Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 17 August 2026

R. Collette
VextCODE

13 February 2026

SIN/IP: Secure In-Network Transport Protocol over IP
draft-collette-sinip-transport-00

Abstract

SIN/IP is a secure, multiplexed transport protocol designed for kernel residency and incremental deployment. It provides authenticated encryption (AEAD), multiplexed streams over a single connection to avoid head-of-line blocking, connection IDs for NAT rebinding and path migration, and pluggable congestion control and pacing. SIN/IP can run as a native IP protocol in controlled networks or be encapsulated in UDP for Internet deployment. This document specifies the wire format, packet and frame layout, connection establishment, packet protection, loss recovery, flow control, congestion control, path management, and connection termination. It also defines IANA registries for SIN/IP parameters and codepoints.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfcXXXX (https://www.rfc-editor.org/info/rfcXXXX).

Table of Contents

{:toc}

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

SIN/IP (Secure In-Network over IP) is a transport protocol that provides confidentiality, integrity, multiplexed streams, and connection identity decoupled from the network path. It is designed to run in the kernel for consistent policy, zero-copy integration, and congestion control, while remaining deployable on the public Internet via UDP encapsulation.

2. Goals

* *Secure by default:* All post-handshake data is protected with an AEAD. Key exchange uses X25519; keys are derived with HKDF. Replay is prevented by authenticating packet numbers and optional token binding.
* *Multi-stream within one connection:* Multiple streams share one connection; each stream has its own offset space and reassembly. Loss on one stream does not block delivery on others (no head-of-line blocking).
* *Pluggable congestion control and pacing:* The design exposes hooks for CC modules (e.g., CUBIC, BBR) and pacing; ACK ranges feed loss detection and RTT estimation.
* *NAT rebinding support:* Connection IDs decouple the logical connection from the 4-tuple; path validation (PATH_CHALLENGE / PATH_RESPONSE) confirms reachability before migrating traffic.
* *Kernel residency:* The transport is intended to run in the kernel to enable zero-copy, stable socket API, policy enforcement, and observability.

3. Non-Goals

* *Replacing TLS for application security semantics:* SIN/IP provides transport-layer encryption and optional server authentication (e.g., Ed25519 signature over the handshake). Application-level certificate verification and TLS-specific features remain the responsibility of the application or an optional TLS layer above SIN/IP.
* *Full Internet deployability without encapsulation:* A new IP protocol number is often blocked or altered by middleboxes. SIN/IP specifies UDP encapsulation as the primary deployable mode for Internet use; native IP protocol is for controlled networks only.
* *Perfect compatibility with every existing socket behavior:* The API is designed to be familiar (stream, seqpacket, datagram) but may not mirror every legacy TCP or UDP quirk.

4. Rationale and Design Principles

Modern applications need low-latency connection setup, multiple logical streams without head-of-line blocking, mobility across NATs, and encryption by default. TCP plus TLS adds RTTs and does not multiplex; QUIC meets many of these goals but is typically implemented in userland, which complicates kernel policy, zero-copy, and observability. SIN/IP targets environments where kernel control and transport evolution matter: datacenters, enterprise edges, and embedded or gateway deployments.

Design principles include: (1) authenticate and encrypt all post-handshake traffic; (2) use a single, well-defined wire format for interoperability; (3) resist ossification via GREASE and "MUST ignore unknown" rules; (4) limit amplification and state allocation until the client is validated; (5) support both native IP and UDP encapsulation with the same wire format.

5. Protocol Overview (One-Page Summary)

A *connection* is identified by connection IDs (dcid, scid) and carries one or more *streams*. Each *packet* has a fixed 32-byte header and a payload of *frames*. *Packet numbers (PN)* are per space (Initial, Handshake, 1-RTT) and drive ACKs and nonce derivation.

Lifecycle: The client sends an INITIAL packet (type 0) with its X25519 public key and connection ID. The server may respond with RETRY (type 5) and a token for stateless retry; the client resends INITIAL with the token. The server responds with SIN_ACK (type 1) with its public key. Both derive the shared secret via X25519 and HKDF. The client sends CONFIRM (type 2); once both have sent and received 1-RTT packets (type 3), the connection is ESTABLISHED. Data transfer uses 1-RTT packets carrying STREAM, ACK, PING, and other frames. Teardown uses CONNECTION_CLOSE or FIN; STATELESS_RST (type 6) allows the server to reject unknown connections without state.

Deployment: In *native* mode, SIN/IP is carried directly in IP (IPv4 or IPv6) with a dedicated protocol number. In *UDP-encapsulated* mode, each SIN/IP packet is sent inside a UDP datagram. UDP encapsulation is the primary deployable mode for Internet use; implementations intended for general Internet use MUST support it.

6. Conventions and Definitions

6.1. Requirements Language

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 all capitals, as shown here.

