HTTP/3: Features, Vulnerabilities, and a Crypto-Cost-Aware Handshake Admission Defense

Samer khanji

Abstract

HTTP/3 maps HTTP semantics onto the QUIC transport and is now a completed Internet standard. The design reduces handshake delay, removes TCP-level head-of-line blocking, and encrypts transport headers by default. Those same properties change the attack surface. This article reviews the architecture and features of HTTP/3, compares the protocol with HTTP/1.1 and HTTP/2 using published measurements, and organizes documented weaknesses into a compact taxonomy covering handshake resource exhaustion, 0-RTT replay, connection migration, QPACK state, privacy leakage, and gateway desynchronization. A recurring gap is identified: RFC 9000 limits unvalidated response volume to three times the received datagram size, which bounds byte amplification but does not bound the cryptographic CPU work spent on Initial and CRYPTO processing. The paper proposes REAP-H3 (Reputation-Enforced Adaptive Protection for HTTP/3), a server-side admission algorithm that combines address validation, a per-source crypto-cost budget, bounded incomplete-CRYPTO state, load-triggered puzzles, and a sliding Bloom filter for 0-RTT anti-replay. Discrete-event

simulation, with cryptographic units calibrated to median OpenSSL timings of the TLS 1.3 primitives that RFC 9001 uses, shows that a byte-only baseline collapses under high-rate Initial floods, whereas REAP-H3 preserves high legitimate handshake success against both spoofed Initials and address-validated CRYPTO floods until the botnet becomes comparable in width to the server's source budget. Sensitivity to C_max, ρ, U_max, and W is reported, together with an operational deployment profile.

Keywords: HTTP/3; QUIC; handshake flooding; 0-RTT replay; denial of service; QPACK; admission control; REAP-H3

1. Introduction

The Hypertext Transfer Protocol remains the dominant application protocol of the public Web. HTTP/1.1 serialized requests on a TCP connection and incurred costly connection setup when browsers opened many parallel sockets. HTTP/2 introduced binary framing, header compression, and stream multiplexing over a single TCP channel, yet a lost TCP segment still stalls every multiplexed stream. HTTP/3, published as RFC 9114, relocates HTTP onto QUIC, a UDP-based transport that integrates TLS 1.3, independent streams, and connection identifiers that can survive path changes [1], [2], [3]. HTTP

semantics themselves are specified independently in RFC 9110, so HTTP/3 is a mapping rather than a new resource model [4].

Deployment is no longer experimental. W3Techs reported that 40.0% of surveyed websites advertised HTTP/3 support in August 2026 [5]. Advertisement, however, is not the same as request share. Sites often signal HTTP/3 through Alternative Services or HTTPS DNS records while a large fraction of objects, first visits, or middlebox-constrained clients still complete over TCP. Measurement studies have therefore been careful to separate protocol availability from realized performance [6]. The practical picture is mixed: HTTP/3 tends to help under high latency and on mobile paths, and it does not uniformly dominate HTTP/2 on well-provisioned wired access [6].

Security research has lagged the performance narrative. Surveys now document a wide set of QUIC and HTTP/3 issues spanning cryptographic modes, handshake flooding, transport-frame abuse, and privacy leakage [7], [8]. Internet-telescope observations showed that QUIC reconnaissance and flood events were already present in 2021, with handshake resource exhaustion compared to TCP SYN floods [9]. Subsequent implementation studies found that production stacks remain uneven: fuzzing and differential testing continue to uncover denial-of-service logic bugs, including CRYPTO-frame floods that force servers to buffer incomplete handshake state [8], [10].

This paper makes five contributions. First, it presents a self-contained, citation-backed account of HTTP/3 features as they are specified, including limits that popular commentary often omits (per-stream head-of-line blocking, 0-RTT replay, and UDP filtering). Second, it compares HTTP/1.1, HTTP/2, and HTTP/3 using published latency and adoption statistics rather than anecdotal claims. Third, it organizes vulnerabilities into a taxonomy that distinguishes protocol-inherent risks from implementation and gateway failures. Fourth, it proposes REAP-H3, an algorithm that treats cryptographic CPU time as a first-class scarce resource during connection admission and that binds 0-RTT admission to single-use tickets. Fifth, it calibrates the simulation units to OpenSSL measurements of the RFC 9001 primitives, re-runs the comparison, and reports parameter sensitivity and botnet-width scaling. The algorithm is compatible with RFC 9000 Retry and anti-amplification rules [2]. No live attack traffic is generated against operational systems.

The remainder of the article is organized as follows. Section 2 reviews related measurement and security work. Section 3 describes the HTTP/3 architecture. Section 4 presents comparative tables and figures. Section 5 develops the vulnerability taxonomy. Section 6 specifies REAP-H3. Section 7 reports calibrated simulation results. Section 8 treats operational feasibility. Section 9 discusses limitations, and Section 10 concludes.

2. Related Work

2.1 Performance and adoption measurements

Perna, Trevisan, Giordano, and Drago conducted a large-scale campaign on thousands of HTTP/3 sites and compared page-load metrics with HTTP/2 under emulated delay, bandwidth caps, and loss [6]. With 200 ms of extra latency, median onLoad times were 6.4 s, 5.8 s, and 5.4 s for HTTP/1.1, HTTP/2, and HTTP/3, respectively [6]. In the native (unimpaired) setting, about half of the sites loaded faster with HTTP/3 and half did not, which is an essential caution against unconditional performance claims [6]. An earlier conference version of the same measurement line reported similar qualitative conclusions [11]. Saif, Lung, and Matrawy compared HTTP/2 and HTTP/3 with Lighthouse quality-of-experience scores and likewise found that gains depend on workload and network context [12].

These results align with the protocol structure. QUIC can deliver application data after one round trip on a first visit and, when a session ticket is cached, in the first flight using 0-RTT [2], [3]. Independent streams recover from loss without stalling the entire connection, which matters on cellular paths. None of these mechanisms magically increase bottleneck capacity. When bandwidth is ample and delay is low, HTTP/2 over a tuned TCP stack remains competitive [6], [12].

2.2 Security surveys and empirical attacks

Joarder and Fung surveyed QUIC security and privacy, covering handshake, cryptographic, transport, and traffic-analysis threats, and argued that performance papers had outnumbered security analyses [7], [13]. Chatzoglou, Kouliaridis, Karopoulos, and Kambourakis combined a literature review with fuzzing of six production QUIC servers and reported practical zero-day conditions that could overwhelm server resources [8]. Nawrocki, Hiesgen, Schmidt, and Wählisch quantified QUIC scans and denial-of-service floods from telescope data and observed that handshake exhaustion was already being combined with other flood vectors [9].

Wang, Chen, Chen, Zhuge, Bai, and Duan introduced MerCuriuzz, a black-box fuzzer aimed at logical rather than memory-safety bugs, and reported fourteen previously unknown issues across sixteen implementations, including a CRYPTO-flood pattern in which fragmented handshake crypto forces costly buffering [10]. That finding is central to the gap addressed here: a server can obey the three-to-one byte limit and still spend unbounded CPU and memory on incomplete cryptographic transcripts.

Defense proposals remain sparse relative to the attack literature. Joarder and Fung later explored deep reinforcement learning for handshake-flood mitigation [14]. Learning-based controllers can adapt, but they introduce training cost, opaque

decision boundaries, and a need for representative traces. REAP-H3 is deliberately non-learning: it uses token buckets, bounded buffers, and an optional puzzle whose difficulty rises only when the server is already near capacity. The intent is a mechanism that operators can reason about and that stays within the Retry and address-validation framework of RFC 9000 [2].

2.3 Standards context

The protocol family is now a cluster of RFCs. RFC 9000 specifies QUIC transport, including streams, connection IDs, path validation, Retry, and anti-amplification [2]. RFC 9001 binds TLS 1.3 into QUIC packet protection and states explicitly that 0-RTT application data is not replay-protected [3]. RFC 9002 describes loss detection and congestion control [15]. RFC 9114 maps HTTP onto QUIC streams and frames [1]. RFC 9204 defines QPACK so that header compression does not re-introduce a total order across streams [16]. HTTP/2 was revised as RFC 9113 in the same cycle [17]. QUIC version 2 (RFC 9369) exists primarily to resist ossification; it does not change the HTTP/3 application mapping [18]. Version-independent wire properties are recorded in RFC 8999 [19]. Adjacent deployments such as DNS over QUIC illustrate that the transport is not Web-only [20].

3. HTTP/3 Architecture and Features

3.1 Stack placement

Figure 1 contrasts the three stacks. HTTP/1.1 and HTTP/2 ride TLS over TCP. HTTP/3 replaces that pair with QUIC over UDP. TLS 1.3 is not an optional add-on in HTTP/3: handshake messages travel in CRYPTO frames and packet protection is applied to nearly all fields beyond the public header [3]. The integration removes a round trip that TCP-plus-TLS often paid on a cold connection and conceals transport metadata that middleboxes previously used to classify TCP flows. The cost is reduced visibility for operators and, on some networks, UDP blocking or rate-limiting that forces a fallback to HTTP/2 or HTTP/1.1.

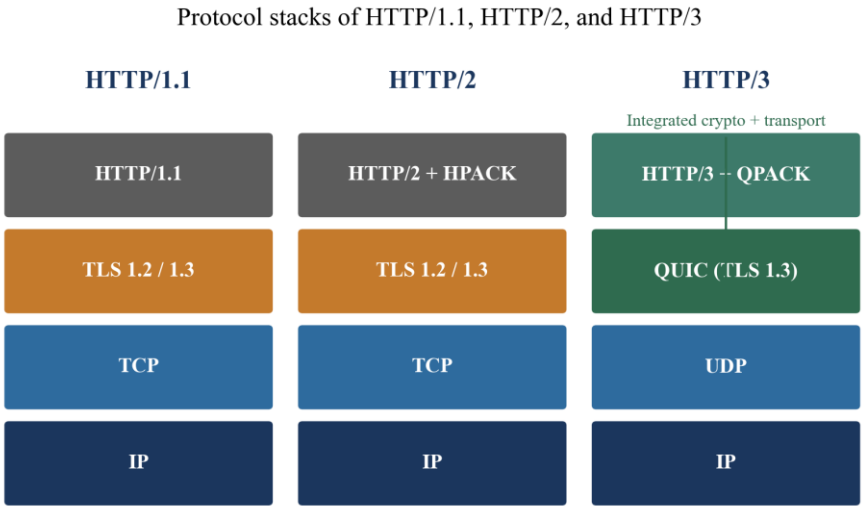

Figure 1. Protocol stacks of HTTP/1.1, HTTP/2, and HTTP/3. HTTP/3 integrates TLS 1.3 into QUIC and runs over UDP [1], [2], [3].

