ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Performance Comparison of API Protocols: A Systematic Review of REST, gRPC and GraphQL

Munir Ali Mohammed, Shra Fatima, Ajaz Husain Warsi,

Computer Sci. & Engineering, Integral University, Lucknow, India

DOI: https://doi.org/10.51583/IJLTEMAS.2026.150500132

Received: 15 May 2026; Accepted: 20 May 2026; Published: 08 June 2026

ABSTRACT

The correct choice of API protocols in the modern distributed systems defines the level of performance, scalability, and resource efficiency. The current paper contains a systematic literature review (SLR) of 20 empirical studies in total published in 2020-2025 by comparing the top three architectural styles, namely, REST, gRPC, and GraphQL. This review offers a review of the evidence-based framework of protocol selection by synthesizing data on latency, throughput, resource utilization, and scalability.

Our results suggest that gRPC provides the shortest latency with small message payloads since it uses HTTP/2 multiplexing and Protobuf serialization, thus this is best suited to inter-service communication. REST has the best raw throughput (1500-2000 RPS on average) but it has head-of-line blocking, and data over-fetching. On the other hand, GraphQL reduces memory usage (around 17 percent versus 20 percent with REST/gRPC) due to accurate field selection, but has high CPU cost in query parsing and resolution.

The review has been found to have significant research gaps, the most notable of which is absence of standardized three way benchmarking and performance analysis of the production scales. This paper ends with evidence-based suggestions of hybrid architecture solutions to provide an architectural and research base of selecting approaches to microservices in architectural projects.

Keywords: API protocols, REST, gRPC, GraphQL, performance evaluation, distributed systems, protocol comparison, benchmarking

INTRODUCTION

Both monolith and modern distributed systems use the Application Programming Interfaces (APIs) as the basis of the communication layer that allows the easy interaction between distributed services and applications.[1]. Development of architectural systems, of monoliths, to systems of microservices has heightened the requirements to have efficient, scalable and maintainable API protocols[2], [3]. Three architectural solutions have become leading choices, including REST (Representational State Transfer), gRPC (Google Remote Procedure Call), and GraphQL, all of which solve isolated communication problems at different trade-offs[4].

REST is the commonly used standard of web APIs, having become the standard adopted by the underlying semantics of the HTTP protocol and being stateless with a client-server architecture[5]. Despite the popularity, REST has a few limitations, such as over-and under-fetching, and in high-throughput situations, performance issues are present [6]. Google's gRPC, the implementation of Google based on the functionality of HTTP/2 and Protocol Buffers, was launched to solve these performance problems, particularly in inter-service communication within distributed systems.[7]. Facebook introduced GraphQL, which is a query language allowing clients to define precise data needs, doing away with inefficiencies of the fixed endpoint format of REST. [8], [9].

Empirical research done recently has indicated obvious performance differences between these protocols. Broad benchmarking undertaken by Niswar et al. [10] revealed that gRPC responded at 233.84ms versus 1,113.33ms

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

of REST when running with 100 active requests. On the same note, studies conducted by Weerasinghe et al. [11] regarding inter-service communication mechanisms have validated the high throughput and latency being the best in gRPC. However, the choice of protocols goes beyond the raw performance scales, aspects such as the complexity of development, tooling ecosystem maturity and browser compatibility as well as team expertise are common considerations when the teams finally have to transition off the theorization into live production[12], [13].

This paper provides a systematic review and comparative study of the REST, gRPC and GraphQL protocols, and synthesizes the findings based on the academic literature of other Scholars. Our objectives include:

1. Examining the architectural predicates and design principles.
2. Measuring the attributes of performance using empirical standards.
3. Analyzing security practices and mechanisms.
4. Determining the best use cases and rules of choice, and
5. Discussing new trends and combination strategies.

This paper has Section II which gives reviews, related work and methodology. In Sections III-V present detailed analyses of REST, gRPC, and GraphQL, respectively. Section VI entails comparative performance evaluation. Sections VII-VIII discuss security considerations and tooling ecosystems. Section IX shows the framework of decision and use case analysis. Section X concludes with the directions of future research.

RESEARCH METHODOLOGY

A. Systematic Literature Review

A systematic process as shown in fig. 1 was followed to conduct the review. To discover the academic publications on API protocol performance and architecture, we systematically performed a literature review according to the available guidelines[14].We have used the IEEE Xplore, ACM Digital Library, Scopus and Google Scholar as our search databases, involving publications from 2018 to 2025. The following combinations of terms were used as search queries: REST API, GRPC, GraphQL, performance comparison, microservices communication and API protocols.

