

Article

An SLA-Aware Priority Management System for HTTP/2 Based on RFC 9218: Design, Implementation, and Performance Evaluation in Service-Based Architectures

Ahmed Lateef Salih Al-Karawi ¹ and Rafet Akdeniz ^{2,*}

¹ Defne Telekomünikasyon A.Ş., Maslak Mahallesi, Maslak Meydan Sokak, Spring Giz Plaza, No:5, İç Kapı:37, Kat:9, Sarıyer, İstanbul 34485, Türkiye; ahmedal-karawi@stu.aydin.edu.tr

² Department of Computer Engineering, Faculty of Engineering and Natural Sciences, Atlas University, Anadolu Caddesi No:40, Kağıthane, İstanbul 34408, Türkiye

* Correspondence: rafet.akdeniz@atlas.edu.tr; Tel.: +90-5423-427-501

Abstract

Service-Based Architectures (SBAs) in 5G core and cloud-native deployments require differentiated treatment for service classes with heterogeneous latency, reliability, and throughput expectations. Although HTTP/3 over QUIC is an important evolution of the HTTP ecosystem, HTTP/2 remains operationally relevant in SBA environments where TCP/TLS-based infrastructures and 3GPP service-based interfaces continue to rely on HTTP/2 communication. This paper therefore focuses on HTTP/2 priority signaling and the problem of translating application-level Service Level Agreement (SLA) policies into protocol-level priority metadata. To address this problem, the paper presents an SLA-aware priority management system built around the RFC 9218 extensible prioritization scheme, specifically its urgency and incremental parameters. The system integrates three coordinated subsystems: a rule-based Priority Classification Engine (PCE), a feedback-driven Dynamic Priority Mapping Algorithm (DPMA), and a runtime priority-update manager that applies bounded priority adjustments under changing network and load conditions. The revised evaluation reports a 7200-observation baseline campaign covering four operating modes, ten service classes, nine network profiles, and twenty repetitions per service–profile–mode combination, together with a 14,880-observation scalability and overhead campaign across increasing concurrent-stream levels. Compared with the unmanaged HTTP/2 baseline, DPMA reduced mean latency by 24.8%, P95 latency by 35.1%, P99 latency by 38.0%, and SLA violations by 19.9 percentage points. Compared with the legacy RFC 7540 baseline, DPMA reduced mean latency by 39.0%, P95 latency by 49.3%, P99 latency by 49.9%, and SLA violations by 21.1 percentage points. Compared with the static RFC 9218 baseline, DPMA reduced mean latency by 38.7%, P95 latency by 48.1%, P99 latency by 50.6%, and SLA violations by 21.4 percentage points. The scalability analysis shows that DPMA maintained P95 latency between 126.8 ms and 128.2 ms over the tested 1–100 concurrent-stream range, with priority-update decision overhead below 0.004 ms per request. The results indicate that SLA-aware use of RFC 9218 priority metadata can improve latency and SLA-compliance behavior in controlled SBA-like HTTP/2 environments while preserving a transparent and auditable prioritization policy.



Academic Editor: Reiko Heckel

Received: 23 May 2026

Revised: 29 June 2026

Accepted: 13 July 2026

Published: 17 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](https://creativecommons.org/licenses/by/4.0/)

[Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: 5G; HTTP/2; microservices; network performance; priority management; quality of service; RFC 9218; service level agreement; service-based architecture

1. Introduction

The transition from monolithic web services to Service-Based Architectures (SBAs) has resulted in systems composed of tens to hundreds of independent microservices with heterogeneous latency, reliability, and throughput requirements [1,2]. This complexity is particularly evident in 5G network deployments where service classes such as Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and Massive Machine Type Communication (mMTC) place different and sometimes competing demands on latency, reliability, and capacity [3,4]. In the 5G core, the technical realization of service-based interfaces specifies HTTP/2-based communication with JSON serialization and TLS protection, which makes HTTP/2 prioritization directly relevant to SBA signaling and service interaction rather than only to conventional web-page delivery [5,6].

This study focuses on HTTP/2 rather than HTTP/1.x because HTTP/2 remains deployed in existing SBA and cloud-native infrastructures, while HTTP/3 over QUIC, although increasingly important, introduces a different transport substrate and a different set of deployment assumptions [7,8]. Recent comparative work indicates that QUIC/TCP and HTTP/3/HTTP/2 trade-offs are condition-dependent rather than uniformly one-directional, especially under network stress and attack scenarios [7]. Therefore, a focused HTTP/2 study remains technically justified when the target environment is an HTTP/2-based SBA stack and the research question concerns how application-level SLA requirements can be translated into HTTP priority signals.

HTTP/2 was originally specified in RFC 7540 [9] and is currently defined by RFC 9113, which deprecates the RFC 7540 priority scheme while retaining the legacy frame definitions for interoperability [10]. The original scheme used a dependency tree in which each stream could be assigned a weight and an optional parent stream. Although expressive, this model has shown inconsistent performance in practice because implementations differ in how they interpret, ignore, or approximate priority signals [11,12]. The primary challenge lies in deriving efficient scheduling decisions from arbitrary dependency trees, as the protocol does not mandate a single scheduling algorithm.

To address these limitations, the Internet Engineering Task Force (IETF) introduced RFC 9218, “Extensible Prioritization Scheme for HTTP,” in June 2022 [13]. This standard defines an extensible priority signal based on two parameters: urgency ($u \in [0, 7]$, where 0 represents the highest priority), and incremental (i , a Boolean). These parameters are communicated via the Priority HTTP header or the PRIORITY_UPDATE frame. This simplified design reduces implementation complexity while retaining sufficient flexibility for practical prioritization needs [13].

However, RFC 9218 intentionally specifies priority signaling semantics rather than an SLA-management policy. It does not define how application-level Service Level Agreement (SLA) requirements should be translated into urgency values, nor how dynamic conditions should trigger priority updates. Static mappings between service classes and urgency levels fail to adapt to runtime factors such as congestion, server overload, or SLA deadline proximity. These limitations motivate the development of an adaptive, SLA-aware prioritization framework. In this work, QoS is treated operationally through five measurable outcomes: latency, SLA violation rate, throughput, success rate, and priority differentiation. Latency and SLA violation rate are the primary optimization targets, while throughput and success rate are monitored to expose trade-offs rather than to claim simultaneous optimization of all QoS dimensions.

The remainder of this paper is organized as follows: Section 2 reviews related work and positions the HTTP/2 scope relative to HTTP/3. Section 3 presents the system architecture. Section 4 describes the implementation details. Section 5 provides experimental evaluation

and explains the metric selection. Section 6 discusses the results and deployment scope. Section 7 concludes the paper.

The main contributions of this work are summarized as follows:

- C1 (System Architecture): A complete SLA-aware HTTP/2 priority management framework for SBA environments, comprising a Priority Classification Engine, a Dynamic Priority Mapping Algorithm, and a PRIORITY_UPDATE Manager, designed for integration with existing HTTP/2 server stacks.
- C2 (SLA-Driven Classification): A rule-based classification mechanism that maps service requests to RFC 9218 urgency levels based on service type, SLA constraints, HTTP method, URL patterns, and runtime network metrics, implemented using a human-readable YAML configuration with runtime reload capability.
- C3 (Dynamic Adaptation): A feedback-driven control loop that dynamically adjusts priorities of active streams based on system load, network congestion, and packet error rate, with bounded update intervals to improve stability and avoid unbounded priority escalation.
- C4 (Experimental Evaluation): A controlled evaluation consisting of 7200 baseline observations across four operating modes, ten service categories, nine network conditions, and twenty repetitions per service–profile–mode combination, complemented by a 14,880-observation scalability and overhead analysis across increasing concurrent-stream levels.

2. Related Work and Background

2.1. HTTP/2 Prioritization: RFC 7540 and Its Limitations

The prioritization model originally defined in RFC 7540 assigns each stream a weight in the range 1–256 and an optional dependency, forming a hierarchical priority tree [9]. RFC 9113 retains the frame format but deprecates the semantics of this priority scheme and recommends the HTTP extensible priority mechanism instead [10]. In principle, servers may use weights and dependency relationships to guide resource allocation; in practice, behavior has varied substantially across implementations.

