Network Time Protocols
Internet-Draft
Intended status: Experimental
Expires: 5 February 2026

G. McCollum
Cisco Systems

4 August 2025

Time Synchronization over QUIC
draft-mccollum-ntp-tsq-01

Abstract

This document proposes a modern, secure, and extensible time synchronization protocol designed to operate over the QUIC transport protocol. Known as TSQ (Time Synchronization over QUIC), this protocol aims to address the limitations of traditional NTP by leveraging QUIC's encryption, widespread UDP/443 acceptance, and multiplexed stream capabilities. TSQ is designed for contemporary deployment environments, including enterprise networks, cloud-native systems, containers, and mobile devices, where traditional UDP-based NTP struggles with security, scalability, or operational reliability.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 February 2026.

Copyright Notice

Table of Contents

1. Introduction

Time synchronization is foundational to modern computing. It underpins authentication systems, log correlation, distributed transactions, and more. NTP, the current standard, was designed in a different era and brings challenges related to security, deployment compatibility, and extensibility. TSQ is proposed as a new protocol built directly on top of QUIC, leveraging its modern transport features to provide secure, authenticated, and operationally-friendly time synchronization.

2. QUIC Overview for TSQ

TSQ is built on top of the QUIC transport protocol, as defined in [RFC9000] and extended with Datagram support in [RFC9221]. This section summarizes key QUIC features relevant to TSQ.

QUIC is a secure, multiplexed, and reliable transport protocol that operates over UDP. It integrates TLS 1.3 into its handshake for encryption and authentication. QUIC was designed to support fast connection setup, mobility, and user-space implementation.

TSQ uses two QUIC delivery mechanisms:

* *QUIC Streams:* Bidirectional, ordered, and reliable channels. TSQ can use a single QUIC stream per exchange to deliver requests and responses. Stream Mode provides delivery guarantees and allows inclusion of signatures for auditability.

* *QUIC Datagrams:* Unordered, unreliable messages delivered over QUIC connections. Datagrams are similar to UDP packets, with no retransmission. TSQ's Datagram Mode trades reliability for reduced latency and overhead.

QUIC connections are established over UDP, typically using port 443. TSQ clients MUST negotiate the application protocol identifier "tsq" via ALPN during the QUIC handshake. QUIC handles encryption, retransmission (for streams), congestion control, and connection migration internally, allowing TSQ to focus on the semantics of time synchronization.

Unlike Stream Mode, Datagram Mode does not provide reliable delivery or built-in integrity guarantees. Unless the response includes a Signature Block TLV (Type 255), clients have no cryptographic proof of authenticity or freshness. Applications requiring authentication MUST use Stream Mode or enforce signed Datagram responses.

Further details on Stream and Datagram modes are provided in Section 4. Readers unfamiliar with QUIC are encouraged to review [RFC9000] for core behavior and [RFC9221] for datagram extensions.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

* *TSQ:* Time Synchronization over QUIC protocol.

* *QUIC:* A secure, multiplexed transport protocol over UDP.

* *TLV:* Type-Length-Value, a flexible message format.

* *Precision Mode:* An OPTIONAL TSQ feature for low-jitter synchronization.

* *RTT:* Round-trip time.

* *NTP:* Network Time Protocol. A widely deployed protocol for synchronizing clocks over IP networks, formally specified in [RFC5905]. TSQ inherits its timestamp format from NTP but differs in transport, security, and extensibility.

4. Scope and Goals

TSQ is intended to:

* Provide secure and authenticated time synchronization

* Support modern deployment scenarios

* Operate in environments where UDP/123 is blocked

* Be extensible and future-proof

* Scale for enterprise and cloud

TSQ is _not_ intended to:

* Replace NTP in ultra-precise or constrained devices

* Replace public NTP infrastructure without optimization

5. TSQ Protocol Overview

TSQ uses QUIC as its transport protocol, establishing secure connections for the exchange of time synchronization messages. TSQ supports two transport modes: QUIC streams and QUIC datagrams. Both modes carry the same TLV-formatted message content, but differ in reliability, latency, and use cases.

Stream Mode: Uses QUIC's reliable, ordered streams. This mode ensures delivery and supports features like message signatures, making it suitable for authenticated and auditable time synchronization exchanges.

Datagram Mode: Uses QUIC DATAGRAM frames [RFC9221] for lower-latency, unordered delivery. This mode is suitable for fast, opportunistic synchronization but offers no retransmission or delivery guarantees. Applications requiring integrity validation SHOULD use Stream Mode.

A typical TSQ exchange (in either mode) consists of the following steps:

1. Client opens a QUIC connection to the TSQ server (typically using UDP/443) and negotiates the TSQ protocol via ALPN.

2. Client sends a TSQ Request containing a cryptographically secure nonce. To support stateless server deployments and ensure robustness against replay attacks, clients MUST generate nonces that are unpredictable and sufficiently long to avoid collisions.

3. Server replies with a TSQ Response containing the echoed nonce, the time the request was received (T2), and the time the response was sent (T3).

4. Client locally records its own transmit (T1) and receive (T4) times, computes the round-trip time (RTT), and derives the clock offset for synchronization.

6. TSQ Protocol Operation

6.1. Connection Establishment

TSQ uses QUIC for transport, leveraging its handshake for mutual authentication and encryption. Clients MUST negotiate the TSQ protocol using the ALPN identifier "tsq" during the QUIC handshake. Short-lived or resumed QUIC connections MAY be used to optimize performance.