3.2 Connection establishment

A first-time HTTP/3 client typically sends an Initial packet carrying the TLS ClientHello. The server replies with its own Initial and Handshake packets, including the certificate chain when TLS authentication is used. After the client finishes the handshake, 1-RTT keys protect subsequent HTTP/3 frames [2], [3]. If the client already holds a session ticket, it may send 0-RTT packets in the first flight. Figure 2 summarizes both sequences. The latency benefit of 0-RTT is real; the freshness guarantee is not. RFC 9001 requires applications to restrict 0-RTT to operations that remain safe if replayed, and HTTP/3

inherits that constraint [1], [3]. Servers may reject early data and fall back to 1-RTT.

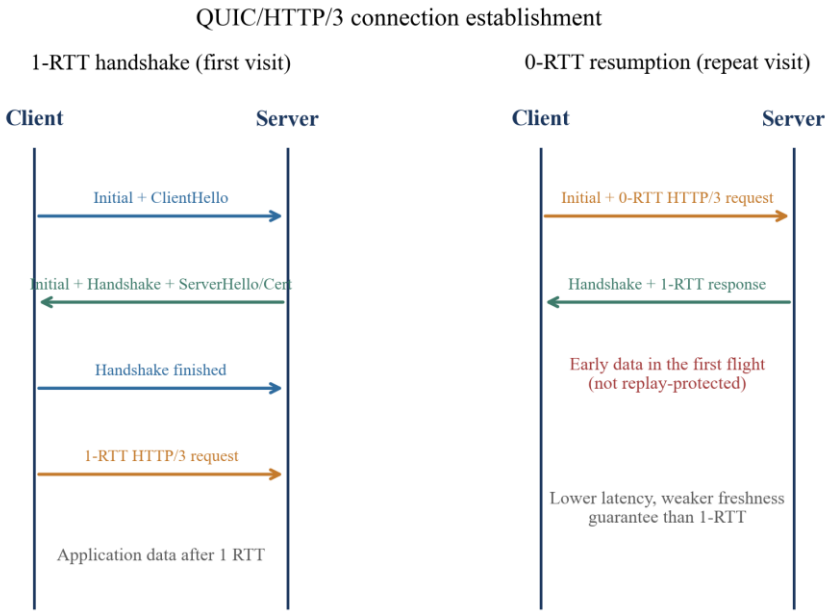

Figure 2. QUIC/HTTP/3 handshakes. 0-RTT places application data in the first flight and does not provide replay protection [3].

Until the client address is validated, a server must not send more than three times the number of bytes it has received [2]. Validation occurs when a valid token is presented, when a Retry completes, or when the client demonstrates possession of the path during the handshake. The three-to-one rule is a byte budget, not a CPU budget. That distinction is the design premise of Section 6.

3.3 Streams, loss recovery, and head-of-line blocking

QUIC multiplexes streams inside encrypted packets. Each stream has its own flow control and offset space [2]. A lost packet that carries only stream B can be repaired without delaying delivery of stream A or C. HTTP/2 cannot offer the same isolation because TCP delivers a single ordered byte stream [17]. Figure 3 illustrates the difference. Two caveats are required for scientific accuracy. First, a stream remains ordered: loss still delays that stream. Second, a connection still shares congestion control, so a lossy stream consumes bytes that other streams might have used [15]. HTTP/3 therefore eliminates transport-level head-of-line blocking among streams; it does not eliminate all coupling among them.

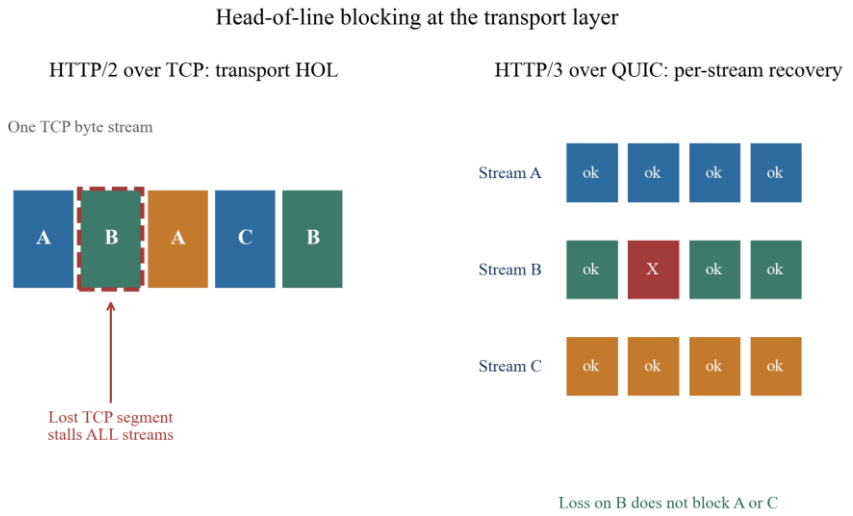

Figure 3. Transport head-of-line blocking. HTTP/2 over TCP stalls all streams after a lost segment; HTTP/3 recovers per stream [2], [17].

3.4 Connection identifiers and migration

A QUIC connection is identified by connection IDs rather than by the UDP four-tuple alone [2]. When a client moves from Wi-Fi to cellular, it can continue the same HTTP/3 session after path validation. Path validation is mandatory before the new path is fully trusted, because otherwise an off-path adversary could redirect traffic or use the server as an amplifier [2]. Migration is therefore a feature with an explicit liveness check, not an unconditional roaming capability.

3.5 QPACK and HTTP mapping

HTTP/3 encodes requests and responses as frames on QUIC streams. Each request occupies a client-initiated bidirectional stream [1]. Header fields use QPACK, which splits encoder and decoder streams from request streams so that a stalled header-table update does not freeze unrelated exchanges [16]. HPACK in HTTP/2 assumed a total order of header blocks; applying it unchanged to QUIC would have recreated head-of-line blocking at the compression layer [16], [17]. HTTP/3 also forbids chunked transfer coding and treats content length and stream closure as the body delimiters [1]. Those choices shrink several classic HTTP/1.1 smuggling gadgets, but they do not automatically sanitize conversions performed by reverse proxies, as Section 5 discusses.

3.6 Always-on encryption and ossification resistance

Most QUIC header fields that TCP would have left in cleartext are encrypted or protected [3]. The remaining public header is intentionally small. QUIC version 2 further perturbs version numbers and long-header type bits so that middleboxes cannot ossify on a single fingerprint [18]. The privacy gain is partial. Packet sizes, timing, and the shape of the handshake still leak information, which is why traffic-analysis attacks remain in the taxonomy [7].

4. Comparative Analysis and Published Statistics

4.1 Feature comparison

Table 1 summarizes protocol properties that matter for both performance and security. The entries follow the RFCs rather than vendor marketing. Handshake RTT counts refer to a typical browser talking to an origin that already supports the corresponding stack and do not include DNS. Figure 4 gives a qualitative radar view of the same dimensions. HTTP/3 scores strongly on handshake speed, loss isolation, migration, and mandatory encryption, and weakly on middlebox compatibility because UDP is filtered on a non-trivial set of enterprise and hotel networks.

Property	HTTP/1.1	HTTP/2	HTTP/3
Transport	TCP	TCP	QUIC over UDP
Typical first-visit RTTs to request	2–3 (TLS)	1–2 (TLS 1.3)	1

Property	HTTP/1.1	HTTP/2	HTTP/3
0-RTT application data	No	TLS 1.3 early data	Yes (replayable)
Stream multiplexing	No (pipelining only)	Yes	Yes (independent)
Transport HOL among streams	Yes	Yes (TCP)	No
Header compression	None	HPACK	QPACK
Connection migration	No	No	Yes, after path validation
Transport encryption	Optional TLS	De facto TLS	Mandatory TLS 1.3
Primary DoS analogue	SYN / Slowloris	Stream / HPACK abuse	Initial / CRYPTO flood

Table 1. Feature comparison of HTTP/1.1, HTTP/2, and HTTP/3 based on RFC 9110, RFC 9113, RFC 9114, RFC 9000, RFC 9001, and RFC 9204 [1]–[4], [16], [17].

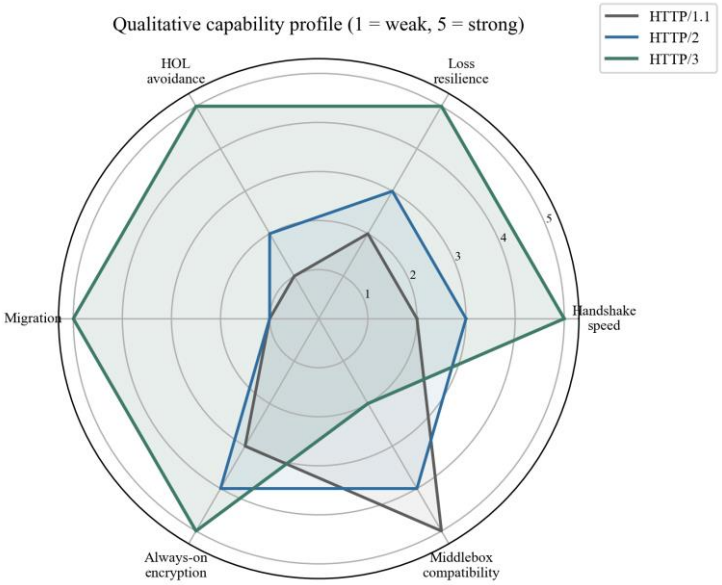

Figure 4. Qualitative capability profile. Scores are ordinal summaries of protocol structure, not measured throughputs.