6.2. Terminology

* *Connection:* The long-lived association between two endpoints, keyed by connection IDs.
* *Stream:* A unidirectional or bidirectional logical channel within a connection, with its own offset space for data.
* *Frame:* A typed unit inside the protected payload (STREAM, ACK, PING, PATH_CHALLENGE, etc.).
* *Packet:* A unit of transmission consisting of a fixed 32-byte header plus payload (frames) and, when encrypted, a 16-byte AEAD tag.
* *Packet number (PN):* A monotonically increasing value per packet number space, used for acknowledgments, loss detection, and nonce derivation.
* *Connection ID (CID):* An opaque identifier used for demultiplexing. The receiver uses the *destination CID (dcid)* to identify the connection; the *source CID (scid)* identifies the sender.
* *Endpoint:* Either peer in a SIN/IP connection (client or server).

6.3. Notation

* All multi-byte integer fields in the wire format are *big-endian* unless otherwise specified.
* Byte ranges and offsets are 0-based.
* "Payload" refers to the protected payload (frames) only; the 16-byte AEAD tag, when present, is not part of the payload length.

7. Architecture and Service Model

7.1. Connection Model

A SIN/IP connection is identified by connection IDs. The receiver demultiplexes packets by destination connection ID (dcid). The connection persists across path changes (e.g., NAT rebind) as long as the peer can be reached and path validation succeeds. Each connection has distinct packet number spaces (Initial, Handshake, 1-RTT) and keying material per direction.

7.2. Streams and Application Semantics

Streams are identified by a stream ID. Each stream has an independent send offset and receive offset. The protocol ensures ordered delivery within a stream but not across streams. Streams can be full-duplex. The receiver reassembles data by stream_id and offset; loss on one stream does not block delivery on others.

7.3. Reliability vs Datagram Service (Optional Extension)

The default service is reliable, ordered delivery per stream. Optional extensions (e.g., capability version 1.1) may define partial reliability (deadline-based drop, unordered delivery) with normative semantics when enabled. Such extensions are out of scope for the base wire format and are signaled via transport parameters or capability negotiation.

8. Deployment Modes

8.1. UDP Encapsulation Mode

In UDP encapsulation mode, each SIN/IP packet is sent as the payload of a UDP datagram. This is the *primary deployable mode* for use on the public Internet, because many middleboxes drop or alter non-TCP/UDP traffic. Implementations intended for general Internet use MUST support UDP encapsulation.

The same 32-byte fixed header and frame encoding are used; only the outer delivery (UDP/IP) and PMTU handling differ. Implementations MUST support a conservative default MTU (e.g., 1200 bytes for UDP payload) or process ICMP Packet Too Big when available.

8.2. Native IP Protocol Mode

In native mode, SIN/IP is carried directly in IPv4 or IPv6 with a dedicated protocol number (to be assigned by IANA). This mode is for controlled networks (datacenter, enterprise) where the operator controls or trusts middleboxes. The wire format is identical to UDP encapsulation; only the outer layer (IP next header / protocol) differs.

8.3. Middlebox Traversal Considerations

Use of a new IP protocol number on the public Internet is NOT RECOMMENDED. SIN/IP specifies UDP encapsulation as the normal mode for Internet deployment. Middleboxes that allow UDP typically allow SIN/IP over UDP; those that block or alter non-TCP/UDP traffic will not affect UDP-encapsulated SIN/IP. NAT rebinding is handled by connection IDs and path validation; no middlebox changes are required.

9. Packet Format

9.1. Common Packet Structure

Every SIN/IP packet has:

1. *Fixed header:* 32 bytes (see {{fixed-header}}).
2. *Payload:* Zero or more bytes. For handshake packets (INITIAL, SIN_ACK, CONFIRM, RETRY), the payload is unencrypted. For 1-RTT and 0-RTT, the payload is encrypted and authenticated.
3. *AEAD tag:* 16 bytes, present only when the payload is encrypted. Immediately follows the payload.

The total length of a packet is 32 + payload_len + (16 if encrypted). Implementations MUST reject packets shorter than 32 bytes. If the payload_len field is inconsistent with the actual received length (e.g., extends beyond the packet), the packet MUST be discarded.

9.2. Fixed Header

The fixed header is exactly 32 bytes. All multi-byte fields are big-endian.

Offset	Size	Name	Description
0	1	ver_type	High 4 bits: version (see {{version-registry}}). Low 4 bits: packet type (see {{packet-types}}).
1	1	flags	Header flags (ACK_ELICITING, FIN, KEY_PHASE, etc.). See {{header-flags}}.
2	1	hlen_words	Header length in 4-octet units. For version 1,

			this MUST be 8 (32 bytes).
3	1	reserved	Reserved. Senders MUST set to 0; receivers MUST ignore.
4	2	epoch	Key phase (handshake, 0-RTT, 1-RTT).
6	2	payload_len	Length in bytes of the protected payload only (excluding the 16-byte AEAD tag when present).
8	1	stream_id	Optional default stream for this packet (0 means no default).
9	1	reserved2	Reserved. Senders MUST set to 0; receivers MUST ignore.
10	4	pn	Packet number in the space implied by packet type.
14	8	dcid	Destination connection ID (8 bytes).
22	8	scid	Source connection ID (8 bytes). May be zero in INITIAL.
30	2	hdr_ext	Extension / reserved. Senders MAY set to 0; receivers MUST ignore unless specified.