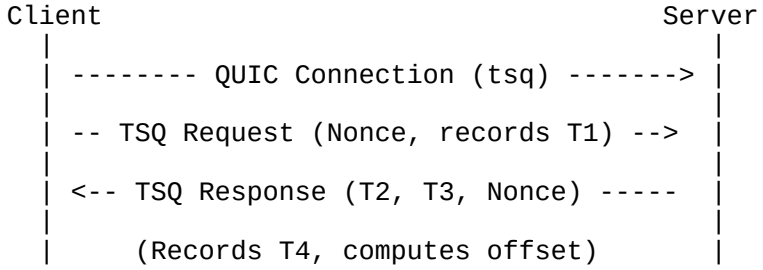
QUIC supports bidirectional stream multiplexing, allowing multiple independent exchanges to occur simultaneously over a single connection. While most TSQ clients use one stream per request-response cycle, multiplexing can be useful in advanced deployments. For example, issuing parallel time and metadata queries, allowing multiple subsystems to share a connection, or supporting audit and monitoring tools without blocking time synchronization traffic. This capability is optional and typically not needed for simple time clients.

6.2. Message Exchange Flow

Each TSQ exchange consists of a request and response over a QUIC stream. Clients and servers MUST capture the following timestamps:

* *T1 (Client Records Time):* The moment the client sends the TSQ Request.
* *T2 (Server Receive Time):* The moment the server receives the TSQ Request.
* *T3 (Server Transmit Time):* The moment the server sends the TSQ Response.
* *T4 (Client Records Time):* The moment the client receives the TSQ Response.

These timestamps enable the client to calculate round-trip time (RTT) and clock offset. T1 and T4 are recorded by the client and are not transmitted. T2 and T3 are included in the TSQ Response message payload.



After recording T1 and T4 locally and receiving T2 and T3 from the server, the client calculates synchronization metrics using the following formulas:

* *Round-Trip Time (RTT):* (T4 T1) (T3 T2)
* *Clock Offset:* ((T2 T1) + (T3 T4)) / 2

These formulas are equivalent to those used in NTP [[RFC5905]] and provide a symmetric view of network delay and server time. The use of a cryptographic nonce ensures freshness and guards against replay, while optional digital signatures provide authenticity guarantees for T2 and T3 if required.

6.3. Message Format and TLVs

6.3.1. TLV Type Registry

The following table summarizes the currently defined TSQ TLV types:

Type	Name	Direction	Required	Description
1	Nonce	Request and Response	Yes	Identifies exchange and prevents replay
2	Receive Timestamp	Response	Yes	Time server received request (T2)
3	Send Timestamp	Response	Yes	Time server sent response (T3)
4246	Reserved		No	Reserved for future extension
247	Metadata Query Request	Request	No	Client requests structured metadata in response
248	Clock Quality Info	Response	No	Estimated clock accuracy and stability
249	Error	Response (Stream Only)	Conditional	Structured error report from server
250	Precision Mode Req/ Ack	Request and Response	No	Negotiate symmetric message sizing
251	Time Source Info	Response	No	Stratum and source metadata
252	Signature Request	Request	No	Client requests signed response
253	Metadata	Request	No	Deployment

		and Response		annotations (e.g., site ID)
254	Padding	Request and Response	No	Zero-filled padding for alignment
255	Signature Block	Response	Conditional	Cryptographic signature of message

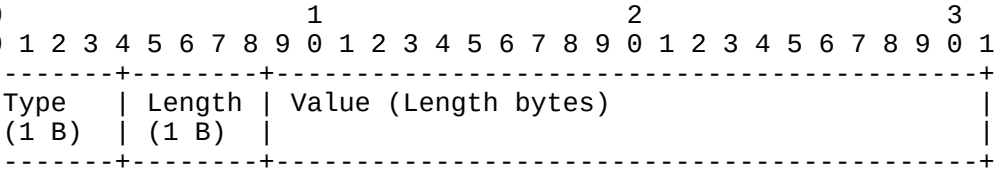
Table 1

TLV Type values 4 through 246 are reserved for future use and MUST NOT be assigned without IETF review.

All TLV types not listed here are reserved. Future extensions MUST use unique type values and follow the TLV format rules defined in this section. Implementations MUST ignore unknown TLV types unless explicitly required by policy.

6.3.2. TSQ Message Formats

TSQ uses a Type-Length-Value (TLV) format to encode fields within TSQ Request and Response messages. Each TLV is self-contained and designed for easy extensibility. The structure is as follows:



- Type: 8-bit unsigned integer identifying the TLV type.
- Length: 8-bit unsigned integer specifying the length of the Value field.
- Value: A variable-length field determined by Length.

All multi-byte fields within the Value are encoded in network byte order (big-endian).

TSQ messages consist of a sequence of TLVs (Type-Length-Value elements). This format is consistent across both stream-based and Datagram modes. In Datagram Mode, all TLVs MUST still appear in a well-defined order, beginning with the Nonce (Type 1). Since QUIC DATAGRAM frames do not provide ordering or framing guarantees, implementations MUST treat the entire datagram payload as a single TSQ message. Partial or fragmented TLVs MUST NOT be used.

All TLVs follow a consistent format: a one-byte Type, a one-byte Length, and a variable-length Value. TLVs MUST be processed sequentially, and unknown TLV types MUST be ignored unless policy dictates otherwise. The Signature Block TLV, if present, MUST appear last.

TSQ Request Message TLVs:

* *Nonce (Type 1):* 16 bytes. A cryptographically secure random value generated by the client. Used to associate responses and ensure freshness. Copied back verbatim in the response.

* *Precision Mode Request (Type 250):* 0 bytes. Included by the client to request Precision Mode behavior.