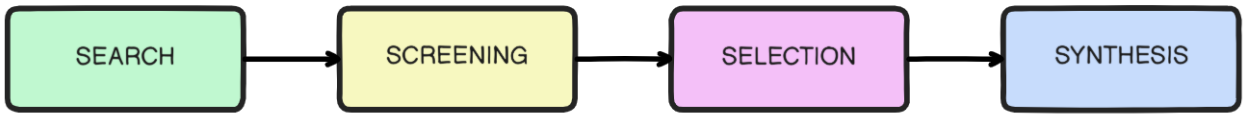

Fig. 1 Systematic Review Process

B. Inclusion and Exclusion Criteria

Inclusion criteria required:

1. Paper based conference proceedings or journal articles.
2. The empirical performance tests or architectural tests.
3. Publications in English, and
4. Focus on REST, gRPC, or GraphQL protocols. We have filtered away grey literature, opinion pieces that lack empirical evidence and studies that do not describe methodology.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

The search resulted in 234 papers with 73 articles passing the inclusion criteria through abstract screening and reviewing the whole text.

C. Performance Evaluation Framework

The synthesis of performance evaluation was concerned with the standardized measures: response time (latency), throughput (requests per second), CPU usage, memory usage, bandwidth usage and load scalability. We gave importance to research studies that were carried out using controlled experiments designs and had a well-defined workload and system configurations [15], [16].

Rest API: Architecture and Characteristics

A. Architectural Foundations

REST defines six architectural constraints that characterize the compliant systems as client-server separation, statelessness, cacheability, uniform interface, layered system architecture and optional code-on-demand [5],[17]. The uniform interface constraint splits into the following four sub-constraints: resource identification using URIs, resource manipulation using representations, self-descriptive messages and hypermedia as the engine of application state(HATEOAS) [17].

Statelessness requires that no session state is maintained on the server in between requests or that each request must contain all information needed to process that request. By eliminating the server-side session management overhead and load distribution issues, this constraint enhances scalability [18]. Loose coupling and technological diversity is fostered by the separation of client and server components, which gives the client and server component the opportunity to develop separately[19].

B. Technical Implementation

REST APIs usually use the human readable data exchange format, JSON (JavaScript Object Notation), as their standard data format because of its popularity across all modern languages, and because it is always compatible with JavaScript [20]. Communication is done over HTTP/1.1, using standard (GET, POST, PUT, DELETE, PATCH) which is semantically equivalent to CRUD (Create Read, Update, Delete) operations [21]. Resources have unique URIs and hierarchy structures that indicate relationships between resources -e.g. /users/123/posts/456 is a post 546 or user 123.

C. Performance Characteristics

Bermback et al. [17] conducted a high variety of benchmarks on quality metrics of web API of 14 production endpoints, and assessed pingability, accessibility, successability and latency. They obtained performance variability with all the means of latencies between 100ms and 1400ms across geographic regions and endpoint implementation. The use of HTTP/1.1 in REST exposes head-of-line blocking, one slow request blocks other requests on that connection [22].

The migration of REST services between JSON and Protocol buffers serialization showed by Shatnawi et al. [23] that payload sizes decreased by 60-80 percent and the response times were shortened by 80 percent, which means that the format of serialization has a significant effect on the performance of REST. But these changes are not in line with traditional practices of the REST and make browsers difficult to use.

D. Advantages and Limitations

Simple, universal support of the infrastructure of HTTP, a rich ecosystem of tools and developed caching mechanisms are the main strengths of REST [24]. The statelessness character allows horizontal scaling and it is easy to recover faults [25]. REST is however plagued with over-fetching (sharing more data than is necessary) and under-fetching (having to make multiple requests to retrieve related information) issues [26]. Muszyński and Koziel [27] found out security weak points in the systems of REST authentication, specifically in the

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

weakness in the implementation of JWT (JSON Web Token) and in the vulnerability to exposing and sharing tokens.

GRPC: High-Performance Rpc Framework

A. Architecture and Design Principles

gRPC uses the paradigm of Remote Procedure Call (RPC), which allows clients to call functions on remote servers as if they were local function calls [7], [28]. There are four major elements of the framework:

1. Protocol Buffers and Serialization mechanisms as the Interface Definition Language (IDL).
2. The transport protocol is HTTP/2..
3. Auto-generation of code in various programming languages and
4. Four patterns of communication; unary, server streaming, client streaming and bidirectional streaming [29] are supported.

Service contracts and messages structure in Protocol Buffers are defined as language-neutral .proto files. The protoc compiler produces type safe client stubs and server skeletons, which do away with boiler plate code and provide contract compliance in heterogeneous environments [30]. It is a contract-first method, which improves maintainability and eliminates interface drift in distributed systems [31].

B. HTTP/2 and Protocol Buffers

The many performance improvements that the HTTP/2 has over the HTTP/1.1 are as follows; multiplexing, header compression (HPACK) and several push (proactive resource delivery) and stream prioritization (optimal resource allocation)[32], [33]. In the study, Moreira et al. [38] tested the performance of HTTP/2 in 5G architecture-based service systems, where they reported that the latency dropped dramatically when working with high bursts of traffic contrasting with HTTP/1.1.

Protocol Buffers uses compact binary encoding to ensure high serialization efficiency [34]. It has been shown in comparative studies to reduce size and recognize 3-10x smaller than JSON, with speeds of serialization benign 10x higher than typical payloads[35], [36]. The schema-based validation of binary format helps avoid bad data transmission and it improves the reliability of the system[37].

