Network Working Group
Internet-Draft
Expires November 11, 2025

T. Lancaster
Independent

May 10, 2025

XRUDP
eXtended Reliable UDP Protocol Specification
draft-xrudp-specification-00

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."

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Abstract

XRUDP (eXtended Reliable UDP) is a modern UDP-based transport protocol providing reliable, in-order, message-oriented delivery with congestion control, security, NAT traversal, and extensibility. It is designed for scenarios demanding reliable and efficient message delivery over unreliable networks, such as real-time signaling, IoT, gaming, and edge computing. This document specifies the XRUDP packet format, state machine, negotiation, and interaction with applications.

Table of Contents

1. Introduction

XRUDP is designed to provide a transport layer suitable for modern internet applications that require more control and performance characteristics than TCP, but need more reliability and features than raw UDP. It is built on top of UDP to leverage its firewall and NAT compatibility, while adding essential transport features like reliability, ordered delivery (for DATA and STREAM packets), congestion control, security, and efficient message delineation.

The protocol supports reliable message delivery, in-order transmission (per stream), congestion control mechanisms (pluggable and negotiated), built-in NAT traversal capabilities, mandatory security (unless explicitly negotiated otherwise), and optional advanced features like multipath and multistream operation.

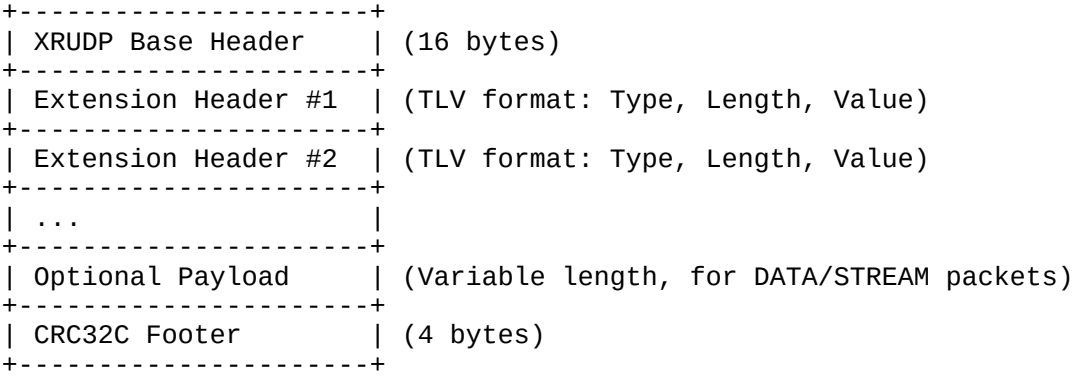
XRUDP aims to offer a balance between the simplicity of UDP and the feature richness of protocols like QUIC and SCTP, with a focus on flexibility and extensibility through its TLV-based header format and negotiation mechanisms.

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 they appear in all caps, as shown here.

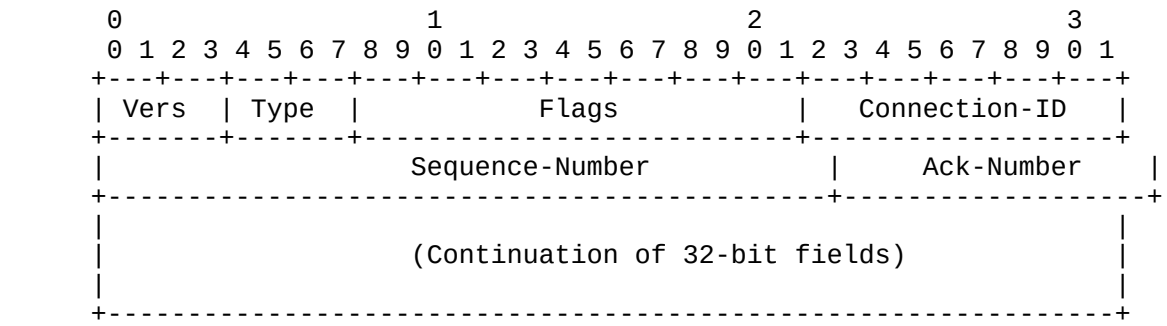
2. XRUDP Packet Structure

XRUDP packets are self-contained units encapsulated within UDP datagrams. A single UDP datagram MAY contain multiple XRUDP packets, but this is not RECOMMENDED for initial implementations and requires careful handling of packet boundaries and CRC calculation. Typically, one UDP datagram contains one XRUDP packet.