* *Signature Request (Type 252):* An OPTIONAL zero-length TLV included by the client to request that the server include a cryptographic authentication block in the TSQ Response. This block enables integrity and source verification using either symmetric (e.g., HMAC) or asymmetric (e.g., Ed25519) methods. If supported, the server MUST include a valid Signature Block TLV (Type 255) in the response. This mechanism allows clients to request authenticated responses on demand, without requiring pre-negotiation or a separate profile.

TSQ Response Message TLVs:

* *Nonce (Type 1):* 16 bytes. Echoed from the request to associate response with request.

* *Receive Timestamp (Type 2):* 8 bytes. The server’s local time when the request was received, in NTP format.

* *Send Timestamp (Type 3):* 8 bytes. The server’s local time when the response was sent, in NTP format.

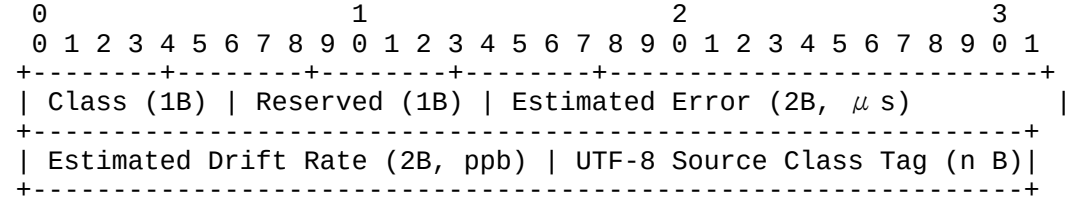
Optional TLVs are used to extend TSQ without altering the core format. These TLVs MUST follow standard TLV structure and be included after the required fields in the message. Each is defined as follows:

* *Metadata Query Request (Type 247):* An OPTIONAL zero-length TLV included by the client in a TSQ Request to indicate interest in structured metadata about the responding server. If present, the server MAY include a Metadata TLV (Type 253) in its response, populated with deployment-specific key-value annotations such as hostname, site ID, region, clock model, or software version.

  This TLV allows for dynamic querying of server identity or status and may be used for monitoring, diagnostics, or debugging. The Metadata TLV format remains unchanged: a sequence of UTF-8 key-value pairs, each with a single-byte length prefix. Keys are implementation-defined and MAY vary by deployment.

  The Metadata Query Request TLV contains no value bytes and MUST appear only in TSQ Requests. Its presence serves as a hint and does not guarantee server metadata will be returned. Servers MAY ignore it or suppress metadata responses based on policy.

* *Clock Quality Info (Type 248):* An OPTIONAL TLV included by the server in TSQ Responses to indicate estimated clock accuracy, drift rate, or classification. This TLV enables clients to assess the quality and stability of the server's time source.



  Field descriptions:

  Class (1 byte): Integer code describing the clock class:

  - 0: Unknown
  - 1: GNSS Disciplined
  - 2: PTP Grandmaster
  - 3: NTP-Synchronized
  - 4: Local OCXO
  - 5: Unstable / Holdover

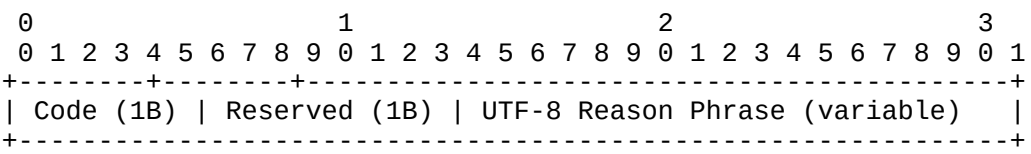
Estimated Error (2 bytes): Approximate ± error in microseconds (unsigned integer). Example: 500 = ±500 μs.

Estimated Drift Rate (2 bytes): Approximate frequency drift in parts per billion (ppb). Example: 50 = ±50 ppb.

Source Class Tag: An OPTIONAL UTF-8 string describing the clock type, e.g., "gpsdo", "ptp-bmc", "ntp-v4". Length is inferred from TLV length.

Clients MAY use this information to guide server weighting, filtering, or telemetry logging. This TLV is advisory and MAY be omitted. If present, it SHOULD appear after the mandatory timestamp TLVs and before the Signature Block.

* *Error (Type 249):* Used by the server in Stream Mode to report structured error conditions. This TLV MUST appear alone in the response, without timestamps or other TLVs. The value format is as follows:



Field definitions:

- *Code (1 byte):* A numeric error code (e.g., 0x01 = malformed request, 0x02 = unsupported TLV, 0x03 = authentication required).

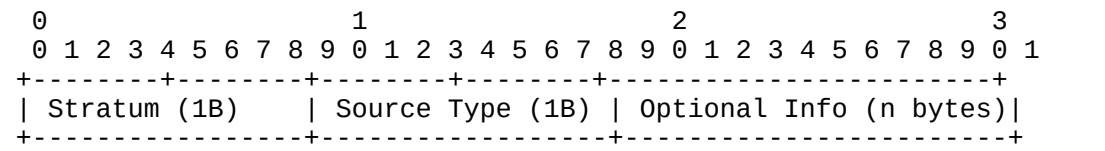
- *Reserved (1 byte):* MUST be set to zero. Reserved for future use.

- *Reason Phrase:* Optional UTF-8 string describing the error. Clients MAY log or display this for diagnostics. The reason phrase MUST NOT exceed 255 bytes.

This TLV MUST only appear in TSQ Responses sent over QUIC streams. Clients receiving an Error TLV MUST treat the request as failed and MUST NOT attempt to parse any additional TLVs in the response. If a Signature Block is present, the Error TLV MUST be included in the signed data.

* *Precision Mode Acknowledgment (Type 250):* 0 bytes. Echoed by the server to confirm support.

