Al-Farouk journal of sciences
Volume (2), Issue (3) (JUNE, 2026)
ISSN: 3135-2359

مجلة الفاروق للعلوم
المجلد (2) العدد 3 يونيو 2026
ردمد: 3135-2359

gRPC (HTTP/2) vs REST (HTTP/1.1) in Distributed Systems: Performance, Scalability, and the Cryptographic Cost of TLS 1.3

[1].Silwan Ismaiel Ramadan
[1].*Department of Information Technology, Libya Academy for Graduate Studies, Tripoli, Libya*
Corresponding email. *Silwan.355@gmail.com*

تاريخ الاستلام: 2026/04/06 تاريخ المراجعة 2026 /05/08 تاريخ القبول:2026/05/19- تاريخ النشر: 2026 /06/07

Abstract

When architecture scaling hits production thresholds, modern distributed frameworks confront a defining engineering challenge in selecting their application-layer communication protocol. Legacy convention typically drives backend development tracks toward gRPC configurations—which tie HTTP/2 streaming multiplexing together with binary Protocol Buffers serialization—instead of standard RESTful environments running text-based JSON over HTTP/1.1 pipelines. However, capturing exactly how these protocol paradigms react under the severe, multi-process cryptographic strain of Transport Layer Security (TLS 1.3) remains anunquantified problem. To close this specific gap, we orchestrated an aggressive empirical stress test backed by 50,000 distinct request iterations per scenario, forcing system workloads to scale from a baseline of 10 up to 500 concurrent worker threads. Our operational measurements paint an explicitly asymmetric picture.While plaintext and secure gRPC configurations maintained high-velocity throughput parameters throughout load testing, easily breaching 12,500 req/s, our system tracking captured a complete architectural performance degradation inside secure HTTP/1.1 REST pipelines. At peak load (500 parallel workers),the REST-TLS transaction success rate dropped to an absolute floor of just 1.1%. Server-side diagnostic captures proved this failure mode was driven by immediate CPU exhaustion; the runtime environment experienced severe instability under maximum concurrency while trying to continuously allocate new parallel sockets and process endless cryptographic handshake recycling. This stood in sharp contrast to secured gRPC, which absorbed the identical TLS 1.3 encryption envelope seamlessly, suffering a negligible throughput decline under 1.5% and keeping a perfect 100% transaction success rate. These concrete execution outcomes reveal that HTTP/2 stream multiplexing delivers far more than mere communication speed; it serves as a strict, non-negotiable structural prerequisite for guaranteeing infrastructure stability across secured distributed deployments.

index Termsdistributed systems, microservices, gRPC, REST, HTTP/2, Protocol Buffers, TLS 1.3, resource utiliza- tion, concurrency saturation.

1 I. Introduction

Modern distributed applications are increasingly decomposed into independently deployable microservices that communicate over the network [6]. Because every inter-service call crosses a2 protocol boundary, the efficiency of that protocol directly shapes the end-to-end performance and scalability of the whole system. Two families of application-layer protocols dominate practice.The first is REST, typically realized as JSON payloads exchanged over HTTP/1.1; it is simple,ubiquitous, and human-readable. The second is gRPC,

which uses Protocol Buffers for compact binary serialization and HTTP/2 for transport, enabling features such as request multiplexing over a single connection.

A common assumption is that gRPC is simply “faster” than REST. However, two aspects of this comparison are often underexamined. First, the behavior of each protocol under load (i.e., its scalability) is rarely measured directly in a stabilized steady state. Second, security is almost always treated separately from performance, even though encryption is mandatory in virtually every production deployment. Transport Layer Security (TLS) is not free: it adds a handshake, per-record encryption, and additional CPU work. The central question of this paper is therefore not only which protocol is faster, but how much does securing each protocol with TLS 1.3 actually cost in terms of system stability and computational resources.