C. Performance Evaluation

Niswar et al. [10] thoroughly tested the performance of the microservices architecture of REST, GraphQL and gRPC under 100 to 500 concurrent requests workloads. The outcome showed that gRPC was faster in general: average response time of 233.84ms (100 requests) to 2606.59ms (500 requests) versus RSET of 1113.33ms to 4009.83ms and it was considerably faster than GraphQL with its significantly high latencies. The analysis of CPU utilization has shown that gRPC used 30.11% when there were 100 requests but can reach 84.04% when there are 500 requests, which is much lower than GraphQL but greater than the use of REST at the same load.

Weerasinghe et al. [11] were able to compare inter-service communication mechanisms and established the benefits of gRPC operations that are sensitive to latency. Their experimental data indicated that gRPC in a way decreased their inter-service communication latency with one-fifth of the alternatives of the basis of REST. Arora et al [38] compared REST with gRPC in microservices-based ecosystems that showed the better throughput characteristics of gRPC in high-concurrency cases.

Johansson et al. [39] conducted comparative studies of REST and gRPC in established software architecture. They found that gRPC is more effective with higher data size transmission with performance gaps that expand proportionally with the data size sent to them. This characteristic makes gRPC particularly suitable for data-intensive applications.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

D. Use cases and Best Practices

gRPC is ideal in high performance and low latency [40] scenarios. It is used in applications such as:

1. **Microservices Communication:** Inter-service communication between the backend system benefit from the advantages of gRPC efficiency and strong typing [40]
2. **Real-time Applications:** Bidirectional streaming supports chat systems, live dashboards and collaborative editing [41].
3. **IoT and Edge Computing:** Compact payloads save on bandwidth usage in resource constrained setups [42]
4. **Polyglot Environments:** Type-safe interoperability in programming languages is guaranteed by code generation. [29]

Best practices involve proper deadline management that helps avoid hanging requests, client-side load balancing of distributed systems, interceptors to cross-cutting concerns (logging, authentication and monitoring) and a critical look at streaming patterns to eliminate extraneous connection overhead[43].

GRAPHQL: Flexible Query Language

A. Architectural Principles

GraphQL offers an API query language and runtime that allows clients to define precise data needs using a strongly-typed schema [8], [44]. GraphQL has one endpoint in contrast to REST, which has many endpoints or gRPC which has multiple procedure calls, which submit declarative queries that specify desired data shape and nesting [45].

Its architecture consists of three basic components:

1. Schema defining type system and available operations.
2. Resolvers that fill out data in every field with arbitrary data sources and, and
3. Operations including queries (read), mutations (write) and subscriptions (real-time updates) [46]. The schema is a contract between servers and the clients and it allows the client to have powerful introspection and tooling abilities [47].

B. Addressing Data Fetching Inefficiencies

The main value proposition of graphql is that it avoids over and under-fetching issues of REST APIs [48]. Over-fetching is the response of endpoint which return fixed data structure that include fields that are not required by particular client use. Under-fetching requires a series of requests to collect related resources consecutively [49]. GraphQL allows customers to include specific field selection and obtain full sets of requirements in individual calls [50].

Nevertheless, these problems are not automatically addressed by GraphQL-specific patterns of implementation, such as Fragments, DataLoaders, query optimization are necessary [51]. Bad resolver design may contribute to performance issues by the N+1 query anti pattern [52].

C. Performance Analysis and N+1 Problem

Systematic performance comparison of REST and GraphQL was performed by Seabra and Nazario [53], with the result that GraphQL migration increased performance in two of three performance measures of the tested applications (in two-thirds cases) in terms of requests per second and data transfer rates. Nonetheless, the

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

workloads exceeding 3000 simultaneous requests decreased the performance and the REST provided better throughput.

GraphQL has a serious performance issue, the N+1 problem [52]. In the most naive approaches, N+1 database calls are made when retrieving a list of N related items, N to get the list, and N to get relationships. This is addressed by the DataLoader pattern of Facebook, which batches and caches requests [54]. An industry implementation known as Dataloader 3.0 proposes a breadth-first data loading strategy to address the N+1 query problem. However, no peer-reviewed evaluation of the approach is currently available.

Dhika et al. [55] contrasted GraphQL and RESt architectures in educational games applications with the results showing the successful implementation of GraphQL and comments on the trade-offs in complexity. The quality of resolver implementation and database access patterns is vital in performance [56].

Comparative Performance Analysis

The overall comparisons of the published empirical studies of the last 5-7 years shows that REST, gRPC and GraphQL have different performance characteristics in various aspects. This part is a synthesis of results in 20 research papers, which discusses the performance features in a controlled experimental scenario to offer evidence-based solutions in protocol choice in both monolith and distributed systems architecture. Table I gives an overall overview of the literature reviewed including the performance results and research settings.

