Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 16 September 2026

Maxine

15 March 2026

Open Internet Time Protocol (OITP)
draft-nightglow-oitp-00

Abstract

This document specifies the Open Internet Time Protocol (OITP), a network time synchronization protocol for decimal time systems that divide the civil day (86400 SI seconds) into 1000 primary units called beats, each exactly 86.4 SI seconds. OITP enables clients to synchronize decimal time clocks over packet-switched networks with sub-beat precision, using techniques derived from existing network time protocols adapted for the decimal time domain.

The reference timescale uses UTC+1 as its reference meridian, with midnight at UTC+1 corresponding to beat zero.

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 16 September 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Background

Decimal time systems that divide the solar day into 1000 equal parts have a long history, dating back to French Revolutionary Time (1793). In 1998, Swatch introduced "Swatch Internet Time" (.beat), which re-popularized this concept for the internet era, proposing a universal time notation without time zones, where all locations share the same numeric time value referenced to UTC+1 (BMT, Biel Mean Time).

Despite decades of software implementations in numerous programming languages and operating systems, no network protocol has been specified for synchronizing decimal time clocks between hosts. Existing implementations derive their time locally from system clocks, inheriting whatever accuracy and precision the underlying operating system provides, but with no mechanism for direct decimal-time synchronization.

1.2. Motivation

OITP provides a purpose-built synchronization mechanism for decimal time, offering:

* Native decimal time wire format (clients never need to parse or convert UTC timestamps)
* Compact wire format optimized for the decimal time domain
* Synchronization precision down to the nanobeat level (~86.4 ns)
* Compatibility with existing decimal time displays and applications
* A minimal, self-contained protocol implementable in around 1500 lines of code in most languages

A server running OITP performs the UTC-to-decimal conversion once, distributing the result directly. Clients receive native decimal time with no conversion step, which reduces code size for constrained devices. The protocol also defines a shared synchronization layer for communities where decimal time is the primary timekeeping system -- something a local NTP-to-decimal conversion on each client cannot provide.

1.3. Relationship to NTP

An alternative approach would be to distribute decimal time as an NTP extension field, piggybacking on NTP's existing infrastructure and security mechanisms (NTS [RFC8915]). OITP is a separate protocol for several reasons:

* NTP extension fields are processed by the client after UTC time is already received; the decimal conversion still occurs on every client. OITP moves the conversion to the server.
* Decimal time requires a different notion of "accuracy": a 1-millibeat error (~86ms) is often acceptable where NTP targets microseconds. A dedicated protocol can have a simpler algorithm tuned to this precision target.
* OITP's wire format is native decimal: timestamps, poll intervals, and delay fields are all expressed in beats and millibeats. An NTP extension field would carry decimal time embedded inside a UTC-based protocol, creating a conceptual mismatch and complicating implementations.

* A self-contained protocol is easier to implement on constrained devices that may not need a full NTP stack.

OITP does not replace NTP. Deployments are expected to use NTP to synchronize the UTC reference clocks that OITP servers derive their time from; this requirement is normatively stated in Section 9.1.

1.4. Scope

OITP is designed for decimal time synchronization over IP networks. It is not intended to replace NTP [RFC5905] or PTP [IEEE1588] for applications requiring UTC or TAI synchronization. OITP servers derive their reference time from existing UTC sources (typically NTP-synchronized system clocks or GPS/PPS receivers) and serve it in the decimal time domain.

1.5. Document Organization

Section 2 defines terminology. Section 3 describes the decimal time system and notation. Section 4 specifies the 64-bit timestamp format. Section 5 provides protocol overview and transport. Section 6 defines the wire packet format. Sections 7 and 8 specify the synchronization algorithm and clock discipline. Sections 9 and 10 define server and client behavior. Section 11 describes the stratum hierarchy. Section 12 addresses security considerations. Section 13 covers IANA considerations.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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 capitalized form, as shown here.

The following terms are used throughout this specification:

beat: The primary unit of decimal time, equal to exactly 86.4 SI seconds, dividing the civil day (86400 SI seconds) into 1000 equal parts.

millibeat: 1/1000 of a beat, equal to 86.4 milliseconds.

microbeat: 1/1,000,000 of a beat, equal to 86.4 microseconds.

nanobeat: 1/1,000,000,000 of a beat, equal to 86.4 nanoseconds.

day epoch: The start of beat 0 for a given day, defined as midnight at the reference meridian (UTC+1), which corresponds to 23:00:00 UTC of the preceding calendar day.

system epoch: The reference point for absolute day numbering in OITP, defined as 1998-10-23T00:00:00+01:00 (midnight UTC+1 on 23 October 1998), day number 0. This date corresponds to the public launch of Swatch Internet Time, ensuring that OITP day numbers align with existing .beat implementations.

linearization: The process of converting an OITP 64-bit timestamp to a uniform integer representation for arithmetic operations. Because the beat field uses only values 0-999 (not the full 10-bit range 0-1023), raw timestamp subtraction is incorrect. See Section 7.1.

Kiss-o'-Death (KoD): A server response with stratum 3 and a non-zero Reference ID containing a diagnostic code (Section 9.4), signaling that the server is refusing or restricting service.

precision: The inherent accuracy of a server's time source, expressed as a signed integer representing floor(log2(E)) where E is the maximum error in beats.

3. Decimal Time System

3.1. Time Division

The civil day (86400 SI seconds) is divided into exactly 1000 beats, each exactly 86.4 SI seconds. Each beat is subdivided according to SI-style decimal prefixes:

Unit	Beats	SI Seconds	Notation
1 beat	1	86.4	@XXX
1 millibeat	0.001	0.0864	@XXX.XXX
1 microbeat	0.000001	0.0000864	-
1 nanobeat	0.000000001	0.0000000864	-

Table 1

3.2. Reference Meridian

The decimal day begins (beat 0) at midnight at the UTC+1 meridian. This corresponds to 23:00:00 UTC of the preceding civil day.

The decimal time value is globally uniform: at any given instant, the beat value is the same for all observers regardless of geographic location. There are no time zones or daylight saving adjustments in decimal time.