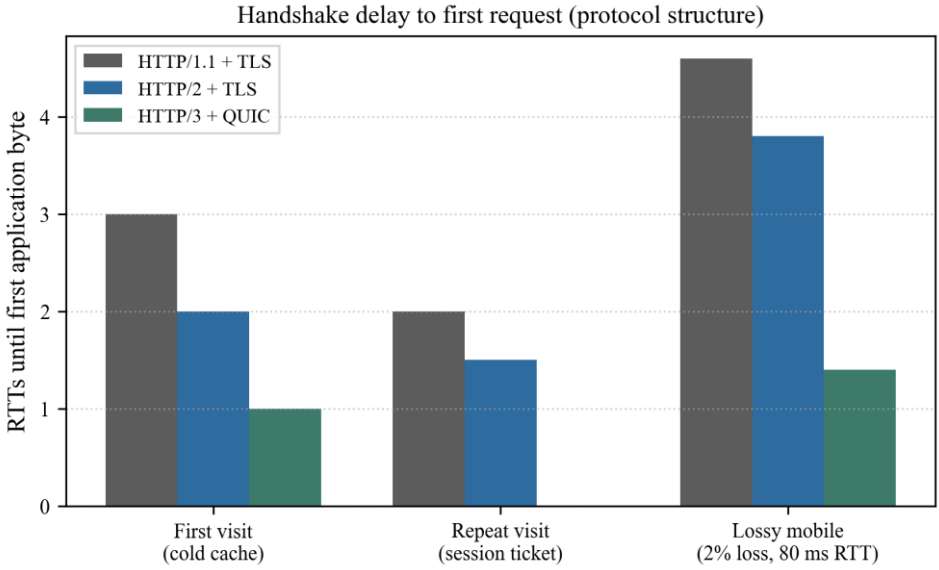

Figure 5. Round trips until the first application byte, derived from handshake structure [2], [3], [17]. The lossy-mobile bar is illustrative of extra recoveries, not a field measurement.

4.2 Adoption

Figure 6 plots two different quantities that are often conflated. Site-level HTTP/3 advertisements reported by W3Techs rose from 27.8% in early 2024 to 40.0% in August 2026 [5]. Request-level shares published from Cloudflare's vantage point were 28% in May 2023 and 21.1% in the first quarter of 2026 [21]. The gap is expected. A site can advertise HTTP/3 while browsers still use TCP for the first connection, for third-party origins without QUIC, or when UDP is blocked. Perna et al. observed that many HTTP/3 pages still fetch a large

fraction of objects from hosts that speak only HTTP/2 or HTTP/1.1, which dilutes end-to-end gains [6].

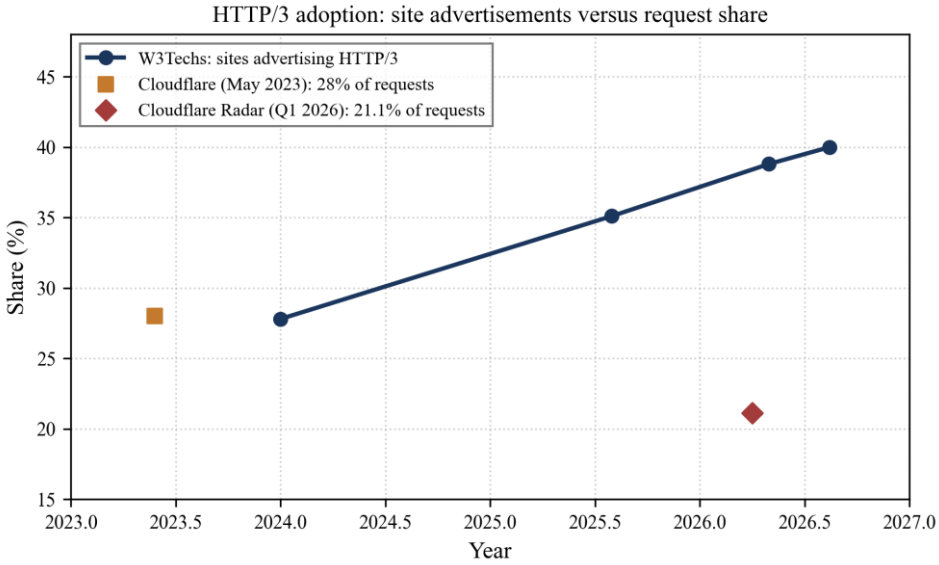

Figure 6. HTTP/3 adoption. W3Techs points are website-level advertisements [5]. Cloudflare points are request shares [21] and are not directly comparable.

4.3 Page-load evidence

Figure 7 restates the onLoad evidence from Perna et al. [6]. Under 200 ms of added latency, HTTP/3 reduced median onLoad relative to HTTP/2 by 0.4 s and relative to HTTP/1.1 by 1.0 s. The share of sites that loaded faster with HTTP/3 than with HTTP/2 increased from about 50% in the native setting to 70%, 73%, and 74% with 50 ms, 100 ms, and 200 ms of extra delay [6]. The same study did not find a comparable general advantage under all loss and bandwidth scenarios, and it reported no benefit for the adaptive-streaming setup they tested

[6]. Those negatives are as informative as the positives: HTTP/3 is not a universal accelerator.

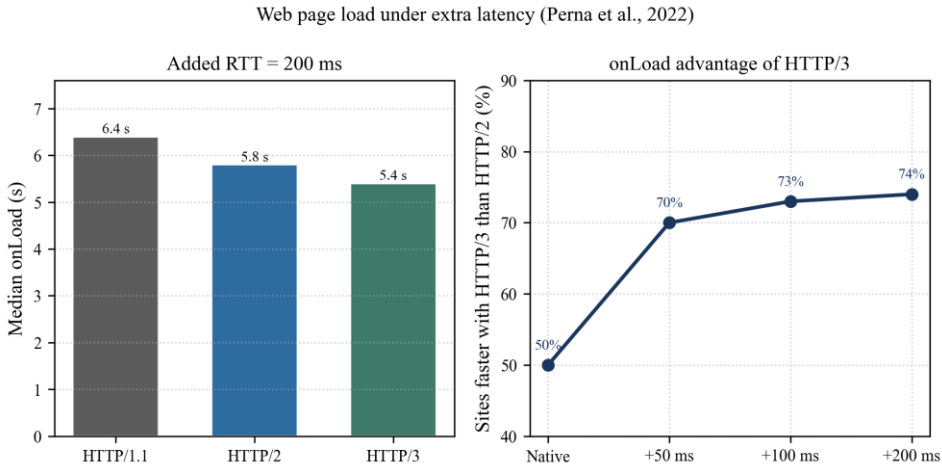

Figure 7. Page-load comparison reconstructed from Perna, Trevisan, Giordano, and Drago [6]. Left: median onLoad with 200 ms extra latency. Right: fraction of sites faster with HTTP/3 than HTTP/2.

5. Vulnerability Taxonomy

Figure 8 groups HTTP/3 risks into handshake resource exhaustion, cryptographic freshness, transport and privacy, and application-gateway mapping. The grouping follows the literature rather than a single CVE database [7], [8], [10]. This section describes mechanisms at the architectural level. It does not provide exploit procedures, payloads, or proof-of-concept code.

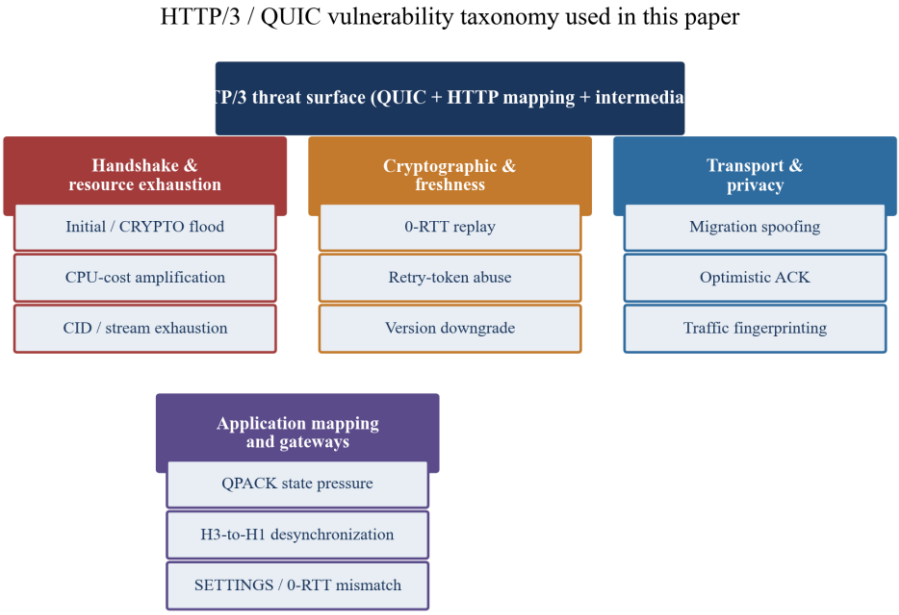

Figure 8. Vulnerability taxonomy used in this paper, spanning transport, cryptography, privacy, and HTTP mapping [7], [8], [10].

5.1 Handshake and resource exhaustion

An unauthenticated UDP datagram that looks like a QUIC Initial can force a server to allocate connection state, parse a ClientHello, and run HKDF and header protection [2], [3]. RFC 9000 caps the bytes the server may send before address validation, which limits reflective amplification [2]. It does not cap the CPU cycles spent on that datagram. Nawrocki et al. already compared this pattern to SYN floods and documented real QUIC flood events [9]. Chatzoglou et al. showed that production servers can be driven into overload by handshake fuzzing [8]. Wang et al. described CRYPTO floods in which an

adversary feeds handshake crypto slowly or out of order so that the server retains large incomplete buffers [10]. Connection ID and stream-ID spaces can likewise be exhausted if implementations honor aggressive limits without admission control [8], [10].