Each XRUDP packet is structured as follows:



XRUDP Base Header (16 bytes, Network Byte Order):



Field descriptions:

- Version (1 byte): The protocol version. This specification defines version 1 (0x01). Incompatible versions MUST result in connection failure (e.g., via a RESET packet with a Version Mismatch error).
- Type (1 byte): The packet type, defining the packet's primary purpose (see Section 3).
- Flags (2 bytes): A set of bit flags that modify the interpretation or indicate characteristics of the packet (see Section 4).
- Connection-ID (4 bytes): A unique identifier for the XRUDP connection. The client initiates this ID in the INIT packet. The server responds with the same client-chosen ID and its own server-chosen ID in the INIT-ACK. All subsequent packets from the client to the server use the server's Connection-ID, and all subsequent packets from the server to the client use the client's Connection-ID. This allows both peers to identify the connection quickly upon receiving a packet. IDs are unique per peer endpoint.
- Sequence-Number (4 bytes): A 32-bit monotonically increasing number assigned by the sender for packets requiring reliability (DATA, STREAM, possibly others as defined by extensions). The sequence number space wraps around at 2^32. Senders MUST handle wrap-around gracefully, maintaining sufficient state to distinguish between old and new sequence numbers within the active window. For non-reliable packets (e.g., ACK, HEARTBEAT), this field MAY carry a different sequence, be set to 0, or used for a packet-specific purpose, as defined by the packet type.
- Ack-Number (4 bytes): A 32-bit cumulative acknowledgment number. This indicates the sequence number of the highest in-order reliable packet received by the sender of this packet. Packets with Sequence-Number less than or equal to Ack-Number are considered acknowledged.

Extension Headers (TLV Format):

Extension headers provide flexibility to add optional parameters, control information, or data to any packet type.

- Ext-Type (2 bytes): Identifier for the extension type (see Section 10). Extensions are typically negotiated during the handshake. The upper bit MAY be used to indicate criticality (e.g., MSB = 1 for critical). The mechanism for criticality MUST be negotiated or defined in the base protocol.
- Ext-Length (2 bytes): The length of the Ext-Value field in bytes.
- Ext-Value (variable): The content of the extension. Its format and meaning are defined by the Ext-Type. Multiple extension headers can be chained after the base header.

Payload:
Present only in DATA and STREAM packet types, and potentially in other packet types defined by extensions (e.g., EXT). It carries the application-layer data or extension-specific large data blocks.

Footer:
- CRC32C (4 bytes): A Cyclic Redundancy Check using the Castagnoli polynomial (0x1EDC6F41). The checksum is calculated over the entire XRUDP packet, *including* the base header, extension headers, and payload, but *excluding* the 4 bytes of the CRC32C field itself (these bytes are treated as zero during calculation). This provides protection against corruption of the packet contents. Implementations MUST verify the CRC32C upon reception and discard packets with invalid checksums.

3. Packet Types

XRUDP defines the following base packet types. Additional types MAY be defined by extensions.

Type	Name	Description
0x01	INIT	Initiate a new connection. Carries negotiation parameters (versions, features, security profiles, extensions, initial window sizes, etc.) and the client’s proposed Connection-ID.
0x02	INIT-ACK	Response to INIT. Indicates acceptance or modification of parameters, carries the server’s Connection-ID, and confirms negotiated settings.
0x03	DATA	Carries reliable, in-order application data for the main connection stream (stream 0). Includes a Sequence-Number for reliability tracking.
0x04	ACK	Acknowledges received reliable packets. Contains the cumulative Ack-Number and MAY include Selective Acknowledgement (SACK) blocks as an extension header to indicate receipt of out-of-order packets.

0x05	NACK	Negative acknowledgement. Explicitly requests retransmission of one or more specific packets, identified by sequence numbers or ranges, typically using an extension header. NACKs are OPTIONAL and MAY be used alongside or instead of timeout-based retransmissions.
0x06	HEARTBEAT	A lightweight packet exchanged periodically to maintain NAT/firewall mappings and measure RTT. MAY carry minimal state or sequence numbers for liveness checks.
0x07	CLOSE	Initiates a graceful connection shutdown. Indicates the sender will send no more application data.
0x08	CLOSE-ACK	Confirms receipt of a CLOSE packet and indicates the sender will send no more application data in response. Follows a CLOSE packet to complete the graceful shutdown sequence.
0x09	RESET	Abruptly terminates a connection. Carries a Status Code extension to indicate the reason for termination (see Section 11). No further packets for this connection are expected.
0x0A	STREAM	Carries reliable, in-order application data for a logical substream. Requires a STREAM-ID extension header and carries a stream-local sequence number or offset for ordering within that stream.
0x0B	PATH-PROBE	Used in multipath operation (Section 7) to probe the viability and characteristics of a potential or alternative network path.
0x0C	PATH-ACK	Response to a PATH-PROBE, confirming reachability and potentially carrying path metrics.
0x0D	EXT	A packet type reserved for extension-defined protocol messages. This packet type’s content consists entirely of one or more extension headers and potentially an extension-defined payload.