3.3. Conversion from UTC

Given a UTC time expressed as hours (h), minutes (m), and seconds (s) with fractional part, the current beat value is:

beat = ((h + 1) * 3600 + m * 60 + s) / 86.4

This formula is illustrative. The divisor 86.4 is not exactly representable in IEEE 754 binary floating-point, which may introduce rounding errors. Implementations SHOULD use the integer-safe equivalent:

millibeats = ((h + 1) * 3600 + m * 60 + s) * 1000 / 86400

or derive decimal time from continuous integer timescales (e.g., Unix nanoseconds) to avoid floating-point precision issues entirely.

If the result is >= 1000, subtract 1000 (the UTC+1 offset causes a day boundary crossing at 23:00 UTC).

Note: This formula assumes exactly 86400 SI seconds per day and does not account for UTC leap seconds. During a positive leap second (23:59:60 UTC), the formula produces values beyond 1000 before wraparound, resulting in a brief discontinuity. Implementations SHOULD derive decimal time from continuous timescales (e.g., CLOCK_REALTIME on POSIX systems where the kernel handles leap seconds via smearing or stepping) rather than applying this formula directly to sexagesimal time components.

During UTC leap second events, the conversion formula may produce beat values outside 0-999. Servers SHOULD use a continuous timescale that smears leap seconds (distributing the adjustment over a window), preserving decimal time continuity. Servers SHOULD NOT freeze or clamp beat values at 999, as this creates a period of stopped time that may trigger false drift detection in clients. The Leap Indicator flag (Section 6.2) signals pending leap seconds; clients SHOULD increase their tolerance for server disagreement when this flag is set.

3.4. Display Notation

The RECOMMENDED display notation for decimal time uses the "@" prefix followed by three digits for the integer beat, optionally followed by a decimal point and fractional digits:

```
@000        midnight (UTC+1)
@500        midday (UTC+1)
@999        just before midnight
@248.573    beat 248, millibeat 573
```

Implementations SHOULD display at least three digits for the integer beat (zero-padded). The number of fractional digits is implementation-defined based on available precision.

3.5. Combined Date-Time Notation

OITP defines two combined date-time notations. Both place the "@" character immediately after the date component with no intervening space, serving as the date-time separator.

3.5.1. Calendar Form

The calendar form uses the calendar date at UTC+1 with dot separators. Its ABNF [RFC5234] grammar is:

```
calendar-form = year "." month "." day "@" beat "." millibeat
year          = 4DIGIT
month         = 2DIGIT          ; 01-12
day           = 2DIGIT          ; 01-31
beat          = 3DIGIT          ; 000-999
millibeat     = 3DIGIT          ; 000-999
DIGIT         = %x30-39         ; 0-9
```

Examples:

```
2026.03.09@438.760  9 March 2026, beat 438, millibeat 760
1998.10.23@000.000  system epoch
```

The date component uses dots (.) as separators rather than hyphens to visually distinguish OITP date-time notation from ISO 8601. The year MUST be four digits; the month and day MUST be zero-padded to two digits. The date portion MUST be the calendar date at the reference meridian (UTC+1).

The calendar form is RECOMMENDED for human-facing displays, shared timestamps, log entries, and data interchange.

3.5.2. Day Form

The day form uses the OITP day number (Section 4.1) instead of a calendar date. Its ABNF [RFC5234] grammar is:

```
day-form  = day-number "@" beat "." millibeat
day-number = "0" / (NZDIGIT *DIGIT)  ; no leading zeros except "0"
NZDIGIT   = %x31-39                  ; 1-9
```

where day-number is a decimal integer with no leading zeros (except for day 0 itself). The 24-bit protocol field supports values up to 16,777,215.

Examples:

```
0@000.000         system epoch (1998-10-23)
9999@438.760      day 9999, beat 438, millibeat 760
10000@500.000     day 10000, beat 500
16777215@999.999  maximum representable day
```

The day form is RECOMMENDED for protocol-level representations, compact logging, and contexts where the OITP day number is more meaningful than a calendar date. It is also the most compact unambiguous representation of an OITP instant.

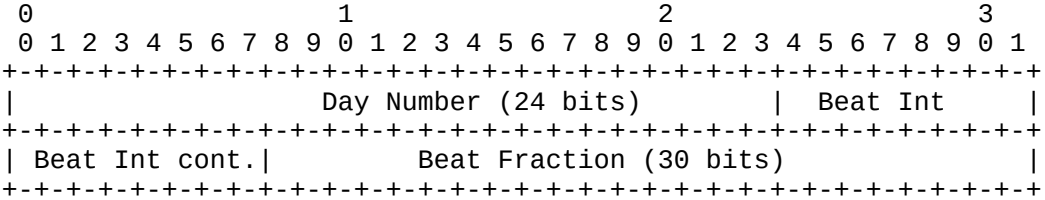
3.5.3. Usage

When displaying only the time component (without date), the "@" prefix notation (Section 3.4) SHOULD be used. When a full date-time representation is needed, implementations SHOULD support at least the calendar form. The day form MAY be offered as an alternative.

4. Timestamp Representation

4.1. OITP Timestamp Format

OITP uses a 64-bit unsigned integer timestamp with the following structure:



Day Number (24 bits): Unsigned integer counting days elapsed since the system epoch (1998-10-23). Range 0 to 16,777,215, covering approximately 45,929 years.

Beat Integer (10 bits): The integer part of the current beat, range 0 to 999. Beat integer values outside the range 0-999 are invalid; receivers MUST discard packets containing out-of-range beat integers. The remaining values 1000-1023 representable in 10 bits are reserved and MUST NOT be transmitted.

Beat Fraction (30 bits): The fractional part of the current beat as a 30-bit unsigned fixed-point number. The value represents the fraction N/2^30 of one beat. This provides a resolution of approximately 0.00000000093 beats, or roughly 0.93 nanobeats, exceeding the nanobeat precision target.

4.2. Design Rationale

The 10-bit beat integer field uses only 1000 of 1024 possible values, making the raw 64-bit timestamp non-linear. An alternative design could use a single 40-bit fractional day field (linear), but the current format allows direct inspection of the day number and beat value in a hex dump or packet capture without computation. This human-readable wire format follows the same design philosophy as NTP’s split seconds/fraction fields. The linearization cost (Section 7.1) is a single multiply-add per timestamp, which is negligible.