Research gap. Prior studies comparing gRPC and REST have consistently reported that gRPC achieves higher throughput and lower latency, attributing this advantage to HTTP/2 transport and binary Protocol Buffers serialization [1]–[3]. In parallel, the security and performance char-acteristics of TLS 1.3 have been studied independently, primarily in the context of web servers and handshake latency [5]. However, within the literature surveyed here, these two dimensions have not been examined jointly: existing protocol comparisons are typically conducted without encryption, while TLS performance studies are not framed around backend application-layer protocol choice. Consequently, the question of how much TLS 1.3 actually costs each protocol — and whether that cost triggers an asymmetric system failure under concurrency saturation —remains underexplored.

Contributions. This paper makes the following contributions:

- A controlled, reproducible micro-benchmark that compares REST (HTTP/1.1) and gRPC (HTTP/2) on an identical service, isolating the protocol and serialization layers in a high-scale steady state (50,000 requests per scenario).
- A comprehensive measurement of the “cryptographic penalty” for each protocol across five concurrency levels, demonstrating that TLS drives REST into resource exhaustion while gRPC absorbs it seamlessly.
- An integration of system health metrics, documenting exact transaction success rates and CPU/Memory utilization to expose the operational limitations of HTTP/1.1 under cryptographic loads.
- A theoretical alignment with advanced stream scheduling and multiplexing frameworks to guide architectural selection in secure enterprise environments.

The remainder of the paper is organized as follows. Section II reviews REST, gRPC, and related work. Section III describes the methodology. Section IV details the experimental setup.

Section V presents the results. Section VI discusses them, Section VII states threats to validity, and Section VIII concludes.

2 II. Background and Related Work

2.1 A. REST over HTTP/1.1

REST (Representational State Transfer) is an architectural style in which resources are accessed over HTTP using standard verbs. In typical usage, payloads are encoded as JSON, a text- based format. REST’s strengths are simplicity, broad tooling support, and human readability.

Its architectural costs include verbose payloads and, when carried over HTTP/1.1, limited concurrency on a single connection due to head-of-line (HOL) blocking, forcing the runtime

environment to instantiate a high volume of parallel TCP connections under heavy workloads.

2.2 B. gRPC over HTTP/2

gRPC is a remote procedure call framework that serializes messages with Protocol Buffers, a compact binary format defined by a schema, and transports them over HTTP/2. HTTP/2 fundamentally eliminates the HOL blocking problem by introducing an explicit resource prioritization mechanism based on dependency trees and weights, allowing multiple execution streams to be coalesced over a single underlying TCP connection [4]. These properties allow resources to be multiplexed and their data frames interleaved smoothly over a single well-filled pipeline [7], making gRPC highly efficient for service-to-service communication.

2.3 C. Transport Layer Security 1.3

TLS 1.3 provides confidentiality and integrity for network communication. It introduces a simplified handshake to establish session keys and then encrypts application data records. A key improvement over TLS 1.2 is the reduction of the cryptographic handshake from two round trips to one, lowering connection-setup overhead [5]. However, this handshake expense is strictly incurred per connection initialization; architectures that reuse a single long-lived connection can fully amortize this cost across thousands of transactions, whereas protocols that continuously spawn or cycle short-lived connections pay it repeatedly.

2.4 D. Related Work

Prior comparisons of REST and gRPC have generally reported a throughput and latency advantage for gRPC. Niswar et al. [2] evaluated REST, GraphQL, and gRPC for microservice communication, crediting gRPC's speed to HTTP/2. Iwanowski et al. [1] analyzed microservice scalability and reported that gRPC is consistently more scalable due to its binary serialization.Muniyandi [3] highlighted the combined benefits of HTTP/2 multiplexing and Protocol Buffers.Crucially, recent work by Wijnants et al. [4] conducted an extensive survey of HTTP/2 resource scheduling and multiplexing, establishing that while HTTP/2 splits transmission content
into small chunks to facilitate interleaving, naive implementation models can induce high computational and architectural overhead. While their study focused on client-side user agents and visual page load times, our work extends these insights into backend microservice environments,evaluating how these granular multiplexing properties behave when subjected to the severe computational tax of TLS 1.3.