Table 1

There is no padding between the header and the payload; the payload immediately follows the header.

9.3. Header Field Semantics

* *ver_type:* The high nibble is the SIN/IP wire version. The low nibble is the packet type. Unknown versions or types MUST cause the packet to be discarded (see {{ossification}}).
* *payload_len:* Must not exceed the remaining packet length. If payload_len indicates more data than received, the packet MUST be discarded.
* *pn:* Monotonically increasing within each packet number space. Used for ACKs, loss detection, and nonce derivation (see {{nonce}}).
* *dcid, scid:* Opaque 8-byte identifiers. The receiver uses dcid to demultiplex to the correct connection.

9.4. Connection IDs

Connection IDs are 8 bytes. The *destination connection ID (dcid)* is the identifier of the connection at the receiver; the *source connection ID (scid)* identifies the sender. In an INITIAL packet, scid MAY be zero. The receiver MUST use dcid to look up the connection. CIDs allow the connection to survive address changes (NAT rebind, path migration) because the connection identity is not tied to the 4-tuple.

9.5. Packet Numbers and Spaces

Packet numbers are maintained in separate spaces: *Initial*, *Handshake*, and *1-RTT* (and optionally *0-RTT*). Each space has its own keys and PNs. Within a space, PN is monotonically increasing. Key update MUST be triggered before PN wrap (e.g., at 2^31 - 1 or by byte/time limits) so that (key, PN) remains unique for the lifetime of the key.

9.6. Versioning and Extensibility

The wire version (high 4 bits of ver_type) identifies the packet format. Version 1 is defined in this document. Unknown versions MUST be discarded. Reserved and GREASE values (e.g., 0x0F) MUST be handled without fatal error (see {{ossification}}). New wire-incompatible formats require a new version; new optional behaviors may be negotiated via transport parameters or capability version within the same wire version.

9.7. Ossification Resistance

Implementations MUST ignore *unknown packet types* (ver_type low nibble not in the assigned set): such packets MUST be discarded without failing the connection. Implementations MUST ignore *unknown frame types*: skip the frame using the length field and continue parsing. Implementations MUST ignore *unknown transport parameter IDs* within TRANSPORT_PARAMS. Reserved and GREASE values (e.g., ver_type 0x0F, packet type 15, frame type 255) MUST be handled without causing a fatal error. Receivers MUST NOT treat unknown or reserved values as a reason to abort the connection. This prevents ossification and allows future extensions.

10. Frame Format

10.1. Frame Encoding

Frames are encoded as TLV: 1 byte type, 2 bytes length (big-endian), then value. *Frames MUST NOT span packets;* each frame is fully contained in a single packet's payload. If a frame's length field indicates a value that extends beyond the remaining payload, the packet MUST be discarded. *Unknown frame types* MUST be ignored: the implementation skips the frame (using the length field to advance) and continues parsing; unknown frames MUST NOT cause the connection to fail. Reserved and GREASE frame type values MUST be handled the same way.

10.2. Frame Type Registry (Normative Table)

Type	Name	Description
0	PADDING	Padding; may be used for alignment or GREASE.
1	STREAM	Stream data (stream_id, offset, length, data).
2	STREAM_FIN	Stream data with end-of-stream.
3	ACK	Acknowledgment with ranges (largest PN, delay, ranges).
4	PING	Keepalive; ACK-eliciting.
5	PATH_CHALLENGE	Path validation (8 bytes data).

6	PATH_RESPONSE	Path validation response (echo 8 bytes).
7	TRANSPORT_PARAMS	Transport parameter TLV.
8	NEW_CONNECTION_ID	Issue new connection ID.
9	RETIRE_CONNECTION_ID	Retire connection ID.
10	CONNECTION_CLOSE	Close connection with reason code.
11-254	Reserved	Reserved for future use.
255	GREASE	Reserved for GREASE.

Table 2

Allocation policy for new frame types: Specification Required.

10.3. STREAM

STREAM carries data for a stream. Format: type (1) = 1, length (2), then stream_id (1), offset (8), data length (2), data (variable). The receiver reassembles by stream_id and offset; duplicate data (same stream_id and offset) is deduplicated.

10.4. STREAM_FIN

Same as STREAM with an end-of-stream indication (e.g., a flag or separate type 2). The receiver delivers data in order and then signals end-of-stream to the application.

