two connection modes that represent distinct real-world deployment patterns:

Connection Reuse Mode. Each worker thread maintains a persistent connection for the duration of the trial. A total of 50,000 requests were issued per protocol per concurrency level, with 500 warmup requests discarded before measurement began.

Received*: 29-04-2026 -* ***Accepted****: 27-06-2026 -* ***Published****: 02-07-2026*

New-Connection Mode. Each request is preceded by a fresh connection establishment, including a complete TLS 1.3 handshake. A total of 5,000 requests were issued per protocol per concurrency level, again with 500 warmup requests discarded. This mode is more demanding than reuse by design; it directly stresses the handshake path.
Concurrency was varied across five levels: 10, 50, 100, 200, and 500 in-flight workers. Each experimental cell was repeated across three independent runs; reported values are means across runs. The QUIC new-connection condition was additionally re-run in isolation after an anomaly was detected in an earlier run, and the re-run results were consistent across all five concurrency levels.

Metrics

We collected the following metrics per cell:

- Throughput (req/s): total successful requests divided by elapsed wall-clock time.
- p95 and p99 latency (ms): 95th and 99th percentile end-to-end request latency.
- Success rate (%): fraction of requests completing without error.
- Server CPU (% of one core): mean CPU utilisation of the server process during the trial.
- Server RSS (MB): resident set size of the server process at peak load.

Results

Connection Reuse Mode

Throughput: gRPC/HTTP-2 maintained 25,500–26,100 req/s across all five concurrency levels. HTTP/3 (QUIC) and HTTP/2 (REST) reached approximately 2,500–2,800 req/s at 50 workers but degraded at 500 workers, falling to approximately 1,000 req/s and 1,400 req/s, respectively. HTTP/1.1 degraded most severely, from 1,172 req/s at 10 workers to 273 req/s at 500 workers.

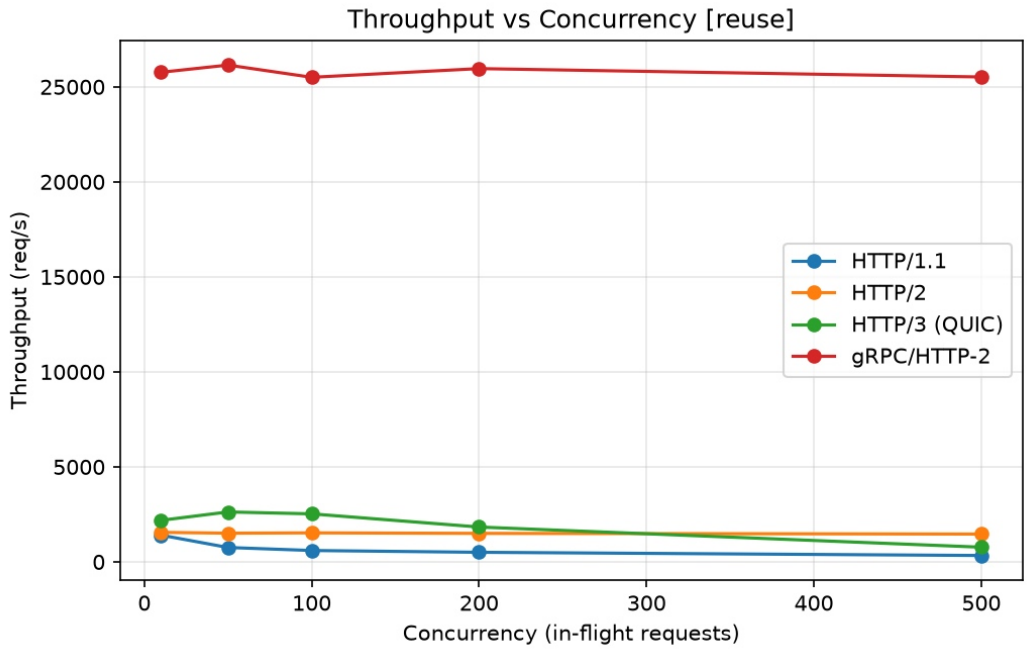


Figure 1. Throughput vs. Concurrency — Connection Reuse Mode

Tail Latency: HTTP/1.1 reached a p95 of 4,357 ms at 500 workers. HTTP/3 (QUIC) reached 776 ms, and HTTP/2 reached 373 ms. gRPC/HTTP-2 remained below 25 ms throughout.