Empirical work on HTTP/2 prioritization shows that priority behavior can materially affect web performance and that incorrect or inconsistent priority handling can reduce the benefit of multiplexing [11]. Comparative studies of HTTP/2 and HTTP/3 further indicate that resource multiplexing, transport behavior, and priority scheduling interact in non-trivial ways [12]. These findings support the view that the RFC 7540 priority model is difficult to implement consistently across heterogeneous client–server systems.

2.2. RFC 9218: Design Rationale and Key Parameters

To address these limitations, RFC 9218 introduces a simplified prioritization model based on two scalar parameters: urgency ($u \in [0, 7]$) and incremental (i) [13]. The urgency parameter defines relative priority levels, where lower values indicate higher priority, while the incremental flag indicates whether responses can be delivered progressively.

This design significantly reduces scheduling complexity by eliminating the dependency tree and enabling queue-based prioritization strategies. Implementations can maintain a priority queue ordered by urgency, with optional preference for incremental streams under congestion [13]. Recent measurement work on the extensible priority scheme, particularly in HTTP/3 deployments, shows increasing implementation interest but also highlights that support and usage remain implementation-dependent [14]. Recent EPS-oriented studies also explore how urgency and incremental delivery can improve web Quality of Experience in HTTP/3, reinforcing that extensible priority signaling is an active research topic rather than only a standards issue [14]. Existing studies focus mainly on

web-resource loading, transport behavior, and browser/server support; they do not define an SLA-aware control loop for SBA workloads or evaluate SLA-class differentiation under controlled service-profile and network-stress combinations.

2.3. HTTP/3/QUIC Context and Justification for the HTTP/2 Scope

HTTP/3 maps HTTP semantics over QUIC and changes the transport substrate from TCP to QUIC/UDP while preserving the general HTTP request–response semantics used by applications [8]. This evolution is important because QUIC can reduce transport-level head-of-line blocking and supports connection-management features that are not available in TCP-based HTTP/2. Nevertheless, the existence of HTTP/3 does not eliminate HTTP/2 as a research target. In SBA environments, 3GPP service-based interfaces continue to specify HTTP/2 protocol usage, and many operational deployments still depend on HTTP/2-capable proxies, gateways, and service meshes [6].

Recent comparative studies also show that the HTTP/2/TCP versus HTTP/3/QUIC decision depends on network condition, implementation, and threat model. For example, Simpson et al. report that QUIC and TCP exhibit different efficiency and security behavior under denial-of-service, man-in-the-middle, and traffic-analysis scenarios rather than one protocol dominating in all cases [7]. Similarly, HTTP/3 priority measurements show that the extensible prioritization scheme remains implementation-dependent in practice [14]. This paper therefore positions HTTP/3 as an important adjacent protocol, while limiting the experimental scope to HTTP/2 because the research objective is SLA-to-priority mapping for HTTP/2-based SBA communication, not a general HTTP version benchmark.

2.4. Service-Based Architectures and QoS in 5G

The 5G system architecture defined in 3GPP TS 23.501 adopts a Service-Based Architecture (SBA) where Network Functions (NFs) interact through service-based interfaces [5]. The technical realization of these interfaces is specified in 3GPP TS 29.500, which defines the protocol stack for SBA communication and uses HTTP/2 with JSON serialization and TLS protection [6]. These interactions are governed by Service Level Agreements (SLAs) specifying constraints on latency, throughput, availability, and reliability.

Different service classes impose distinct QoS requirements. As discussed by Taleb et al. [15], eMBB focuses on high throughput, URLLC requires ultra-low latency and high reliability, and mMTC emphasizes massive connectivity with relaxed latency constraints. Existing QoS-aware frameworks, including network slicing approaches, primarily operate at the network or orchestration layer [16]. Consequently, application-layer prioritization, particularly within HTTP/2-based communication, remains insufficiently addressed.

2.5. SLA-Aware and Learning-Based Resource Management

Recent cloud-native resource-management research increasingly uses learning-based or analytically assisted control to preserve SLOs under dynamic workloads. Examples include ML-based QoS-aware resource managers for microservices, fine-grained SLO-oriented reprovisioning frameworks, lightweight analytical controllers that decompose end-to-end latency budgets across service tiers, and bi-level learning-assisted controllers for SLO-targeted microservices [17–20]. These approaches are important contemporary comparators for resource allocation, autoscaling, and service-level control. However, they solve a different control problem from the one addressed here: they allocate CPU, memory, throttling targets, or service-level resources, whereas this paper focuses on translating application-level SLA policies into HTTP/2 priority metadata.

The rule-based DPMA was selected for three methodological reasons: First, SLA-to-priority mappings used in operational environments must be auditable and explainable to operators, especially when high-priority treatment affects competing service classes. Second, RFC 9218 defines a compact urgency space with eight ordered levels, making deterministic mappings a natural and transparent mechanism for validating the relationship between SLA budgets and HTTP priority metadata. Third, a rule-based policy provides a reproducible reference design against which future learning-assisted schedulers can be compared. Learning-based controllers such as Sinan, FIRM, Ursa, and Autothrottle address related SLO-control problems, but their primary control variables are service resources, throttling, or multi-tier allocation decisions rather than direct HTTP priority metadata assignment [17–20]. The present work therefore uses a deterministic policy to keep the prioritization logic interpretable while still allowing for empirical comparison against unmanaged, legacy RFC 7540, and static RFC 9218 configurations.

2.6. SLA Management in Distributed Systems

SLA management mechanisms in distributed systems typically focus on monitoring, violation detection, prediction, and resource allocation. Keller and Ludwig [21] formalized SLA management for web services, while Buyya et al. [22] established cloud-oriented concepts such as elastic provisioning and service-level resource management. Later work investigated predictive SLA-violation detection using runtime monitoring and learning-based methods [23]. However, these systems primarily target infrastructure-level or application-level adaptation and do not integrate SLA awareness into HTTP prioritization mechanisms.

2.7. Summary of Research Gaps

The existing literature reveals several critical gaps that motivate this work:

- Existing RFC 9218 deployments and studies generally focus on protocol-level priority signaling rather than explicit SLA-to-priority translation for SBA workloads.
- HTTP/3/QUIC studies provide important transport-level comparisons, but they do not remove the need to study HTTP/2 in environments where 3GPP SBA interfaces and existing service infrastructures continue to rely on HTTP/2.
- Static mappings between service classes and urgency levels fail to adapt to dynamic network conditions such as congestion, server load, and packet error rate.
- The PRIORITY_UPDATE mechanism remains underutilized for proactive, SLA-driven priority adjustment.
- Prior studies provide limited evidence on multi-condition SLA-aware adaptation under controlled SBA-like workloads with explicit latency, SLA-violation, throughput, and success-rate measurements.

3. System Design and Architecture

3.1. Overview

The proposed system is designed as a middleware layer positioned between HTTP/2 clients and application servers. It intercepts incoming requests, performs classification, assigns RFC 9218-compatible Priority metadata, and applies bounded runtime priority adjustments through the prototype control path. This design evaluates the SLA-aware decision logic and the priority-control workflow independently of vendor-specific server internals, which is important because RFC 9218 standardizes priority signaling semantics but leaves concrete scheduling decisions to implementations.

Table 1 summarizes the implementation boundary used in the evaluation.

Table 1. Implementation boundary for RFC 9218-related behavior.