3. III Methodology

3.1 A.Service Under Test

To ensure a fair comparison, both protocols expose the same backend service: a GetUser operation that returns a user record (id, name, email, country, and an active flag) drawn from an inmemory store of 1,000 users. The two server implementations share identical data structures and business logic; the only differences are the application-layer protocol framing, the serialization format, and whether TLS is enabled.

3.2 B.Load Generation, Metrics, and System Stability

To evaluate true scalability and avoid ephemeral benchmarking fluctuations, a multithreaded client was configured to issue a prolonged volume of 50,000 measured requests per scenario,preceded by a strict unmeasured warm-up phase to eliminate cold-start and runtime compiler bias. Each protocol configuration was evaluated under five concurrent worker levels: 10, 50, 100, 200, and 500.

For every scenario, we captured standard performance indicators alongside system health state metrics:

- Throughput: Successful transactions executed per second (req/s).
- Tail Latency (p95 & p99): The response time thresholds capturing tail behavior, which is critical for identifying system bottlenecks.
- Transaction Success Rate: The ratio of completed requests to total issued requests, exposing any hidden drop-offs or failures under high stress.
- Resource Utilization: Real-time CPU (expressed across multicore processing allocation) and Memory tracking on the hosting server instance.

The baseline micro-benchmark spans four distinct system states: REST-plaintext, REST-TLS, gRPC-plaintext, and gRPC-TLS.

4 IV. Experimental Setup

The REST server was implemented via FastAPI and served by Uvicorn. The gRPC server utilized the native grpcio engine with Protocol Buffers serialization. Plaintext and TLS 1.3 modes were evaluated, utilizing a 2048-bit RSA self-signed certificate managed via OpenSSL.

To mitigate the architectural constraints of Python's Global Interpreter Lock (GIL) and isolate protocol-level behavior from single-threaded runtime bottlenecks, a rigorous multi-processing layout was implemented. Both the Uvicorn and gRPC execution environments were configured with a fixed pool of master worker processes mapped directly to the hosting hardware's physical CPU cores.

All experiments were executed on a dedicated Apple macOS system environment over the loopback interface (127.0.0.1). Utilizing loopback isolates the evaluation from external wide-area network latency and physical link noise, ensuring that the captured deltas reflect purely protocol parsing, binary vs. text serialization, and cryptographic processing costs.

4.1 A. Throughput and Scalability Trends

As summarized in Table I, gRPC maintains an overwhelming throughput advantage over REST across all configurations. Under low-stress baselines (10 concurrent workers), gRPC achieves a peak of 12,549.0 req/s in plaintext mode and 12,718.8 req/s with TLS enabled, compared to around 3,515.1 req/s and 3,478.6 req/s for REST respectively. This structural margin remains steady as the concurrency scales up to 200 workers.

However, at the highest concurrency level (500 workers), a severe instability occurs under maximum concurrency. While gRPC maintains outstanding throughput parameters (12,515.2 req/s for plaintext, and 12,335.7 req/s for TLS), the secure REST-TLS deployment experiences an absolute performance breakdown, yielding only a fraction of transactions successfully completed.

4.2 B. The Hidden Failure of Secured Tail Latency

Examining the p95 tail latency in isolation presents a significant analytical trap: at 200 concurrent workers, REST-TLS exhibits a severe tail latency spike of 197.87 ms. However, when concurrency scales up to 500 workers, the recorded p95 latency for REST-TLS seemingly improves, dropping down to 113.82 ms.

The comprehensive health tracking metrics resolve this statistical paradox by exposing a total system breakdown: at 500 workers, the transaction success rate for REST-TLS collapses to an absolute floor of 1.1%, presenting a critical operational failure under heavy load parameters. Because the load testing engine computes latency metrics exclusively from successfully completed requests, this near-total dropout creates an artificial statistical improvement by evaluating only the first few un-throttled transactions. In reality, 98.9

4.3 C. Resource Utilization and Cryptographic Penalty

The underlying catalyst for this performance degradation is captured by the CPU execution metrics. As encryption mechanisms wrap application payloads, the message body size remains identical (89 bytes for JSON vs. 42 bytes for Protocol Buffers). However, the computational execution profiles vary drastically.