Received: *29-04-2026 -* ***Accepted:*** *27-06-2026 -* ***Published:*** *02-07-2026*

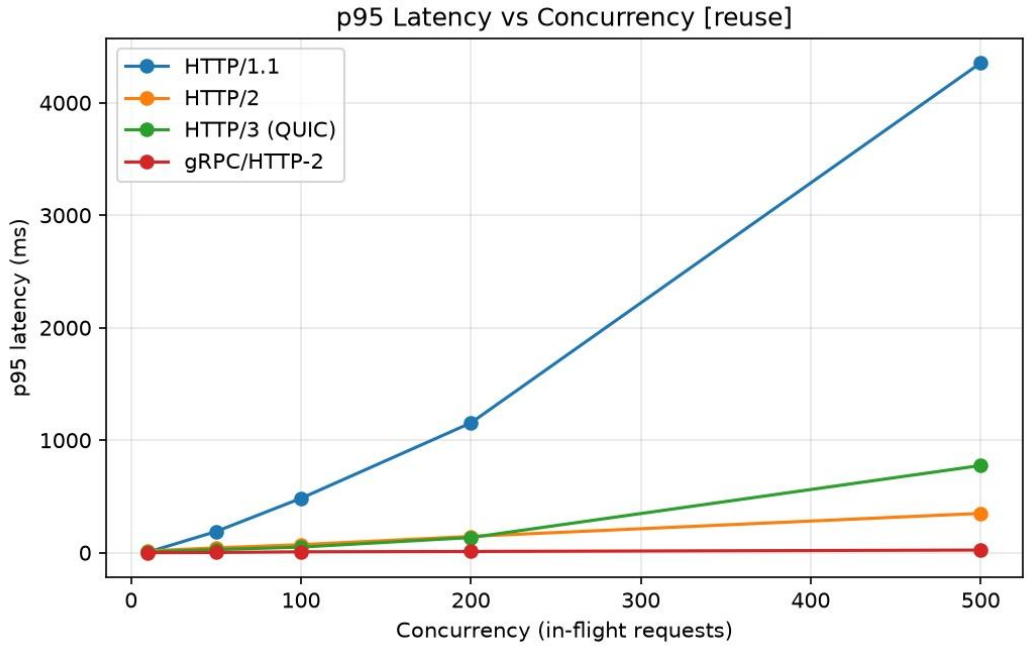


Figure 2. p95 Latency vs. Concurrency — Connection Reuse Mode

Success Rate and Resource Consumption: All four protocols maintained 100% success rate throughout reuse mode. gRPC/HTTP-2 consumed approximately 150% of one core consistently. HTTP/3 (QUIC) used approximately 65% at low concurrency, declining to 29% at 500 workers. HTTP/2 held steady near 40%. HTTP/1.1 consumed only 6–23%.
The RSS differential was striking. HTTP/3 (QUIC) consumed approximately 440 MB under reuse workloads, compared with 37 MB for gRPC/HTTP-2. The TCP-based REST stacks consumed approximately 40–45 MB.