4.3. Resolution and Range

The 30-bit fractional field provides:

* Resolution: 1/1,073,741,824 of a beat = ~0.93 nanobeats = ~80.5 ns
* This is comparable to NTP’s 32-bit fractional second field, which provides ~233 picosecond resolution

The 24-bit day field provides:

* Range: 16,777,216 days = ~45,929 years from epoch
* The epoch (1998-10-23) places the end of range at approximately year 47960, which is considered sufficient

4.4. Special Values

A timestamp of all zeros (day 0, beat 0, fraction 0) represents the system epoch itself (1998-10-23T00:00:00+01:00). A timestamp of all ones (0xFFFFFFFFFFFFFFFF) is reserved and MUST NOT be used in protocol messages; it indicates an invalid or uninitialized timestamp.

In protocol fields where zero indicates "not set" or "not applicable" (e.g., the Origin and Receive Timestamp fields of a client request), this usage takes precedence: the value zero in those fields means the field is absent, not that the timestamp equals the epoch. Receivers MUST NOT interpret a zero in a "not set" field as a valid epoch timestamp.

5. Protocol Overview

5.1. Transport

OITP operates over UDP [RFC768] for time synchronization. The RECOMMENDED port number is 8640 (see Section 13 for IANA registration).

Implementations MAY additionally expose an HTTP interface for lightweight, human-readable time queries (see Appendix D). The HTTP interface is not part of the synchronization protocol and does not provide offset or delay calculation.

5.2. Operating Modes

OITP defines two operating modes:

Basic mode: A client sends a request and receives a response containing the server’s current decimal time. This mode provides time transfer without round-trip delay compensation. Suitable for applications where beat-level accuracy is sufficient.

Full mode: The default mode. A client and server exchange timestamps allowing computation of clock offset and network delay, similar to NTP’s client-server mode. This mode provides sub-millibeat precision on typical networks.

5.3. Protocol Exchange

In full mode, a single request-response exchange proceeds as follows:

1. At time T1, the client records its local decimal timestamp and sends a request packet containing T1 in the Transmit Timestamp field.

2. At time T2, the server receives the request and records T2.

3. At time T3, the server sends a response packet containing T1 (copied from the request), T2, and T3.

4. At time T4, the client receives the response and records T4.

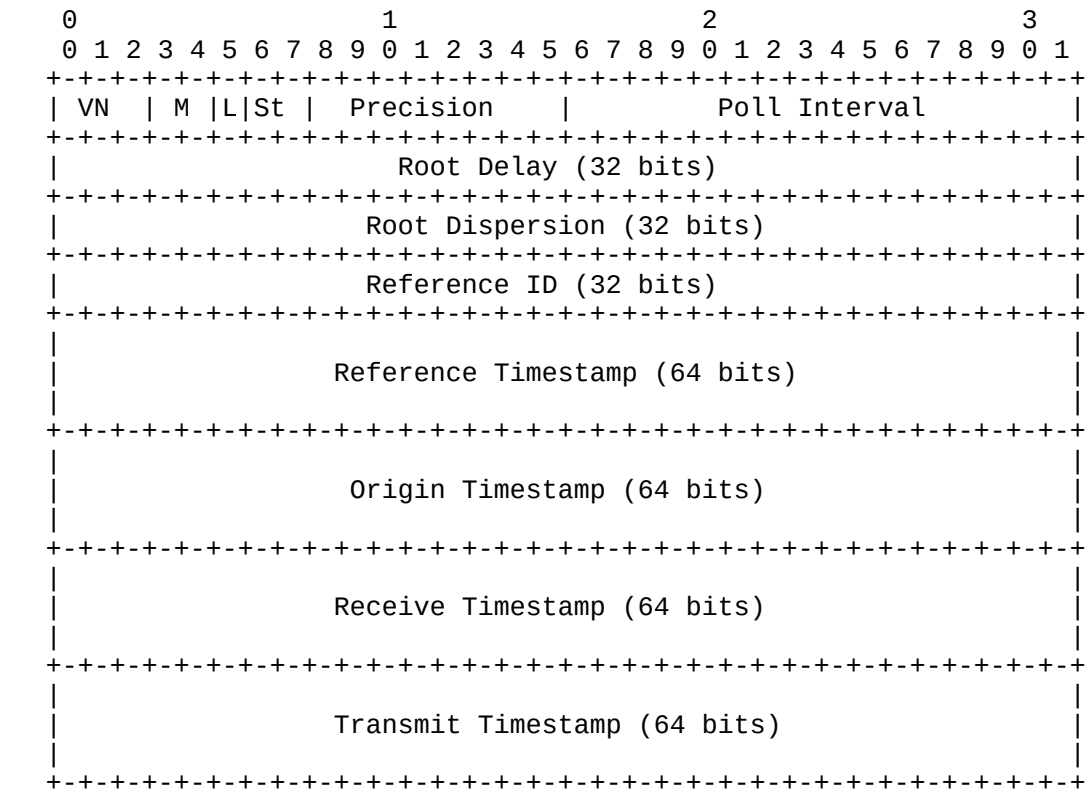
The client then computes the clock offset and round-trip delay using the four timestamps.

6. Packet Format

6.1. Header

All multi-octet fields in OITP packets MUST be transmitted in network byte order (big-endian). Bit 0 of each field is the most significant bit (MSB-first), following the IETF convention established in [RFC791].

All OITP packets share a common format:



Total packet size: 48 octets (same as NTP for ease of implementation on shared infrastructure).

6.2. Field Descriptions

Version Number (VN, 3 bits): The protocol version number. This specification defines version 1. Implementations MUST set this field to 1.

Mode (M, 2 bits): The operating mode of the sender:

Value	Mode
0	Reserved
1	Basic Client
2	Full Client
3	Server

Table 2

Leap Indicator (L, 1 bit): Set to 1 if the server’s UTC source indicates a pending leap second within the current UTC day (i.e., a leap second is scheduled at 23:59:60 UTC of the current day). The flag SHOULD be asserted no earlier than beat 0 of the affected day and SHOULD be cleared once the leap second has passed. Since decimal time does not observe leap seconds, this serves as an advisory flag only. Clients MAY use this to flag reduced accuracy during the leap second adjustment period.