* *Time Source Info (Type 251):* Variable length. Optionally included by the server in the TSQ Response to indicate the current stratum and synchronization source. This allows clients to assess the distance from the primary time source (e.g., stratum-1) and apply local trust or weighting policies accordingly. The format is as follows:



- Stratum: 0 (unsynchronized), 1 (reference), 2+ (as in NTP)
- Source Type: 1 = GPS, 2 = PTP, 3 = NTP, 4 = Manual, 5 = Unknown
- Optional Info: UTF-8 string (e.g., "ntp.example.com", "us-west1")

  Stratum: Describes the server’s distance from the primary time source, consistent with NTP terminology.

  Source Type: Indicates the technology or source used to discipline the server’s clock.

  Optional Info: Provides human-readable metadata about the time source. This may include the hostname of the upstream time server, the data center region, or a clock quality tag. The field is UTF-8 encoded and its length is inferred from the enclosing TLV’s total length.

  The stratum field in TSQ does not enforce a hierarchical time distribution model, as used in NTP. Instead, it serves as advisory metadata to help clients assess the quality and proximity of the server’ s time source. Implementations MAY use the stratum value to apply weighting, filtering, or quorum selection policies when querying multiple servers. TSQ does not impose stratum-based path validation or strict level enforcement.

  Clients MAY use this TLV to assess time accuracy or for logging/ auditing. Servers MAY omit this field, and clients MUST tolerate its absence.

* *Metadata (Type 253):* Variable length. Encoded as a series of UTF-8 key-value pairs. Each key and value is prefixed with a single-byte length. Example: [0x04] "site" [0x07] "us-east". MAY be repeated for multiple annotations. Clients MUST ignore unknown keys. This TLV is intended to carry deployment or implementation-specific annotations, such as region identifiers, server version, environment tags, or clock quality metadata.

Example encoding:

[0x04] ’site’ [0x07] ’us-east’ --> 04 73 69 74 65 07 75 73 2d 65 61 73 74

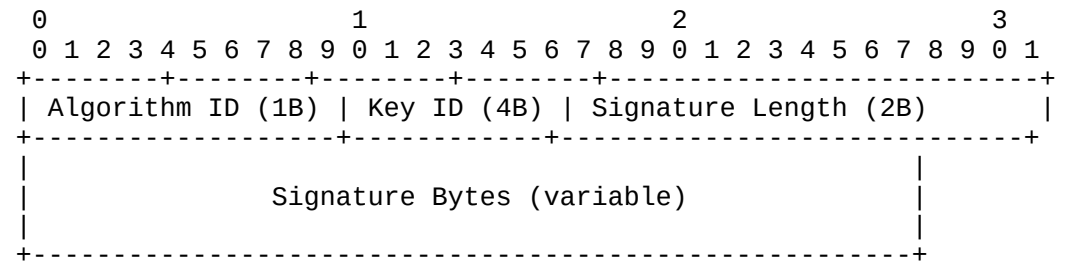
To ensure consistent signature validation, the Metadata TLV (Type 253) MUST be serialized in a canonical form before inclusion in a signed message. Specifically, key-value pairs MUST be encoded in lexicographic order by key, with exact UTF-8 byte sequences preserved. Implementations MUST NOT reorder, collapse, or normalize UTF-8 content during parsing or signing. This ensures that both sender and recipient derive the same byte-for-byte representation for verification.

* *Padding (Type 254):* Optional TLV used to align the total TSQ message size for symmetry between request and response. This reduces timing bias and improves accuracy, especially in Precision Mode. The value MUST consist of zero-value bytes (0x00).

  Padding TLVs MUST appear only at the end of a TSQ message. They MUST be included in the Signature TLV coverage if present. Type 0 is reserved and MUST NOT be used for padding.

* *Signature Block (Type 255):* MUST be the final TLV in the message. The block supports both symmetric (e.g., HMAC-SHA256) and asymmetric (e.g., Ed25519) methods. It allows clients to verify message integrity and authenticity. Asymmetric signatures are suitable for auditability and long-term verification, while symmetric authentication offers lower overhead for real-time validation. If included, the Signature Block MUST cover the Nonce (Type 1), Receive Timestamp (Type 2), Send Timestamp (Type 3), and all included optional TLVs, in canonical TLV order.

  When used in Datagram Mode, the Signature Block TLV (Type 255) is the only mechanism that provides end-to-end authentication and integrity protection. QUIC Datagram frames lack built-in reliability and integrity, so any unauthenticated response may be spoofed by an attacker. If a client includes a Signature Request TLV (Type 252), and the server supports authentication, then the response MUST include TLV 255. Clients operating in Datagram Mode without authentication MUST treat all responses as unauthenticated and advisory only.



Field descriptions:

- *Algorithm ID (1 byte):* Indicates the signature algorithm in use. See below.

- *Key ID (4 bytes):* An identifier for the signing key (e.g., for key rotation).

- *Signature Length (2 bytes):* Total number of signature bytes to follow.

- *Signature Bytes:* The raw signature output from the selected algorithm.

Signatures MAY use [RFC8032] (Ed25519) or [RFC2104] (HMAC-SHA256), depending on deployment policy.

All timestamps use the NTP format, a 64-bit unsigned fixed-point representation: the upper 32 bits represent seconds since the NTP epoch (January 1, 1900), and the lower 32 bits represent fractional seconds. This format ensures sub-microsecond resolution and compatibility with existing time protocols.

6.4. Precision Mode

Precision Mode is an OPTIONAL feature of TSQ designed to reduce timing variance introduced by serialization delay and queuing asymmetries. It ensures that TSQ Request and Response messages are of equal size to enable low-jitter, symmetric round-trip time measurement.