Component	Evaluation Role	Interpretation
SLA classification rules	Implemented	Operational middleware logic for service classification
Urgency and incremental mapping	Implemented	RFC 9218-compatible priority-metadata assignment
Runtime priority-state registry	Implemented	Control state for active requests and bounded updates
Priority-update action	Implemented in the prototype control path	Dynamic adaptation of request priority state under runtime conditions
Server/proxy scheduler integration	Deployment-dependent	The design is compatible with implementations that preserve and act on RFC 9218 metadata
HTTP/3/QUIC execution	Discussed as cross-protocol context	Outside the HTTP/2 experimental scope of this study

The design has four goals: (G1) protect latency-sensitive service classes by assigning low urgency values to deadline-critical traffic; (G2) prevent uncontrolled promotion of ordinary traffic by bounding urgency adjustments; (G3) translate explicit SLA policies into protocol-level priority metadata; and (G4) preserve deployability by separating SLA decision logic from the HTTP/2 server or proxy implementation. The system does not attempt to optimize all QoS metrics simultaneously. Instead, it prioritizes latency and SLA-compliance objectives for critical services while measuring throughput, success rate, and lower-priority delay as trade-off indicators.

The overall architecture consists of four tightly coupled subsystems, as illustrated in Figure 1:

- Priority Classification Engine (PCE): Responsible for analyzing request-level attributes and assigning an internal priority level based on service type, SLA requirements, and contextual metadata.
- Dynamic Priority Mapping Algorithm (DPMA): Translates internal priority levels into RFC 9218 parameters (u, i), where urgency (u) and incremental (i) values are dynamically adjusted according to real-time network conditions and system state.
- PRIORITY_UPDATE Manager (PUM): Maintains state information for active streams, detects potential SLA violations, and proactively issues PRIORITY_UPDATE frames to adjust stream priorities during execution.
- SLA Monitoring Framework (SMF): Continuously collects system-level and network-level metrics, including latency, throughput, and packet error rate, and provides feedback to both the DPMA and PUM for adaptive decision-making.

The middleware design preserves backward compatibility at the HTTP-message level. Servers that do not implement RFC 9218 scheduling can still process the forwarded requests because unknown priority metadata can be ignored, while servers or intermediaries that support RFC 9218 can use the supplied urgency and incremental signals. The actual scheduling benefit therefore depends on the selected HTTP/2 stack and its support for extensible priority handling.

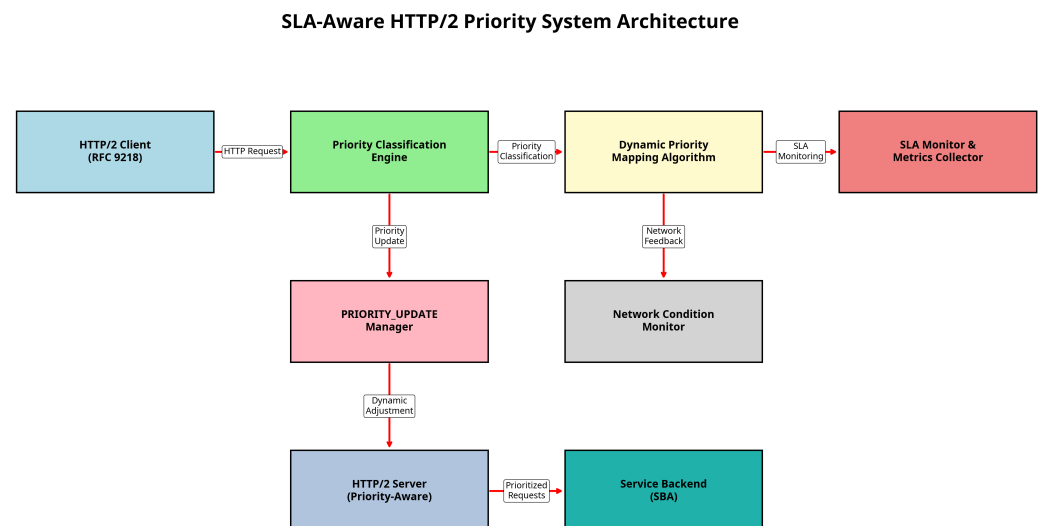


Figure 1. Architecture of the SLA-Aware HTTP/2 Priority Management System.

3.2. Priority Classification Engine

3.2.1. Priority Levels

The Priority Classification Engine (PCE) assigns each incoming request to one of four internal priority levels based on SLA requirements, as summarized in Table 2.

Table 2. Internal priority levels and associated SLA parameters.

Level	Label	Urgency (u)	Incremental (i)	Illustrative SLA Target	Typical Use
0	URLLC	0	true	<1 ms	Safety alarms, control signaling
1	Interactive	1	false	<10 ms	User-facing transactions
2	Standard	3	false	<200 ms	API queries, database access
3	Background	6	true	<15,000 ms	Analytics, bulk transfer

The values in Table 2 define the experimental policy used in this study. They should not be interpreted as universal QoS thresholds for all HTTP/2 deployments. Operators can tune the SLA targets, urgency values, and incremental flags according to service-domain requirements, but the central design principle remains unchanged: deadline-critical traffic receives lower urgency values, while delay-tolerant traffic is prevented from consuming high-priority resources by default. The revised evaluation examines this policy under ten service categories and nine network profiles, which exposes how the mapping behaves under normal, congested, error-prone, and mixed-impairment conditions.

3.2.2. Classification Criteria

The PCE evaluates each request against an ordered rule set. Rules are processed sequentially, and the first matching rule determines the assigned priority level. Each rule may include one or more of the following attributes:

- URL path pattern: Regular expressions applied to the request target. For example, paths under `/api/v*/control/*` or `/api/v*/emergency/*` map to Level 0, while `/api/v*/analytics/*` or `/api/v*/logs/*` map to Level 3.
- HTTP method: POST requests targeting actuator endpoints are promoted by one priority level relative to equivalent GET requests.
- Custom headers: The presence of `X-Service-Type`: URLLC enforces Level 0 assignment. The `X-Critical` flag promotes requests to at least Level 1.
- Source identity: Client IP addresses or bearer-token claims can trigger user-class-based priority adjustments.

Rules are defined in YAML format and can be reloaded at runtime without requiring a server restart. Requests that do not match any rule are assigned Level 2 (Standard) by default to prevent uncontrolled consumption of high-priority resources.

3.3. Dynamic Priority Mapping Algorithm

3.3.1. Base Mapping

Each internal priority level is mapped to default RFC 9218 parameters. Specifically, Level 0 maps to $(u = 0, i = \text{true})$, Level 1 to $(u = 1, i = \text{false})$, Level 2 to $(u = 3, i = \text{false})$, and Level 3 to $(u = 6, i = \text{true})$.

3.3.2. Dynamic Adjustment Rules

The Dynamic Priority Mapping Algorithm (DPMA) continuously monitors three network metrics: congestion level $C \in [0, 1]$, server utilization $S \in [0, 1]$, and packet error rate $E \in [0, 1]$. These metrics are updated every 100 ms using exponential moving averages.

The adjusted urgency u_adj is computed from the base urgency u_0 as follows:

```

if C > 0.7:
if level in {0,1}: u\_adj = max(0, u0 - 1)
if level in {2,3}: u\_adj = min(7, u0 + 1)

if S > 0.8:
if level in {2,3}: u\_adj = min(7, u0 + 2)

if E > 0.1:
if level == 0: u\_adj = 0

u\_adj = clamp(u\_adj, 0, 7)

```

The 100 ms update interval represents a trade-off between responsiveness and stability. Shorter intervals may introduce oscillatory behavior, whereas longer intervals increase the risk of SLA violations.

3.4. PRIORITY_UPDATE Manager

The PRIORITY_UPDATE Manager (PUM) maintains a mapping between active stream identifiers and their current (u, i) assignments, along with estimated SLA deadlines. The following events trigger PRIORITY_UPDATE frames:

- SLA deadline proximity: When a stream exceeds 80% of its SLA budget, urgency is increased (i.e., $u \leftarrow \max(0, u - 1)$).
- Network metric changes: A variation exceeding 20% in C , S , or E between consecutive measurements triggers updates for all active streams.
- Load rebalancing: When Level 0 streams exceed a configurable threshold (default: 16), Level 1 streams are temporarily promoted to prevent starvation.

- Application-triggered updates: External applications may invoke a REST interface to request explicit priority adjustments.

3.5. SLA Monitoring Framework

The SLA Monitoring Framework (SMF) records per-request metrics in a local SQLite database with a 72-h retention window. Collected metrics include end-to-end latency, server processing time, HTTP status codes, assigned priority levels, urgency values, and concurrent stream counts.