Stratum (St, 2 bits): The distance from the reference source:

Value	Meaning
0	Reference clock (GPS/PPS-disciplined)
1	NTP-synchronized or UTC reference
2	Synchronized to stratum 0 or 1 server
3	Unsynchronized / Kiss-o’-Death (9.4)

Table 3

Precision (8 bits): Signed integer (two’s complement) representing the precision of the server’s clock, expressed as floor(log2(E)) where E is the maximum error expressed in beats. For example:

Value	Precision	Approximate Error
-10	~0.001 beats	~1 millibeat
-20	~0.000001 beats	~1 microbeat
-30	~1 nanobeat	~86.4 nanoseconds

Table 4

A typical NTP-backed server would advertise precision of approximately -13 to -16 (sub-millibeat).

Poll Interval (16 bits): Unsigned integer indicating the RECOMMENDED minimum interval between client requests, expressed in beats. A value of 0 indicates no recommendation. Servers under load SHOULD set this field to request reduced query rates. Clients SHOULD set this field to 0 in requests (the field carries server-to-client guidance; clients have no poll interval to communicate in a single request).

Root Delay (32 bits): Total round-trip delay to the primary reference source, expressed as a 16.16 fixed-point number in beats. The integer part occupies bits 31-16 and the fractional part occupies bits 15-0; a value of 0x00010000 represents exactly 1 beat (86.4 seconds). Clients SHOULD use this field for server selection (Section 11.2) when multiple servers are available; single-server clients MAY ignore it.

Root Dispersion (32 bits): Total dispersion to the primary reference source, expressed as a 16.16 fixed-point number in beats, using the same encoding as Root Delay. Clients SHOULD use this field for server selection (Section 11.2); single-server clients MAY ignore it.

Reference ID (32 bits): Identifies the particular reference source. For stratum 0 and stratum 1, this is a four-character ASCII string (left-justified, zero-padded) identifying the reference source type:

ID	Source
GPS\0	GPS or GNSS hardware reference clock
PPS\0	Pulse-per-second hardware reference
NTP\0	Time derived from an NTP source

Table 5

For stratum 2, this field contains the IPv4 address of the upstream OITP server, or the first 32 bits of the SHA-256 [RFC6234] hash of the upstream server's IPv6 address (16-byte binary representation per [RFC8200]). For IPv4-mapped IPv6 addresses (::ffff:0:0/96), implementations MUST use the embedded IPv4 address (the final 4 octets) directly rather than hashing the full 16-byte IPv6 representation, to ensure consistent Reference IDs across dual-stack deployments. This serves only as a non-cryptographic identifier for loop detection; collision resistance is not a security requirement.

For stratum 3, this field contains an ASCII diagnostic code (Section 9.4) for a Kiss-o'-Death response, or zero for unsynchronized clocks. Clients MUST treat a non-zero Reference ID in a stratum 3 response as a KoD code and handle it per Section 9.4; a stratum 3 response with a zero Reference ID indicates an unsynchronized server and MUST NOT be used for synchronization.

Reference Timestamp (64 bits): The time at which the server's clock was last set or corrected, in OITP timestamp format (Section 4). Servers SHOULD update this field after each successful clock discipline operation.

Origin Timestamp (64 bits): In a server response, this field contains the client's transmit timestamp copied from the request (Section 9.2). In a client request, this field SHOULD be set to zero. The client's local time is carried in the Transmit Timestamp field instead (Section 10.2).

Receive Timestamp (64 bits): The decimal time at which the request was received by the server. Set to zero in client requests.

Transmit Timestamp (64 bits): The decimal time at which the packet was sent. In Full mode (Section 10.3), both clients and servers MUST set this field to their local decimal time at the moment of transmission. A Full mode request with a zero Transmit Timestamp is invalid; servers SHOULD discard such requests silently. In Basic

Client mode (Section 10.2), the value zero is a valid transmit timestamp; the server echoes it back in the origin field without interpretation, and no offset or delay calculation is performed.

7. Synchronization Algorithm

7.1. Offset and Delay Calculation

Given the four timestamps from a full mode exchange (T1, T2, T3, T4 all in OITP 64-bit format), the clock offset theta and round-trip delay delta are computed as:

```
theta = ((T2 - T1) + (T3 - T4)) / 2
delta = (T4 - T1) - (T3 - T2)
```

Implementations MUST discard samples where the computed round-trip delay (delta) is negative, as this indicates inconsistent timestamps caused by path asymmetry changes, packet reordering, or malicious modification.

Because the 10-bit beat integer field uses only values 0-999 (not the full 0-1023 range), the raw 64-bit timestamp is not a uniform linear representation. Implementations MUST linearize timestamps before arithmetic by converting to: day * 1000 * 2^30 + beat * 2^30 + fraction. The results are in beat-fraction units where one beat equals 2^30 units. The maximum linearized value (day=16,777,215, beat=999, frac=2^30-1) is approximately 1.80 * 10^19, which exceeds the range of a signed 64-bit integer (maximum 9.22 * 10^18). Implementations MUST therefore use wider-than-64-bit arithmetic (128-bit integers or arbitrary precision) for linearized timestamp values. However, timestamp *differences* for practical synchronization (exchanges spanning fewer than approximately 8,500 days) fit in a signed 64-bit integer; implementations that restrict their operational window accordingly MAY use 64-bit arithmetic for differences only. On platforms where the default integer width is 32 bits, intermediate products MUST be explicitly cast to a wider type before multiplication to avoid silent overflow. Packets that would produce an out-of-range beat integer after linearization MUST be discarded per Section 4.1.

The offset theta represents the estimated correction needed to bring the client’s clock into alignment with the server. A positive value indicates the client is behind; a negative value indicates the client is ahead.

All four timestamp differences (T2-T1, T3-T4, T4-T1, T3-T2) MUST be computed as signed 64-bit integers. Although linearized timestamps are non-negative, their differences can be negative, and implementations using unsigned 64-bit types will produce incorrect results due to wrap-around.