TABLE I: Core Empirical Studies on API Protocol Performance (2020-2025)

Study	Protocol	Scale & Methodology	Key Performance Metrics	Principal Findings
[57]	REST, GraphQL, gRPC	Load Testing: 1000 records stress testing conditions	Latency, throughput, CPU, memory, failure rate	REST: highest throughput, higher failures (2-5%). GraphQL: 17% memory vs 20% REST/gRPC, higher latency (2.9s) gRPC: balanced, 2.0s latency
[58]	gRPC, REST, SOAP	Empirical benchmarking, Variable message sizes	Latency, Scalability, TLS impact	gRPC: optimal small messages (<1KB) Best scalability under concurrent load. Minimal TLS performance degradation
[59]	REST, GraphQL	Serverless (AWS Lambda). Network Latency variation	RTT, end-to-end latency, cost	GraphQL: 25-67% improvements over REST. Single-request reduces RTT Especially effective under high network latency
[60]	REST, gRPC	.NET/C# microservices Task classification analysis	Communication efficient by task type	gRPC: advantages for specific communication patterns. Task-dependent protocol selection criteria. IoT/Industry 4.0 optimization

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

[4]	REST, WebSocket, gRPC, GraphQL	Python Implementation CRUD benchmarking	Comparative efficiency, reliability	gRPC: most efficient and reliable. GraphQL: slowest, library implementation issues protocol-language interaction effects

A. Latency Characteristics

Experiments on load testing with 1000 records of students illustrate that GraphQL has an average latency of 2.9 seconds whereas RESTful has average latency of 1.6 seconds and gRPC has average performance attributes. Comparison of the results of various studies shows that gRPC has the lowest p95 latency values, especially when the message payload is smaller and Protocol Buffer serialization has tangible benefits. Since binary encoding format is less costly to parsing than a deserialization of different formats, it has shorter response times to requests.

Empirical benchmarking of gRPC, REST and GraphQL shows that gRPC has a lower response time, and then comes REST, and GraphQL in microservices communication contexts. It increases the performance difference in the presence of high-concurrency conditions, with the HTTP/2 multiplexing features of gRPC allowing the full utilization of connection without the head-of-line blocking problems of the HTTP/1.1 used by conventional REST implementations.

The main reasons why GraphQL has higher latency features are the overhead of query parsing, validation and resolution. Complex nested queries that need multiple resolver executions add more processing time over REST with its simple request-response model. Nevertheless, in situations where network round-trip time is the most important factor in latency, the capability of GraphQL to load up related data on a single request can lead to lower total end-to-end latency of a request even though the per-request processing time is high.

B. Throughput Analysis

A study on the throughput properties has shown that REST has the best raw request-per-second measures in benchmark controlled environments. The ease of process model of request processing in REST, the availability of developed HTTP/1.1 server code and the ubiquitous optimization of serialization libraries in the field of JSON helps to achieve high throughput capability. This throughput benefit however, needs to be counterbalanced with the behaviour of REST to make several requests when it is necessary to retrieve complex data cases which may negate the per-request performance benefit when considering the end-to-end application paths.

gRPC exhibits competitive throughput behaviour, and in particular, this addresses the connection multiplexing of HTTP/2. The fact that the protocol can multiplex several outstanding requests to the same TCP connection minimizes the connection establishment overhead, and also enhances the total throughput in high-concurrent settings. Experiments on the performance of gRPC under different message sizes have shown that payloads between 1KB and 100KB have the best throughput due to the maximum efficiency of Protocol Buffer serialization but without paying an exorbitant connection overhead.

The nature of GraphQL throughput depends greatly on the query complexity and the effectiveness of the caching. Simple queries with less resolver execution can be used to reach similar throughput as that of REST, and complex queries with high data graph traversal have lower performance is adversely affected by the absence of in-built HTTP caching in the GraphQL implementations which requires an application-level caching policy to realize competitive throughput with read-intensive workloads.

C. Resource Utilization Patterns

Memory usage analysis during load testing reveals GraphQL demonstrates lower memory consumption at 17% compared to RESTful and gRPC implementations which exhibit identical memory at approximately 20%. This efficiency advantage stems from GraphQL's precise field selection, reducing the volume of data structures that must be instantiated and maintained in memory during request processing.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

The characteristics of CPU utilization are inverse, where the GraphQL utilization is higher than gRPC and REST during communication between microservices. Query parsing, validation and executing dynamic resolver overheads the CPU, making the use of gRPC more cost-effective to use with complex queries with deep nesting to large field selections. gRPC also shows efficient CPU usage with binary Protocol Buffer parsing that uses fewer computational units than JSON deserialization and has efficient wire format compression.

The network I/O efficiency of different protocols differs greatly, with gRPC having the most efficient network usage on Protocol Buffer compression, HTTP/2 which normally implies a 20-30% smaller payload that what would be in JSON based REST implementations.

GraphQL offers a middle way level of network efficiency as it selectively fetches fields removing over-fetching issues, but with the cost of serializing to JSON maintained. REST has the worst network bandwidth usage between verbose JSON encoding and the behaviour of giving back full resource representations when not needed by the client.