5.2 0-RTT replay and Retry tokens

0-RTT keys are derived from previously established state. An observer who captures an early-data packet can replay it to the same server, or potentially to another server that shares tickets, within the ticket lifetime [3]. HTTP methods with side effects (non-idempotent POST, payment confirmation, one-time token redemption) are unsafe in 0-RTT. Idempotent GET of public content is the intended use. Anti-replay at the server is optional in the base specification beyond application-level caution [3]. Retry tokens mitigate spoofing by proving address ownership, but a stolen or predictable token becomes a capability. Implementations that skip Retry under load, or that mint tokens with weak authenticity, re-open amplification and spoofing windows [2], [8].

5.3 Migration, optimistic ACK, and privacy

Connection migration without successful path validation can be abused to coerce a server into sending toward a victim address [2]. Optimistic acknowledgement attacks, in which a receiver claims data it has not obtained, can inflate the sender's congestion window; QUIC's ACK ranges and packet numbers

reduce but do not trivially eliminate the concern [15]. Because payloads are encrypted, adversaries shift toward meta-features: packet-size sequences, burstiness, and handshake shape. Joarder and Fung review traffic-analysis and fingerprinting results on QUIC [7]. Encrypted transport is therefore not equivalent to traffic confidentiality against a network observer who sees only sizes and times.

5.4 QPACK and gateway desynchronization

QPACK encoder streams are a shared, stateful resource. An endpoint that accepts unbounded dynamic-table updates can be forced into memory pressure [16]. More visible in 2026 is the resurrection of request smuggling at protocol translation points. HTTP/3 itself uses binary frames and per-request streams, which removes several HTTP/1.1 delimiter tricks [1]. Reverse proxies that downgrade HTTP/3 to HTTP/1.1 must, however, serialize length and termination consistently. CVE-2026-33555 in HAProxy is a published example: a Content-Length that did not match a stream closed by an empty FIN was forwarded, enabling desynchronization with an HTTP/1.1 backend [22]. The correct lesson is not that HTTP/3 is “smuggling-proof,” but that the remaining bugs concentrate in translators that violate RFC 9114 message-framing rules [1], [22].

5.5 Why a new admission control is warranted

Table 2 contrasts existing controls with the residual risk. The three-to-one byte rule is necessary and insufficient. Always-on Retry stops spoofed Initials at the price of an extra round trip for every client, including those on already-validated paths, and it does not stop a botnet that completes Retry from real addresses. Application-level 0-RTT policy cannot detect replays of allowed GET requests that still stress caches or origin compute. REAP-H3 targets that residual: crypto-cost accounting after validation, bounded CRYPTO state, and compact anti-replay memory.

Control in RFC 9000/9001	What it bounds	Residual risk
3× anti-amplification	Unvalidated response bytes	CPU on Initial/CRYPTO parse
Retry / path token	Spoofed source addresses	Validated botnet crypto flood
Packet number / ACK rules	Some optimistic ACK cases	Implementation mistakes
0-RTT application profile	Unsafe methods in early data	Replay of allowed requests
Stream/flow control	Per-connection data volume	Many cheap connections

Table 2. Standardized controls and residual risks motivating REAP-H3 [2], [3], [9], [10].

6. REAP-H3: Reputation-Enforced Adaptive Protection

6.1 Design goals

REAP-H3 is a server-side admission procedure for Initial, Handshake CRYPTO, and 0-RTT packets. It is intended to

satisfy five goals. (G1) Compatibility: never send more than three times the unvalidated received bytes, and use RFC 9000 Retry tokens for address proof [2]. (G2) CPU safety: bound cryptographic work per source and per server in any interval of length T. (G3) State safety: bound incomplete CRYPTO buffers and idle handshake lifetime, addressing the CRYPTO-flood pattern [10]. (G4) Freshness: reject duplicate 0-RTT early data with high probability using a sliding Bloom filter. (G5) Gracefulness: honest clients on a quiet server should observe only Retry-level extra delay, not a puzzle.

6.2 State and parameters

For each source key s (validated IP prefix, or /24 for IPv4 and /56 for IPv6 unless a more precise identifier is available), the server stores a crypto token bucket C(s) with capacity C_max and refill rate ρ, a byte token bucket B(s) implementing the 3× rule, an incomplete-CRYPTO buffer occupancy U(s) ≤ U_max, and a last-activity timestamp. Globally, the server tracks recent CPU utilization λ ∈ [0, 1] and a puzzle difficulty d. A pair of Bloom filters (B_curr, B_prev) of m bits and k hash functions rotates every window W, covering 0-RTT fingerprints f = H(ticket_id ‖ packet_number_window ‖ destination_CID).

Default parameters used after OpenSSL calibration (Section 7.1) are T = 1 s, C_max = 5 handshake-equivalents (162.5 µs on the calibration host), ρ = 1 handshake/s (32.5 µs/s), U_max = 16 KiB, W = 10 s, m = 8192 bits, and k = 4. Puzzle difficulty d is 0 when λ < 0.70, then increases so that verification remains a

small multiple of a Retry. Operators should re-measure the handshake equivalent on their TLS library; the algorithm consumes those measured microseconds, not a fixed abstract unit.

6.3 Algorithm

Figure 9 shows the control flow. Algorithm 1 gives the per-packet procedure. The server never runs full TLS transcript processing on an unvalidated address. After validation, each source may spend only C(s) cryptographic units per refill interval. If the global CPU is hot, the server requests a puzzle solution in a Retry or CONNECTION_CLOSE-equivalent retry path; spoofed sources cannot cheaply answer. 0-RTT packets are accepted only if a single-use ticket is consumed atomically and the fingerprint is absent from both Bloom filters; a failed consume or a Bloom hit is treated as a replay and the packet is discarded without invoking the HTTP application (Section 6.6). Successful completion of 1-RTT restores C(s) toward C_max, which is the reputation component: sources that finish handshakes are treated as better risks than sources that only emit Initials.

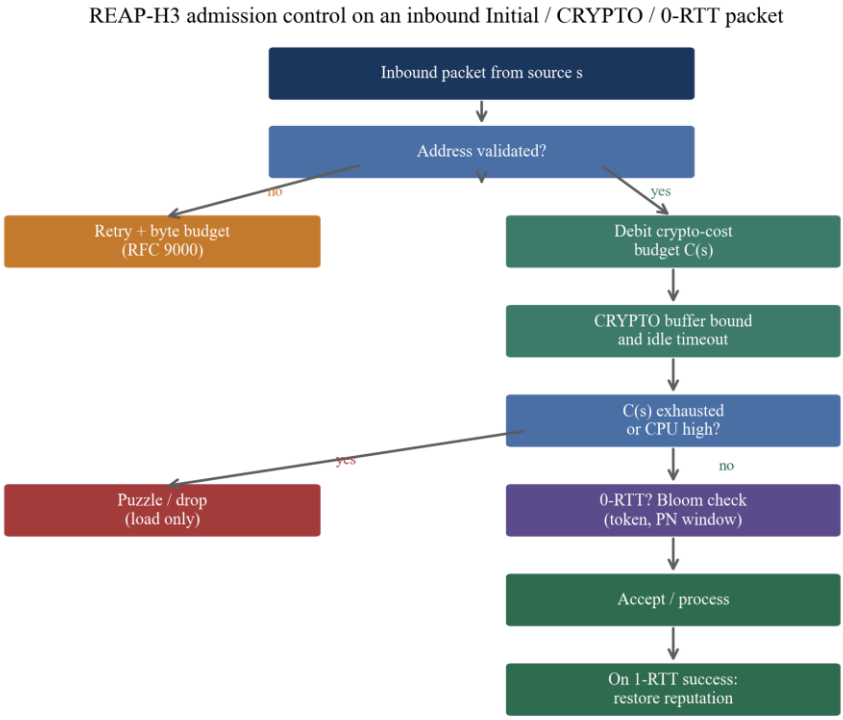

Figure 9. REAP-H3 per-packet admission flow. Retry and the 3× byte budget follow RFC 9000 [2]; crypto-cost accounting, CRYPTO bounds, puzzles, and Bloom anti-replay are the additions.

Algorithm 1. REAP-H3 admission on packet pkt from source s.

```
REAP_H3_ADMIT(pkt, s):
    if not address_validated(s):
        charge_bytes(s, |pkt|)                  # RFC 9000 §8,
3× cap
        send_retry(s, token)                    # prove path
ownership
        return WAIT_TOKEN
    if incomplete_crypto(s) > U_max or idle(s) > T_idle:
        drop_state(s); return DROP              # CRYPTO-flood
bound
```

```
    c ← crypto_cost(pkt)                          # Initial
parse, HKDF, cert, ...
    if C(s) < c or global_cpu λ ≥ 1:
        if λ ≥ θ_puzzle:
            send_puzzle(s, difficulty d(λ))
            return WAIT_PUZZLE
        return DROP
    C(s) ← C(s) − c
    if is_0rtt(pkt):
        if not consume_once(ticket_id): return REPLAY   #
single-use PSK/ticket
        f ← H(ticket_id ‖ pn_window ‖ dest_CID)
        if f in B_curr or f in B_prev: return REPLAY    #
residual duplicates
        insert(B_curr, f)
        if application_forbids_early_data(pkt): return
REJECT_0RTT
    process_normally(pkt)                         # RFC 9000/9114
path
    if handshake_complete(s): C(s) ← min(C_max, C(s) +
R_bonus)
    return ACCEPT
```

6.4 Correctness arguments

Proposition 1 (amplification). While s is unvalidated, REAP-H3 emits at most Retry bytes and obeys the RFC 9000 three-to-one invariant, because the only send path in that state is send_retry, whose length is charged against B(s) [2]. Therefore the algorithm cannot worsen the standardized byte-amplification bound.