To apply a computed offset to a local timestamp, the inverse of the linearization formula is:

```
linear  = day * 1000 * 2^30 + beat * 2^30 + frac
day     = floor(linear / (1000 * 2^30))
remainder = linear mod (1000 * 2^30)   ; non-negative
beat    = floor(remainder / 2^30)
frac    = remainder mod 2^30
```

where mod denotes the non-negative (floored) modulo operation. Implementations MUST use floor division (toward negative infinity) rather than truncation division (toward zero) to ensure correct de-linearization for negative intermediate values on signed platforms.

7.2. Filtering

Implementations SHOULD maintain a window of the most recent N exchange results (RECOMMENDED N=8) and apply a selection algorithm to reject outliers. The sample with the minimum delay in the window is RECOMMENDED as the best estimate, as lower delay generally correlates with more symmetric paths and therefore more accurate offset estimates. When two samples have equal delay, either may be selected; implementations SHOULD prefer the more recent sample as it reflects current network conditions.

7.3. Day Boundary Handling

Special care MUST be taken when timestamps in a single exchange span a day boundary (beat rolling from 999.xxx to 000.xxx). Implementations MUST compare day numbers in the timestamps and adjust calculations accordingly. An exchange spanning a day boundary is still valid provided the total round-trip delay is less than 500 beats (12 hours).

7.4. Precision Limits

The achievable synchronization precision depends on the server’s stratum and the network characteristics:

Scenario	Expected Precision
LAN, stratum 0 server	~0.01 millibeats (~1ms)
WAN, stratum 0 server	~0.1-1 millibeats
WAN, stratum 1 server	~1-10 millibeats

Table 6

These figures assume typical network jitter and are provided for guidance only.

8. Clock Discipline

8.1. Initial Synchronization

When an OITP client starts with no prior state, it SHOULD send a burst of 4 requests at short intervals (RECOMMENDED 2 seconds apart) to rapidly acquire an initial time estimate. The best sample (minimum delay) from this burst SHOULD be used for the initial clock step. If all burst queries fail to elicit a valid response (e.g., due to network unreachability), the client SHOULD retry the burst after a backoff interval (RECOMMENDED starting at 16 beats, doubling on each consecutive failure) rather than proceeding with an unsynchronized clock.

8.2. State Persistence

Implementations on devices with non-volatile storage SHOULD persist the last known good clock state (day number, approximate beat, and the estimated drift rate if tracked) across restarts. On restart, a persisted state MAY be used to skip the initial burst if the elapsed time since the last save is short and the confidence interval is acceptable, subject to implementation-defined bounds. This is particularly valuable for battery-powered or constrained devices where network operations are expensive.

8.3. Steady-State Operation

After initial synchronization, the client SHOULD poll the server at regular intervals. The RECOMMENDED default poll interval is 64 beats (approximately 92 minutes). Implementations MAY adjust the poll interval dynamically based on observed clock drift and network conditions, within the range of 16 beats (~23 minutes) to 1000 beats (~24 hours).

8.4. Clock Adjustment

For offsets smaller than 1 beat, implementations SHOULD slew the clock gradually rather than stepping it, to avoid discontinuities in the local decimal time. A slew rate of 0.5 millibeats per beat (approximately 43.2 ms/86.4s, or 500 ppm) is RECOMMENDED. When a new offset correction is computed while a previous slew is still in progress, the new correction SHOULD replace the remaining slew rather than accumulate with it. This is correct because the new measurement's timestamps (T1, T4) already reflect any partial slew applied since the previous correction.

For offsets larger than 1 beat, implementations SHOULD step the clock immediately, as slewing would take an impractical amount of time. Applications that read the clock continuously (e.g., for display) SHOULD be designed to tolerate sudden jumps at step events; implementations MAY emit a local notification (e.g., a log message or signal) when a step correction is applied, to allow dependent subsystems to react.

8.5. Panic Threshold

If the offset exceeds 50 beats (approximately 72 minutes), the client SHOULD assume a severe error condition and refuse to adjust the clock automatically. Operator intervention or a full re-initialization is RECOMMENDED.

Implementations MAY track the clock frequency error (drift rate) between successive offset corrections and apply predictive corrections between polls to reduce inter-poll drift. Frequency discipline algorithms are beyond the scope of this document and are expected to be addressed in a future revision of this specification.

9. Server Operations

9.1. Reference Clock Derivation

An OITP server derives its decimal time from a UTC reference source. The server MUST maintain an accurate UTC-to-decimal-time conversion, applying the formula in Section 3.3 with the highest precision available from its UTC source. When the server operates with its own clock discipline loop (e.g., synchronizing to an upstream OITP or NTP server), timestamps in responses MUST reflect the clock-corrected time rather than the raw system clock, so that accumulated corrections benefit clients immediately.

OITP servers MUST derive their time from a traceable UTC source (e.g., a GPS/PPS receiver, or an NTP-synchronized system clock). A server MUST NOT advertise stratum 0 or 1 unless it is in fact synchronized to such a source.

Servers with direct access to GPS or PPS reference clocks SHOULD operate at stratum 0. Servers deriving time from NTP-synchronized system clocks SHOULD operate at stratum 1.

9.2. Request Processing

Upon receiving a packet, the server MUST silently discard packets where:

* The version field is not 1
* The mode field is not 1 (Basic Client) or 2 (Full Client)
* The packet length is less than 48 octets

In particular, servers receiving mode 3 (Server) packets MUST discard them silently to prevent reflection attacks. The server MUST NOT send an error response to discarded packets; silent discard prevents information disclosure and avoids amplification of malformed traffic.

Upon receiving a valid client request, the server:

1. Records the receive timestamp T2
2. Sets the mode field to 3 (Server) in the response, regardless of the client's mode field (Basic or Full)
3. Copies the client's transmit timestamp to the origin timestamp field
4. Sets the receive timestamp field to T2
5. Sets all server metadata fields (stratum, precision, reference ID, root delay, root dispersion)
6. Records the transmit timestamp T3 as late as possible before sending
7. Sends the response

The server SHOULD minimize processing time between recording T2 and T3 to reduce uncertainty.