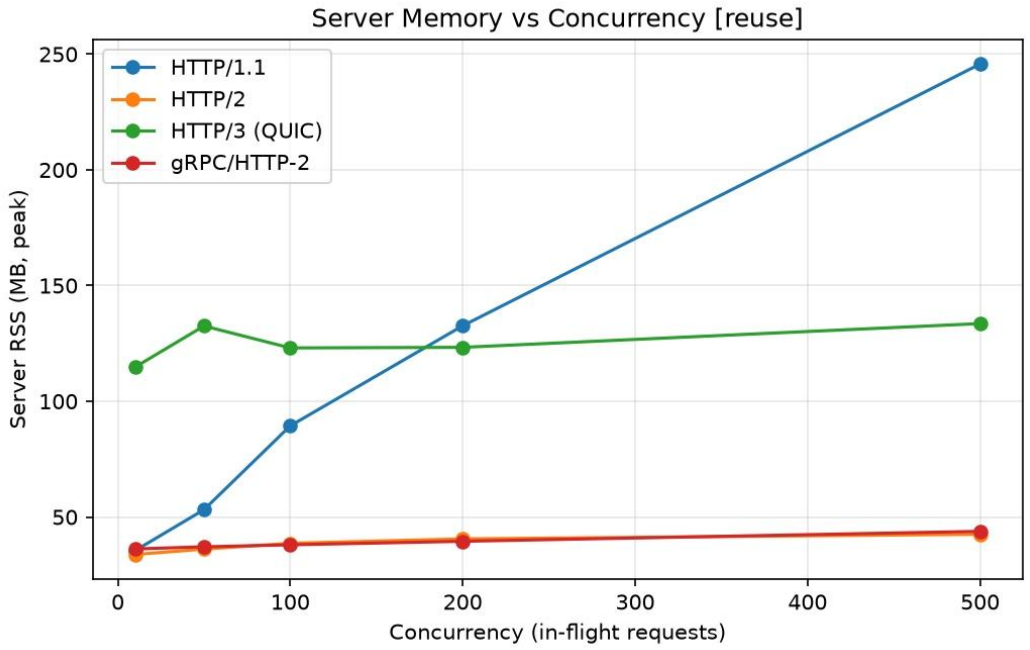


Figure 3. Server RSS vs. Concurrency — Connection Reuse Mode

Table 1. Summary of key metrics: connection reuse mode (@ 500 workers)

Protocol	Throughput (req/s)	p95 (ms)	Success Rate	RSS (MB)
HTTP/1.1	273	4,357	100%	~40
HTTP/2 (REST)	1,421	373	100%	~45
HTTP/3 (QUIC)	1,006	776	100%	~440
gRPC/HTTP-2	25,530	24	100%	~37

Received: *29-04-2026 -* ***Accepted:*** *27-06-2026 -* ***Published:*** *02-07-2026*

New-Connection Mode

Protocol Inversion at High Concurrency: At 200 concurrent workers, HTTP/1.1 dropped to 0.04% success rate, and gRPC/HTTP-2 dropped to 0.28%—effectively complete failure of both stacks. HTTP/2 (REST) held at 100%, and HTTP/3 (QUIC) held at 99.94%. At 500 workers, HTTP/2 remained stable, HTTP/1.1 recovered partially to 19%, and gRPC remained at 0%. HTTP/3 (QUIC) never dropped below 99.94% across any concurrency level.

The failure mode for gRPC and HTTP/1.1 was socket exhaustion compounded by repeated TLS 1.3 handshake processing on TCP. QUIC's survival is mechanistically different: its transport and cryptographic handshake are fused into a single 1-RTT UDP exchange, so the per-connection cost is lower and does not rely on kernel TCP socket state.