* *Negotiation:* Precision Mode is negotiated using TLV Type 250. The client includes a zero-length _Precision Mode Request_ TLV (Type 250) in the TSQ Request to indicate support for Precision Mode. If the server also supports Precision Mode, it MUST include a zero-length _Precision Mode Acknowledgment_ TLV (Type 250) in the TSQ Response. If the server does not support Precision Mode, it MUST omit this TLV. Clients MUST interpret the absence of the

acknowledgment TLV as lack of support and MAY either proceed without Precision Mode or fail the exchange, based on local policy.

* *Fixed-Length Packet Symmetry:* When Precision Mode is successfully negotiated, both endpoints MUST ensure that the total byte length of the TSQ Request and Response messages are identical. To do so, implementations MUST include a Padding TLV (Type 254) to pad the message as needed. Padding MUST consist entirely of zero-value bytes and MUST appear after all other TLVs except the Signature Block. If used with the Signature Block TLV, Padding MUST appear immediately before it.

* *Interoperability:* Clients that require Precision Mode for accurate clock synchronization SHOULD implement a fallback behavior if the server does not acknowledge support. The use of a dual-role TLV (Type 250) avoids the need for separate type registration and keeps the negotiation mechanism simple. This TLV MAY be extended in the future to include payload data for enhanced negotiation, though current implementations MUST treat the TLV as zero-length.

6.5. QUIC Datagram Mode

In addition to stream-based transport, TSQ optionally supports transmission over QUIC Datagrams [RFC9221] for low-latency, unreliable synchronization exchanges. This is referred to as Datagram Mode.

* *Operational Flow:* In Datagram Mode, the TSQ Request and Response are each encoded as a single TLV-encoded message and transmitted in a single QUIC DATAGRAM frame. This eliminates the need for a stream-based handshake, reducing round-trip latency. There is no flow control or retransmission of TSQ messages when sent this way.

* *Message Format:* The message format is identical to stream-based TSQ: a sequence of TLVs beginning with a Nonce (Type 1) and including Server Time (Type 2), Receive/Send Timestamps, and optionally Padding, Metadata, and Signature TLVs. The same message structure applies; only the transport differs.

* *Negotiation:* Datagram Mode is implicitly negotiated via ALPN and application profile. If both client and server support it, the client MAY send the first TSQ Request as a DATAGRAM. If the server replies using a DATAGRAM, the mode is established. If the server does not support DATAGRAM frames, it will not acknowledge or will fall back to a stream-based response. Clients MUST be prepared to retry using stream-based transport.

* *Trade-Offs:* Datagram Mode offers reduced latency and avoids head-of-line blocking, but provides no delivery guarantees. This makes it suitable for approximate or opportunistic synchronization, but not for authenticated, auditable timestamps. Applications that require signature validation, retransmission, or ordering SHOULD use stream-based mode.

6.6. Error Handling

TSQ implementations MUST validate all TLV structures. In Datagram Mode, malformed messages SHOULD result in silent discard to avoid amplification or response spoofing. In Stream Mode, servers MAY respond with a structured error using the Error TLV (Type 249) described below. This allows clients to distinguish protocol errors from transport issues. Clients MUST validate that the echoed nonce in the TSQ Response matches the nonce originally sent in the request. If the nonce is missing, incorrectly formatted, or does not match, clients MUST discard the response, log a diagnostic warning, and treat the exchange as invalid. Clients MAY retry with a different server or after a delay, subject to local rate-limiting policy.

Timestamp anomalies include any condition where the computed round-trip time (RTT) or clock offset is implausible or mathematically invalid. This includes, but is not limited to:

* RTT calculation results in a negative or zero duration.

* Offset calculation exceeds deployment-defined thresholds (e.g., 24 hours).

* Timestamps appear non-monotonic (e.g., T3 earlier than T2).

* Receive time (T4) is earlier than transmit time (T1).

When encountering such anomalies, clients SHOULD discard the result, log a warning or error for diagnostic purposes, and MAY retry the exchange with a different server or after a short delay. Implementations MAY apply sanity checks or rate limits to prevent excessive retries in unstable environments.

Deployments requiring strict synchronization SHOULD implement anomaly detection thresholds that align with operational tolerances (e.g., tolerable drift or jitter; see [RFC8633]). Clients MUST NOT apply any synchronization adjustment based on a TSQ Response be applied unless all four timestamps (T1T4) are consistent and validated.

7. Security Considerations

TSQ leverages QUIC’s security model to ensure confidentiality, integrity, and replay protection. All TSQ messages are encapsulated within encrypted QUIC streams or datagrams, depending on mode.

TSQ relies on the Application-Layer Protocol Negotiation (ALPN) extension in QUIC to distinguish TSQ traffic from other application protocols. Implementations MUST verify that the negotiated ALPN value is exactly "tsq". Clients MUST reject any QUIC connection where the negotiated ALPN does not exactly match "tsq". Failure to enforce ALPN validation could result in downgrade attacks where the connection is interpreted as TSQ despite not meeting protocol security requirements.

The QUIC handshake provides mutual authentication if certificates are validated. In TSQ, this ensures both the client and server are communicating with authenticated endpoints.

Replay protection is achieved via QUIC’s transport guarantees. However, implementations MUST validate echoed nonces and timestamp monotonicity to guard against application-level replay attacks.

QUIC supports session resumption, which allows a client to reconnect to a server using a previously issued resumption ticket. TSQ clients MAY use resumption to reduce handshake latency or cryptographic overhead for repeated exchanges. Resumption can be implemented in a stateless manner, where the server stores no session data and relies on opaque tokens. TSQ does not retain application-layer state between exchanges, and its nonce-based request validation ensures replay protection even when using 0-RTT resumption. Implementations SHOULD ensure that resumption tokens do not leak identifying information unless explicitly required by deployment policy.