TABLE II Synthesis of Performance Characteristics Across Protocols

Performance Dimension	**REST**	**gRPC**	**GraphQL**	**Synthesis Quality**
Latency	1.6s avg (1000 records) Moderate baseline	2.0s avg Lowest for small messages Best P95 characteristics	2.9s avg Higher per-request 25-67% total RTT reduction	High consistency across studies
Throughput	Highest: 1500-2000 RPS 95-98% success rate	Moderate-High: 1200-1800 RPS 98-99% success rate	Moderate: 1000-1500 RPS 97-99% success rate	Moderate consistency
CPU Utilization	Moderate baseline JSON parsing overhead	Low-Moderate Binary efficiency advantage	High Query parsing/validation overhead	Limited cross-study consistency
Memory Usage	20% allocation JSON structure overhead	20% allocation Protocol buffer efficiency	17% allocation Selective instantiation	High consistency in limited studies

Performance metrics derived from primary empirical research [58, 59, 61, 63, 65] . Evidence strength refers to the number of studies contributing to each dimension; synthesis quality refers to the cross-study consistency. Critical observations: (1) Critical observations regarding the context of the latency comparison of REST-GraphQL are network RTT vs. processing time; (2) gRPC message size sensitivity means that workload-specific evaluation is required; (3) GraphQL's CPU-memory tradeoff means that careful capacity planning is needed.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

D. Scalability Characteristices

Empirical research on the protocol's behavior under increasing load conditions shows different scalability patterns. gRPC shows the best horizontal scalability characteristics and does not increase with the number of concurrent connections due to the usage of the HTTP/2 multiplexing protocol and the efficient connection pooling mechanism. The stateless nature of the protocol and efficient use of resources allow the protocol to be distributed across multiple instances of a server without substantial coordination overhead.

Scalability benefits of REST can be explained and described by the lack of state and a universal adoption of established caching strategies.

The HTTP-based caching, which can be deployed at various levels, such as client, an intermediate proxy, and a source server, offers a means of significantly decreasing the load of workloads with high percentage of read operations.

However, the nature of the RESTful paradigm that would cause repeated and sequential requests of related data can induce cascading failures when the load is high, with a latency experienced by a single service cascading throughout service chains.

In GraphQL scalability, it is important to be careful with how the limits of query complexities are implemented as well as how resolver optimization is done. The scalability in GraphQL should be achieved through the implementation of some advanced caching policies, the integration of data loader tools to address N + 1 query problems, and the sensible schema design determining the balance between flexibility and high-performance requirements.

Future Directions

The context of edge computing is a valuable future area of API protocol performance studies. The resource, network, and latency demands of edge deployments vary significantly compared to the traditional cloud or data center setups. Knowledge of protocol performance in edge scenarios such as scarce computational means, intermittent connectivity, and low network latency would be useful in architecture choices to the upcoming Internet of Things and edge computing applications.

The selection of protocols with the help of artificial intelligence is one of the promising research directions that make use of machine learning methods to choose the protocols according to the characteristics of the workload and the constraints of the infrastructure and performance requirements. Past execution history in a wide variety of deployment environments might be used to train models that would make predictions on the best choice of protocol to use in new application situations, which would automate architecture decisions that currently have to be made manually and through expert judgment.

There would be cost-performance analysis that includes the infrastructures cost, development cost, and operational overhead which would give more comprehensive decision frameworks in choosing the protocols. A study ought to be done on the total cost of ownership keeping in view the raw performance measurements but also development productivity, operation complexity and infrastructure efficiency. This kind of analysis would allow making informed trade-off analysis between performance maximization and resource constraints and budget limitations.

The study of hybrid architecture that seeks to analyze systems using more than one and two or more protocols at a time is a line of research that is significant based on the tendencies in the deployment of systems in the real world. Most systems that deploy REST also use external public API and gRPC to communicate with other systems (internal microservices) or GraphQL to communicate with particular client applications. The knowledge of the best patterns to use in the protocol mixing, the design of gateways, and implementation of the translation layer would be a good guide on the complex implementation of distributed systems architecture.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

CONCLUSION

This systematic literature review examines twenty 2020-2025 scientific articles and assesses performance comparison between REST, gRPC and GraphQL protocols in monolithic and distributed architectures. All protocols have their own advantages and disadvantages: gRPC is best used in low-latency, microservice-based applications, which use Protocol Buffer and HTTP/2 to make the best use of the network; REST can be used with massively higher throughput and better caching, whereas public APIs are more appropriate; and GraphQL provides better CPU and memory utilization and uses fetching data on-demand, although it has been observed to require more attention to performance. This analysis identifies gaps in the research, such as the lack of standardized three-way comparisons and production-scale analysis, which the future research should construct around benchmarking frameworks and domain-specific performance analysis. Finally, the choice of protocol must be supported with systematic consideration of application requirements and team capabilities and the protocols should be recommended to be integrated to combine their respective strengths and overcome architectural issues.