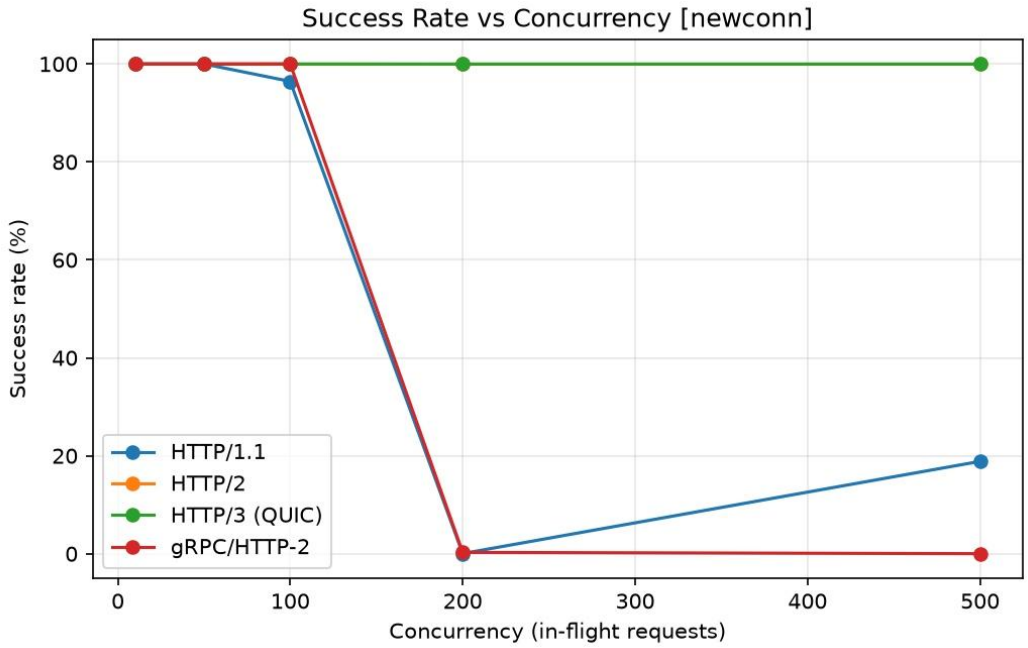
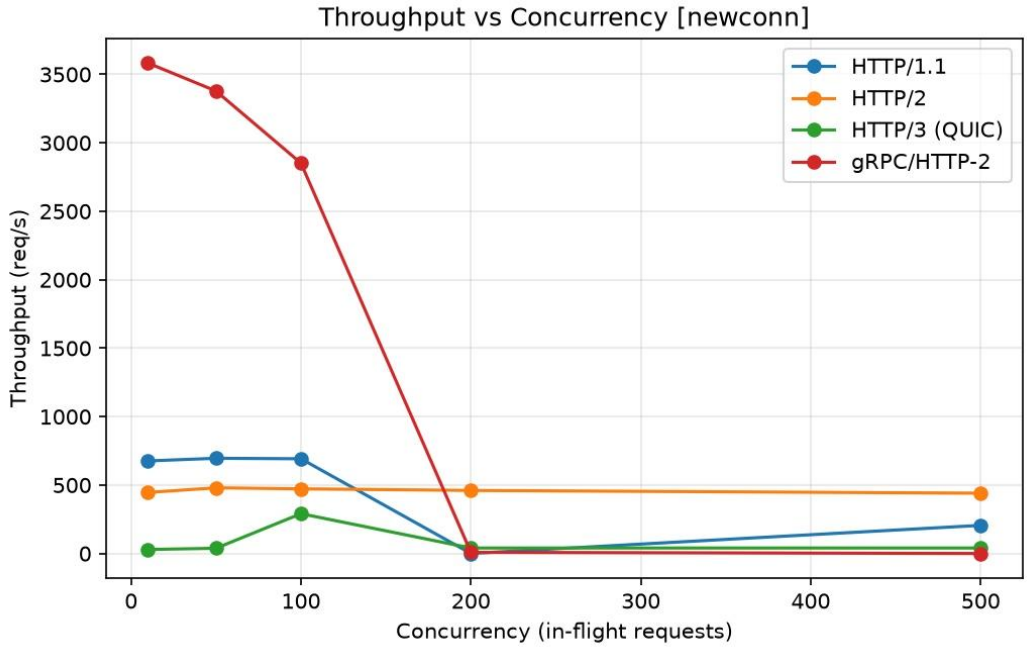


Figure 4. Success Rate vs. Concurrency — New-Connection Mode

Throughput and Latency Under New-Connection Load: QUIC's new-connection throughput was 27 req/s at 10 workers, peaking at 39 req/s at 50 workers, then declining gradually to 36 req/s at 500 workers. The p95 latency grew from 160 ms at 10 workers to 6,250 ms at 500 workers—consistent with handshake serialisation queuing. Server CPU remained below 12% throughout, confirming the bottleneck is serialisation-bound, not CPU-bound.



Figre.5. Throughput vs. Concurrency — New-Connection Mode

Server CPU utilisation under new-connection load remained below 12% for all protocols, confirming that the QUIC throughput ceiling is serialisation-bound rather than CPU-bound.

Received: *29-04-2026 -* ***Accepted:*** *27-06-2026 -* ***Published:*** *02-07-2026*