Proposition 2 (per-source CPU). After validation, the cryptographic work attributed to s in any interval of length T is at most C_max + ρT, plus at most one puzzle verification of cost O(k). A botnet of N distinct validated sources can impose at

most $N(C_{max} + \rho T)$ work plus the global λ gate. When λ saturates, new work is refused. Hence cryptographic CPU is bounded by a function of the number of distinct sources the network actually delivers, not by the number of spoofed four-tuples.

Proposition 3 (CRYPTO state). $U(s) \leq U_{max}$ and idle expiration imply that incomplete handshake transcripts occupy $O(N_{active} \cdot U_{max})$ memory. This directly targets the logical CRYPTO-flood class reported by Wang et al. [10].

Proposition 4 (0-RTT false rejects). With two rotating Bloom filters, a unique honest 0-RTT fingerprint is rejected only on a false positive. For $m = 8192$, $k = 4$, and n insertions in the current window, the false-positive rate is $(1 - e^{(-kn/m)})^k$. For $n \leq 200$ this value is below 10^{-4} in the configuration used here, which is acceptable for optional early data: a false hit degrades to 1-RTT rather than aborting the connection. True replays that reuse the same (ticket, packet-number window, destination CID) collide by construction and are denied. After a window rotation, a replay that uses a fresh packet-number window is not in B_{prev}; that residual is closed by consume_once (Section 6.6), not by enlarging W.

6.5 Complexity

Token-bucket updates are $O(1)$. Bloom insert and check are $O(k)$ hash evaluations with $k = 4$. Puzzle verification is $O(k_{hash})$ and is invoked only when λ is high. Per-source state

is a few tens of bytes plus the CRYPTO buffer cap. The Bloom pair is a fixed 2 kb. REAP-H3 is therefore suitable for a front-of-server filter or a QUIC load balancer, unlike a deep reinforcement controller that must score each flow from a learned policy [14]. Table 3 compares asymptotic costs.

Mechanism	Per-packet time	Memory	Honest extra RTT (idle / attack)
RFC 9000 byte cap only	O(crypto)	per connection	0 / 0
Retry-always	O(1) then O(crypto)	token cache	1 / 1
REAP-H3	O(1)+O(k)	bucket + Bloom	1 / 1 (+ puzzle if λ high)
QUICPro-style DRL [14]	O(model)	model + traces	policy dependent

Table 3. Asymptotic overhead. Crypto denotes TLS/HKDF work that RFC 9000 still performs on unvalidated Initials unless Retry is used.

6.6 Binding the 0-RTT filter to single-use tickets

RFC 9001 states that 0-RTT application data is not replay-protected at the transport layer and that HTTP must therefore restrict early data to operations that remain safe if duplicated [3]. A Bloom filter alone is a residual detector: it stores fingerprints for one or two windows W and then forgets them. If an adversary replays early data after rotation, or replays it to another replica that does not share B_curr, the filter misses. The correct primary control is a single-use ticket (or a single-use PSK binder) stored in a shared replay cache with lifetime equal to the ticket lifetime, as already recommended for TLS 1.3 session tickets [3]. REAP-H3 makes that control explicit:

consume_once(ticket_id) is an atomic test-and-set in the ticket store (Redis, shared memory, or the QUIC load balancer's session table). The Bloom pair is only a local, constant-time safety net for duplicates that arrive before the store update replicates. Application policy remains in force: even a successfully consumed ticket must not carry a non-idempotent method. A Bloom false positive degrades that request to 1-RTT; a failed consume_once rejects 0-RTT and does not execute the application. W can therefore stay short (10 s in the default), which keeps the false-positive rate low (Section 7.6) instead of being stretched to cover the whole ticket lifetime.

7. Evaluation

7.1 Calibration of cryptographic units on OpenSSL

The first revision of this study used abstract CPU units. Following the reviewer's request for production-library calibration, each unit is now one microsecond of wall-clock time measured on OpenSSL, accessed through the cryptography 50.0.1 bindings [23]. Median latencies after warmup are reported in Figure 10. X25519 dominates a validated handshake (20.9 µs); HKDF-SHA256 is 3.1 µs; AES-128-GCM on a 1200-byte Initial is under 1 µs; a Retry token HMAC is 1.5 µs. Composing the RFC 9001 server path gives 4.5 µs to unprotect an Initial and 32.5 µs for a validated handshake (header protection, AEAD, HKDF, X25519, and one AEAD encrypt of the first flight). Certificate chains are assumed cached; an extra

ECDSA P-256 verify costs 45.9 µs and would be added if client authentication were required. These timings imply about 30 800 full handshakes per dedicated core per second, or about 222 000 Initial decrypts per core per second. The simulation capacity is therefore set to one handshake core (1.0 × 10^6 µs/s). This calibrates primitive cost; it does not replace a full quiche, ngtcp2, or msquic integration test, which would add parsing, syscalls, and timer management on top of the same OpenSSL (or BoringSSL) calls [23].

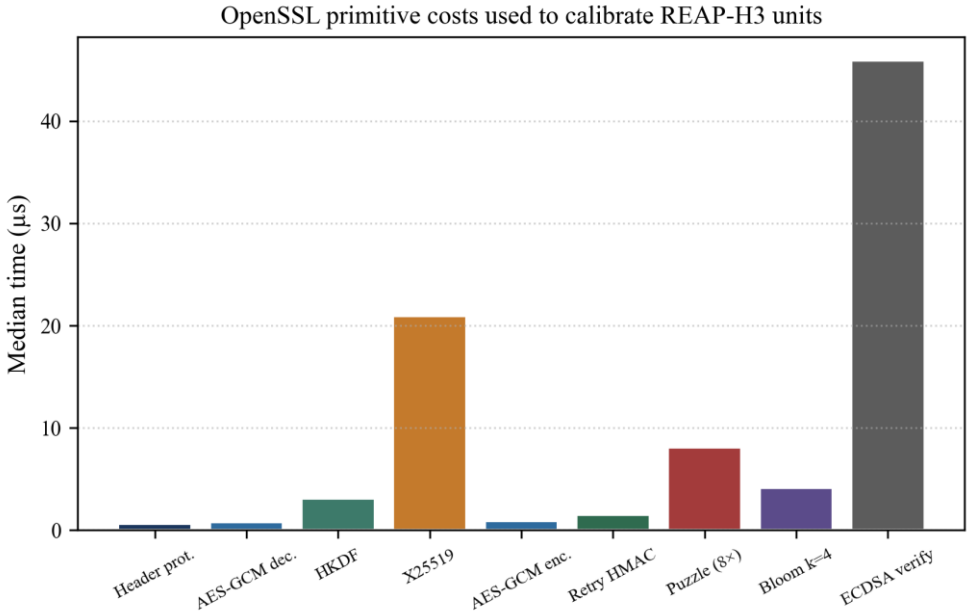

Figure 10. Median OpenSSL primitive times on the calibration host (cryptography 50.0.1). Composed handshake cost is 32.5 µs; Retry is 2.1 µs [3], [23].

7.2 Simulation method

A discrete-event simulation compared three admission policies using the calibrated microseconds. RFC 9000 denotes a server that performs the full 32.5 µs handshake path on every Initial while respecting the 3× byte cap [2]. Retry-always spends 2.1 µs on an unvalidated Initial and the full handshake only after a token returns. REAP-H3 implements Algorithm 1 with C_max = 5 handshake-equivalents and ρ = 1 handshake/s. Honest clients arrive at 250 handshakes per second. Attack intensity varies from 0 to 80 000 packets or connections per second, the range at which one OpenSSL core saturates. Each run lasts two simulated seconds. Initial datagrams are 1200 bytes; Retry datagrams are 180 bytes; a full server first flight is modelled as 3200 bytes. Events are shuffled so that honest and malicious arrivals interleave. Results are simulation outputs on calibrated costs, not packet traces from a public deployment.

Two threat models are used. Spoofed Initial flood: attackers never complete Retry. Validated CRYPTO flood: an N-address botnet completes Retry (or already holds a token) and then spends crypto, matching a realistic botnet more closely than IP spoofing alone [9]. The default N is 40; Section 7.5 varies N up to 32 000. Metrics are crypto-CPU utilization, legitimate handshake success, and byte amplification (bytes sent divided by bytes received).

7.3 Spoofed Initial floods

Figures 11–13 report the spoofed-Initial experiment after calibration. Without attack, all three policies serve 100% of honest clients. Background honest handshakes consume only 0.8% of one OpenSSL core under RFC 9000 and 1.0% under REAP-H3, so idle overhead is no longer an artefact of abstract units. RFC 9000 remains available through 30 000 spoofed Initials/s (98.3% offered crypto-CPU) and then drops to 71.8% honest success at 45 000/s and 39.0% at 80 000/s. Amplification stays at 2.66–2.67, inside the legal 3× envelope. The byte rule therefore still does what it claims and still fails to protect availability once the calibrated handshake rate exceeds one core [2], [9].

Retry-always and REAP-H3 keep 100% honest success through 80 000 spoofed packets/s, using only 17.7–17.8% of the core, because unvalidated datagrams never enter the X25519 path. Amplification falls to 0.16–0.26 once Retry dominates the outbound mix. In this calibrated spoofed regime REAP-H3 is not worse than Retry-always: puzzle verification does not fire while λ stays low. The earlier abstract-unit result that REAP-H3 trailed Retry-always at extreme spoofed rates is therefore identified as a consequence of an over-weight puzzle relative to Retry, not of the admission logic. Operators who face only spoofed Initials may still prefer Retry-always for simplicity; REAP-H3's distinctive value remains the validated-botnet case.