9.3. Rate Limiting

Servers SHOULD implement rate limiting to prevent abuse. A reasonable default is to allow no more than one request per beat (86.4 seconds) per source IP address during steady-state operation. To accommodate initial synchronization bursts (Section 8.1), servers SHOULD allow a short burst of up to 8 requests from a previously unseen source IP before enforcing the per-beat rate limit. Servers

MAY use the poll interval field to communicate desired polling rates to clients.

9.4. Kiss-o'-Death

If a server wishes to deny service to a client, it MUST respond with stratum 3 and a Reference ID containing an ASCII diagnostic code:

Code	Meaning
DENY	Access denied
RATE	Rate limit exceeded
RSTR	Access restricted
STEP	Server stepping clock

Table 7

Clients receiving a DENY or RSTR response SHOULD cease querying that server. Clients receiving a STEP response SHOULD wait one full poll interval before retrying, as the server is in the process of adjusting its own clock. Clients receiving a RATE response SHOULD double their poll interval and retry after the increased interval; if RATE is received again, the client SHOULD continue doubling up to the maximum poll interval before ceasing queries. Because this version of OITP lacks authentication, clients SHOULD NOT permanently blacklist a server based on a single KoD response; implementations SHOULD require consistent KoD responses across multiple poll cycles before ceasing queries permanently, to mitigate spoofed KoD denial-of-service attacks (Section 12.2).

10. Client Operations

10.1. Server Selection

Clients SHOULD be configured with at least one OITP server address. Server addresses MAY be specified as hostnames or as IP address literals (IPv4 dotted-decimal or IPv6 bracketed notation); constrained devices without access to a DNS resolver SHOULD use IP address literals directly to avoid the dependency on DNS resolution. When multiple servers are available, clients SHOULD query all of them and use standard intersection and clustering algorithms to select the best source and reject falsetickers.

10.2. Basic Mode Client Behavior

In basic mode, the client:

1. Sets mode to 1 (basic client)
2. Sets stratum to 3 (unsynchronized)
3. Sets the transmit timestamp field to zero (or any value; the server copies it to the origin field but it is not used for offset calculation)
4. Sets all other fields to zero
5. Sends the request to the server on port 8640

Upon receiving a response, the client MUST verify that the version field is 1 and the mode field is 3 (Server) before using the response. The client then reads the server's transmit timestamp (T3) as the current decimal time. No offset or delay compensation is applied. This mode provides beat-level accuracy suitable for display purposes where sub-beat precision is not required. Basic mode is particularly appropriate for battery-powered or highly constrained devices where minimizing computation and state is more important than sub-beat precision.

Basic mode clients MUST NOT interpret stratum 3 responses as Kiss-o'-Death. Basic mode clients SHOULD silently discard any response with stratum 3. Since basic mode uses a zero or predictable transmit timestamp, the origin timestamp match provides no protection against spoofed KoD responses.

10.3. Full Mode Client Behavior

In full mode, the client:

1. Records its current local decimal time as T1
2. Sets the transmit timestamp field to T1
3. Sets mode to 2 (full)
4. Sets stratum to 3 (unsynchronized) if the client has not yet synchronized, or to the client's current stratum otherwise
5. Sets the remaining fields as follows:
    * Precision: the client's estimated clock precision if synchronized, or 0 if unsynchronized
    * Poll Interval: 0
    * Root Delay: 0x00000000
    * Root Dispersion: 0x00000000
    * Reference ID: 0x00000000
    * Reference Timestamp: 0x0000000000000000
6. Sends the request to the server on port 8640

10.4. Response Validation

Upon receiving a response, the client MUST verify:

1. The source address of the response matches the destination address of the corresponding request
2. The version number is recognized (clients MUST discard responses with an unrecognized version number)
3. The mode field is 3 (server)
4. The origin timestamp matches the T1 sent in the request
5. The stratum is not 3 with a kiss-o'-death reference ID
6. The transmit timestamp is non-zero
7. All timestamp fields in the response (receive timestamp and transmit timestamp) MUST have beat integer values in the range 0-999. Packets with out-of-range beat integers MUST be discarded.

If any check fails, the response MUST be discarded.

10.5. Display

Clients providing user-facing time display SHOULD format the current decimal time according to Section 3.4. The number of fractional digits displayed SHOULD reflect the actual synchronization precision achieved, not the protocol's theoretical maximum.

11. Precision Hierarchy

11.1. Stratum Levels

OITP uses a simplified stratum hierarchy compared to NTP:

Stratum 0: The server has a direct hardware reference clock (GPS receiver with PPS output) and performs the UTC-to-decimal conversion locally. Expected precision: microbeats to nanobeats.

Stratum 1: The server derives its time from NTP (or another UTC source) and converts to decimal time. The precision is bounded by the NTP synchronization quality, typically sub-millibeat on well-connected hosts.

Stratum 2: The server synchronizes to a stratum 0 or stratum 1 OITP server. Each additional hop adds uncertainty from the OITP synchronization process. This is the maximum operational stratum; further cascading is not supported by the 2-bit field. A stratum 2 server MUST NOT synchronize to another stratum 2 server. Allowing stratum 2 chains would create unbounded hierarchy depth invisible to clients.

Version 1 of OITP targets small-to-medium deployments where one or two hops of OITP-native hierarchy is sufficient. A typical deployment consists of one or more stratum 0 servers (GPS/PPS-disciplined) with clients and stratum 1/2 servers synchronizing to them. The 2-bit stratum field is a deliberate simplicity trade-off for this scope.

Should future deployments require deeper native OITP hierarchies (e.g., large-scale networks of independent reference clocks), a subsequent version of this protocol could extend the stratum field. The 3-bit version number field (Section 6.2) and the extension mechanism (Section 12.8) provide the necessary upgrade path.

Stratum 3: Reserved for unsynchronized clocks and Kiss-o'-Death (KoD) responses. A stratum 3 packet with a non-zero Reference ID is a KoD; clients MUST handle it per Section 9.4 regardless of whether the specific code is recognized (unknown codes SHOULD be treated as DENY). A stratum 3 packet with a zero Reference ID indicates an unsynchronized server. Clients MUST NOT synchronize to stratum 3 servers. Servers MUST NOT advertise stratum 3 during normal operation. A server transitioning to stratum 3 (e.g., upon loss of its reference source) MUST set the Reference ID to zero unless it is intentionally sending a Kiss-o'-Death response. Failure to clear a stale Reference ID would cause clients to misinterpret the response as KoD.