10.5. ACK

ACK carries the largest received PN, an ACK delay, and a list of (count, first_pn) ranges indicating which packets were received. Format: type (1) = 3, length (2), largest_pn (4), ack_delay (2), num_ranges (1), then for each range: count (2), first_pn (4). This provides SACK-like information for loss recovery.

10.6. PING

Type 4, length 0 (or minimal). ACK-eliciting; used for keepalive or RTT measurement.

10.7. PATH_CHALLENGE

Type 5, length 8, value = 8 random bytes. Sent to validate a new path; the peer echoes the bytes in PATH_RESPONSE.

10.8. PATH_RESPONSE

Type 6, length 8, value = 8 bytes (echo of PATH_CHALLENGE). Proves the peer received the challenge and holds the connection keys.

10.9. TRANSPORT_PARAMS

Type 7. Value is a TLV-encoded set of transport parameters (parameter ID, length, value). Used in handshake for version negotiation, flow control limits, max streams, etc. Unknown parameter IDs MUST be ignored.

10.10. CONNECTION_CLOSE

Type 10. Carries a 16-bit reason code (see {{error-codes}}) and optional reason phrase. Signals graceful or error close.

10.11. NEW_CONNECTION_ID

Type 8. Issues a new connection ID to the peer for migration or load balancing. Format and semantics are implementation-defined; typically includes sequence number and the new CID.

10.12. RETIRE_CONNECTION_ID

Type 9. Tells the peer to retire a previously issued connection ID.

10.13. Reserved / GREASE Frames

Frame type 255 and reserved types (11-254) MUST be ignored. Senders MAY send PADDING or GREASE for ossification resistance. Receivers MUST NOT treat unknown or reserved frame types as fatal.

11. Transport Parameters

11.1. Encoding and Negotiation Rules

Transport parameters are carried in TRANSPORT_PARAMS frames (or in the INITIAL/SIN_ACK payload in some implementations). Each parameter has a 16-bit ID, 16-bit length, and value. Parameters are negotiated during the handshake; the server can send its parameters in SIN_ACK, the client in CONFIRM or INITIAL. Unknown parameter IDs MUST be ignored.

11.2. Defined Transport Parameters

At minimum, the following are used for interoperability:

* *Supported versions / capability version:* List or single value indicating supported capability set (e.g., 1.0, 1.1). The selected version is the highest mutually supported.
* *Max streams:* Maximum number of streams the endpoint will accept.
* *Initial flow control window:* Initial connection-level and/or stream-level flow control window in bytes.
* *Idle timeout:* Connection idle timeout in milliseconds.

Additional parameters (ECN support, pacing, etc.) may be defined in extension documents. Allocation policy: Specification Required.

11.3. Defaults and Error Handling

If a required parameter is missing, implementations MAY use a safe default or close the connection with PROTOCOL_VIOLATION. Invalid or out-of-range values SHOULD result in CONNECTION_CLOSE with an appropriate error code.

12. Connection Establishment

12.1. Handshake Overview

The handshake is 1-RTT: INITIAL -> SIN_ACK -> CONFIRM. Optionally, the server sends RETRY before SIN_ACK to enforce address validation (stateless retry). 0-RTT data may be sent with CONFIRM if the client has a session ticket.

12.2. Packet Types

Type	Name	Description
0	INITIAL	Client first flight; carries client public key, CID, optional token.
1	SIN_ACK	Server response; carries server public key, selected version.
2	CONFIRM	Client confirmation; may carry early 1-RTT or 0-RTT data.
3	1RTT	Encrypted 1-RTT data (frames only).
4	0RTT	Encrypted 0-RTT data (with 0-RTT

		keys).
5	RETRY	Stateless retry; server sends token, no state allocated.
6	STATELESS_RST	Stateless reset; server rejects unknown connection.
7-15	Reserved	Reserved / GREASE.

Table 3

12.3. Client State Machine

CLOSED -> (send INITIAL) -> INITIAL_SENT -> (recv RETRY -> resend INITIAL with token) -> (recv SIN_ACK) -> SIN_ACK_RCVD -> (send CONFIRM) -> (recv 1RTT) -> ESTABLISHED. On timeout or invalid response, transition to CLOSED.

12.4. Server State Machine

LISTEN -> (recv INITIAL) -> INITIAL_RCVD -> (optional: send RETRY) -> (send SIN_ACK) -> SIN_ACK_SENT -> (recv CONFIRM) -> ESTABLISHED. The server does not allocate full per-connection state until CONFIRM is received (see {{anti-amplification}}).

12.5. Stateless Retry

The server MAY respond to an INITIAL with RETRY, sending a token. The token MUST be integrity-protected (e.g., HMAC with server secret) and SHOULD bind the client address and expire (e.g., 60 seconds). The client MUST resend INITIAL including the token. The server MUST NOT allocate full connection state until the client has echoed a valid token. This limits state exhaustion and supports anti-amplification.