REFERENCES

1. V. Kanvar, R. Jain, and S. Tamilselvam, “Handling Communication via APIs for Microservices,” Aug. 02, 2023, arXiv: arXiv:2308.01302. doi: 10.48550/arXiv.2308.01302.
2. A. Lercher, “Managing API Evolution in Microservice Architecture,” May 2024, pp. 195–197. doi: 10.1145/3639478.3639800.
3. K. Alanezi and S. Mishra, “Utilizing Microservices Architecture for Enhanced Service Sharing in IoT Edge Environments,” IEEE Access, vol. PP, pp. 1–1, Jan. 2022, doi: 10.1109/ACCESS.2022.3200666.
4. Ł. Kamiński, M. Kozłowski, D. Sporysz, K. Wolska, P. Zaniewski, and R. Roszczyk, “Comparative review of selected Internet communication protocols,” Dec. 14, 2022, arXiv: arXiv:2212.07475. doi: 10.48550/arXiv.2212.07475.
5. R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” 2000.
6. “Analysis and Design of Microservices Architecture with GraphQL as an API Gateway for Higher Education Information System | Request PDF,” in ResearchGate, doi: 10.1109/ICOSTECH54296.2022.9829090.
7. M. Niswar et al., "Performance evaluation of microservices communication with REST, GraphQL, and gRPC," Int. Journal of Electronics and Telecommunications, 2024.
8. G. Brito and M. T. Valente, "REST vs GraphQL: A Controlled Experiment," 2020 IEEE International Conference on Software Architecture (ICSA), 2020.
9. [9] P. Ina and P. Hakik, “(PDF) GraphQL: A Comprehensive Analysis of Its Advantages, Challenges, and Best Practices in Modern API Development,” ResearchGate, Nov. 2025, doi: 10.55549/epstem.1598445.
10. M. Niswar, R. Arisandy Safruddin, A. Bustamin, and I. Aswad, “Performance evaluation of microservices communication with REST, GraphQL, and gRPC,” International Journal of Electronics and Telecommunications, pp. 429–436, June 2024, doi: 10.24425/ijet.2024.149562.
11. S. Weerasinghe and I. Perera, “Evaluating the Inter-Service Communication on Microservice Architecture,” Dec. 2022, pp. 1–6. doi: 10.1109/ICITR57877.2022.9992918.
12. Y. Gan and C. Delimitrou, “The Architectural Implications of Cloud Microservices,” IEEE Comput. Arch. Lett., vol. 17, no. 2, pp. 155–158, July 2018, doi: 10.1109/LCA.2018.2839189.
13. G. C. Desina, “Evaluating The Impact Of Cloud-Based Microservices Architecture On Application Performance,” May 19, 2023, arXiv: arXiv:2305.15438. doi: 10.48550/arXiv.2305.15438.
14. K. Barbara and C. Stuart, “Guidelines for performing Systematic Literature Reviews in Software Engineering,” Keele University and University of Durham, Keele, UK, Technical Report EBSE-2007-01, July 2007. [Online]. Available: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf
15. D. Bermbach and E. Wittern, “Benchmarking Web API Quality – Revisited,” Journal of Web Engineering, Oct. 2020, doi: 10.13052/jwe1540-9589.19563.
16. N. Bjørndal, M. Mazzara, A. Bucchiarone, N. Dragoni, and S. Dustdar, “Migration from Monolith to Microservices: Benchmarking a Case Study.,” The Journal of Object Technology, vol. 20, p. 3:1, Jan. 2021, doi: 10.5381/jot.2021.20.2.a3.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