Aggregate statistics (mean, median, P95, and P99 latency, success rate, and SLA violation rate) are computed every 5 s using a sliding 60 s window. These metrics are exposed via a REST API to enable integration with external monitoring and orchestration systems.

4. Implementation

4.1. Software Stack

The middleware prototype was implemented in Python 3.11. A Flask-based management layer handled experiment orchestration and observability, while HTTP/2 frame-level behavior was represented through an h2-style control abstraction. Because RFC 9218 PRIORITY_UPDATE support is implementation-dependent, the prototype should not be described as a production-grade HTTP/2 server with native RFC 9218 scheduling. Production deployment would require an HTTP/2 server, reverse proxy, or extension module that explicitly supports the required priority signaling and exposes a scheduler capable of acting on it.

Table 3 summarizes the core software dependencies used in the implementation.

Table 3. Software dependencies.

Component	Library/Technology	Version
HTTP/2 framing	h2	4.x
Web framework	Flask	3.x
Async I/O	Python asyncio	stdlib
Configuration	PyYAML	6.x
Metrics storage	SQLite (sqlite3)	stdlib
Rule compilation	re (compiled RegEx)	stdlib

4.2. Priority Classification Engine Implementation

The Priority Classification Engine (PCE) is implemented as a rule-driven classification module responsible for mapping each incoming request to an internal priority level. Rule sets are loaded from a YAML configuration file and compiled at initialization and during hot-reload events to ensure efficient runtime evaluation.

The classification process is based on an ordered rule set. Each incoming request is evaluated sequentially against the rules, and the first matching rule determines the assigned priority level. If no rule matches, the request is assigned a default Level 2 (Standard) classification.

In Listing 1, `path` denotes the HTTP request target, `method` denotes the HTTP verb, `headers` denotes the request-header dictionary, `source_ip` denotes the client or upstream source address, `rule` denotes one YAML-defined classification rule, and `Priority_Info` denotes the returned tuple containing internal priority level, adjusted urgency, incremental flag, matched rule name, and request identifier.

Listing 1. Priority Classification Procedure.

```

Input: Request (path, method, headers, source_ip)
Output: PriorityInfo object
request_id ← UUID()
for each rule in rule set do
    if MatchRule(rule, request) = true then
        u_adj ← ApplyDynamicAdjustments(rule.urgency, rule.level)
        return Priority_Info(rule.level, u_adj, rule.incremental, rule.name, request_id)
end for
return Default_Classification(request_id)

```

The rule-matching process evaluates only the attributes defined within each rule. URL path conditions are checked using pre-compiled regular expressions, header constraints are verified through constant-time dictionary lookups, and source identity filtering is performed using pre-parsed address rules.

In Listing 2, `R.path_pattern` denotes the compiled path-matching expression, `R.method` denotes the rule's allowed HTTP method or method set, and required headers denote key-value constraints specified in the YAML policy. The Boolean output is true only when all rule constraints are satisfied.

Listing 2. Rule Matching Procedure.

```

Input: Rule R, Request (path, method, headers, source_ip)
Output: Boolean
if path does not match R.path_pattern then return false
if method does not match R.method then return false
if required headers are not satisfied then return false
if source_ip is not allowed then return false
return true

```

This design ensures low computational overhead in the request-processing path. Rule compilation is performed offline, eliminating runtime regex compilation costs. Header evaluation operates in constant time, and the sequential rule evaluation introduces a worst-case complexity of $O(N)$, where N is the number of rules. In practice, the rule set remains small, and classification latency is negligible under typical operating conditions.

Furthermore, the first-match rule strategy provides deterministic and interpretable behavior, which is critical for SLA-driven systems. The default classification mechanism prevents unclassified traffic from being promoted to high-priority levels, thereby preserving resources for latency-sensitive services.

4.3. Dynamic Priority Adjustment Implementation

The Dynamic Priority Mapping Algorithm (DPMA) is implemented as a runtime adjustment function that modifies the base urgency value according to current network conditions. The function `_apply_dynamic_adjustments` applies rule-based transformations using three monitored metrics: congestion level (C), server load (S), and packet error rate (E). These metrics are updated by a background coroutine that polls the SLA Monitoring Framework (SMF) REST API every 100 ms.

In Listing 3, u_0 denotes the base urgency assigned by the PCE, u denotes the working urgency value, u_{adj} denotes the final adjusted urgency, and L denotes the internal priority level. The functions $\max(\cdot)$, $\min(\cdot)$, and the final clamp operation ensure that the returned urgency remains within the valid RFC 9218 range $[0, 7]$.

Listing 3. Dynamic Priority Adjustment Procedure.Input: base_urgency u_0 , priority_level L , metrics (C, S, E) Output: adjusted urgency u_{adj} $u \leftarrow u_0$ If $C > 0.7$: If $L \leq 1$: $u \leftarrow \max(0, u - 1)$ Else: $u \leftarrow \min(7, u + 1)$ If $S > 0.8$ and $L \geq 2$: $u \leftarrow \min(7, u + 2)$ If $E > 0.1$ and $L = 0$: $u \leftarrow 0$ Return $\max(0, \min(7, u))$

This design supports adaptive prioritization under dynamic network conditions. High-priority traffic is protected during congestion, while lower-priority traffic is progressively deprioritized. The bounded adjustment mechanism reduces the risk of oscillatory behavior and ensures that urgency values remain within the valid RFC 9218 range.

Fairness is supported through bounded urgency adjustment and continued service eligibility for lower-priority traffic. The DPMA keeps all urgency values within the RFC 9218 range and avoids unrestricted promotion of individual requests. In operational deployments, high-priority classification should be bound to authenticated service identity, untrusted client-supplied priority headers should be rewritten at the ingress boundary, and high-priority classifications should be rate-limited and audited. These controls strengthen the prioritization mechanism against priority-escalation abuse while preserving the intended SLA mapping.

4.4. PRIORITY_UPDATE Manager Implementation

The PRIORITY_UPDATE Manager (PUM) is responsible for updating stream priorities during runtime. It maintains a registry of active requests and triggers updates when priority changes are required. The system uses an `asyncio.PriorityQueue` to ensure that streams closest to SLA deadlines are processed first.

In Listing 4, `request_id` identifies the active request or stream tracked by the middleware, `new_priority` denotes the updated internal priority object, `active_requests` denotes the in-memory mapping from request identifiers to current priority state, and `update_callbacks` denotes the registered integration hooks used to translate control decisions into stack-specific priority-update actions.

Listing 4. Priority Update Management Procedure.Input: `request_id`, `new_priority`State: `active_requests`, `update_callbacks`If `request_id` exists in `active_requests`: `old_priority` $\leftarrow$ `active_requests[request_id]` `active_requests[request_id]` $\leftarrow$ `new_priority` For each callback in `update_callbacks`: `callback(request_id, old_priority, new_priority)`

Callback functions registered by the HTTP/2 server or proxy layer translate priority changes into RFC 9218-compatible priority updates when the selected stack supports the required signaling. This decoupled design enables flexible integration while maintaining a clear separation between control logic and transport-layer signaling.

For M active streams, the PUM maintains $O(M)$ state. Priority-change lookup is constant time for a known request identifier, while queue operations depend on the selected priority-queue implementation and are typically $O(\log M)$. The present experiments did not separately instrument CPU cycles, memory allocation, frame-serialization time, or

control-loop latency for each PRIORITY_UPDATE operation; therefore, the computational-overhead discussion is analytical rather than a measured overhead benchmark.

4.5. Management Interface

A browser-based management interface is implemented using HTML5 and JavaScript to support operational control and monitoring. The interface provides four primary capabilities:

- A real-time monitoring dashboard with 5 s auto-refresh for system metrics.
- A rule editor for modifying YAML-based classification rules without restarting the system.
- A request simulation tool for validating configuration changes prior to deployment.
- A performance analytics module with exportable charts for offline analysis.

All interface components interact with the system through the same REST API exposed to external monitoring tools. This unified design ensures consistency between graphical and programmatic access while simplifying system integration.