12.6. 0-RTT Data

If the server provided a session ticket, the client MAY send 0-RTT data with CONFIRM. The server MUST apply replay protection (e.g., accept 0-RTT only once per ticket or within a time window). Non-idempotent 0-RTT data SHOULD be treated as potentially replayed; the server MAY reject or delay it.

12.7. Address Validation and Anti-Amplification

Until the client has been validated (e.g., by echoing a RETRY token or completing the handshake), the server MUST NOT send more than *3 times* the number of bytes it has received from that client. That is, for each unvalidated client address, (bytes sent in response) <= 3 * (bytes received from that client). The server allocates full per-connection state only after receiving a valid CONFIRM. Under load, the server MAY reject new INITIALs with RETRY or STATELESS_RST.

13. Packet Protection

13.1. Cryptographic Agility

SIN/IP version 1 uses X25519 for key exchange, HKDF-SHA256 for key derivation, and either XChaCha20-Poly1305 or AES-256-GCM for AEAD. Future versions may define other algorithms via the version or transport parameters.

13.2. Key Exchange

The client sends its X25519 public key (32 bytes) in the INITIAL payload; the server sends its X25519 public key (32 bytes) in SIN_ACK. Both compute the 32-byte ECDH shared secret. Optional server authentication: the server may include an Ed25519 signature over (client_pubkey || server_pubkey); the client MUST verify it when server identity is required.

13.3. Key Schedule

The shared secret is input to HKDF-SHA256. The info string (e.g., "SINIPv1 Key Material") and output length (e.g., 56 bytes) yield: 32-byte session key, 12-byte client send IV, 12-byte server send IV. The initiator uses client IV for sending and server IV for receiving; the responder does the opposite.

13.4. Nonce Construction

The nonce is 12 bytes: *nonce = IV XOR (0x00000000 || PN)* where PN is 32 bits big-endian in the low four bytes. This ensures a unique nonce per (key, PN). PN spaces are separate, so keys differ per space; key update MUST occur before PN wrap.

13.5. AEAD and Associated Data

The AEAD authenticated data (AD) is the entire 32-byte header. The plaintext is the payload (frames); the ciphertext and 16-byte tag replace the plaintext on the wire. Tampering with the header causes verification to fail.

13.6. Key Update and Epoch Handling

Rekeying can be triggered by bytes encrypted, time, or packet number window. The KEY_PHASE bit in the header indicates the phase. New keys are derived from the existing key or a new handshake. The endpoint MUST trigger key update before PN wrap in a space.

13.7. Replay Protection

Each packet uses a distinct nonce; replay of a ciphertext is detected (same PN under same key). For 0-RTT, the server MUST apply replay protection (e.g., one-time use per ticket or time window).

14. Reliability and Loss Recovery

14.1. ACK Generation

The receiver SHOULD send an ACK for every ACK-eliciting packet. It MAY coalesce ACKs within a short delay (e.g., max ack_delay). ACK frames carry the largest received PN and ranges; ACK delay can be included for RTT estimation.

14.2. Retransmission and PTO

Lost data is retransmitted in *new* packets with new packet numbers. The sender declares a packet lost when it has been outstanding for at least the PTO (RTO) or when a reorder threshold (e.g., 3 later packets acked) is met. RTO is computed using an RFC 6298-style algorithm: SRTT, RTTVAR, RTO = SRTT + 4*RTTVAR, clamped to 1-60 seconds. Implementations MUST use a single normative algorithm for a given capability version.

14.3. Reordering, Duplicate Detection, and Thresholds

The receiver deduplicates by (stream_id, offset) for STREAM data. The sender deduplicates ACKs by packet number. A reorder threshold (e.g., 3 packets) avoids declaring loss too early when packets are reordered. The reorder window (in packets or time) MUST be defined to minimize spurious retransmits.

15. Flow Control

15.1. Connection-Level Flow Control

The receiver advertises a connection-level flow control window (total bytes in flight across all streams). The sender MUST NOT send beyond this window. Window updates are sent via transport parameters or dedicated frames.

15.2. Stream-Level Flow Control

Each stream has an advertised receive window. The sender MUST NOT send stream data beyond the stream's window. When the advertised window is zero, the sender MUST send periodic probes (e.g., PING or minimal STREAM) at most every T seconds (e.g., 5) so the receiver can send a window update.

15.3. Stream Limits

The maximum number of streams and the maximum stream ID are negotiated via transport parameters. Exceeding the limit results in connection close or stream rejection.

16. Congestion Control

16.1. Requirements

Implementations MUST implement a congestion controller. The algorithm MUST reduce the send rate in response to loss (and to ECN CE when ECN is in use). The default MAY be NewReno-like, CUBIC, or BBR.

16.2. Default Congestion Controller

This document does not mandate a single algorithm; implementations choose among standard algorithms (e.g., CUBIC {{?RFC8312}}, BBR). The choice is configurable (e.g., socket option or sysctl).