4. Packet Flags

XRUDP packet flags (2 bytes, 16 bits) modify packet interpretation or indicate characteristics of the packet. Flags are carried in the Base Header.

Bit (MSB 0)	Name	Description
0	ENCRYPTED	If set, the payload (and potentially certain extension values) of this packet is encrypted using the negotiated session key. This flag MUST be set for DATA, STREAM, and any other packet types carrying application data or sensitive control information once security is established.

1	AUTH	If set, the packet includes an authentication tag (e.g., MAC) appended after the payload or extension headers, calculated over the base header, all extension headers, and the payload. This flag MUST be set if authenticated encryption or authentication is negotiated.
2	COMPRESSED	If set, the payload (and possibly some extension values) of this packet has been compressed using a negotiated algorithm. Decompression MUST be applied before interpretation of the payload.
3	FEC	If set, this packet contains Forward Error Correction parity or repair data, or is an FEC-protected data packet. The specific FEC scheme is negotiated and indicated by an extension.
4	MULTIPATH	If set, this packet is being sent over a non-primary or alternative network path as part of a multipath connection (Section 7). Requires a PATH-ID extension.
5	MULTISTREAM	If set, this packet belongs to a logical substream within the connection (Section 7). This flag MUST be set for STREAM packets and for any other packets (e.g., ACK, NACK) that carry stream-specific information. Requires a STREAM-ID extension.
6	ECN	Explicit Congestion Notification. The usage of this flag and its interpretation (e.g., ECN-CE) follows standard IP ECN conventions. Peers MUST negotiate support for ECN.
7	PRIORITY	If set, this packet is considered high-priority. Implementations MAY use this flag for scheduling or queueing decisions. The specific meaning is application-defined but could influence retransmission timers or path selection in multipath.
8-15	Reserved	These bits are reserved for future use. Senders MUST set them to zero. Receivers MUST ignore them.

5. Connection Management

XRUDP connection establishment is a three-way handshake designed to negotiate parameters and establish shared state, including security keys.

1. **Initiation (Client to Server):** The client sends an **INIT** packet to the server’s well-known or negotiated UDP endpoint. The INIT packet includes:
   * Client’s proposed Connection-ID.

* Supported XRUDP versions.
* Proposed transport parameters (e.g., initial window size, timer values, max packet size).
* Proposed security parameters (e.g., supported cipher suites, key exchange methods, ephemeral public key).
* List of supported and requested extensions, including parameters for those extensions.

2. **Response (Server to Client):** The server processes the INIT packet. If acceptable, it responds with an **INIT-ACK** packet to the client's source IP/port. The INIT-ACK packet includes:
* The client's original Connection-ID (from the INIT).
* The server's chosen Connection-ID.
* The selected XRUDP version.
* Confirmed or modified transport parameters.
* Selected security parameters, including the server's ephemeral public key and potentially key material derived from the client's key.
* List of accepted extensions and their negotiated parameters.
* If the INIT is not acceptable, the server MAY send a RESET with an appropriate error code (e.g., Protocol Version Mismatch, Parameter Negotiation Failed, Encryption Required).

3. **Confirmation (Client to Server):** Upon receiving a valid INIT-ACK, the client processes the server's response, derives the session keys using the negotiated parameters, and establishes the connection state. The client sends an **ACK** packet to the server using the server's Connection-ID. This ACK confirms the client is ready and implicitly acknowledges the INIT-ACK. This ACK packet MUST be authenticated if security is negotiated. From this point, both peers SHOULD use the negotiated Connection-IDs and security parameters for all subsequent packets.