5. Experimental Evaluation

5.1. Evaluation Objectives and Metric Selection

The evaluation is designed to test whether the proposed SLA-aware mapping improves the behavior that motivated the system design. Four evaluation questions guide the experiment. RQ1 asks whether SLA-driven urgency assignment creates observable latency separation between priority classes. RQ2 asks whether adaptive priority updates reduce SLA violations compared with unmanaged HTTP/2 and a legacy RFC 7540-style baseline. RQ3 asks what throughput and success-rate trade-offs accompany latency prioritization. RQ4 asks whether priority ordering remains useful under stress conditions such as congestion, high server load, and elevated packet error rate.

The selected metrics follow directly from these questions. Mean, median, P95, and P99 latency capture both central behavior and tail behavior; SLA violation rate measures whether latency budgets are respected; throughput indicates whether prioritization reduces aggregate service capacity; success rate identifies failures caused by stress conditions; and statistical tests assess whether observed separation between priority classes is larger than expected random variation under the controlled workload. These metrics do not represent every possible QoS dimension. They are selected because they correspond to the stated design goals of latency protection, SLA compliance, and controlled trade-off visibility.

5.2. Experimental Setup

5.2.1. Hardware Testbed

All experiments were conducted on an isolated testbed to eliminate interference from external traffic and to improve repeatability under controlled conditions. This isolation reduces background network variability, but it also limits external validity: the measured gains should be attributed to the complete prototype and testbed configuration rather than to native RFC 9218 scheduling behavior in arbitrary HTTP/2 deployments.

The testbed consists of two dedicated physical servers interconnected via a 10 Gbps Ethernet switch. The server node is equipped with an Intel Xeon E5-2680 v4 processor (14 cores at 2.40 GHz) and 64 GB DDR4 ECC memory, running Ubuntu 22.04 LTS. The load generator operates on identical hardware to avoid asymmetry-induced bottlenecks and to ensure that observed performance effects reflect the system under test rather than hardware asymmetry.

Traffic generation is performed using the wrk2 benchmarking tool with custom Lua scripts. The use of wrk2 is motivated by its constant-rate request generation model, which provides precise control over offered load and avoids coordinated omission effects that can distort latency measurements in bursty workloads.

Network conditions are emulated using Linux Traffic Control (`tc netem`), enabling controlled manipulation of delay, queue length (to model congestion), and packet loss rate. This approach allows systematic exploration of network behaviors that are difficult to reproduce consistently in live environments, such as high-loss or satellite-like conditions.

5.2.2. Service Scenarios

Ten service scenarios are defined to represent a broad range of SBA-like workloads. The latency thresholds are used as experimental SLA budgets that emulate control-plane, interactive, media, standard-data, bulk-transfer, background-analytics, and telemetry traffic. They are not presented as universal 3GPP latency guarantees; rather, they provide a controlled policy space for evaluating whether the priority mechanism can differentiate among competing service classes. The service scenarios and their corresponding SLA latency thresholds are summarized in Table 4.

Table 4. Service scenarios and SLA latency thresholds.

ID	Service Class	Assigned Urgency	SLA Latency Limit
S1	URLLC Control	0	1 ms
S2	URLLC Data	0	2 ms
S3	Interactive High	1	10 ms
S4	Interactive Medium	1	15 ms
S5	Voice Call	1	60 ms
S6	Video Streaming	2	100 ms
S7	Standard Data	3	120 ms
S8	Bulk Transfer	5	1000 ms
S9	Background Analytics	6	1200 ms
S10	IoT Telemetry	7	1500 ms

The selected scenarios cover a wide range of QoS requirements in SBA environments, from ultra-low-latency control traffic to delay-tolerant telemetry and analytics processing. This diversity is essential for evaluating whether the proposed prioritization mechanism can protect critical services while making the trade-offs imposed on lower-priority services explicit.

5.2.3. Network Conditions

Nine network profiles are defined using Linux traffic control and `tc netem`. The profiles vary bandwidth, one-way delay, jitter, packet loss, queue limit, and server-load percentage so that normal, congested, error-prone, mixed-impairment, and extreme conditions are all represented. The exact replay command template used by the supplementary artifact is: `tc qdisc replace dev IFACE parent 1:10 handle 10: netem delay DELAYms JITTERms loss LOSS% limit LIMIT`, preceded by the corresponding HTB rate class for the profile bandwidth.

The parameter ranges cover both typical and stress conditions. High congestion and mixed impairment profiles expose queuing and packet-loss effects, while high-server-load profiles test the adaptation logic under resource pressure.

For reproducibility, the baseline dataset contains $4 \times 10 \times 9 \times 20 = 7200$ observations across four operating modes: `no_priority`, `legacy_rfc7540`, `static_rfc9218`, and `dpma`. The scalability dataset contains 14,880 observations across the same four modes and the tested concurrent-stream levels. These observations are used directly to compute the tables and figures presented in Section 6. The nine network-emulation profiles applied throughout the evaluation are summarized in Table 5.

Table 5. Network-emulation profiles used in the reproducible evaluation.

ID	Profile	Bandwidth	Delay	Jitter	Loss	Server Load
N1	Normal	1000 Mbit/s	0.5 ms	0.1 ms	0.00%	0%
N2	Slightly Congested	800 Mbit/s	0.8 ms	0.2 ms	0.00%	15%
N3	Moderately Congested	500 Mbit/s	1.0 ms	0.2 ms	0.01%	25%
N4	Heavily Congested	250 Mbit/s	1.5 ms	0.4 ms	0.02%	35%
N5	High Server Load	100 Mbit/s	3.0 ms	0.8 ms	0.05%	45%
N6	High Error Rate	50 Mbit/s	5.0 ms	1.0 ms	0.08%	55%
N7	Mixed Congestion–Load	25 Mbit/s	6.0 ms	1.5 ms	0.10%	65%
N8	Mixed Congestion–Error	10 Mbit/s	8.0 ms	2.0 ms	0.20%	75%
N9	Extreme Conditions	5 Mbit/s	12.0 ms	3.0 ms	0.50%	85%

The principal execution settings and reproducibility parameters are summarized in Table 6.

Table 6. Reproducibility settings and execution policy.

Item	Specification
Traffic-control replay	Exact tc/netem profiles N1–N9 corresponding to the network conditions reported in Table 5
Workload generator	wrk -t8 -c256 -d30s -R500 -latency -s lua/sba_workload.lua SERVER_URL
Run order	Block randomization using the fixed seed 9218
CPU/NIC preparation	CPU-governor configuration, queue-discipline reset, and NIC metadata collection before execution
Server-load generation	stress-ng -cpu N -cpu-load P -timeout 3600s -metrics-brief
Raw baseline observations	7200 observations covering four priority-management modes, ten service scenarios, nine network profiles, and twenty repetitions
Raw scalability observations	14,880 observations covering the evaluated concurrent-stream levels and overhead measurements
Processed outputs	Baseline-mode, network-profile, service-scenario, scalability, and overhead summaries used to generate the tables and figures in Section 6

5.2.4. Experiment Design

The baseline experiment follows a full-factorial structure combining four operating modes, ten service scenarios, nine network profiles, and twenty repetitions per combination. The four operating modes are: (i) `no_priority`, in which priority metadata are not applied; (ii) `legacy_rfc7540`, in which legacy dependency/weight-style priority behavior is used as a comparator; (iii) `static_rfc9218`, in which fixed RFC 9218 urgency values are applied without dynamic adaptation; and (iv) `dpma`, in which the proposed SLA-aware dynamic mapping is enabled. This structure permits comparison against both older and contemporary priority-signaling alternatives while preserving identical service and network conditions.

Each service–profile–mode combination was repeated twenty times, yielding 7200 baseline observations. A separate scalability campaign evaluates concurrent-stream levels of 1, 5, 10, 20, 50, and 100 streams, yielding 14,880 observations. Latency was measured as testbed-level response latency under the configured network-emulation profile. This measurement boundary included the application request path and configured emulation effects in the isolated hardware testbed; it was not presented as a radio-access, carrier-backhaul, or public-Internet end-to-end measurement.