16.3. Pacing

Sending MUST be paced according to the congestion controller's allowed rate to avoid bursts. Pacing is a normative requirement for capability versions that specify it (e.g., v1.1).

16.4. PLPMTUD (If Supported)

Over UDP encapsulation, endpoints MUST implement safe MTU handling: process ICMP Packet Too Big and reduce the effective payload size, or use a conservative default (e.g., 1200 bytes). Probe-based PLPMTUD (e.g., RFC 4821) may be added in a future version.

17. Path Management and Mobility

17.1. NAT Rebinding

Connection IDs decouple the connection from the 4-tuple. When the client’s address changes (e.g., new NAT binding), it continues to use the same dcid/scid so the server can route packets to the correct connection. No change to the wire format is required.

17.2. Connection Migration

Before using a new path for data, the endpoint MUST complete path validation (PATH_CHALLENGE / PATH_RESPONSE). Until validation succeeds, only PATH_CHALLENGE and PATH_RESPONSE frames MAY be sent on the new path. This prevents redirect attacks.

17.3. Path Validation Using PATH_CHALLENGE/PATH_RESPONSE

The endpoint sends PATH_CHALLENGE with 8 random bytes. The peer echoes them in PATH_RESPONSE. Only the holder of the connection (with the right keys) can produce a valid response. After validation succeeds, the endpoint may use the new path for 1-RTT data.

18. Connection Termination

18.1. Graceful Close

Either endpoint sends a packet with the FIN flag or a CONNECTION_CLOSE frame. The peer acknowledges and may send its own FIN. The connection enters TIME_WAIT (see {{time-wait}}). State machine: ESTABLISHED -> FIN_WAIT_1 -> FIN_WAIT_2 or CLOSING -> TIME_WAIT -> CLOSED.

18.2. Stateless Reset

The server may send STATELESS_RST (packet type 6) without looking up the connection. The payload is typically an HMAC of the dcid with a server secret, truncated. Only the server can generate a valid reset. The client treats it as connection closed.

18.3. TIME_WAIT and Reuse Rules

The endpoint in TIME_WAIT MUST retain state for at least 2*MSL or 30 seconds, whichever is greater. During this time, the same (4-tuple, dcid) MUST NOT be reused for a new connection. This prevents delayed segments from being mistaken for segments of a new connection.

19. Middlebox Considerations

Middleboxes that allow UDP typically allow SIN/IP over UDP. No middlebox changes are required. Use of a new IP protocol number on the public Internet is NOT RECOMMENDED; UDP encapsulation is the primary deployable mode. NAT rebinding is handled by connection IDs and path validation; the 4-tuple may change without breaking the connection.

20. Implementation Considerations

20.1. Kernel Integration Considerations

When SIN/IP is implemented in the kernel: (1) Session keys and keying material MUST be zeroized when the connection is destroyed or the key phase is retired. (2) Header and frame parsers MUST validate all lengths and bounds before use; invalid or truncated data MUST NOT be dereferenced. (3) Protocol updates may be decoupled from kernel release cadence (e.g., loadable module). (4) The API between kernel and user space MUST NOT expose raw keys or unvalidated packet data; only decrypted, reassembled data and metadata appropriate to the socket abstraction MAY be exposed.

20.2. Resource Limits and DoS Hardening

Implementations SHOULD enforce connection limits, rate limiting per address, and idle timeouts. The amplification limit (Section 8.7) and state allocation rules (state only after CONFIRM) are normative. Under load, the server MAY send RETRY or STATELESS_RST to shed load.

20.3. Buffering, Reassembly, and Zero-Copy

The receiver reassembles by stream_id and offset. Implementations may use scatter-gather and zero-copy (e.g., sendfile, splice) when the transport is in the kernel; such optimizations are implementation-defined.

21. Security Considerations

* *Confidentiality and integrity:* All post-handshake traffic is protected with an AEAD; the header is authenticated as associated data. Passive and active attackers cannot read or modify payloads without detection.
* *Replay:* Unique nonces (IV XOR PN) prevent replay. 0-RTT requires server-side replay protection.
* *Downgrade:* Version and capability are in the wire and transport params; implementations MUST NOT accept unsupported versions or fall back to weaker options.
* *DoS:* Amplification is limited to 3*; state is allocated only after client validation. Stateless retry and STATELESS_RST allow the server to shed load.
* *Key material:* Keys MUST be zeroized when no longer needed. Kernel implementations MUST NOT expose keys to user space.

22. Privacy Considerations

SIN/IP does not intentionally expose identifiers beyond what is necessary for demultiplexing (connection IDs). Connection IDs are opaque and may be changed (NEW_CONNECTION_ID / RETIRE_CONNECTION_ID). Packet numbers and timing may leak side-channel information; implementations should consider constant-time crypto and traffic analysis resistance where applicable.