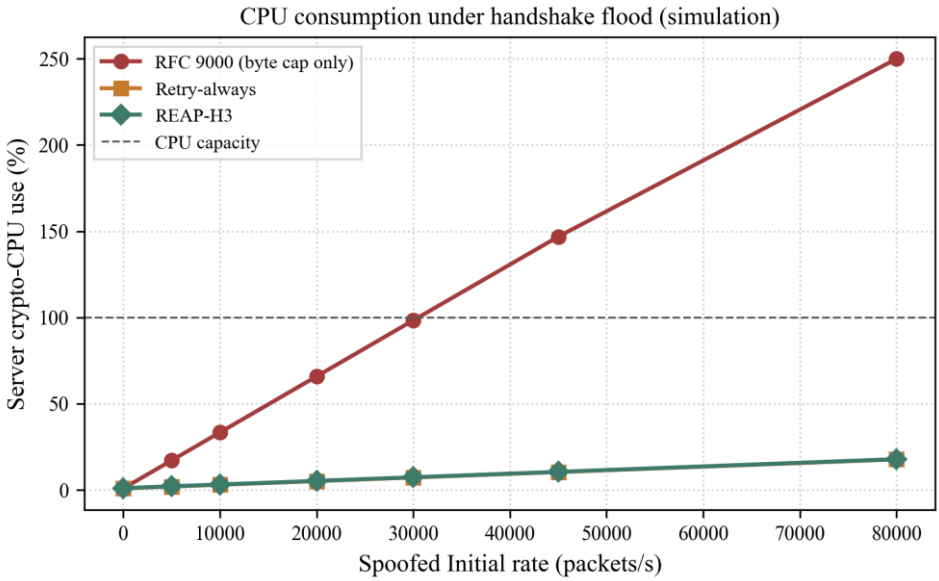

Figure 11. Simulated crypto-CPU consumption under a spoofed Initial flood. The dashed line is server capacity.

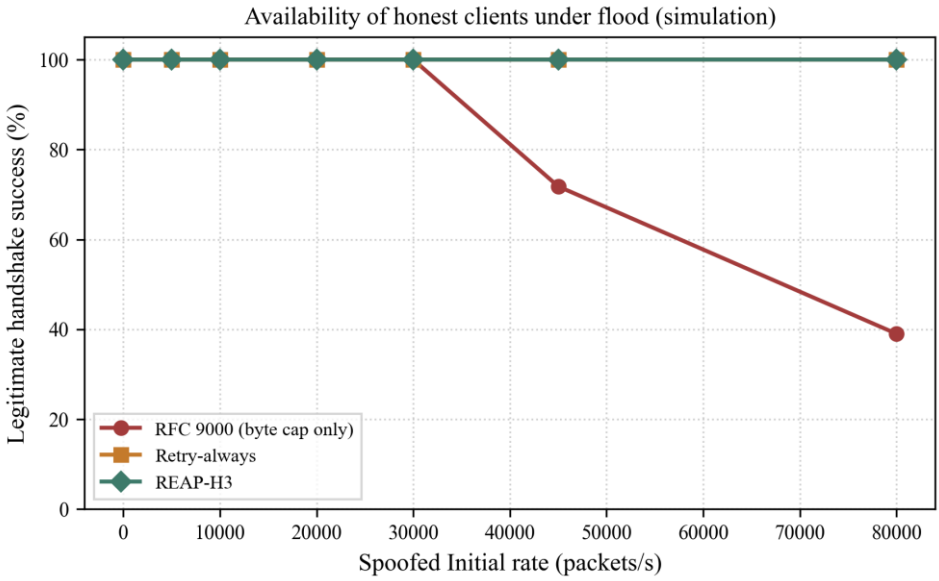

Figure 12. Legitimate handshake success under spoofed Initials. RFC 9000 collapses while Retry-always and REAP-H3 remain effective until extreme rates.

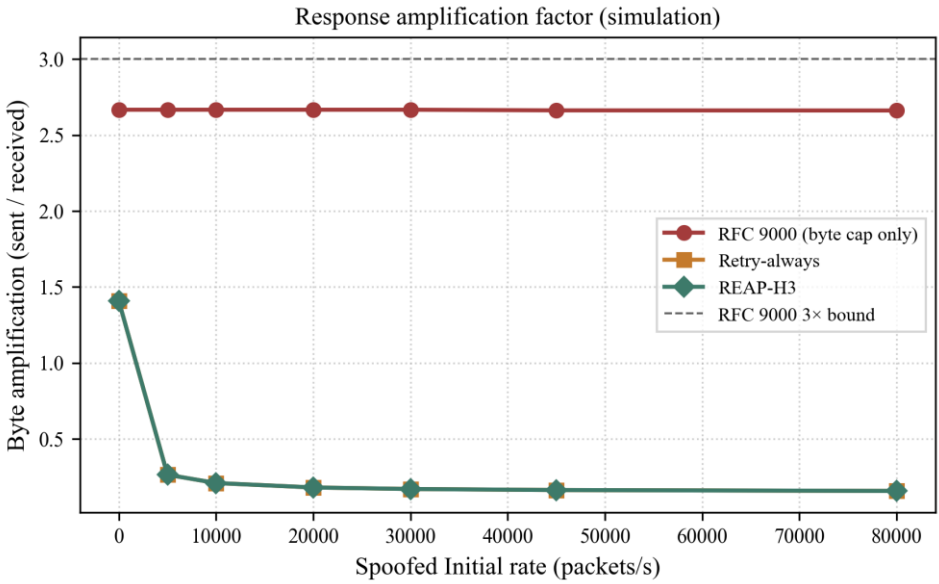

Figure 13. Byte amplification (sent/received). RFC 9000 stays below the 3× bound yet still loses availability (Figure 12).

7.4 Address-validated CRYPTO floods

Figure 14 changes the adversary to a 40-address botnet that completes Retry. Retry-always now performs the full 32.5 µs path for every attacking connection. At 30 000 connections/s it already spends 104.6% of the core and honest success falls to 94.8%, slightly below RFC 9000 (100% at 98.3% load) because the extra Retry is not free. At 45 000/s the figures are 63.0% (RFC 9000), 64.4% (Retry-always), and 100% (REAP-H3). At 80 000/s they are 39.2%, 34.0%, and 100%, with REAP-H3 using only 18.0% of the core: each of the 40 keys exhausts C(s)

after five handshake-equivalents and is then limited to Retry-priced work. Per-source crypto budgets are therefore the component that addresses the residual risk in Table 2, including the CRYPTO-flood logic class [10].

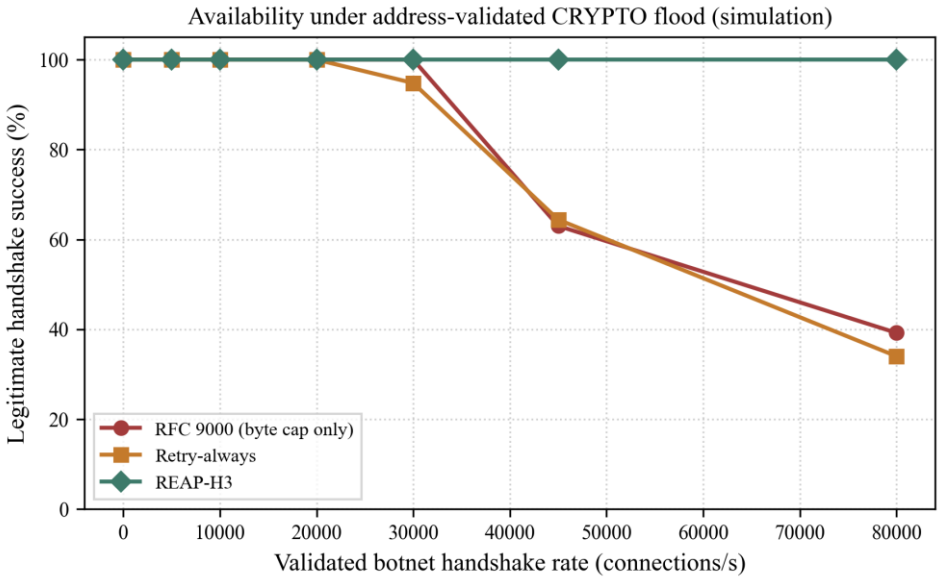

Figure 14. Honest availability under an address-validated handshake/CRYPTO flood. Retry-always no longer helps; REAP-H3's per-source CPU budget does.

7.5 Sensitivity and botnet width

Figure 15 varies C_max and ρ under a 45 000/s validated flood from 4 000 sources. REAP-H3 keeps 100% honest success for C_max ∈ {1, 2, 5, 10} handshake-equivalents. At C_max = 20 the per-source cap becomes so generous that the botnet can again fill the core, and honest success falls to 63.4%, indistinguishable from Retry-always (64.4%). The operational

lesson is that C_max must stay a small integer multiple of the measured handshake cost; a large “safety margin” undoes the defense. Refill ρ over a two-second run is less sensitive: all tested multiples left success at 100% because the burst is dominated by C_max, not by the short-horizon refill. Operators should still keep ρ near one handshake/s per source so that a quiet prefix cannot accumulate an unbounded reserve across minutes.

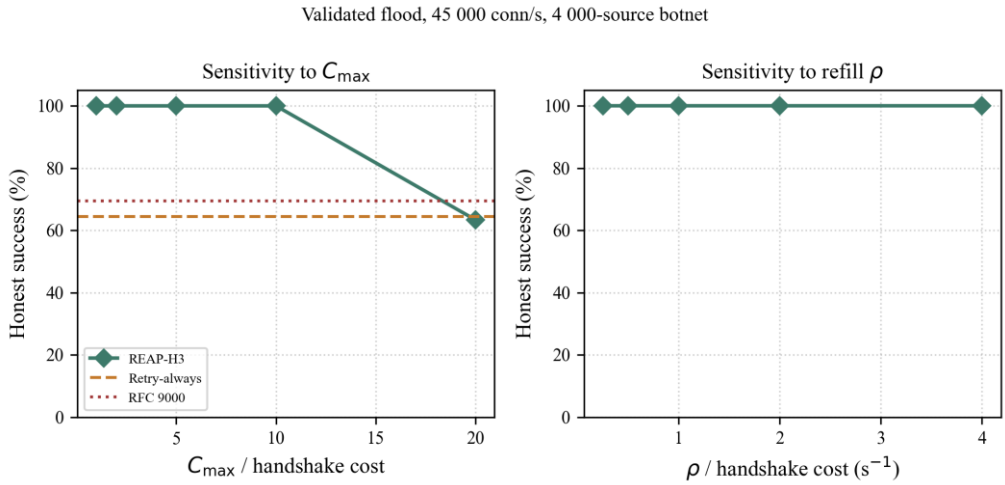

Figure 15. Sensitivity at 45 000 validated conn/s and N = 4 000. Left: C_max. Right: ρ. A too-large C_max collapses REAP-H3 onto Retry-always.