5.3. Statistical Methodology

The statistical analysis uses raw per-observation latency and SLA-violation records. Because latency measurements are heavy-tailed, the evaluation reports mean, median, P95, and P99 latency rather than relying on the mean alone. SLA compliance is evaluated using

the service-specific thresholds in Table 4. One-way ANOVA is used to test whether latency differs across the four operating modes, and Welch pairwise comparisons are used for DPMA against each baseline. Cohen's d is reported as a standardized effect-size measure, while practical relevance is interpreted through tail-latency and SLA-violation reductions.

6. Results and Analysis

6.1. Four-Mode Baseline Comparison

The revised baseline evaluation compares dpma against three operating modes under the same ten service classes, nine network profiles, and twenty repetitions per service–profile–mode combination. Table 7 summarizes the complete 7200-observation baseline dataset.

Table 7. Four-mode baseline comparison across 7200 raw observations.

Mode	Observations	Mean (ms)	Median (ms)	P95 (ms)	P99 (ms)	SLA Violation (%)	Success (%)
no_priority	1800	459.12	74.11	1929.49	2719.43	50.28	94.89
legacy_rfc7540	1800	565.62	76.59	2467.90	3364.81	51.50	97.17
static_rfc9218	1800	562.98	76.63	2413.06	3417.48	51.83	96.06
dpma	1800	345.07	76.01	1252.08	1686.61	30.39	96.00

Compared with no_priority, DPMA reduces mean latency by 24.8%, P95 latency by 35.1%, P99 latency by 38.0%, and SLA violations by 19.9 percentage points. Compared with legacy_rfc7540, DPMA reduces mean latency by 39.0%, P95 latency by 49.3%, P99 latency by 49.9%, and SLA violations by 21.1 percentage points. Compared with static_rfc9218, DPMA reduces mean latency by 38.7%, P95 latency by 48.1%, P99 latency by 50.6%, and SLA violations by 21.4 percentage points. These results show that the dynamic SLA-aware mapping provides additional benefit beyond both unmanaged operation and static priority signaling.

The distributional behavior is shown in Figure 2. The logarithmic scale is used because the workload intentionally combines very different SLA classes, from sub-2 ms URLLC-like traffic to delay-tolerant telemetry and analytics traffic. The boxplot confirms that the primary DPMA gain appears in the upper tail rather than only in the central tendency.

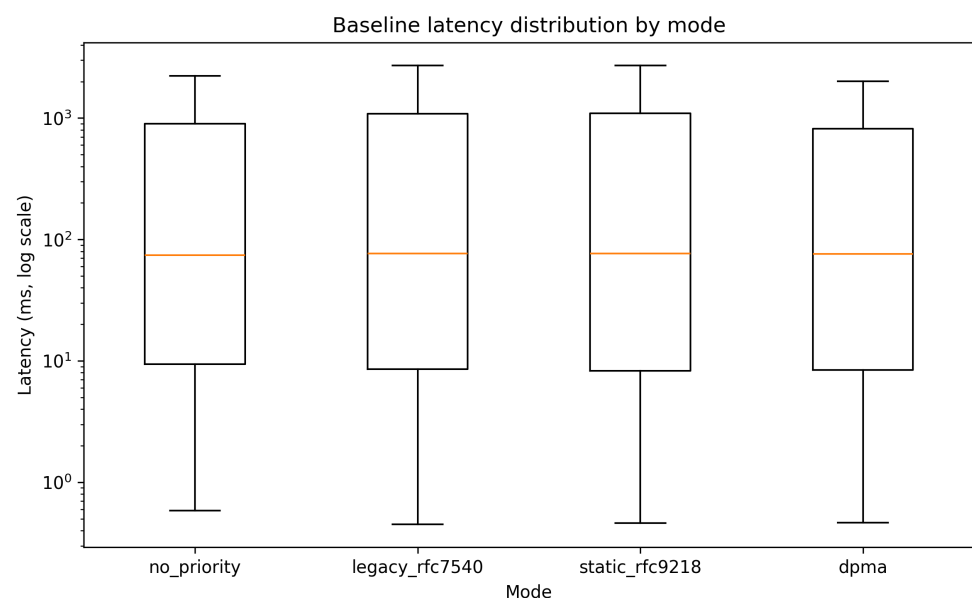


Figure 2. Latency distribution by operating mode across the 7200-observation baseline dataset. The log-scale axis is used to make both ultra-low-latency and high-delay service classes visible in the same figure.

Figure 3 summarizes the P95-latency and SLA-violation comparison across modes. The same ordering is observed in both metrics: DPMA provides the lowest tail latency and the lowest aggregate SLA-violation rate.

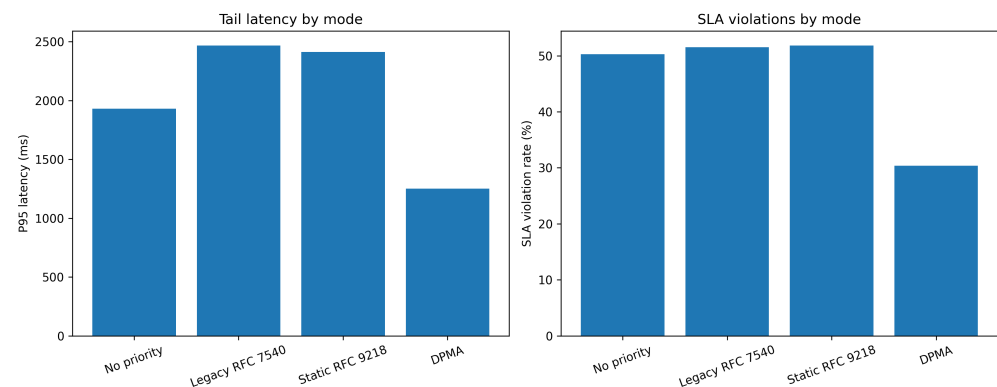


Figure 3. Tail-latency and SLA-violation comparison across operating modes. DPMA shows the lowest P95 latency and lowest SLA-violation rate under the tested service and network conditions.

6.2. Network-Stress Behavior

Table 8 compares DPMA with the unmanaged baseline across the nine network profiles. The strongest reductions appear under congested and mixed-impairment conditions, where dynamic priority adaptation has more opportunity to affect queuing behavior and SLA violations.

Table 8. DPMA versus unmanaged baseline across network profiles.

Network Profile	DPMA Mean	No-Priority Mean	Mean Reduction	DPMA SLA Viol.	No-Priority SLA Viol.	SLA Reduction
Normal	278.76	293.67	5.1%	0.0%	0.0%	0.0 pp
Slightly Congested	286.49	350.42	18.2%	0.5%	0.5%	0.0 pp
Moderately Congested	345.19	442.08	21.9%	18.0%	58.5%	40.5 pp
Heavily Congested	399.52	593.71	32.7%	72.5%	100.0%	27.5 pp
High Server Load	310.99	387.57	19.8%	5.0%	14.5%	9.5 pp
High Error Rate	269.54	325.74	17.3%	0.0%	0.0%	0.0 pp
Mixed Congestion–Load	385.31	530.95	27.4%	55.0%	100.0%	45.0 pp
Mixed Congestion–Error	359.30	472.36	23.9%	26.5%	79.0%	52.5 pp
Extreme Conditions	470.57	735.62	36.0%	96.0%	100.0%	4.0 pp

The network-profile results indicate that DPMA is most beneficial when prioritization decisions interact with congestion, queueing, or mixed impairment. Under normal conditions, the SLA-violation rate is already low and the absolute improvement is correspondingly smaller. This pattern is consistent with the purpose of the DPMA, which is to adapt priority metadata when operating conditions begin to threaten service-specific latency budgets.

6.3. Scalability and Overhead Analysis

The reviewers requested explicit overhead metrics. The revised scalability campaign therefore reports CPU utilization, memory consumption, classification time, priority-update decision time, and latency under increasing concurrent-stream levels. Figure 4 summarizes the main scalability trends, and Table 9 reports the DPMA overhead values. CPU utilization is reported as process-level aggregate CPU percentage across available cores; therefore, values above 100% indicate multi-core execution rather than a measurement error.