17. M. Biehl, RESTful API Design. API-University Press, 2016.
18. O. Chaplia and H. Klym, “Designing a Node.js Project Architecture for RESTful Microservices,” Sept. 2023, pp. 808–811. doi: 10.1109/IDAACS58523.2023.10348681.
19. D. Damyanov and Z. Varbanov, “An application about server communication: Using the Command pattern on Web API requests,” in 2024 16th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), June 2024, pp. 1–5. doi: 10.1109/ECAI61503.2024.10607395.
20. Ś. Mariusz and P. Beata, “(PDF) Performance comparison of programming interfaces on the example of REST API, GraphQL and gRPC,” ResearchGate, Aug. 2025, doi: 10.35784/jcsi.2744.
21. S. D. Meglio and L. Libero Lucio Starace, “Evaluating Performance and Resource Consumption of REST Frameworks and Execution Environments: Insights and Guidelines for Developers and Companies,” IEEE Access, vol. 12, pp. 161649–161669, 2024, doi: 10.1109/ACCESS.2024.3489892.
22. G. Ilya, “High Performance Browser Networking - Google Books.” Accessed: Dec. 20, 2025. [Online]. Available: https://www.google.co.in/books/edition/High_Performance_Browser_Networking/KfW-AAAAQBAJ?hl=en&gbpv=1&dq=HTTP/1.1+vs+HTTP/2:+What%27s+the+Difference&pg=PA215&printsec=frontcover
23. A. Shatnawi, A. Bahri, B. Niang, and B. Verhaeghe, “Enhancing Data Serialization Efficiency in REST Services: Migrating from JSON to Protocol Buffers:,” in Proceedings of the 20th International Conference on Software Technologies, Bilbao, Spain: SCITEPRESS - Science and Technology Publications, 2025, pp. 193–200. doi: 10.5220/0013459500003964.
24. C. Rodríguez et al., “REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices,” in Web Engineering, A. Bozzon, P. Cudre-Maroux, and C. Pautasso, Eds., Cham: Springer International Publishing, 2016, pp. 21–39. doi: 10.1007/978-3-319-38791-8_2.
25. H. Subramanian and P. Raj, Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs. Packt Publishing Ltd, 2019.
26. S. Kanthed, “Rest vs. GraphQL: Comparative Analysis of API Design Approaches,” IJMRGE, vol. 4, no. 1, pp. 984–991, 2023, doi: 10.54660/.IJMRGE.2023.4.1.984-991.
27. M. Tomasz and K. rzegorz, “(PDF) A security analysis of authentication and authorization implemented in web applications based on the REST architecture,” ResearchGate, Aug. 2025, doi: 10.35784/jcsi.1925.
28. A. Gupta and S. Dubey, "Utilizing gRPC for High-Performance Inter-Service Communication in .NET," Int. Journal of Engineering Research & Technology, 2020.
29. R. Sikora and A. Postoliuk, "Comparative Analysis of Architectural Styles REST, GraphQL, and gRPC," Herald of Khmelnytskyi National University, 2025.
30. H. Babal, GRPC Microservices in Go. Simon and Schuster, 2023.
31. K. Indrasiri and D. Kuruppu, gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes. O’Reilly Media, Inc., 2020.
32. H. de Saxcé, I. Oprescu, and Y. Chen, “Is HTTP/2 really faster than HTTP/1.1?,” in 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Apr. 2015, pp. 293–299. doi: 10.1109/INFCOMW.2015.7179400.
33. J. Min and Y. Lee, “An Experimental View on Fairness between HTTP/1.1 and HTTP/2,” in 2019 International Conference on Information Networking (ICOIN), Jan. 2019, pp. 399–401. doi: 10.1109/ICOIN.2019.8718119.
34. J. B. Moreira, “Microservices with HTTP/1.1, HTTP/2 and HTTP/3, which one?,” in Proc. IEEE/ACM Conf. on Network Softwarization (NetSoft), Lisbon, Portugal, 2019, pp. 1–8.
35. C. Currier, “Protocol Buffers,” in Mobile Forensics – The File Format Handbook: Common File Formats and File Systems Used in Mobile Devices, C. Hummert and D. Pawlaszczyk, Eds., Cham: Springer International Publishing, 2022, pp. 223–260. doi: 10.1007/978-3-030-98467-0_9.
36. A. Shatnawi, A. Bahri, B. Niang, and B. Verhaeghe, “Enhancing Data Serialization Efficiency in REST Services: Migrating from JSON to Protocol Buffers:,” in Proceedings of the 20th International Conference on Software Technologies, Bilbao, Spain: SCITEPRESS - Science and Technology Publications, 2025, pp. 193–200. doi: 10.5220/0013459500003964.
37. J. Vanura and P. Kriz, "Performance Evaluation of Java, JavaScript and PHP Serialization Libraries for XML, JSON and Binary Formats,” in Services Computing – SCC 2018, J. E. Ferreira, G. Spanoudakis, Y. Ma, and L.-J. Zhang, Eds., Cham: Springer International Publishing, 2018, pp. 166–175. doi: 10.1007/978-3-319-94376-3_11.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