TSQ supports a negotiation mechanism for clients that require signed responses. This is achieved using the Signature Request TLV (Type 252), which signals that the client requires a digitally signed TSQ Response. Servers that receive this TLV MUST include a valid Signature Block TLV (Type 255) in the response or fail the request. If no Signature Block is present in the response, the client MUST treat the exchange as unauthenticated and discard it. This mechanism ensures deterministic policy enforcement for deployments requiring auditability.

TSQ supports optional cryptographic authentication using either symmetric (e.g., HMAC-SHA256) or asymmetric (e.g., Ed25519) methods. Asymmetric signatures enable auditability and long-term validation, while symmetric authentication offers lower overhead for real-time integrity verification.

When symmetric authentication is used (e.g., HMAC-SHA256), clients and servers must be provisioned with a shared secret key. TSQ does not define a built-in key exchange or authentication infrastructure. Shared keys MUST be provisioned out-of-band, such as through configuration management, provisioning tools, or enrollment protocols.

In contrast, when asymmetric authentication is used (e.g., Ed25519), the server signs the response using a private key, and clients verify using a known public key. This method supports stateless validation and is suitable for broadcast, audit logging, or public deployments.

For security implications specific to Datagram Mode, see Section 7.1.

The Time Source Info TLV (Type 251) provides optional metadata about the server's synchronization source and quality. While useful for transparency and diagnostics, clients MUST treat this TLV as advisory. It MUST NOT be used as a substitute for cryptographic integrity or freshness validation. Implementers SHOULD validate time quality using signatures, replay protection, and out-of-band policy when accuracy is critical.

Privacy-conscious deployments SHOULD minimize identifying metadata in TSQ messages and MAY use ephemeral QUIC connections to reduce correlation risk. Servers MAY also implement rate-limiting to mitigate client fingerprinting or denial-of-service vectors.

7.1. Security Implications of Datagram Mode

QUIC Datagram mode provides reduced latency and eliminates head-of-line blocking, but introduces several important security trade-offs compared to stream-based TSQ exchanges.

* *Unreliability:* QUIC Datagram frames are not retransmitted or acknowledged. Messages may be lost, reordered, or duplicated.

* *Spoofing Risk:* Since datagrams are stateless and unidirectional, responses can be spoofed by an on-path attacker unless validated using digital signatures.

* *No Built-In Integrity:* QUIC Datagram mode lacks framing guarantees. Without the Signature Block TLV, clients have no cryptographic assurance of authenticity or freshness.

As a result, clients that operate in Datagram Mode SHOULD treat responses as advisory unless they include a valid Signature Block. Implementations that require auditability, integrity guarantees, or traceability SHOULD prefer stream-based TSQ exchanges.

Where Datagram Mode is used, deployments MAY enforce signature usage via the Signature Request TLV and validate all incoming messages against known server credentials.

8. Signature Format and Verification

TSQ supports optional message-level signatures for deployments that require auditability or long-term verifiability beyond QUIC's ephemeral session security. This mechanism is distinct from QUIC's built-in authentication and confidentiality, and SHOULD only be enabled when such properties are required.

When enabled, the TSQ Response MUST include a TLV extension of type 255 (0xFF), known as the Signature Block.

* Signature Algorithm ID (1 byte)

* Key ID (4 bytes)

* Signature Length (2 bytes)

* Signature Bytes

Supported signature algorithms:

* 0x01: Ed25519 using the standard Curve25519 curve

* 0x02: HMAC-SHA256 (requires shared secret)

The data to be signed MUST consist of the exact, contiguous byte sequence of all TLVs in the message *preceding* the Signature Block TLV (Type 255), in the order they appear on the wire. The Signature Block itself MUST NOT be included in the signed data.

This typically includes the following TLVs:

* The echoed Nonce (Type 1)

* Receive Timestamp (Type 2)

* Send Timestamp (Type 3)

* Optional TLVs (e.g., Metadata, Time Source Info, Padding), if present

Keys are provisioned out-of-band and identified by the Key ID. TSQ does not define a key distribution mechanism; implementers MAY use mechanisms such as DNSSEC, enrollment protocols, or manual configuration.

Use of signatures SHOULD be negotiated via a policy or deployment profile. Clients that require signatures but do not receive them MUST treat the response as unauthenticated.

Signatures mitigate threats such as long-term forgery, post-session repudiation, and auditing in compliance-sensitive deployments. They do not replace or extend QUIC's real-time transport security and SHOULD NOT be used as a substitute for encrypted transport.

The presence of the Signature Request TLV (Type 252) in the TSQ Request mandates the inclusion of a valid Signature Block TLV in the response. Failure to comply MUST result in rejection of the response by the client. This provides a lightweight, in-band method to enforce signature requirements on a per-request basis.

9. Scalability and Deployment Considerations

TSQ is designed to operate efficiently in large-scale, heterogeneous environments. It is suitable for use in enterprise, cloud-native, and mobile scenarios where traditional NTP may encounter limitations due to UDP/123 filtering, firewall traversal, or security posture.

TSQ's use of QUIC over UDP/443 improves deployability in networks that restrict traditional NTP. UDP port 123 is often blocked in enterprise, campus, or guest network environments due to DDoS concerns or policy constraints. In contrast, UDP/443 is widely permitted to support QUIC-based web services, increasing the likelihood of successful time synchronization in such environments. However, QUIC may also be restricted in some legacy networks, and implementers SHOULD perform reachability testing or provide fallback mechanisms as needed.

QUIC's stream multiplexing and NAT rebinding support allow TSQ to serve mobile clients that frequently change IP addresses or network paths. Use of QUIC on UDP/443 also enables easier traversal through firewalls and middleboxes.