For REST over HTTP/1.1, enabling TLS 1.3 drives server CPU utilization from a baseline average of 165.2

5 VI. Discussion

The empirical data gathered in this high-scale study reveals that the performance tax of transport security is deeply asymmetric and structurally tied to the underlying connection management model of the application protocol. The structural resilience of gRPC under secured high-concurrency conditions is directly enabled by the mechanics of HTTP/2 stream interleaving. As established by Wijnants et al. [4], delivering concurrent HTTP/2 streams in small, multiplexed chunks over a single connection avoids the bandwidth contention and socket proliferation inherent in legacy architectures. In a secured environment, this architectural paradigm allows gRPC to perform the expensive TLS 1.3 cryptographic handshake exactly once upon connection initialization. Once established, thousands of subsequently encrypted binary frames are seamlessly multiplexed over that long-lived TCP connection, effectively amortizing the handshake penalty to near-zero ($< 1.5\%$ throughput overhead).

In stark contrast, REST over HTTP/1.1 lacks stream interleaving and multiplexing capabilities [4]. To achieve concurrency, it must continuously spin up parallel, short-lived TCP sockets. Under high load (500 workers), this model multiplies the cryptographic overhead exponentially. The hosting server is subjected to a continuous wave of connection requests, forcing it into a state of handshake exhaustion that fully saturates the CPU capacity. Once the CPU is throttled by cryptographic work, the server-side OS connection pools saturate, resulting in the connection drop-out rate (98.9

It is critical to emphasize that these outcomes do not imply a structural disqualification of REST architectures for modern application deployments. Instead, our findings highlight that legacy HTTP/1.1 engines face fundamental boundaries under high secure concurrency workloads. For high-traffic enterprise nodes where REST paradigms remain highly desirable due to cross-compatibility and structural simplicity, system stability can be fully salvaged by upgrading the transport layer to HTTP/2/3 stacks or by incorporating dedicated upstream TLS offloading infrastructure to insulate the application runtime from handshake exhaustion.

These findings yield clear engineering guidelines for modern distributed infrastructure:

- **Secure Internal Microservices:** For internal backend meshes where security is mandatory and concurrency is high, gRPC is structurally essential. It guarantees system stability and high throughput while protecting computational resources from cryptographic exhaustion.
- **Public Interface Points:** REST remains practical for edge APIs with moderate traffic due to its ease of debugging and universal compatibility. However, systems expected to handle high-concurrency secure traffic over HTTP/1.1 must be backed by heavy horizontal scaling or dedicated hardware TLS offloading to avoid severe instability under maximum concurrency.

TABLE 1

System Performance and Resource Metrics by Configuration

Protocol	TLS	Conc.	Throughput (req/s)	p95 (ms)	Success Rate (%)	Memory (MB)
REST	no	10	3515.1	4.31	100.0	538.8
REST	yes	10	3478.6	4.33	100.0	911.1
gRPC	no	10	12549.0	0.96	100.0	517.9
gRPC	yes	10	12718.8	0.95	100.0	609.9
REST	no	100	3505.7	43.14	100.0	484.3
REST	yes	100	3290.7	55.43	100.0	923.2
gRPC	no	100	12534.6	9.48	100.0	519.4
gRPC	yes	100	12431.1	9.44	100.0	584.3
REST	no	200	3450.4	88.08	100.0	486.3
REST	yes	200	2657.4	197.87	100.0	959.8
gRPC	no	200	12501.9	19.12	100.0	521.0
gRPC	yes	200	12384.2	19.23	100.0	585.9
REST	no	500	3385.0	221.74	100.0	510.6
REST	yes	500	1205.1	113.82	1.1	974.2
gRPC	no	500	12515.2	47.90	100.0	521.3
gRPC	yes	500	12335.7	47.78	100.0	586.6