Connection State Machine: Implementations manage connections using a state machine that transitions based on received packets and internal events (timeouts, application calls). Typical states include:
* CLOSED: No active connection state.
* LISTEN: Waiting for incoming INIT packets.
* SYN-SENT: Client has sent INIT, waiting for INIT-ACK.
* SYN-RECEIVED: Server has received INIT, sent INIT-ACK, waiting for client ACK.
* ESTABLISHED: Handshake complete, data can be exchanged.
* FIN-WAIT-1: Application initiated graceful close, sent CLOSE.
* FIN-WAIT-2: Received ACK for CLOSE, waiting for peer's CLOSE.
* CLOSE-WAIT: Received peer's CLOSE, application hasn't closed yet.
* LAST-ACK: Application closed, sent CLOSE-ACK, waiting for ACK.
* CLOSING: Sent CLOSE, received CLOSE, waiting for ACK for sent CLOSE.
* TIME-WAIT: Sent final ACK in certain close sequences, waiting for a period to ensure packet loss doesn't leave peer in a confused state.
* RESETTING: Sent or received a RESET packet.

Graceful Shutdown: Initiated by sending a **CLOSE** packet. The peer responds with a **CLOSE-ACK**. After sending CLOSE, no new application data is sent on reliable streams. After receiving CLOSE-ACK, the connection can transition towards a final state. Both sides MUST ensure all in-flight reliable data is delivered and acknowledged before fully closing. Streams MUST be closed before the connection.

Abortive Termination: Initiated by sending a **RESET** packet. This immediately tears down the connection state at the sender’s side. Upon receiving a RESET, the receiver MUST immediately tear down its connection state associated with the Connection-ID in the packet. RESET packets are not reliable and do not require acknowledgment.

6. Reliability, Flow, and Congestion Control

XRUDP provides reliable, ordered delivery for DATA packets (the main connection stream, conceptually stream 0) and STREAM packets (logical substreams).

Sequence Numbers: Reliable packets (DATA, STREAM) are assigned a unique, 32-bit Sequence-Number by the sender within the scope they apply to (connection-wide for DATA, stream-local for STREAM). These numbers increase monotonically and wrap around. Senders MUST handle wrap-around gracefully, maintaining sufficient state to distinguish between old and new sequence numbers within the active window. For non-reliable packets (e.g., ACK, HEARTBEAT), this field MAY carry a different sequence, be set to 0, or used for a packet-specific purpose, as defined by the packet type.

Acknowledgments (ACK): Receivers use the **ACK** packet type to inform the sender about received reliable packets. The base header’s Ack-Number provides a cumulative acknowledgment, indicating that the receiver has successfully received all reliable packets with Sequence-Number up to and including Ack-Number, *in order*. To acknowledge out-of-order packets, receivers SHOULD include a **SACK (Selective Acknowledgment) extension header** in ACK packets. A SACK extension contains blocks specifying contiguous ranges of sequence numbers received *beyond* the cumulative Ack-Number. Senders use cumulative ACKs and SACKs to determine which packets need retransmission.

Negative Acknowledgments (NACK): Receivers MAY send **NACK** packets to explicitly request retransmission of missing packets. NACKs are typically sent for sequence numbers below the highest received sequence number but higher than the cumulative Ack-Number. A NACK extension header identifies the specific missing packets or ranges. NACKs can supplement or accelerate loss recovery compared to relying solely on retransmission timeouts.

Retransmission: Senders maintain a retransmission queue for sent reliable packets. Packets are removed from the queue upon cumulative acknowledgment or SACK indicating reception. If a packet is not acknowledged within a Retransmission Timeout (RTO), the sender SHOULD

retransmit it. RTO calculation SHOULD use methods similar to TCP, based on Round-Trip Time (RTT) measurements derived from the exchange of data packets and their corresponding ACKs.

Flow Control: XRUDP uses a credit-based or sliding window flow control mechanism to prevent a sender from overwhelming a receiver's buffer. The receiver advertises its available buffer space as a "receive window". This window advertisement is typically carried in the **ACK** packet (e.g., in a flow control extension) and potentially in the INIT-ACK. The sender MUST NOT send reliable data that would cause the unacknowledged data to exceed the receiver's advertised window. Flow control is managed independently per stream (including the main data stream 0).

Congestion Control: XRUDP includes congestion control to adapt to network capacity and prevent congestion collapse. Congestion control algorithms are negotiated during the handshake. Common algorithms like TCP RENO, CUBIC, or BBR MAY be supported. The congestion control mechanism operates based on acknowledgments (ACKs, SACKs), packet loss detection (timeouts, duplicate ACKs, NACKs), and potentially ECN signals. The sender maintains a congestion window, limiting the amount of in-flight data, independent of the receiver's flow control window. The effective sending window is the minimum of the flow control window and the congestion window.