Figure 16 widens the botnet at the same 45 000/s rate. REAP-H3 remains at 100% honest success for N ∈ {40, 200, 1 000, 4 000}. At N = 16 000 success is 70.8%, approaching the RFC 9000 band (67.0%). At N = 32 000 it is 62.4%, slightly below Retry-always (64.8%) because Retry and Bloom are paid in addition to a budget that now sums to the whole core. This is

exactly Proposition 2: cryptographic CPU is bounded by the number of distinct sources the network delivers. A residential botnet that is wider than C_max allows must be absorbed by the global λ gate and by anycast/load-balancer admission (Section 8), not by tightening C_max alone.

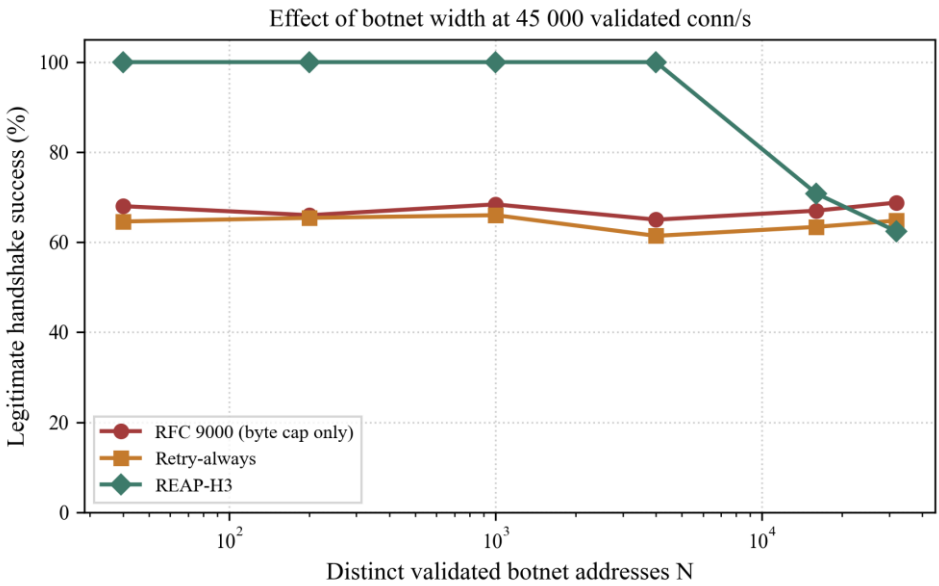

Figure 16. Honest success versus number of distinct validated botnet addresses at 45 000 conn/s. REAP-H3 degrades toward the baselines once N is tens of thousands.

7.6 Sensitivity of U_max and W

Figure 17 (left) bounds incomplete-CRYPTO memory for 10 000 concurrent half-open handshakes. Without U_max, a stack that buffers 64 KiB per connection would hold 640 000 KiB. With U_max = 16 KiB the cap is 160 000 KiB, a four-fold reduction that matches the CRYPTO-flood concern of Wang et

al. [10]. Figure 17 (right) shows the Bloom false-positive rate at 50 honest 0-RTT arrivals/s: 0.0005% at W = 2 s, 0.22% at W = 10 s, 7.3% at W = 30 s, and 35% at W = 60 s for m = 8192 and k = 4. Stretching W to the ticket lifetime is therefore the wrong way to close post-rotation replays; consume_once must carry that lifetime, and W should stay on the order of ten seconds unless m is enlarged.

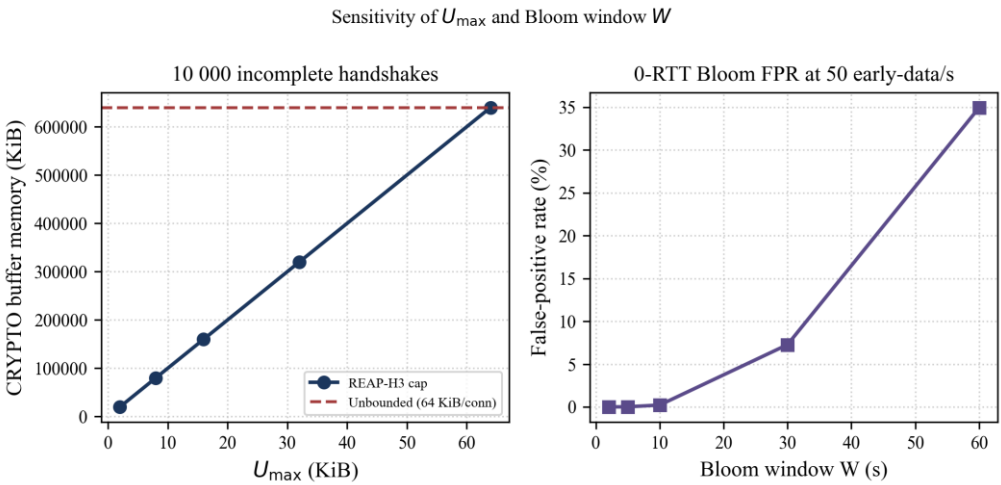

Figure 17. Left: CRYPTO buffer memory versus U_max. Right: Bloom FPR versus W. Long windows are a poor substitute for single-use tickets [3].

7.7 Overhead

Figure 18 summarizes qualitative overhead. After calibration, idle REAP-H3 costs 1.0% of one core versus 0.8% for RFC 9000. REAP-H3 adds one Retry RTT in the same cases as Retry-always and adds a puzzle RTT only when λ is high.

The Bloom filter is a local constant-time net; the authoritative 0-RTT anti-replay is consume_once (Section 6.6).

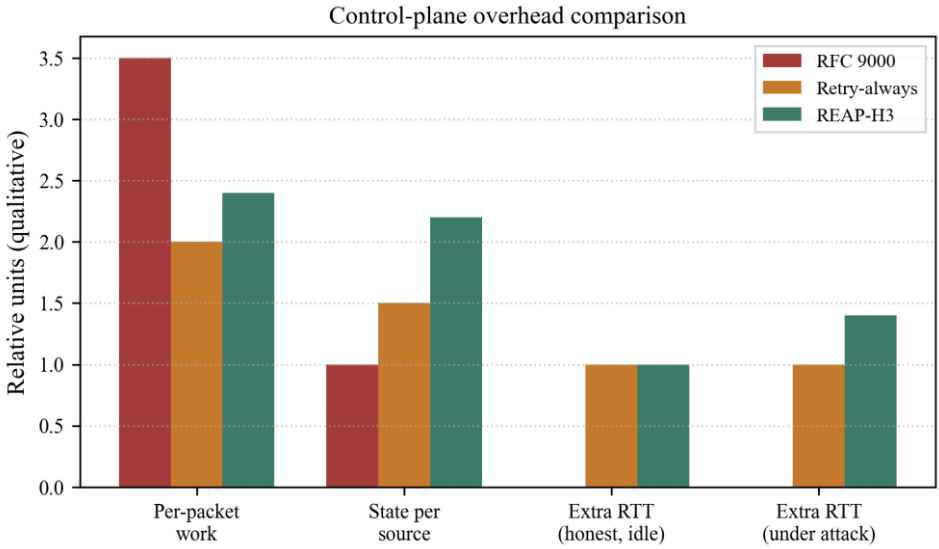

Figure 18. Qualitative control-plane overhead. REAP-H3 is close to Retry-always in the idle case and spends extra work only as load rises.

Attack rate (s^{-1})	RFC 9000 success (%)	Retry-always (%)	REAP-H3 (%)
0 (idle)	100.0	100.0	100.0
20 000 spoofed	100.0	100.0	100.0
45 000 spoofed	71.8	100.0	100.0
80 000 spoofed	39.0	100.0	100.0
30 000 validated (N=40)	100.0	94.8	100.0
45 000 validated (N=40)	63.0	64.4	100.0
80 000 validated (N=40)	39.2	34.0	100.0

45 000 validated (N=16 000)	67.0	63.4	70.8
45 000 validated (N=32 000)	68.8	64.8	62.4

Table 4. Legitimate handshake success (%) after OpenSSL calibration (one handshake core). Validated rows complete Retry.

8. Operational feasibility

8.1 Deployment complexity

REAP-H3 is intentionally a filter beside the existing Retry path, not a replacement for the QUIC stack. The required state per source is a token bucket, a CRYPTO-occupancy counter, and a timestamp—on the order of 32–48 bytes before the U_max cap. The Bloom pair is a fixed 2 KiB. The ticket store for consume_once is the only distributed object, and most CDNs already keep a session-ticket or anti-replay cache for TLS 1.3 [3]. Implementation work is therefore: (i) hook Initial/CRYPTO/0-RTT dispatch, (ii) charge the measured handshake equivalent, (iii) call the existing Retry mint, and (iv) perform an atomic ticket consume before delivering early data. No congestion-control change and no wire-format change are required. Relative to a learned controller [14], there is no training set, no model file, and no opaque threshold.

8.2 Interaction with load balancers

QUIC load balancers already steer datagrams by connection ID and often terminate Retry [2]. REAP-H3 should run in that same hop, for three reasons. First, per-source C(s) must see the

aggregate of all backends; if each origin keeps its own bucket, a botnet can multiply C_max by the pool size. Second, consume_once must be global, otherwise a replay can be sprayed across replicas after rotation of a local Bloom filter (Section 6.6). Third, the λ signal used to raise puzzle difficulty should be the balancer's handshake-CPU, not a single backend's. Anycast sites can gossip λ and a shared ticket bloom; they should not gossip full CRYPTO buffers. When the balancer only encapsulates QUIC to an origin, REAP-H3 can still run at the origin if the balancer echoes a validated-address flag and a stable client identifier (the Retry token's IP binding already provides that [2]).