23. IANA Considerations

IP Protocol Number Allocation: IANA is requested to assign an IP Protocol Number for "SIN/IP" in the "Protocol Numbers" registry, for use as IPv4 Protocol and IPv6 Next Header when SIN/IP is used in native mode. The assignment is for experimental use.

UDP Port Allocation for Encapsulation: IANA is requested to assign a UDP destination port for SIN/IP encapsulation in the "Service Name and Transport Protocol Port Number" registry. Until assignment, implementations MAY use a configurable port (e.g., for testing). Suggested service name: sinip.

SIN/IP Version Registry: IANA is requested to create a new registry "SIN/IP Versions" under "SIN/IP Parameters". Initial allocation:

Value	Description
0x0	Invalid / reserved
0x1	Version 1 (this document)
0x2-0xE	Unassigned
0xF	GREASE

Table 4

Allocation policy: Specification Required.

SIN/IP Packet Type Registry: IANA is requested to create "SIN/IP Packet Types" (low 4 bits of ver_type). Initial values: 0=INITIAL, 1=SIN_ACK, 2=CONFIRM, 3=1RTT, 4=0RTT, 5=RETRY, 6=STATELESS_RST. Values 7-14 reserved; 15 GREASE. Allocation policy: Specification Required.

SIN/IP Header Flags Registry: IANA is requested to create "SIN/IP Header Flags" (8-bit bitmap). Initial: bit 0 = ACK_ELICITING, bit 1 = FIN, bit 2 = KEY_PHASE. Remaining bits reserved. Unknown flags MUST be ignored. Allocation policy: Specification Required.

SIN/IP Frame Type Registry: IANA is requested to create "SIN/IP Frame Types" (8-bit). Initial values as in the table in Section 6.2. Allocation policy: Specification Required.

SIN/IP Transport Parameter Registry: IANA is requested to create "SIN/IP Transport Parameters" (16-bit IDs). Allocation policy: Specification Required.

SIN/IP Error Code Registry: IANA is requested to create "SIN/IP Error Codes" (16-bit). Initial values: 0x0000=NO_ERROR, 0x0001=INTERNAL_ERROR, 0x0002=PROTOCOL_VIOLATION, 0x0003=UNSUPPORTED_VERSION, 0x0004=IDLE_TIMEOUT, 0x0005=INVALID_TOKEN, 0x0006=CONNECTION_REFUSED, 0x0007=RESOURCE_EXHAUSTED, 0x0008=CRYPTO_ERROR. 0x0009-0xFFFF reserved. Allocation policy: Specification Required.

24. References

24.1. 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 (https://www.rfc-editor.org/info/rfc2119).

[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 (https://www.rfc-editor.org/info/rfc8174).

[RFC6298] Paxson, V., et al., "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, https://www.rfc-editor.org/info/rfc6298 (https://www.rfc-editor.org/info/rfc6298).

24.2. Informative References

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, https://www.rfc-editor.org/info/rfc9000 (https://www.rfc-editor.org/info/rfc9000).

[RFC3168] Ramakrishnan, K., et al., "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, https://www.rfc-editor.org/info/rfc3168 (https://www.rfc-editor.org/info/rfc3168).

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, https://www.rfc-editor.org/info/rfc4821 (https://www.rfc-editor.org/info/rfc4821).

[RFC8312] Rhee, I., et al., "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, https://www.rfc-editor.org/info/rfc8312 (https://www.rfc-editor.org/info/rfc8312).

25. Appendix A. Wire Image Examples (Hex + Field Decode)

Example: minimal INITIAL packet (32-byte header only; payload and tag omitted for brevity). Header layout per Section 5.2 (big-endian).

* ver_type = 0x10 (version 1, type INITIAL=0)
* flags = 0x00
* hlen_words = 8
* reserved = 0
* epoch = 0x0000

* payload_len = 0x0029 (41 bytes: client pubkey 32 + CID 8 + version 1)
* stream_id = 0
* reserved2 = 0
* pn = 0x00000000
* dcid = 0x0123456789ABCDEF (example client CID)
* scid = 0x0000000000000000 (zero in INITIAL)
* hdr_ext = 0x0000

Hex (32 bytes): 10 00 08 00 00 00 00 29 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00 00 00 00 00 00 00 00 00

A 1RTT packet carrying encrypted payload would have ver_type = 0x13 (version 1, type 1RTT=3), non-zero pn, and dcid/scid set to receiver/sender CIDs; the payload would contain TLV frames (e.g., type 1 STREAM, length, value) followed by the 16-byte AEAD tag.