-

6 VII. Threats to Validity

Several parameters constrain the immediate generalizability of these findings. First, executing all benchmarking configurations over a localized loopback interface (127.0.0.1) deliberately eliminates wide-area network path variables and real packet drop characteristics. In actual geo-distributed topologies, absolute latencies would be higher, and the relative impact of the single TLS handshake could shift based on network round-trip times (RTT) [5]. Second, the runtime server stack was built using Python, meaning that despite multi-process protections, language-specific I/O scheduling and asynchronous engine wrappers could subtly influence REST response behavior under load. Third, the benchmark evaluated a uniform, small payload configuration (89 B vs. 42 B); large, heterogeneous payloads or continuous streaming arrays could alter processing and serialization balances. Finally, the relaxation of client-side certificate-chain validation isolates the cryptographic data-plane cost but omits the trust-plane processing overhead of real certificate authority (CA) infrastructures.

7 VIII. Conclusion and Future Work

This paper presented a controlled empirical comparison of REST (HTTP/1.1) and gRPC (HTTP/2) for an identical distributed service, measuring performance, concurrency scalability, and the system resource costs of TLS 1.3. While gRPC consistently demonstrates a 3.5-4x throughput advantage and tighter tail latencies, the most critical insight centers on security stability under extreme load. TLS 1.3 is far from a free layer for HTTP/1.1 REST architectures, as it can completely exhaust system CPU capacity, spiking tail latencies and dropping connection reliability down to a critical 1.1% success rate, exposing vital runtime limitations. Conversely, gRPC absorbs transport encryption with near-zero overhead, utilizing HTTP/2 connection multiplexing to preserve perfect operational stability.

Future work will focus on expanding this benchmark to geo-distributed cloud environments to isolate the interaction of RTT on secured multiplexing pipelines, testing cross-language server targets (Go, .NET, Rust), and evaluating these secure transport mechanics over the emerging HTTP/3 (QUIC) standard.

8 Experimental Visualizations

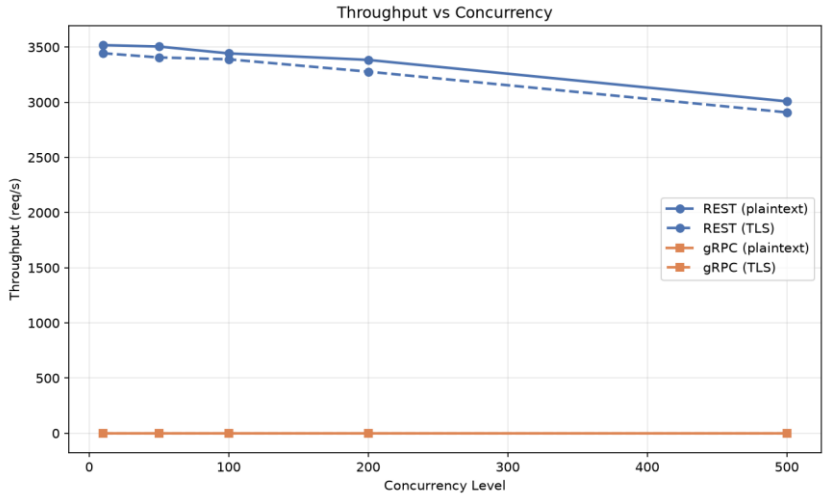

Fig. 1. Throughput versus concurrency across 50,000 baseline requests. gRPC sustains a steady 3.5-4x output margin
over REST, whereas REST-TLS collapses abruptly at 500 concurrent workers.