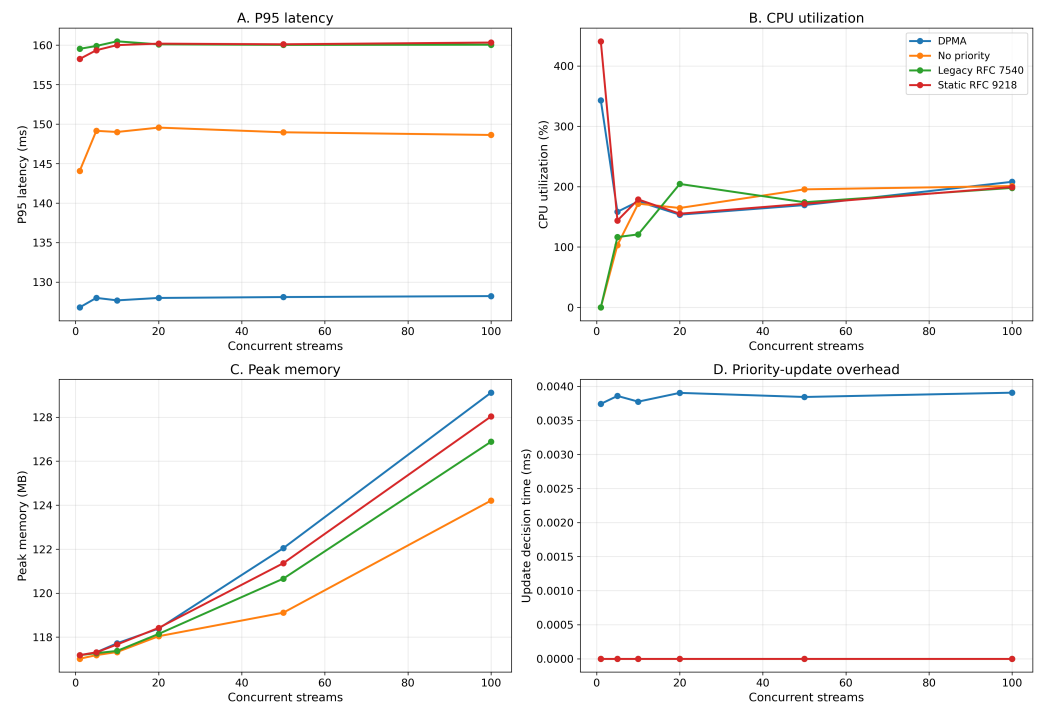


Figure 4. Scalability and overhead analysis under increasing concurrent-stream levels. The panels report P95 latency, CPU utilization, peak memory, and priority-update decision overhead.

Table 9. DPMA scalability and overhead summary.

Concurrent Streams	Requests	P95 Latency (ms)	CPU (%)	Peak Memory (MB)	Class. Time (ms)	Update Time (ms)
1	20	126.80	343.13	117.18	0.0273	0.0037
5	100	128.00	158.38	117.31	0.0355	0.0039
10	200	127.69	175.53	117.72	0.0286	0.0038
20	400	127.99	153.60	118.39	0.0348	0.0039
50	1000	128.11	169.44	122.04	0.0499	0.0038
100	2000	128.23	208.04	129.11	0.0354	0.0039

Across the tested range from 1 to 100 concurrent streams, DPMA maintains P95 latency between 126.80 ms and 128.23 ms. Peak memory grows from 117.18 MB to 129.11 MB, while the priority-update decision time remains below 0.004 ms per request. These results indicate that the decision logic adds small per-request overhead in the tested environment and that the main resource cost is associated with normal process execution and concurrent request handling rather than priority-decision computation.

6.4. Statistical Validation of Observed Effects

Statistical validation is summarized in Table 10. The ANOVA result confirms that latency differs significantly across the four operating modes. Pairwise Welch comparisons show that DPMA differs significantly from each baseline. The standardized effect sizes are modest at the aggregate level because the dataset combines service classes with very different SLA budgets, but the practical effect is visible in the large P95, P99, and SLA-violation reductions reported in Table 7.

Overall, the results show that DPMA achieves the best aggregate mean, P95, and P99 latency and the lowest aggregate SLA-violation rate across the revised baseline dataset. The gains are strongest under congested and mixed-impairment profiles, while improvements are naturally smaller under normal or lightly impaired profiles where SLA violations are already low.

Table 10. Statistical validation of the four-mode baseline comparison.

Test or Metric	Value	Interpretation
Raw baseline observations	7200	Four modes, ten services, nine profiles, twenty repetitions
One-way ANOVA across modes	$F = 34.72, p < 0.001$	Significant latency difference across modes
ANOVA effect size	$\eta^2 = 0.014$	Small aggregate effect across heterogeneous services
DPMA vs. no_priority	$t = -5.69, p < 0.001, d = 0.19$	Significant improvement with small aggregate standardized effect
DPMA vs. legacy_rfc7540	$t = -9.29, p < 0.001, d = 0.31$	Significant improvement with small-to-moderate aggregate effect
DPMA vs. static_rfc9218	$t = -9.23, p < 0.001, d = 0.31$	Significant improvement with small-to-moderate aggregate effect

7. Discussion

7.1. Implications for SBA Deployments

The revised results show that the proposed SLA-aware priority-control path improves latency and SLA-compliance behavior under controlled SBA-like HTTP/2 conditions. The strongest gains occur in tail latency and SLA-violation reduction, which are more relevant to SLA enforcement than average latency alone. This finding is important for service-based systems because SLA breaches are typically triggered by tail behavior rather than by median performance.

The scope of the prototype is HTTP/2, but the design logic is compatible with the broader RFC 9218 abstraction of urgency and incremental delivery. HTTP/3/QUIC remains an important direction because it changes the transport substrate and may expose different behavior under packet loss, congestion, or attack conditions. The revised paper therefore positions HTTP/3 comparison as a cross-protocol extension rather than as a substitute for studying HTTP/2-based SBA communication.

7.2. Scalability, Fairness, and Security Considerations

The PCE and DPMA are largely stateless with respect to individual requests, while runtime monitoring state can be externalized to a shared monitoring backend in clustered deployments. The scalability campaign indicates stable P95 latency and low priority-update decision overhead up to 100 concurrent streams in the tested environment. In Kubernetes or service-mesh deployments, the same policy logic can be deployed at ingress, gateway, or sidecar boundaries, provided that priority metadata are preserved across proxy hops and that monitoring data are shared consistently.

Fairness is handled through bounded urgency mappings rather than unrestricted promotion. Lower-priority traffic remains eligible for service, and urgency updates remain within the RFC 9218 range. For operational deployment, high-priority classification should be bound to authenticated service identity, untrusted client-supplied Priority headers should be removed or rewritten at the ingress boundary, and high-priority classifications should be rate-limited and audited. These controls preserve the intended SLA semantics while reducing the risk of malicious priority escalation.

7.3. Scope of Validation

The evaluation validates the proposed method in a controlled two-server HTTP/2 testbed with explicit raw baseline and scalability observations. The testbed is suitable for isolating the effect of service class, network profile, and priority-control mode. Broader deployment studies can extend the same experimental protocol to multi-node Kubernetes clusters, service-mesh proxies, multi-region topologies, HTTP/3/QUIC implementations, and learning-assisted priority policies. These extensions are methodological continuations of the present validation rather than prerequisites for interpreting the controlled HTTP/2 results.

8. Conclusions

This work presents an SLA-aware HTTP/2 priority management system based on RFC 9218, integrating classification, adaptive mapping, and bounded runtime priority updates within a middleware framework. The study addresses the practical problem of translating application-level SLA policies into HTTP priority metadata for HTTP/2-based SBA communication.