11.2. Stratum Selection

Clients SHOULD prefer lower-stratum servers. When multiple servers of the same stratum are available, the server with the lowest root delay and root dispersion SHOULD be preferred.

12. Security Considerations

12.1. Rate Limiter Hash Collisions

Server implementations that use hash-based per-IP rate limiting (e.g., a direct-mapped hash table indexed by a hash of the source address) are subject to hash collisions: two distinct client addresses that hash to the same slot will share a rate limit bucket, potentially causing one client to be rate-limited due to the other's traffic. This is an implementation artifact and not a protocol vulnerability. Implementations with stringent fairness requirements SHOULD use larger tables or collision-resistant data structures.

12.2. Spoofing

OITP, like NTP, is vulnerable to spoofed responses from off-path attackers when running over plain UDP. Source address validation is normatively required by Section 10.4. The origin timestamp matching (Section 10.4) provides a weak form of authentication against blind spoofing. In particular, an attacker can spoof Kiss-o'-Death responses (Section 9.4) to cause clients to cease querying legitimate servers. Section 9.4 mitigates this by requiring multiple consistent KoD responses before permanent cessation.

The stratum, root delay, root dispersion, and precision fields are self-reported and unauthenticated. A malicious server can advertise optimal values (e.g., stratum 0 with minimal root delay) to be preferentially selected by clients. Multi-server deployments mitigate this through cross-validation of server claims.

12.3. On-Path Attacks

An on-path attacker who can intercept and modify OITP packets in transit can manipulate any or all of the four timestamps (T1-T4) to shift the client's clock by an arbitrary amount. This is a fundamental limitation of unauthenticated time synchronization protocols; NTP [RFC5905] has the same vulnerability.

Applications MUST NOT rely on OITP for security-sensitive timekeeping (e.g., certificate validity checking, session token expiry, audit log correlation) unless an authenticated mode is used. A future authentication extension (Section 12.7) would mitigate this attack. Until such an extension is standardized, deployments requiring tamper-resistant time synchronization SHOULD use NTS-protected NTP [RFC8915].

An on-path attacker or compromised server can gradually shift a client's clock by applying small offsets (less than the step threshold of 1 beat) at each poll cycle. The clock discipline algorithm will slew these adjustments without triggering any alarm. Over successive polls, the client's clock can drift arbitrarily far from true time. Deployments using a single server have no defense against this attack. Implementations SHOULD query multiple independent servers and apply intersection algorithms (Section 10.1) to detect and reject sources providing inconsistent time.

12.4. Amplification

The OITP UDP packet format is symmetric (48 bytes request, 48 bytes response), providing no amplification factor. This makes OITP unsuitable as a DDoS amplification vector via UDP. The optional HTTP interface (Appendix D) operates over TCP, which requires a three-way handshake that prevents source address spoofing; HTTP responses cannot be reflected to a forged source address.

12.5. Privacy

OITP requests do not inherently contain identifying information beyond the source IP address. However, the transmit timestamp reveals the client's current local decimal time estimate, which could theoretically be used to fingerprint client implementations or infer clock quality.

Since the transmit timestamp is essential for offset calculation (Section 7.1), it cannot be randomized without degrading synchronization precision. Clients concerned with privacy SHOULD rely on network-layer protections (e.g., source address anonymization) rather than timestamp manipulation.

12.6. Replay Attacks

An attacker who captures a valid OITP response can replay it to a client at a later time. The origin timestamp check (Section 10.4) mitigates this: the replayed response must carry the correct origin timestamp for the client's current request, which changes with every query. A replayed response will be rejected unless the attacker can also observe and match the client's current transmit timestamp.

12.7. Future Authentication

This specification does not define an authentication mechanism. A future extension could define authentication extension fields appended after the base 48-byte packet, similar to NTS (Network Time Security, [RFC8915]) for NTP. Such an extension could use modern symmetric or asymmetric cryptographic primitives. Future authentication extensions could take inspiration from protocols such as Roughtime [I-D.ietf-ntp-roughtime], which provides cryptographic time authentication with accountability.

12.8. Extension Mechanism

Implementations MUST send packets of exactly 48 octets for version 1 of this protocol. Implementations MUST accept and process packets of at least 48 octets. Implementations MUST silently discard packets shorter than 48 octets. Implementations MUST ignore any bytes beyond the 48-byte base packet. Future versions of OITP could define extension fields appended after the base header. The version field (Section 6.2) will be incremented for incompatible changes. Compatible extensions MUST be designed such that implementations unaware of them can safely discard the extra bytes. Because Version 1 implementations silently ignore bytes beyond the 48-octet base packet, an on-path attacker could strip a future authentication extension by truncating the packet. Any future authentication mechanism MUST use a new protocol version number or a mandatory-to-process signaling mechanism that cannot be silently removed, to prevent downgrade attacks.

Note: Version negotiation (allowing a client to discover and prefer the highest mutually-supported version) is not defined in this specification and is deferred to a future document. Version 1 implementations MUST silently discard requests and responses carrying an unrecognized version number.

13. IANA Considerations

13.1. Port Number

This document requests the assignment of port number 8640 (UDP and TCP) for the Open Internet Time Protocol. The port number 8640 references the fundamental constant of the decimal time system: 86400 seconds per day divided into 1000 beats of 86.4 seconds each. The port falls within the User Port range (1024-49151) and is currently unassigned in the IANA Service Name and Transport Protocol Port Number Registry.

Field	Value
Service Name	oitp
Port Number	8640
Transport Protocol	UDP, TCP
Description	Open Internet Time Protocol
Reference	This document

Table 8

TCP registration is requested because the OPTIONAL HTTP time interface (Appendix D) operates on TCP using the same port number. Registering both transports prevents future port conflicts and follows the recommendation of [RFC6335] Section 7.2 to register both UDP and TCP when either is used.