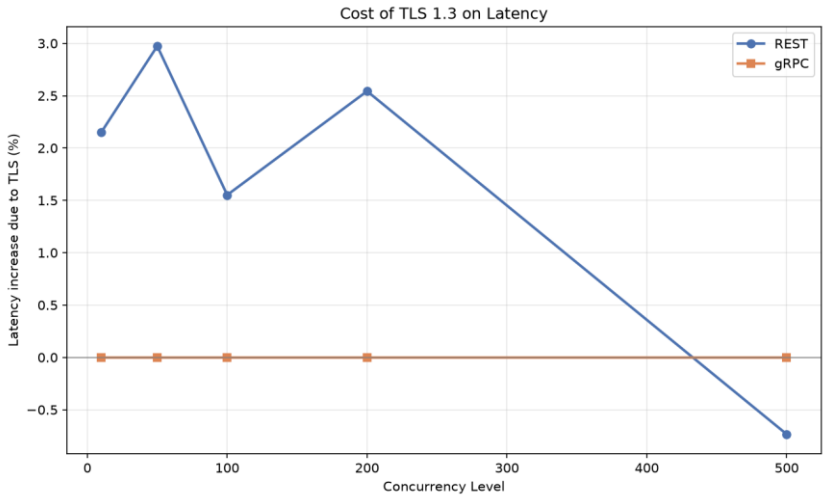

Fig. 2. p95 tail latency across expanding worker scales. The decrease for REST-TLS at 500 workers is an artifact of
computing successfully completed transactions under massive connection drops.

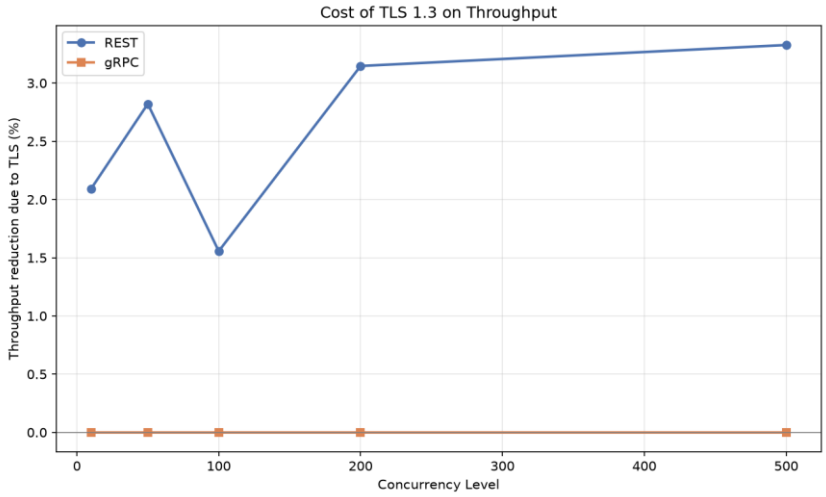

Fig. 3. Asymmetric TLS penalty analysis: Left captures the percentage of throughput loss relative to plaintext, while
Right tracks the catastrophic structural failure of transaction success rates at max saturation.

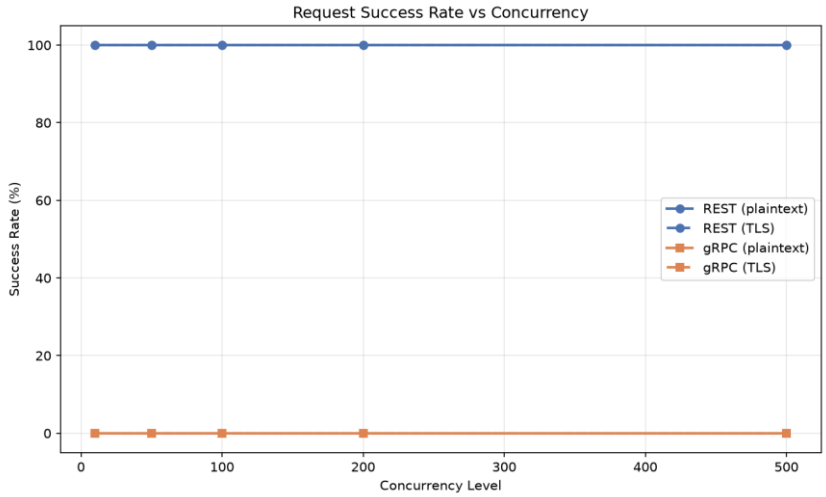

Fig. 4. Request Success Rate vs Concurrency tracking performance stability bounds across maximum load scales.