ECN (Explicit Congestion Notification): If negotiated, XRUDP endpoints SHOULD set the ECN flag (Section 4) in their packets when they receive IP packets marked with Congestion Experienced (CE). Receivers that detect ECN-CE marks in incoming packets MUST signal this information back to the sender (e.g., via an ECN-specific extension in ACK packets) to allow the congestion control algorithm to react.

7. Multipath and Multistream

XRUDP is designed with optional support for Multipath operation and Multistreaming within a single connection.

Multipath: XRUDP MAY establish and utilize multiple distinct network paths simultaneously for a single connection. Each path is identified by a unique **PATH-ID extension** (e.g., a 4-byte identifier assigned during path setup).

* Path Discovery: Paths can be discovered via explicit signaling (e.g., in INIT/INIT-ACK with extensions carrying potential addresses/ports, or using ICE/STUN as per Section 9) or implicitly by receiving packets from a new source address/port pair.
* Path Validation: New paths MUST be validated using **PATH-PROBE** and **PATH-ACK** packets before being used for regular data transfer. PATH-PROBE packets are sent on the potential new path, and a successful PATH-ACK response confirms bidirectional reachability. PATH-PROBE/ACK packets carry the Connection-ID and the proposed PATH-ID extension.

* Path Management: Endpoints monitor path quality (RTT, loss, jitter) and use this information for scheduling packets across paths. Packet reordering across paths must be handled by the receiver's reliability layer. The MULTIPATH flag (Section 4) indicates a packet is sent on a non-primary path.
* Multipath Scheduling: Policies for deciding which packets to send on which path are implementation-defined but can include load balancing, redundancy for critical data, or using the best path based on current metrics. A PATH-ID extension header in regular data packets indicates which path the packet was sent on by the sender.

Multistream: XRUDP supports creating multiple independent, reliable, in-order, bidirectional logical streams within a single connection. This allows applications to manage concurrent data flows without head-of-line blocking between streams.

* Stream Identification: Each stream is identified by a unique **STREAM-ID extension** (e.g., a 32-bit or 64-bit identifier, negotiated or assigned dynamically). The MULTISTREAM flag (Section 4) MUST be set on packets belonging to a specific stream.
* Stream Lifecycle: Streams can be opened and closed independently of the main connection. A stream can be initiated by either peer sending a packet with a new STREAM-ID. A stream is closed when both peers have sent and received all data for that stream and exchanged appropriate stream-level close signals (e.g., flags or extension headers within STREAM packets indicating end-of-stream).
* Stream Reliability & Flow Control: Each stream maintains its own sequence number space (local to the stream) and flow control window. Reliability (retransmission, acknowledgment) and flow control (window advertisement) are handled independently for each stream. ACK and NACK packets MAY include stream-specific acknowledgment information using extensions (e.g., SACK blocks with STREAM-ID).
* Stream Prioritization: Streams MAY be assigned priorities, influencing how packets from different streams are scheduled for transmission on the available network path(s).

8. Security

Security is a fundamental aspect of XRUDP. Implementations MUST negotiate security parameters during the handshake and apply authenticated encryption to protect data in transit, unless an explicit, less-secure profile is mutually agreed upon (which is NOT RECOMMENDED for most use cases).

Security Negotiation: Security parameters are negotiated within the INIT and INIT-ACK packets using dedicated extension headers. This negotiation includes:

* Protocol Version: Indicating support for the security features defined in the XRUDP version.

* Key Exchange Method: REQUIRED to provide forward secrecy. RECOMMENDED methods include ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) or X25519. Ephemeral public keys are exchanged in the handshake extensions.
* Cipher Suite: Negotiating the authenticated encryption algorithm (AEAD) and hashing function. RECOMMENDED cipher suites include AES-128-GCM, AES-256-GCM, and ChaCha20-Poly1305.
* Authentication: Negotiation of methods for authenticating the endpoints themselves (e.g., using pre-shared keys, certificates, or relying solely on the security of the underlying channel or application-layer authentication).
* Extensions: Security-related extensions (e.g., for embedding a full TLS 1.3 handshake) are also negotiated.

Key Derivation: After the key exchange (e.g., ECDHE), both endpoints derive a shared secret. This secret is used with a Key Derivation Function (KDF) (e.g., HKDF) and parameters from the handshake (e.g., Connection-IDs, nonces exchanged in extensions) to generate session keys for encryption and authentication in both directions. Key updates MAY be performed periodically or after a certain amount of data has been exchanged using a rekeying mechanism (via extensions).