13.2. Reference ID Registry

This document requests IANA create an "OITP Reference Identifier" registry. Registry entries consist of:

* *Value*: a 4-octet ASCII string (right-padded with NUL bytes (0x00) if shorter), encoded as a 32-bit big-endian integer
* *Stratum*: the stratum value(s) for which this entry applies
* *Meaning*: description of the reference source or condition
* *Reference*: the document defining this entry

The registration policy is Specification Required [RFC8126]. IANA is requested to designate one or more experts for this registry.

Initial entries:

Value	Stratum	Meaning
NTP\0	1	Time derived from an NTP source
GPS\0	0	GPS or GNSS hardware reference clock
PPS\0	0	Pulse-per-second hardware reference
RATE	3	KoD: rate limit exceeded
DENY	3	KoD: access denied
RSTR	3	KoD: access restricted
STEP	3	KoD: server stepping clock

Table 9

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

14.2. Informative References

[I-D.ietf-ntp-roughtime] Ladd, W. and M. Dansarie, "Roughtime", Work in Progress, Internet-Draft, draft-ietf-ntp-roughtime-17, 21 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-ntp-roughtime-17>.

[IEEE1588] IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE Std 1588-2019, 2019.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <https://www.rfc-editor.org/info/rfc8915>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

Appendix A. Acknowledgments

OITP's design borrows from the Network Time Protocol, originally developed by David L. Mills. The four-timestamp exchange, stratum hierarchy, and Kiss-o'-Death mechanism all derive from NTP's architecture.

Appendix B. Reference Implementation Notes

The reference implementation, named Beatnik, is maintained at: https://oitp.net/download.html (https://oitp.net/download.html)

Beatnik provides:

* beatnik serve: Run an OITP server (stratum 0 with GPS/PPS, or stratum 1 with NTP derivation)
* beatnik query <server>: Single query with offset/delay display
* beatnik sync <server>: Daemon mode with clock discipline
* beatnik now: Display current decimal time

Beatnik targets a single static binary with zero external dependencies beyond the Zig standard library.

Appendix C. Example Exchange

Client (192.0.2.1) queries server (198.51.100.1) at approximately @248.500:

```
Client Request:
  Version: 1, Mode: 2 (Full Client), Leap: 0, Stratum: 3
  Precision: -10, Poll: 0
  Root Delay: 0, Root Dispersion: 0
  Reference ID: 0x00000000
  Reference Timestamp:  0x0000000000000000
  Origin Timestamp:     0x0000000000000000
  Receive Timestamp:    0x0000000000000000
  Transmit Timestamp:   0x0027103E20000000
    (Day 10000, Beat 248, Fraction 0.500)
```

```
       Timestamp encoding breakdown:
         Day 10000  = 0x002710 -> bits 63-40
         Beat 248   = 0x0F8    -> bits 39-30
                                  (10 bits: 00_1111_1000)
         Frac 0.500 = 0x20000000 -> bits 29-0
                                  (0.5 * 2^30 = 536870912)
         Raw: (0x002710 << 40) | (0x0F8 << 30)
              | 0x20000000
            = 0x0027103E20000000

   Server Response:
     Version: 1, Mode: 3 (Server), Leap: 0, Stratum: 1
     Precision: -14, Poll: 64
     Root Delay: 0x00000083
     Root Dispersion: 0x00000041
     Reference ID: "NTP\0"
     Reference Timestamp:  0x0027103E00000000
                           (Day 10000, Beat 248.000)
     Origin Timestamp:     0x0027103E20000000
                           (copied from request)
     Receive Timestamp:    0x0027103E20040000
                           (Beat 248.500244)
     Transmit Timestamp:   0x0027103E20048000
                           (Beat 248.500275)

   Client receives response, records T4:
     T4 (local clock):     0x0027103E20088000
                           (Beat 248.500519)

     T1 = 0x0027103E20000000  (client transmit)
     T2 = 0x0027103E20040000  (server receive)
     T3 = 0x0027103E20048000  (server transmit)
     T4 = 0x0027103E20088000  (client receive)

     Linearization (Section 7.1):
     All four timestamps share Day 10000 and Beat 248, so
     linearization is a no-op in this example. In general,
     toLinear(ts) = day*1000*2^30 + beat*2^30 + frac.

     T2 - T1 = 0x40000 = 262144 raw units
     T3 - T4 = -0x40000 = -262144 raw units

     offset = (262144 + (-262144)) / 2
            = 0 (symmetric path, clocks agree)
     delay  = (T4-T1) - (T3-T2)
            = 0x88000 - 0x8000
            = 524288 raw units
            = 524288 / 1073741.824
```

~ 0.488 millibeats (~42 ms round-trip)

Appendix D. HTTP Time Interface

This appendix is informative. It describes a convention for HTTP-based time queries and is not part of the normative protocol specification.

OITP servers may expose an HTTP [RFC9110] interface alongside the UDP protocol to provide low-friction access to the current decimal time. This interface is optional and informational; it does not participate in the synchronization protocol.

D.1. Endpoints

GET /time (or the root path): Returns the current beat time as plain text in the format @HHH.SSS\n where HHH is the zero-padded three-digit integer beat (000-999) and SSS is the zero-padded three-digit millibeat (000-999), truncated (not rounded) from the internal 30-bit fraction. The trailing newline is required (Content-Type: text/plain).

GET /json: Returns a JSON [RFC8259] object with the following fields (Content-Type: application/json):

```
{
  "timestamp": "2026.03.10@248.573",
  "time": "@248.573",
  "day": 10000,
  "beat": 248,
  "millibeat": 573,
  "date": "2026.03.10"
}
```

Field	Type	Description
timestamp	string	Combined date-time (Section 3.5)
time	string	Formatted beat time (@HHH.SSS)
day	integer	OITP day number since epoch
beat	integer	Integer beat (0-999)
millibeat	integer	Fractional millibeats (0-999)
date	string	Calendar date at UTC+1

Table 10

D.2. Operational Notes

* The HTTP interface should include Access-Control-Allow-Origin: * to enable browser-based clients.
* The HTTP interface runs on TCP port 8640, sharing the same port number as the UDP protocol. Since UDP and TCP are distinct transport protocols, no conflict arises.
* Responses reflect the server's current time at the moment of response generation, without any offset or delay compensation.
* Rate limiting should be applied independently from the UDP rate limiter.

Author's Address

Maxine
Email: max@nightglow.one