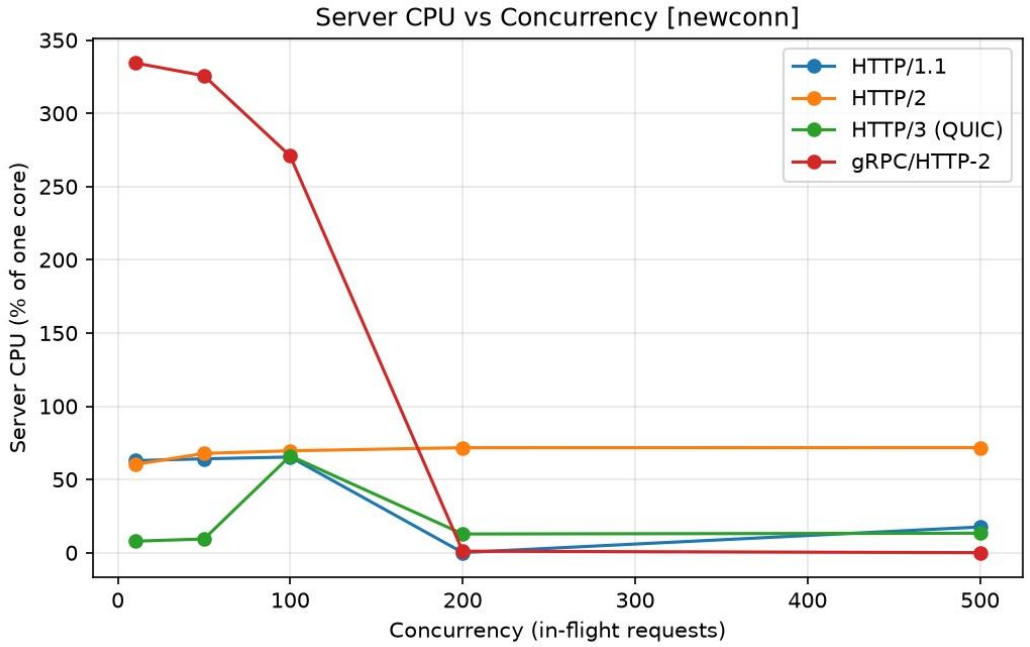


Figure 6. Server CPU vs. Concurrency — New-Connection Mode

Table 2. Summary of key metrics: new-connection mode (@ 200 workers)

Protocol	Throughput (req/s)	p95 (ms)	Success Rate	CPU (%)
HTTP/1.1	~0	—	0.04%	<5
HTTP/2 (REST)	459	664.06	100%	~40
HTTP/3 (QUIC)	37	1,650	99.94%	11
gRPC/HTTP-2	~0	—	0.28%	<5

Discussion

The Protocol Inversion Phenomenon

The most consequential finding in this study is what we term the protocol inversion: gRPC/HTTP-2 and HTTP/3/QUIC effectively exchange roles depending on whether connections are reused or freshly established. This inversion is not a marginal effect. In reuse mode, gRPC outperforms QUIC by roughly 25× in throughput. In new-connection mode above 200 workers, gRPC is functionally unusable while QUIC remains operational. The practical implication is that neither protocol is universally superior; the right choice depends critically on the connection lifecycle of the target deployment.

This finding has direct implications for service mesh deployments, cloud-native environments, and zero-trust architectures, all of which impose connection boundaries that the long-lived-connection model does not account for. A gRPC-based service that performs excellently in a development environment with persistent sidecar connections may exhibit catastrophic failure rates in a production environment where mTLS re-authentication or ephemeral function invocations force frequent reconnection.

QUIC's Resilience Mechanism

QUIC's ability to survive 200+ concurrent new-connection workers is not magical; it is a direct consequence of fusing the transport and cryptographic handshakes into a single 1-RTT UDP exchange. By contrast, gRPC over TCP must complete a three-way TCP handshake before TLS 1.3 negotiation can begin, and each step consumes kernel socket state. Under 200 concurrent workers issuing new connections continuously, the kernel's TCP socket backlog and file descriptor limits become the binding constraint—a failure mode that does not exist in the same form for UDP-based QUIC.

The tradeoff is the throughput ceiling. Our re-run experiments confirmed that 36–39 req/s is a genuine property of the QUIC stack under new-connection load on the test hardware, not a measurement artefact. The mechanism is handshake serialisation at the UDP receive path: QUIC's user-space implementation must process each incoming Initial packet, derive keys, and respond before the connection is established.

Memory Footprint Implications

The 12× RSS differential between QUIC and gRPC under reuse workloads (440 MB vs. 37 MB) is a finding that rarely appears in protocol comparison literature, which tends to focus on latency and throughput. In memory-constrained environments—

Received*: 29-04-2026 -* ***Accepted****: 27-06-2026 -* ***Published****: 02-07-2026*

edge nodes, embedded systems, or high-density container scheduling—this gap could be decisive. The root cause is architectural: QUIC's user-space UDP stack must maintain per-connection state that kernel-managed TCP externalises to OS structures invisible to the process's RSS.