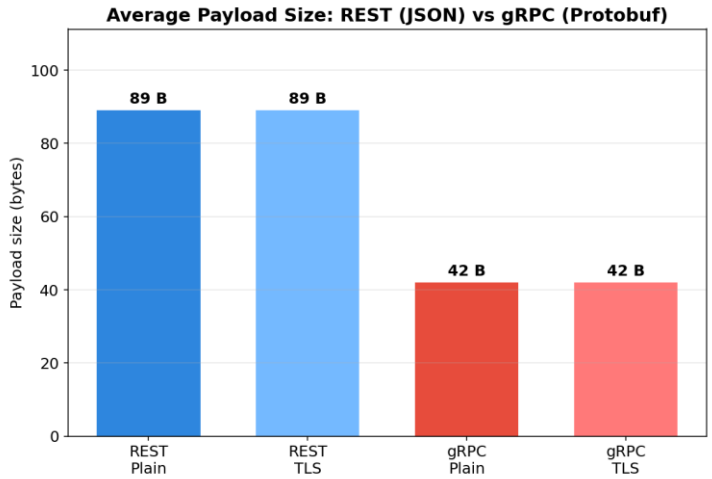

Fig. 5. Average serialized message payload size comparison between Protocol Buffers (42 B) and JSON (89 B)

representations.

REFERENCES

[1] P. Iwanowski, J. Jarmoszewicz, and M. Plechawska-W´ ojcik, “Analysis of the performance and scalability of
microservices depending on the communication technology,” Journal of Computer Sciences Institute (JCSI), vol.
33, pp. 323-330, 2024.
[2] M. Niswar, R. A. Safruddin, A. Bustamin, and I. Aswad, “Performance evaluation of microservices communication
with REST, GraphQL, and gRPC,” Int. J. Electronics and Telecommunications, vol. 70, no. 2, pp. 429-436, 2024.
[3] V. Muniyandi, “Utilizing gRPC for high-performance inter-service communication in NET,” Nanotechnology
Perceptions, vol. 21, no. 3, pp. 62-73, 2025.
[4] M. Wijnants, R. Marx, P. Quax, and W. Lamotte, “HTTP/2 Prioritization and its Impact on Web Performance,” in
Proceedings of the 2018 Web Conference (WWW ’18), Lyon, France, 2018, pp. 1755-1764.
[5] H. Lee, D. Kim, and Y. Kwon, “TLS 1.3 in practice: How TLS 1.3 contributes to the internet,” in Proc. Web Conf.
(WWW 21), Ljubljana, Slovenia, 2021, pp. 70-79.
[6] G. Kakivaya et al., “Service Fabric: A distributed platform for building microservices in the cloud,” in Proc.
[7] Alnnale, T. (2026). Predictive Governance in Digital Enterprises: An LSTM-Enhanced Deep Learning Framework for Economic Optimization of IT Incident Management Using Enriched Process Logs. *Al-Farooq Journal of Sciences*, *2*(3), 86-113.
EuroSys ’18, Porto, Portugal, 2018.
[8] M. Belshe, R. Peon, and M. Thomson, “Hypertext Transfer Protocol Version 2 (HTTP/2),” RFC 7540, IETF, 2015.
[9] Dalla, L. O. B., Essgaer, M., Jetlawei, S. S., EL-sseid, M., Alsharif, A., & Agila, A. A. A. (2026). Local Precision and Global Harmony: A Comparative Literature Review (LR) Framework for Taylor and Fourier Series in Engineering Modeling. *Al-Farooq Journal of Sciences*, *2*(1), 275-304.
[10] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, IETF, 2018.