38. Ritu, S. Arora, A. Bhardwaj, A. Kukkar, and S. Kaur, “A Comparative Analysis of Communication Efficiency: REST vs. gRPC in Microservice- Based Ecosystems,” May 2024, pp. 621–626. doi: 10.1109/INNOCOMP63224.2024.00107.
39. I. Olivos and M. Johansson, "Comparative Study of REST and gRPC for Microservices," Bachelor's Thesis, Linköping University (DiVA portal), 2023.
40. V. Muniyandi, “Utilizing Grpc For High-Performance Inter-Service Communication In .NET,” May 15, 2025, Social Science Research Network, Rochester, NY: 5381441. doi: 10.2139/ssrn.5381441.
41. S. Atri, “gRPC as a low latency backbone for real time financial platforms,” International Journal of Leading Research Publication (IJLRP), vol. 6, no. 8, Aug. 2025, Art. no. IJLRP25081712.
42. [42] J. Kampars, D. Tropins, and R. Matisons, “A Review of Application Layer Communication Protocols for the IoT Edge Cloud Continuum,” in 2021 62nd International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS), Oct. 2021, pp. 1–6. doi: 10.1109/ITMS52826.2021.9615332.
43. G. Gupta and D. Gupta, "Performance Comparison of gRPC and REST for Microservices-based Applications," 2023 IEEE ICCES, 2023.
44. V. Touronen, Microservice Architecture Patterns with GraphQL, M.Sc. thesis, Dept. of Computer Science, University of Helsinki, Helsinki, Finland, Mar. 2019.
45. A. Quiña-Mera, P. Fernandez, J. M. García, and A. Ruiz-Cortés, “GraphQL: A Systematic Mapping Study,” ACM Comput. Surv., vol. 55, no. 10, p. 202:1-202:35, Feb. 2023, doi: 10.1145/3561818.
46. G. Brito et al., "An Empirical Study of GraphQL Schemas," 2019 IEEE 26th International Conference (SANER), 2019.
47. G. Brito et al., "An Empirical Study of GraphQL Schemas," 2019 IEEE 26th International Conference (SANER), 2019.
48. R. N. Muzaki and A. Salam, “Reducing Under-Fetching And Over-Fetching In Rest Api With Graphql For Web-Based Software Development,” Jurnal Teknik Informatika (Jutif), vol. 5, no. 2, pp. 447–453, Apr. 2024, doi: 10.52436/1.jutif.2024.5.2.1725.
49. P. Roksela, M. Konieczny, and S. Zielinski, “Evaluating execution strategies of GraphQL queries,” in 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), July 2020, pp. 640–644. doi: 10.1109/TSP49548.2020.9163501.
50. T. Taskula, Advanced Data Fetching with GraphQL: Case Bakery Service, Master’s thesis, School of Science, Aalto University, Espoo, Finland, Feb. 2019.
51. O. Ali, "Popular API Technologies: REST, GraphQL, and gRPC," Karelia University of Applied Sciences Publications, 2025.
52. O. Hartig and J. Pérez, "Semantics and Complexity of GraphQL," Proceedings of the 2018 World Wide Web Conference (WWW '18), ACM, 2018.
53. M. Seabra, M. F. Nazário, and G. Pinto, “REST or GraphQL? A performance comparative study,” in Proc. XIII Brazilian Symp. on Software Components, Architectures, and Reuse (SBCARS ’19), Salvador, Brazil, Sept. 2019, pp. 1–10, doi: 10.1145/3357141.3357149.
54. A. Quiña-Mera et al., "GraphQL or REST for Mobile Applications?" Springer Nature (ARTIIS), 2022.
55. M. A. Dhika, D. Khairani, S. U. Masruroh, A. Fiade, V. Arifin, and W. A. Tsaqofi, “Comparing GraphQL and ReST Architecture in Arabic Learning Games: A Quality of Service (QoS) Approach,” in 2023 11th International Conference on Cyber and IT Service Management (CITSM), Nov. 2023, pp. 1–5. doi: 10.1109/CITSM60085.2023.10455108.
56. L. Sailer, Mapping GraphQL Queries to SQL, Master’s thesis, Database Systems Research Group, Wilhelm-Schickard-Institut für Informatik, Mathematisch-Naturwissenschaftliche Fakultät, Eberhard Karls Universität Tübingen, Tübingen, Germany, Sept. 2023.
57. S. Chandra and A. Farisi, “Comparative Analysis of RESTful, GraphQL, and gRPC APIs: Performance Insight from Load and Stress Testing,” SISFOKOM, vol. 14, no. 1, pp. 81–85, Jan. 2025, doi: 10.32736/sisfokom.v14i1.2375.
58. M. N. Hedelin, Benchmarking and Performance Analysis of Communication Protocols: A Comparative Case Study of gRPC, REST, and SOAP, Master’s thesis, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, June 2024.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

59. I. A. Khan, H. Mishra, and K. Choubey, “A comparative analysis of REST and GraphQL APIs: Performance, efficiency, and developer experience,” International Journal of Advanced Multidisciplinary Scientific Research (IJAMSR), vol. 8, no. 4, pp. 29–39, Apr. 2025, doi: 10.31426/ijamsr.2025.8.4.8212.
60. M. Bolanowski et al., "Efficiency of REST and gRPC Realizing Communication Tasks in Microservice-Based Ecosystems,” in Frontiers in Artificial Intelligence and Applications, H. Fujita, Y. Watanobe, and T. Azumi, Eds., IOS Press, 2022. doi: 10.3233/FAIA220242.
61. R. Jin, R. Cordingly, D. Zhao, and W. Lloyd, “GraphQL vs. REST: A Performance and Cost Investigation for Serverless Applications,” in Proceedings of the 10th International Workshop on Serverless Computing, Hong Kong Hong Kong: ACM, Dec. 2024, pp. 37–42. doi: 10.1145/3702634.3702956.
62. R. Jin, GraphQL vs. REST: Performance and Scalability Analysis for Serverless Applications, Master’s thesis, Dept. of Computer Science and Systems, University of Washington, Tacoma, WA, USA, 2025.
63. R. Sikora and A. Postoliuk, “Comparative Analysis Of The Effectiveness Of Architectural Styles Rest, Graphql, And Grpc For Scalable Microservice Systems,” Herald of Khmelnytskyi National University. Technical sciences, vol. 355, no. 4, pp. 560–568, Aug. 2025, doi: 10.31891/2307-5732-2025-355-79.
64. M. Niswar, R. Arisandy Safruddin, A. Bustamin, and I. Aswad, “Performance evaluation of microservices communication with REST, GraphQL, and gRPC,” International Journal of Electronics and Telecommunications, pp. 429–436, June 2024, doi: 10.24425/ijet.2024.149562.
65. O. Ali, “Popular API Technologies: REST, GraphQL, and gRPC,” Karelia University of Applied Sciences, Joensuu, Finland, 2025.