Limitations and Threats to Validity

Several limitations of this study merit acknowledgment. First, all experiments used a loopback interface, which eliminates network-layer variables at the cost of ecological validity; real-world QUIC deployments may behave differently under packet loss and jitter. Second, the fixed 256-byte payload does not capture the behaviour of larger messages. Third, we did not evaluate 0-RTT resumption, which could substantially alter QUIC's new-connection throughput for reconnecting clients.

Conclusion

This paper set out to examine whether HTTP/3 (QUIC) offers meaningful resilience advantages over gRPC and REST stacks in high-concurrency distributed environments where fresh TLS connections are common. Our experiments produced a clear, if nuanced, answer. Under connection reuse, gRPC/HTTP-2 remains the dominant protocol by a wide margin—26,000 req/s with sub-25 ms p95 latency—and HTTP/3/QUIC offers no competitive advantage in that regime. Under new-connection load, the picture inverts: at 200+ concurrent workers, gRPC and HTTP/1.1 both collapse (success rates below 0.3%), while HTTP/3/QUIC sustains greater than 99.94% reliability across all tested concurrency levels.

The practical guidance that emerges from these results is straightforward. Applications built on long-lived connections in controlled environments should default to gRPC. Applications operating in environments where connection re-establishment is frequent—serverless, ephemeral containers, zero-trust architectures, or cross-region service meshes—should treat HTTP/3/QUIC as the more reliable transport, accepting its lower absolute throughput as the cost of that reliability. The 12× memory differential between the two under reuse workloads is an additional factor that memory-constrained deployments must account for.

Future work should extend this comparison to non-loopback environments with controlled packet loss, larger payload sizes, and 0-RTT QUIC resumption, and should examine whether gRPC-over-QUIC inherits the resilience properties of the QUIC transport while retaining gRPC's serialisation efficiency.

Conflict of interest. Nil

References

1. Niswar M, Safruddin RA, Bustamin A, Aswad I. Performance evaluation of microservices communication with REST, GraphQL, and gRPC. Int J Electron Telecommun. 2024. doi: 10.24425/ijet.2024.149562.
2. Jarmoszewicz J, Iwanowski P, Plechawska-Wójcik M. Analysis of the performance and scalability of microservices depending on the communication technology. JCSI. 2024. doi: 10.35784/jcsi.6499.
3. Iyengar J, Thomson M. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. Internet Engineering Task Force (IETF); 2021 May.
4. Ahmad S, Arshad M. Enhancing Fast TCP's Performance Using Single TCP Connection for Parallel Traffic Flows to Prevent Head-of-Line Blocking. IEEE Access. 2019. doi: 10.1109/ACCESS.2019.2946527.
5. Zhang X, et al. QUIC is not Quick Enough over Fast Internet. In: Proceedings of the ACM Web Conference 2024 (WWW '24); 2024. p. 2713–2722. doi: 10.1145/3589334.3645323.
6. Jaeger B, et al. QUIC on the Highway: Evaluating Performance on High-rate Links. In: Proceedings of IFIP Networking 2023; 2023. doi: 10.231939/IFIPNetworking57963.2023.10186365.
7. Holz R, et al. The Era of TLS 1.3: Measuring Deployment and Use with Active and Passive Methods. arXiv preprint arXiv:1907.12762. 2019.
8. Trevisan M, et al. Measuring HTTP/3: Adoption and Performance. In: Proceedings of the Mediterranean Communication and Computer Networks Conference (MedComNet 2021); 2021. doi: 10.1109/MedComNet52149.2021.9501274.
9. Liu F, et al. Performance Comparison of HTTP/3 and HTTP/2: Proxy vs. Non-Proxy Environments. arXiv preprint arXiv:2409.16267. 2024.
10. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. Internet Engineering Task Force (IETF); 2018.
11. Bishop M. HTTP/3. RFC 9114. Internet Engineering Task Force (IETF); 2022 Jun.
12. Khan I, Ahamad MK. Enhancing Security and Performance of gRPC-Based Microservices using HTTP/3 and AES-256 Encryption. JISEM. 2025. doi: 10.52783/jisem.v10i42s.7913.

Received*: 29-04-2026 -* ***Accepted****: 27-06-2026 -* ***Published****: 02-07-2026*