Each QUIC Datagram message incurs transport-layer overhead due to the UDP header, QUIC packet header, encryption framing, and QUIC Datagram frame. The total overhead typically ranges from 45 to 55 bytes per message, depending on QUIC implementation and configuration. This overhead is comparable to DTLS and less than full VPN or IPsec encapsulation. Overhead is generally symmetric in both directions but may vary slightly based on connection parameters or congestion control state.

TSQ connections are intended to be short-lived by default, allowing high churn rates with minimal state retention. However, QUIC session resumption MAY be used to optimize recurring exchanges with known servers.

TSQ clients MAY be configured with multiple server endpoints and perform parallel or sequential queries, similar to NTP clients. Implementations MAY apply filtering, clustering, or consensus algorithms across multiple responses to improve synchronization accuracy or resiliency. The use of multiple sources is especially relevant in mobile or cloud deployments where path asymmetry or stratum diversity can vary over time.

Implementations MAY operate in a stateless mode by avoiding long-lived session tracking and relying on QUIC's address validation tokens to prevent abuse. Stateless designs are especially beneficial for high-volume deployments such as cloud-based time services.

TSQ is not intended to replace the global NTP stratum hierarchy or public time pool infrastructure directly. However, it MAY be used in conjunction with such infrastructure, particularly where UDP-based protocols are not viable or additional metadata, security, or auditability is required.

TSQ MAY be used as a time source for intermediary servers that speak NTP or NTS. In such cases, the intermediary server (e.g., a stratum-2 NTP node) may consume TSQ and emit NTP-formatted responses to downstream clients. TSQ provides stratum information and precise timestamps, but does not currently define structured equivalents for NTP-specific fields such as root delay, root dispersion, or reference ID. Implementers MAY synthesize these values based on local policy, or encode additional metadata using the Metadata TLV (Type 253). Future revisions of TSQ MAY introduce dedicated TLVs for conveying these values to improve interoperability with traditional NTP infrastructure.

Servers MAY include a Time Source Info TLV in responses to expose stratum or reference source. This is particularly useful in hierarchical deployments or enterprise deployments with multiple upstream sources. However, TSQ does not impose or rely on a strict time hierarchy, and clients MUST treat this information as advisory.

TSQ servers MAY coexist with NTP or NTS implementations on the same system. Since TSQ uses QUIC over UDP/443 with a distinct ALPN identifier, it operates independently from NTP (UDP/123) or NTS (typically TCP/4460). A single time service MAY offer both protocols in parallel, allowing clients to select based on policy, transport compatibility, or required features such as authentication or precision.

Implementers of dual-stack services SHOULD ensure that time, stratum, and clock quality metadata are consistent across all served protocols. While TSQ does not currently define mappings to NTP header fields such as root delay or dispersion, such translations MAY be considered in future revisions or hybrid deployments.

TSQ supports an optional stateless design by allowing servers to construct responses solely based on information included in the client request, without maintaining per-client session state. Specifically, the client’s request includes a cryptographically secure nonce, and all timestamps necessary for round-trip time (RTT) calculation are echoed or generated in the response. If signature-based integrity protection is used, the server can sign the response using a long-lived private key without needing to store session-specific metadata. This enables large-scale deployments where the server handles a high volume of clients without incurring memory or tracking overhead per connection. However, implementations MUST ensure that nonce uniqueness and freshness are sufficient to prevent replay attacks and ambiguity in response validation. In particular, clients MUST generate cryptographically strong nonces using a secure random number generator. Nonces MUST be unpredictable and collision-resistant, as servers may operate in a stateless mode and rely solely on the nonce for response association. The default 16-byte nonce length is RECOMMENDED to ensure sufficient entropy for all deployment sizes. Reuse or predictability of nonces can lead to forged or replayed responses, especially in datagram mode.

10. Experimental Criteria

This document is published as an Experimental RFC to evaluate the viability and security of time synchronization over the QUIC transport protocol. The goals of this experiment are to:

* Demonstrate successful time synchronization in varied environments (e.g., cloud, mobile, IoT) where traditional NTP/NTS protocols face limitations.

* Validate interoperability between independent TSQ implementations across QUIC versions and TLS stacks.

* Evaluate the robustness of TSQ's security model, including replay prevention, signature validation, and support for stateless operation.

* Assess the practicality of TSQ's Precision Mode and Datagram Mode in real-world deployments.

* Implementers are encouraged to produce and test against at least one reference implementation that supports both Stream and Datagram modes, and to validate fallback behaviors (e.g., unsupported Precision Mode or missing Signature Block) to ensure graceful interoperability across partial implementations.

Feedback from implementers and operators will help determine whether TSQ is suitable for progression to the Standards Track, and whether refinements to the protocol, message formats, or security properties are necessary.

11. IANA Considerations

This document requests IANA to register the following value in the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry:

TSQ uses the ALPN identifier "tsq" to negotiate protocol support during the TLS handshake [RFC7301].

Protocol	Identification Sequence	Reference
tsq	0x03 74 73 71 ("tsq")	This document

Table 2

Per [RFC7301], this identifier is used during the TLS handshake to indicate TSQ support.

12. Acknowledgments

Special thanks to Joe Clarke for insightful comments and review during early iterations of this document.

Thanks also to participants in the NTP and QUIC IETF Working Groups whose discussions and documents shaped many of the ideas in this work.

13. References

13.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", February 1997, <https://www.rfc-editor.org/rfc/rfc2104.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997, <https://www.rfc-editor.org/rfc/rfc2119.html>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stebila, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", July 2014, <https://www.rfc-editor.org/rfc/rfc7301.html>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", 2017, <https://www.rfc-editor.org/rfc/rfc8032.html>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", May 2017, <https://www.rfc-editor.org/rfc/rfc8174.html>.