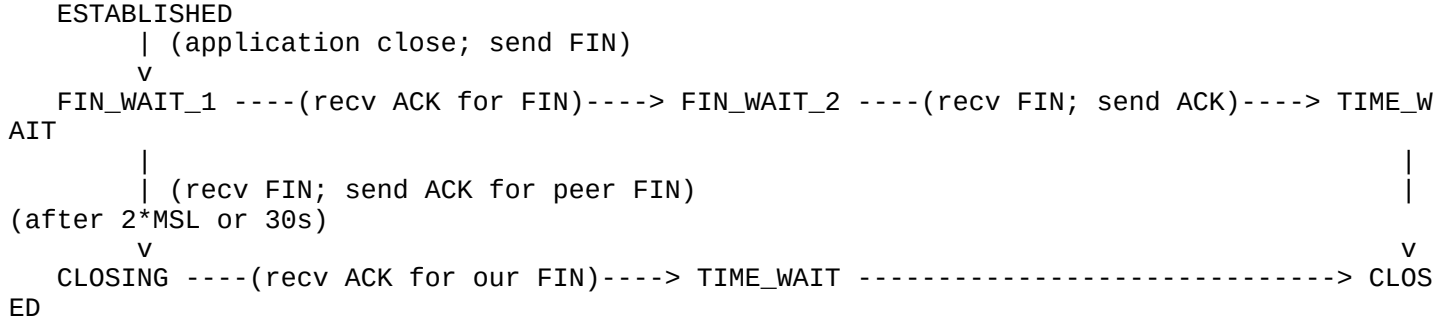
26. Appendix B. State Machines (Full)

Handshake (simplified):

* *Client:* CLOSED -> (send INITIAL) -> INITIAL_SENT -> (recv RETRY -> resend INITIAL with token) -> (recv SIN_ACK) -> SIN_ACK_RCVD -> (send CONFIRM) -> (recv 1RTT) -> ESTABLISHED.
* *Server:* LISTEN -> (recv INITIAL) -> INITIAL_RCVD -> (optional: send RETRY) -> (send SIN_ACK) -> SIN_ACK_SENT -> (recv CONFIRM) -> ESTABLISHED.

Connection close and TIME_WAIT (normative): The endpoint that initiates close sends a packet with the FIN flag (or CONNECTION_CLOSE frame) and enters FIN_WAIT_1. When it receives an ACK for the FIN, it enters FIN_WAIT_2 (if it may still receive data) or CLOSING (if both sides have sent FIN). When the peer's FIN is acknowledged, the endpoint enters TIME_WAIT. The TIME_WAIT duration MUST be at least 2*MSL or 30 seconds, whichever is greater; during this time the endpoint MUST NOT allow a new connection to reuse the same (4-tuple, dcid). After TIME_WAIT expires, the connection enters CLOSED.

Connection-close state diagram:



27. Appendix C. Test Vectors (Crypto)

X25519 key exchange, HKDF key derivation, and AEAD (e.g., ChaCha20-Poly1305 or AES-256-GCM) test vectors for SIN/IP v1 are to be provided in a companion document or a future revision. Implementations should derive keys as specified in Section 9 (Key Schedule): shared secret from X25519 -> HKDF-SHA256 with info "SINIPv1 Key Material" -> 56 bytes (32-byte key, 12-byte client send IV, 12-byte server send IV). Nonce = IV XOR (0x00000000 || PN) with PN in the low 32 bits (big-endian). AEAD authenticated data is the serialized 32-byte header.

28. Appendix D. Design Notes (Non-Normative)

SIN/IP is designed for environments where kernel control and transport evolution matter: datacenters, enterprise edges, and gateway deployments. The same wire format is used for native IP and UDP encapsulation to simplify implementation and testing. Connection IDs and path validation enable mobility without middlebox changes. The Bridge (gateway) component, which terminates SIN/IP and proxies to TCP/UDP, is specified separately and allows incremental deployment toward existing services.

29. Acknowledgments

Thanks to the reviewers and contributors who provided feedback on earlier versions of this document.

30. Authors' Addresses

{: align="left"} Rick Collette VextCODE Email: rcollet@gmail.com

31. Building This Draft

This draft is written in mmark (RFC-style markdown). To produce the canonical .txt and .html for submission:

* Install mmark (https://github.com/mmarkdown/mmark (https://github.com/mmarkdown/mmark)) and xml2rfc (https://pypi.org/project/xml2rfc/ (https://pypi.org/project/xml2rfc/)).
* Run: mmark draft-collette-sinip-transport-00.md > draft-collette-sinip-transport-00.xml
* Run: ./fix-idnits-xml.sh draft-collette-sinip-transport-00.xml (adds date, BCP 14 refs, and ENTITY declarations so idnits passes)
* Run: xml2rfc draft-collette-sinip-transport-00.xml --text --html

Alternatively, use the IETF draft submission tool (https://datatracker.ietf.org/submit/ (https://datatracker.ietf.org/submit/)) to upload the .md or .xml and let it generate the .txt output.

32. Normative References (BCP 14)

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

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

Author’s Address

Rick Collette
VextCODE
Email: rcollet@gmail.com