Packet Protection:
* Confidentiality: Application data payloads and sensitive extension values MUST be encrypted using the negotiated AEAD cipher and session keys when the ENCRYPTED flag is set.
* Integrity and Authenticity: An authentication tag (MAC) MUST be calculated over the base header, all extension headers (in their plaintext form, though some values may be encrypted), and the encrypted payload. This tag is appended to the packet, and the AUTH flag MUST be set. Receivers MUST verify the tag before processing the packet. Packets with invalid tags MUST be silently discarded.
* Associated Data: Fields in the base header and extension headers that are *not* encrypted (e.g., Version, Type, Connection-ID, Sequence-Number, Ack-Number, Ext-Type, Ext-Length) are treated as Additional Authenticated Data (AAD) in the AEAD operation.

Replay Protection: Receivers MUST implement replay protection by keeping track of received valid Sequence-Numbers for a given Connection-ID (and Path-ID/Stream-ID if applicable). A sliding window or similar mechanism SHOULD be used to efficiently detect and discard replayed packets.

TLS 1.3 Integration: For complex authentication or negotiation scenarios, a full TLS 1.3 handshake MAY be encapsulated within XRUDP extension packets (e.g., using a dedicated TLS extension type). In this mode, XRUDP’s handshake might be minimal, primarily establishing the UDP connection and negotiating the TLS extension, with TLS then handling authentication, key exchange, and potentially even application protocol negotiation (ALPN).

9. NAT Traversal and Keepalive

XRUDP is designed to facilitate operation through Network Address Translators (NATs) and firewalls, which often have stateful mappings that time out if no traffic is seen on a UDP flow.

Keepalive (HEARTBEAT): Endpoints SHOULD exchange **HEARTBEAT** packets periodically to keep the NAT/firewall binding state alive in both directions. The interval for HEARTBEATs is negotiated during the handshake (e.g., as a transport parameter extension). A typical interval might be significantly shorter than common NAT timeout values (e.g., every 10-30 seconds). HEARTBEAT packets are small, have no payload, and their primary purpose is to generate traffic on the UDP socket. They MAY include a sequence number or timestamp to assist with RTT measurement but are generally not part of the reliable delivery stream.

Initial NAT Traversal (ICE/STUN): For more complex NAT scenarios (e.g., Symmetric NATs), XRUDP MAY leverage standard NAT traversal techniques like Interactive Connectivity Establishment (ICE) and STUN/TURN. Information required for ICE/STUN (e.g., candidate addresses, STUN/TURN server addresses) MAY be exchanged during the XRUDP handshake within dedicated extension headers in the INIT and INIT-ACK packets. This allows XRUDP to discover and potentially use the public IP address and port mapped by the NAT.

Address and Port Changes: If a peer’s source IP address or UDP port changes during the connection lifetime (which can happen with some NATs or mobility events), the receiving XRUDP endpoint SHOULD recognize the packet as belonging to an existing connection if the Connection-ID matches. New source/port pairs MAY trigger a path validation process (Section 7) in multipath-enabled connections or update the known address for a single-path connection.

10. Extensions and Negotiation

XRUDP’s extensibility is primarily enabled by the **Extension Header** mechanism (Section 2) and the **Negotiation** process during the handshake (Section 5).

Extension Header Format: As detailed in Section 2, extensions use the TLV format: ‘Ext-Type‘ (2 bytes), ‘Ext-Length‘ (2 bytes), ‘Ext-Value‘ (variable).

Extension Negotiation during Handshake:

* INIT Packet: The client includes an extension negotiation extension in its INIT packet. This extension contains a list of the extension types it supports and wishes to use, potentially along with initial parameters or preferences for each.

* INIT-ACK Packet: The server processes the client's list and includes an extension negotiation extension in its INIT-ACK packet. This list indicates which of the client's proposed extensions are accepted, potentially with modified parameters or the server's own parameters. The server MAY also propose extensions not requested by the client, which the client would then accept or reject in its final ACK (or subsequently via a dedicated extension update mechanism).
* Parameter Negotiation: For extensions that require negotiation (e.g., congestion control algorithm, FEC parameters), the 'Ext-Value' in the negotiation extensions (within INIT/INIT-ACK) carries the proposed and accepted parameters.