[RFC9000] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", May 2021, <https://www.rfc-editor.org/rfc/rfc9000.html>.

[RFC9221] Schinazi, D., "Using QUIC Datagrams with HTTP/3", June 2022, <https://www.rfc-editor.org/rfc/rfc9221.html>.

13.2. Informative References

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", June 2010, <https://www.rfc-editor.org/rfc/rfc5905.html>.

[RFC7822] Haberman, B. and D. Mills, "Network Time Protocol Version 4 (NTPv4) Extension Fields", RFC 7822, April 2016, <https://www.rfc-editor.org/rfc/rfc7822.txt>.

[RFC8633] Teichel, K., Malhotra, A., and D. Franke, "Network Time Protocol Best Current Practices", July 2019, <https://www.rfc-editor.org/rfc/rfc8633.html>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Malone, M., and A. Sommers, "Network Time Security for the Network Time Protocol", September 2020, <https://www.rfc-editor.org/rfc/rfc8915.html>.

[I-D.ietf-ntp-roughtime] Langley, A., Dowling, B., and G. Malone, "Roughtime", May 2024, <https://www.ietf.org/archive/id/draft-ietf-ntp-roughtime-14.html>.

Appendix A. Comparison to Existing Protocols

This appendix provides high-level comparison and deployment context. It is non-normative.

The following table compares TSQ with related time synchronization protocols across key properties. TSQ is split into its Stream and Datagram modes to highlight differences in reliability and authentication behavior. Roughtime is included as a cryptographically verifiable but coarse-grained alternative.

Feature	TSQ (Stream)	TSQ (Datagram)	Roughtime	NTP	NTS
Transport Protocol	QUIC Streams	QUIC Datagrams	UDP	UDP	TCP (NTS-KE) + UDP
Transport Reliability	Yes	No	No	Yes	Yes
Authentication	Optional (HMAC/Signature)	Optional (HMAC/Signature)	Always (Signature)	No	Yes (AEAD)
Message Signing Supported	Yes	Yes	Yes	No	Yes (via AEAD tag)
Replay	Yes	Nonce-based	Yes	Weak	Yes

Protection						
Encryption	Yes (QUIC)	Optional (QUIC)	No	No	Yes (TLS/ AEAD)	
Structured Extension Mechanism	TLVs	TLVs	Flat key-value	Yes (Extension Field)	Yes [RFC7822]	
Negotiated Precision Mode	Optional	Optional	No	N/A	N/A	
Metadata / Annotations	Yes (UTF-8)	Yes (UTF-8)	Limited	No	No	
Connection Reuse	Yes	Yes	No	Yes	Yes	
NAT/Path Rebinding Support	Yes	Yes	No	Partial	Partial	
Short Polling Interval Support	Yes	Yes	No	Yes	Yes	
CPU to Establish Connection	Medium (QUIC+TLS)	Medium (QUIC+TLS)	High (Signature Init)	Low	High (TLS)	
CPU per Exchange	LowMedium	Low	Low	Low	LowMedium	
Server Memory per Client	Low (Stateless)	Low (Stateless)	Zero	Low	Medium (Session state)	
Scalable for Large Deployments	Yes	Yes	Yes	Yes	Mixed	
Typical Time Accuracy	High	Medium	LowMedium	Medium	Medium	
Message Overhead (approx)	~75 bytes	~55 bytes	~70 bytes	~48 bytes	~90 bytes	

Table 3

These values are approximate and represent typical protocol behavior. Message overhead includes transport framing, headers, and cryptographic context where applicable. Time accuracy depends on network conditions and server stratum.

Protocols like TSQ, Roughtime, and NTS leverage secure transports (QUIC or TLS), which provide confidentiality by encrypting message contents. While confidentiality is not strictly necessary for time synchronization, it is a side effect of modern transport security. Authentication and integrity protection are more critical for correctness and trustworthiness of time data, and are represented separately in the table.

Roughtime provided a compelling model for cryptographic time synchronization using UDP, but several lessons influenced TSQ's design. These include the need for flexible authentication mechanisms (not always public-key), extensibility via structured TLVs, and transport-layer security. TSQ builds on Roughtime's strengths while addressing limitations in audit scalability, extensibility, and deployment friendliness.

TSQ benefits from QUIC's built-in mobility features such as NAT rebinding and connection migration, which enable seamless operation across network changes or mobile IP transitions. NTP and NTS do not offer transport-layer mobility support but can re-establish sessions after address changes. Roughtime is stateless and does not support connection recovery.

TSQ uses a structured TLV (Type-Length-Value) format to allow future extension without altering the core message structure. This enables features such as metadata queries, error signaling, or clock quality reporting to be added incrementally. NTP also includes an Extension Field mechanism as defined in [RFC7822], which is used by NTS, but its extensibility is limited by field size, version constraints, and legacy interoperability. Roughtime also supports extension, though its model is simpler and less flexible than TSQ's TLV-based design.

CPU and memory estimates are qualitative and vary by implementation, cryptographic library, and connection policy. They are provided here to help differentiate protocol behavior under typical deployment assumptions.

TSQ and Roughtime serve complementary purposes. While Roughtime provides cryptographic attestation of coarse-grained time and bootstraps trust in untrusted environments, TSQ is designed for ongoing synchronization with higher precision, session continuity, and transport-layer security. TSQ may follow Roughtime during startup or cold boot scenarios.

Author’s Address

Garrett McCollum
Cisco Systems
Email: gmccollu@cisco.com