The revised baseline evaluation contains 7200 raw observations across four operating modes, ten service classes, nine network profiles, and twenty repetitions per service–profile–mode combination. Compared with `no_priority`, DPMA reduced mean latency by 24.8%, P95 latency by 35.1%, P99 latency by 38.0%, and SLA violations by 19.9 percentage points. Compared with `legacy_rfc7540`, DPMA reduced mean latency by 39.0%, P95 latency by 49.3%, P99 latency by 49.9%, and SLA violations by 21.1 percentage points. Compared with `static_rfc9218`, DPMA reduced mean latency by 38.7%, P95 latency by 48.1%, P99 latency by 50.6%, and SLA violations by 21.4 percentage points.

The scalability analysis further shows that DPMA maintained P95 latency between 126.80 ms and 128.23 ms across the tested 1–100 concurrent-stream range, while priority-update decision overhead remained below 0.004 ms per request. These findings support the conclusion that SLA-aware dynamic priority mapping can provide measurable latency and SLA-compliance benefits over unmanaged, legacy-priority, and static RFC 9218 configurations in controlled HTTP/2 SBA-like environments.

Author Contributions: A.L.S.A.-K. designed the research and methods, developed the software artifacts, performed the analytic work, and wrote the initial version of the paper. R.A. coordinated supervision, contributed iterative commentary, and refined the manuscript through editorial revisions. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The revised supplementary artifact includes the replay scripts, workload files, raw baseline observations, raw scalability observations, processed summary tables, and generated figures used for the revised analysis. Additional inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to express their sincere appreciation to Atlas University for its valuable technical support. They also gratefully acknowledge Istanbul Aydın University for providing a supportive research environment that significantly contributed to this work. Furthermore, the authors extend their sincere thanks to Defne Telekomünikasyon A.Ş. (<https://www.defne.com.tr/>) for its valuable technical support.

Conflicts of Interest: A.L.S.A.-K. is affiliated with Defne Telekomünikasyon A.Ş., which is acknowledged for technical support. The authors declare that this affiliation did not influence the study design, analysis, interpretation, or manuscript preparation.

Abbreviations

The following abbreviations are used in this manuscript:

SBA	Service-Based Architecture
QoS	Quality of Service
SLA	Service Level Agreement
HTTP/2	Hypertext Transfer Protocol Version 2
IETF	Internet Engineering Task Force
RFC	Request for Comments
PCE	Priority Classification Engine
DPMA	Dynamic Priority Mapping Algorithm
PUM	PRIORITY_UPDATE Manager
SMF	SLA Monitoring Framework
URLLC	Ultra-Reliable Low-Latency Communication
eMBB	Enhanced Mobile Broadband
mMTC	Massive Machine-Type Communication
CI	Confidence Interval
CDF	Cumulative Distribution Function
ANOVA	Analysis of Variance
MEC	Multi-access Edge Computing
NF	Network Function
CPU	Central Processing Unit

References

1. Jamshidi, P.; Pahl, C.; Mendonça, N.C.; Lewis, J.; Tilkov, S. Microservices: The Journey So Far and Challenges Ahead. *IEEE Softw.* **2018**, *35*, 24–35. [\[CrossRef\]](#)
2. Dragoni, N.; Giallorenzo, S.; Lluch-Lafuente, A.; Mazzara, M.; Montesi, F.; Mustafin, R.; Safina, L. Microservices: Yesterday, Today, and Tomorrow. In *Present and Ulterior Software Engineering*; Mazzara, M., Meyer, B., Eds.; Springer: Cham, Switzerland, 2017; pp. 195–216. [\[CrossRef\]](#)
3. Simsek, M.; Aijaz, A.; Dohler, M.; Sachs, J.; Fettweis, G. 5G-Enabled Tactile Internet. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 460–473. [\[CrossRef\]](#)
4. Popovski, P.; Nielsen, J.J.; Stefanović, Č.; de Carvalho, E.; Ström, E.; Trillingsgaard, K.F.; Bana, A.S.; Kim, D.M.; Kotaba, R.; Park, J.; et al. Wireless Access for Ultra-Reliable Low-Latency Communication: Principles and Building Blocks. *IEEE Netw.* **2018**, *32*, 16–23. [\[CrossRef\]](#)
5. 3GPP. *System Architecture for the 5G System (5GS)*; Technical Specification TS 23.501; 3rd Generation Partnership Project: Sophia Antipolis, France, 2020.
6. 3GPP. *5G System; Technical Realization of Service Based Architecture; Stage 3*; Technical Specification TS 29.500, Release 18, Version 18.6.0; 3rd Generation Partnership Project: Sophia Antipolis, France, 2024.
7. Simpson, A.; Alshaali, M.; Tu, W.; Asghar, M.R. Quick UDP Internet Connections and Transmission Control Protocol in unsafe networks: A comparative analysis. *IET Smart Cities* **2024**, *6*, 351–360. [\[CrossRef\]](#)
8. Bishop, M. *HTTP/3*; RFC 9114; IETF: Fremont, CA, USA, 2022. [\[CrossRef\]](#)
9. Belshe, M.; Peon, R.; Thomson, M. *Hypertext Transfer Protocol Version 2 (HTTP/2)*; RFC 7540; IETF: Fremont, CA, USA, 2015. [\[CrossRef\]](#)
10. Thomson, M.; Benfield, C. *HTTP/2*; RFC 9113; IETF: Fremont, CA, USA, 2022. [\[CrossRef\]](#)
11. Wijnants, M.; Marx, R.; Quax, P.; Lamotte, W. HTTP/2 Prioritization and Its Impact on Web Performance. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 1755–1764. [\[CrossRef\]](#)
12. Marx, R.; De Decker, T.; Quax, P.; Lamotte, W. Resource Multiplexing and Prioritization in HTTP/2 over TCP versus HTTP/3 over QUIC. In *Web Information Systems and Technologies*; Springer: Cham, Switzerland, 2020; pp. 96–126. [\[CrossRef\]](#)
13. Oku, K.; Pardue, L. *Extensible Prioritization Scheme for HTTP*; RFC 9218; IETF: Fremont, CA, USA, 2022. [\[CrossRef\]](#)
14. Herbots, J.; Marx, R.; Lamotte, W.; Quax, P. HTTP/3's Extensible Prioritization Scheme in the Wild. In Proceedings of the Applied Networking Research Workshop, Vancouver, BC, Canada, 22 July 2024. [\[CrossRef\]](#)
15. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [\[CrossRef\]](#)

16. Foukas, X.; Patounas, G.; Elmokashfi, A.; Marina, M.K. Network Slicing in 5G: Survey and Challenges. *IEEE Commun. Mag.* **2017**, *55*, 94–100. [[CrossRef](#)]
17. Zhang, Y.; Hua, W.; Zhou, Z.; Suh, G.E.; Delimitrou, C. Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), Virtual, 19–23 April 2021. [[CrossRef](#)]
18. Qiu, H.; Banerjee, S.S.; Jha, S.; Kalbarczyk, Z.T.; Iyer, R.K. FIRM: An Intelligent Fine-Grained Resource Management Framework for SLO-Oriented Microservices. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), Virtual, 4–6 November 2020.
19. Zhang, Y.; Zhou, Z.; Elnikety, S.; Delimitrou, C. Ursa: Lightweight Resource Management for Cloud-Native Microservices. In Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Edinburgh, UK, 2–6 March 2024. [[CrossRef](#)]
20. Wang, Z.; Li, P.; Liang, C.-J.M.; Wu, F.; Yan, F.Y. Autothrottle: A Practical Bi-Level Approach to Resource Management for SLO-Targeted Microservices. In Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), Santa Clara, CA, USA, 16–18 April 2024; pp. 149–165.
21. Keller, A.; Ludwig, H. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Netw. Syst. Manag.* **2003**, *11*, 57–81. [[CrossRef](#)]
22. Buyya, R.; Yeo, C.S.; Venugopal, S.; Broberg, J.; Brandic, I. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Gener. Comput. Syst.* **2009**, *25*, 599–616. [[CrossRef](#)]
23. Hemmat, R.A.; Hafid, A. SLA Violation Prediction in Cloud Computing: A Machine Learning Perspective. *arXiv* **2016**, arXiv:1611.10338.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.