Extension Criticality: To ensure interoperability and enable future evolution, extensions are categorized as either critical or non-critical.
* Non-Critical Extensions: An implementation that receives a packet containing a non-critical extension header type it does not recognize MUST ignore that extension header and continue processing the rest of the packet. This allows new non-critical features to be added without breaking compatibility with older implementations.
* Critical Extensions: An implementation that receives a packet containing a critical extension header type it does not recognize cannot safely process the packet or continue the connection, as the extension is essential to the packet's or connection's meaning or security. Such an implementation MUST terminate the connection, typically by sending a **RESET** packet with the 'Unsupported Extension' error code (Section 11).
* Signaling Criticality: The mechanism for signaling whether an extension is critical MUST be clearly defined. As specified in Section 2, the most significant bit (MSB) of the 'Ext-Type' field MAY be used for this purpose (e.g., MSB=1 for critical, MSB=0 for non-critical). This convention MUST be established by the base protocol version.

Common Extension Categories: The document anticipates extensions for:
* Reliability/Flow Control: SACK blocks, NACK lists, Window updates.
* Congestion Control: ECN feedback, specific algorithm parameters.
* Multipath: PATH-ID, Path metrics, Path validation challenges/responses.
* Multistream: STREAM-ID, Stream flow control updates, Stream state signaling (open, close, reset).
* Security: Public keys, nonces, cipher suite preferences, TLS handshake data.
* NAT Traversal: ICE/STUN attributes.
* Application Signaling: Application-specific control messages or metadata that need reliable or ordered delivery but are not the main data payload.

The **EXT** packet type (Section 3) is available for future extension protocols that may not fit neatly into existing packet types or require a sequence of packets solely comprising extension data.

11. Error Handling and Status Codes

XRUDP connections can terminate gracefully (CLOSE/CLOSE-ACK) or abortively (RESET). The **RESET** packet is the primary mechanism for signaling errors or abnormal termination.

When an endpoint encounters a fatal error, protocol violation, negotiation failure, or resource issue that prevents the connection from continuing, it SHOULD send a **RESET** packet to the peer. The RESET packet carries a Status Code extension header to provide information about the reason for termination.

Status Code Extension Header:
* ‘Ext-Type‘: The designated type for the Status Code extension.
* ‘Ext-Length‘: Typically 2 bytes (for the Code) or more if additional diagnostic data is included.
* ‘Ext-Value‘: Contains a 2-byte Status Code (in network byte order) and MAY be followed by optional diagnostic data (e.g., a text string, error details).

Common error/status codes for the Status Code extension (2 bytes):

Code (Hex)	Meaning	Description
0x0000	No Error	Used in successful closing, or for indicating intent without an actual error.
0x0001	Protocol Version Mismatch	Unsupported XRUDP version.
0x0002	Parameter Negotiation Failed	Peers could not agree on required parameters.
0x0003	Authentication Failed	Security authentication process failed.
0x0004	Encryption Required	Peer refused to negotiate or use encryption when required.
0x0005	Congestion Control Required	Peer refused to negotiate or use congestion control.
0x0006	Flow Control Violation	Peer sent data exceeding the advertised window.
0x0007	Timeout	Connection timed out due to lack of activity or acknowledgments.
0x0008	Resource Exhausted	Endpoint ran out of resources (memory, sockets, etc.).
0x0009	Unsupported Extension	Peer received a critical extension it doesn’t support.
0x000A	Stream Error	An error occurred on a specific stream (details MAY be in diagnostic data).
0x000B	Application Error	Application-specific error caused connection termination.
0xFFFE	Internal Error	Unexpected internal error.
0xFFFF	Unknown Error	Error reason is not specified

or mapped to a known code.

Implementations SHOULD log received error codes for debugging. Specific streams MAY also have their own error states or error codes communicated via stream-specific extension headers before the stream is closed or reset.

12. IANA Considerations

This document requires the assignment of several identifiers by IANA.

1. **XRUDP UDP Port Number:** This document requests the assignment of a well-known UDP port number for XRUDP traffic. The suggested port number is 4444.

2. **XRUDP Packet Type Registry:** This document requests the creation of a registry for XRUDP Packet Types. The registry SHALL assign 1-byte values. Initial assignments are listed in Section 3 (0x01-0x0D). Future assignments SHOULD be made using the "Expert Review" policy [RFC8126].