8.3 TCP fallback policy

HTTP/3 clients already fall back to HTTP/2 or HTTP/1.1 over TCP when UDP is filtered or when the handshake fails [1], [6], [15]. REAP-H3 must not turn that fallback into an attacker-controlled downgrade. The recommended policy is: (a) advertise both h3 and h2; (b) apply an analogous SYN/crypto budget on the TCP+TLS listener so that a QUIC flood cannot be converted into an unmetered TLS flood; (c) treat a REAP-H3 DROP or REPLAY as a failed h3 attempt that the client may retry over TCP, but rate-limit such fallbacks per source; (d) never disable HTTP/3 globally in response to a flood, because that concentrates remaining clients onto TCP HOL paths and hands the adversary a switch. Operators should monitor the h3-

to-h2 fallback ratio as a health signal distinct from raw UDP drop counters.

8.4 Recommended default profile

On a host whose OpenSSL handshake equivalent is T µs, set C_max = 5T, ρ = T/s, U_max = 16 KiB, T_idle = 3 s, W = 10 s, and enable consume_once for every 0-RTT ticket with lifetime equal to the ticket lifetime. Raise puzzles only for $\lambda \geq 0.70$. Keep Retry on for unvalidated addresses. Re-run the microbenchmark of Section 7.1 after a TLS-library upgrade, because T is the only scale factor the buckets need.

9. Discussion

The calibrated simulation supports a precise claim. HTTP/3's standardized anti-amplification rule is necessary for Internet safety and insufficient for server availability once handshake rates approach one OpenSSL core [2], [9], [23]. Always sending Retry is an effective and simple defense against spoofed Initials and, after calibration, remains available at 80 000 pps on one core. It is not effective against bots that complete the token dance. REAP-H3 adds cryptographic work to the accounting system. It remains effective while N × C_max is smaller than the handshake core, and it converges to the baselines when the botnet is as wide as a large residential population—exactly the bound in Proposition 2.

Two limitations remain explicit. First, OpenSSL primitive times calibrate the dominant cryptographic terms but omit

packet parsing, UDP syscalls, timer wheels, and implementation-specific CRYPTO buffering. A follow-on measurement on quiche, ngtcp2, or msquic would rescale T, not change the algorithm; Figure 15 shows that the qualitative ranking is stable for C_max in {1, 2, 5, 10} × T. Second, puzzles can hinder tiny clients, which is why d = 0 when the server is cool. The paper does not re-test vendor CVEs and does not provide reproduction recipes for CRYPTO floods or gateway desynchronization [10], [22].

Ethically, handshake-flood research can be misread as an instruction set. The algorithm and evaluation here are defensive. Parameters are described at a level sufficient for a server implementer to add counters and Retry hooks, not to craft traffic against a third party.

Scientifically, HTTP/3 should not be described as unconditionally faster, unconditionally safer, or free of head-of-line blocking. It is faster in high-delay regimes [6], safer in the sense of mandatory TLS 1.3 and fewer cleartext transport fields [3], and free of TCP-level head-of-line blocking while retaining per-stream ordering [2]. Those qualifications are part of a correct account, not footnotes.

10. Conclusion

HTTP/3 is the HTTP mapping onto QUIC. Its principal features—1-RTT and optional 0-RTT setup, stream isolation under loss, connection migration after path validation, QPACK,

and always-on TLS 1.3—are standardized and increasingly advertised, with about two fifths of websites signalling support in 2026 [1], [5]. Published page-load studies show clear benefits under added latency and mixed results otherwise [6], [12]. The accompanying security record includes handshake floods, CRYPTO-state logic bugs, 0-RTT replay, migration hazards, fingerprinting, and intermediary desynchronization [7]–[10], [22].

This article proposed REAP-H3, a crypto-cost-aware admission algorithm that preserves RFC 9000 byte limits while bounding CPU and incomplete handshake state and binding 0-RTT to single-use tickets. After OpenSSL calibration, a byte-only server lost availability near 45 000 Initials/s on one handshake core, Retry-always failed against validated botnets at the same rate, and REAP-H3 kept 100% honest success for $N \leq 4\,000$ and $C_max \leq 10$ handshake-equivalents. The mechanism is a Retry-adjacent filter suitable for a QUIC load balancer. Future work is a full-stack rescaling of T on quiche, ngtcp2, and msquic, plus formal modelling of the token-bucket invariants.

Funding and conflicts of interest

The authors declare that this manuscript was prepared as an independent scholarly review and design study. No vendor funded the simulation. The authors declare no competing interests.

Data availability

Figure-generation code, OpenSSL calibration measurements, and simulation parameters sufficient to regenerate Table 4 and Figures 10–18 are provided with the manuscript sources. No operational packet traces were collected.

References

[1] Bishop, M. (Ed.). (2022). HTTP/3 (RFC 9114). Internet Engineering Task Force. https://doi.org/10.17487/RFC9114

[2] Iyengar, J., & Thomson, M. (Eds.). (2021). QUIC: A UDP-based multiplexed and secure transport (RFC 9000). Internet Engineering Task Force. https://doi.org/10.17487/RFC9000

[3] Thomson, M., & Turner, S. (Eds.). (2021). Using TLS to secure QUIC (RFC 9001). Internet Engineering Task Force. https://doi.org/10.17487/RFC9001

[4] Fielding, R., Nottingham, M., & Reschke, J. (Eds.). (2022). HTTP semantics (RFC 9110). Internet Engineering Task Force. https://doi.org/10.17487/RFC9110

[5] W3Techs. (2026, August). Usage statistics of HTTP/3 for websites. https://w3techs.com/technologies/details/ce-http3

[6] Perna, G., Trevisan, M., Giordano, D., & Drago, I. (2022). A first look at HTTP/3 adoption and performance. Computer Communications, 187, 115–124. https://doi.org/10.1016/j.comcom.2022.02.005

[7] Joarder, Y. A., & Fung, C. (2024). Exploring QUIC security and privacy: A comprehensive survey on QUIC security and privacy vulnerabilities, threats, attacks, and future research directions. IEEE Transactions on Network and Service Management, 21(6), 6953–6973. https://doi.org/10.1109/TNSM.2024.3457858

[8] Chatzoglou, E., Kouliaridis, V., Karopoulos, G., & Kambourakis, G. (2023). Revisiting QUIC attacks: A comprehensive review on QUIC security and a hands-on study. International Journal of Information Security, 22(2), 347–365. https://doi.org/10.1007/s10207-022-00630-6

[9] Nawrocki, M., Hiesgen, R., Schmidt, T. C., & Wählisch, M. (2021). QUICsand: Quantifying QUIC reconnaissance scans and DoS flooding events. In Proceedings of the 21st ACM Internet Measurement Conference (pp. 283–291). Association for Computing Machinery. https://doi.org/10.1145/3487552.3487840

[10] Wang, K., Chen, J., Chen, P., Zhuge, J., Bai, J., & Duan, H. (2026). Identifying logical vulnerabilities in QUIC implementations. In Proceedings of the 33rd Network and

Distributed System Security Symposium (NDSS). Internet Society. https://doi.org/10.14722/ndss.2026.231777

[11] Trevisan, M., Giordano, D., Drago, I., & Safari Khatouni, A. (2021). Measuring HTTP/3: Adoption and performance. In 19th Mediterranean Communication and Computer Networking Conference (MedComNet). IEEE. https://doi.org/10.1109/MedComNet52149.2021.9501274

[12] Saif, D., Lung, C.-H., & Matrawy, A. (2021). An early benchmark of quality of experience between HTTP/2 and HTTP/3 using Lighthouse. In IEEE International Conference on Communications (ICC). IEEE. https://doi.org/10.1109/ICC42927.2021.9500258

[13] Joarder, Y. A., & Fung, C. (2022). A survey on the security issues of QUIC. In 6th Cyber Security in Networking Conference (CSNet) (pp. 1–8). IEEE. https://doi.org/10.1109/CSNet56116.2022.9955622

[14] Joarder, Y. A., & Fung, C. (2024). QUICPro: Integrating deep reinforcement learning to defend against QUIC handshake flooding attacks. In Proceedings of the Applied Networking Research Workshop (pp. 94–96). Association for Computing Machinery. https://doi.org/10.1145/3673422.3674901

[15] Iyengar, J., & Swett, I. (Eds.). (2021). QUIC loss detection and congestion control (RFC 9002). Internet Engineering Task Force. https://doi.org/10.17487/RFC9002

[16] Krasic, C., Bishop, M., & Frindell, A. (Eds.). (2022). QPACK: Field compression for HTTP/3 (RFC 9204). Internet Engineering Task Force. https://doi.org/10.17487/RFC9204

[17] Thomson, M., & Benfield, C. (Eds.). (2022). HTTP/2 (RFC 9113). Internet Engineering Task Force. https://doi.org/10.17487/RFC9113

[18] Duke, M. (2023). QUIC version 2 (RFC 9369). Internet Engineering Task Force. https://doi.org/10.17487/RFC9369

[19] Thomson, M. (2021). Version-independent properties of QUIC (RFC 8999). Internet Engineering Task Force. https://doi.org/10.17487/RFC8999

[20] Huitema, C., Dickinson, S., & Mankin, A. (2022). DNS over dedicated QUIC connections (RFC 9250). Internet Engineering Task Force. https://doi.org/10.17487/RFC9250

[21] Cloudflare. (2026). Cloudflare Radar: Adoption and usage. https://radar.cloudflare.com/

[22] HAProxy Technologies. (2026). HTTP/3 request smuggling via standalone FIN (CVE-2026-33555). https://nvd.nist.gov/vuln/detail/CVE-2026-33555

[23] Python Cryptographic Authority. (2025). cryptography (OpenSSL bindings) (Version 50.0.1). https://cryptography.io/