3. **XRUDP Extension Type Registry:** This document requests the creation of a registry for XRUDP Extension Types. The registry SHALL assign 2-byte values (‘Ext-Type‘). The mechanism for signaling criticality (e.g., the MSB of the ‘Ext-Type‘) SHOULD be documented alongside the registry policy. Initial assignments SHOULD include types for SACK, NACK, PATH-ID, STREAM-ID, Status Code, Negotiation Parameters, Security Parameters (key exchange, cipher suite lists), ICE/STUN attributes, and FEC parameters. Future assignments SHOULD be made using the "Expert Review" policy [RFC8126].

4. **XRUDP Status Code Registry:** This document requests the creation of a registry for XRUDP Status Codes. The registry SHALL assign 2-byte values. Initial assignments are listed in Section 11 (0x0000-0x000B, 0xFFFE-0xFFFF). Future assignments SHOULD be made using the "Expert Review" policy [RFC8126]. Ranges MAY be allocated for private or application-specific use upon request (e.g., a range for application error codes).

13. Security Considerations

Implementing XRUDP requires careful attention to security to avoid introducing vulnerabilities.

Confidentiality and Integrity: As specified in Section 8, authenticated encryption using AEAD ciphers is paramount. Implementations MUST use robust, negotiated cipher suites and key exchange methods providing forward secrecy (e.g., ECDHE). Disabling or weakening security SHOULD

only be done based on an explicit mutual agreement documented within the protocol negotiation, and is NOT RECOMMENDED for traffic on untrusted networks. The AUTH flag MUST protect critical header fields, extension headers, and the payload from tampering.

Denial-of-Service (DoS) Attacks:

* Handshake Attacks: The initial handshake (INIT/INIT-ACK) is particularly vulnerable as it requires server state. Implementations SHOULD employ stateless cookies or similar techniques during the initial phase to mitigate SYN-flood style attacks. Rate limiting incoming INIT packets and validating the source address (e.g., using a cookie verified in the client's final ACK) are crucial.
* Resource Exhaustion: Malformed packets, excessive connection attempts, or rapid stream creation attempts can exhaust resources (memory, CPU) on endpoints. Implementations MUST validate packet sizes, header lengths, and extension lengths carefully and apply limits to the number of open connections and streams per connection.
* Amplification Attacks: UDP-based protocols can be exploited for amplification attacks if a small request generates a large response. XRUDP's handshake SHOULD be designed such that the server's INIT-ACK response is no significantly larger than the client's INIT request unless the client's address has been verified. The PATH-PROBE/PATH-ACK exchange SHOULD also be limited in size. Applications using XRUDP MUST ensure their data and control messages do not enable amplification.

Replay Attacks: Sequence numbers and authentication tags provide protection against simple packet replay. Implementations MUST maintain state (e.g., a sliding window of seen sequence numbers per connection/stream/path) to detect and discard replayed packets.

Key Management: Secure generation, storage, and management of long-term keys (if used for endpoint authentication) and ephemeral session keys are critical. Keys MUST reside only in trusted memory and be securely erased after use. Rekeying mechanisms (Section 8) limit the amount of data protected by a single set of session keys.

Privacy: While data payloads are encrypted, the XRUDP base header and some extension headers contain metadata (Connection-ID, Sequence-Number, Type, Flags, potentially STREAM-ID or PATH-ID) that could expose information about connection activity, number of streams, or paths used. Applications sensitive to metadata leakage should be aware of this and potentially use traffic shaping or padding. Encryption of certain extension values containing sensitive data (even if the Type/Length are visible) is RECOMMENDED.

Interoperability: Implementations SHOULD be robust to receiving malformed packets or unsupported non-critical extensions, discarding them without crashing or leaking information.

14. References

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

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

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

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

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol (SCTP)", RFC 4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>. (Note: TLS 1.3 [RFC8446] is RECOMMENDED for encapsulation)

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC5389] Rosenberg, J., Ed., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008, <https://www.rfc-editor.org/info/rfc5389>.

[RFC8839] Batharikishnan, M. and V. Singh, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8839, July 2020, <https://www.rfc-editor.org/info/rfc8839>.

15. Acknowledgements

This document is inspired by previous work on reliable UDP and modern transport protocols, including RUDP, TICP, SCTP, MPTCP, and QUIC. The design aims to incorporate lessons learned from these efforts while providing a flexible base for various use cases. Thanks to all contributors in the IETF transport area, researchers, and developers whose work has paved the way for protocols like XRUDP.

Authors' Addresses

Torin Lancaster
Anonymous
New Zealand

Email: admin@wikiped.me