NSCOM01 MCO1 RFC
REQUEST FOR COMMENTS: 6769
CATEGORY: Experimental

R. Gonzaga
K.L. Chiu
DLSU Manila

February 2026

Reliable Data Transfer over UDP

Abstract

This document specifies an application-layer protocol for reliable data transfer over the User Datagram Protocol (UDP). It defines a simple directly implementable design to support guaranteed, in-order file transfers for both GET (download) and PUT (upload) operations. To lessen the unreliability of UDP, the protocol uses a Stop-and-Wait Automatic Repeat Request (ARQ) mechanism, sequence numbering, and timeout-based retransmissions. Additionally, this specification outlines a standardized 16-byte common header, explicit session establishment and teardown procedures, and strict error-handling rules to manage connection faults and state mismatches.

1. Introduction

This document specifies the core protocol for the MCO1 implementation of reliable data transfer over User Datagram Protocol (UDP). It defines the required message exchanges, packet formats, client and server behavior, reliability mechanisms, and error-handling procedures. The goal is to achieve reliable, in-order file transfer for both GET (download) and PUT (upload) operations while maintaining a simple and directly implementable design.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

3. Protocol Overview

3.1. Session Scope

Each session MUST transfer exactly one file (either a GET or PUT operation). The client MUST initiate a session using a SYN message that specifies the desired operation and filename. After the transfer completes, the session SHOULD be terminated using a FIN/FIN_ACK exchange. A new file transfer MUST start an entirely new session.

3.2. Negotiation and Fixed Default

- Negotiated Parameters: CHUNK_SIZE and the Initial Sequence Number (ISN) MUST be established during the SYN/SYN_ACK handshake.

- Fixed Defaults: The sender MUST use a Retransmission Timeout (RTO) of 500ms and a maximum retransmission limit (MAX_RETR) of 10 attempts.

3.3. GET (Download)

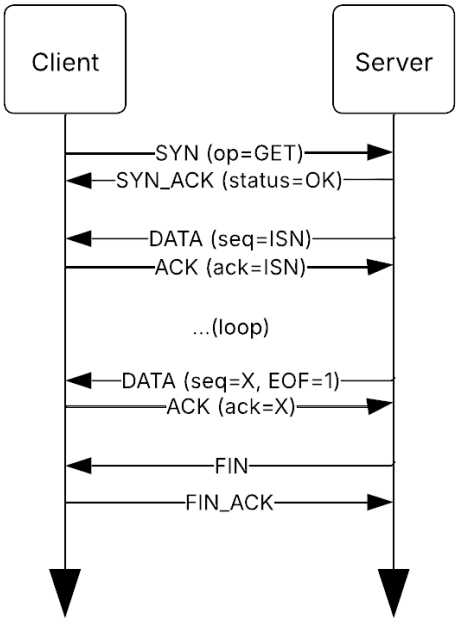

Figure 1. *GET (Download) operation Swimlane Diagram*

3.4. PUT (Upload)

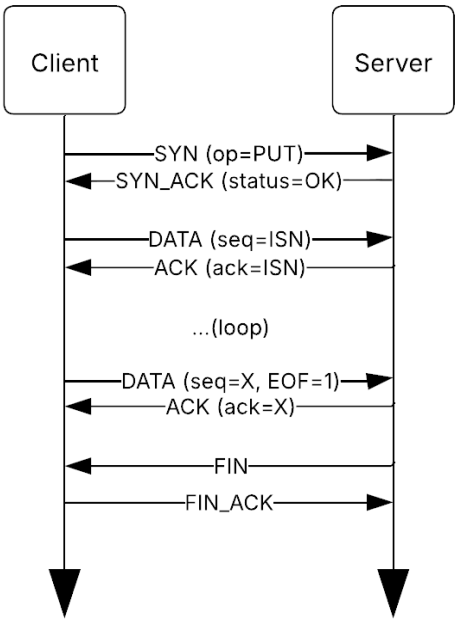

Figure 2. *PUT (Upload) operation Swimlane Diagram*

3.5. FILE_NOT_FOUND (0x01)

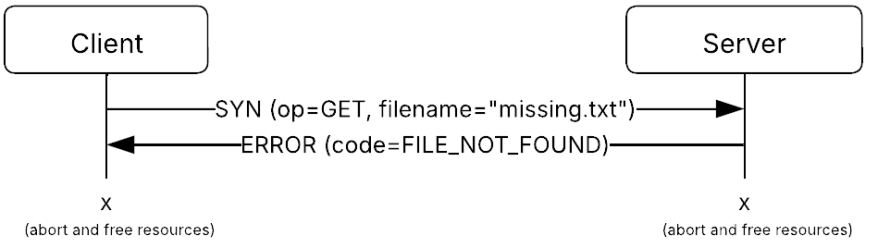

Figure 3. *FILE_NOT_FOUND Error Swimlane Diagram*

3.6. BAD_REQUEST (0x02)

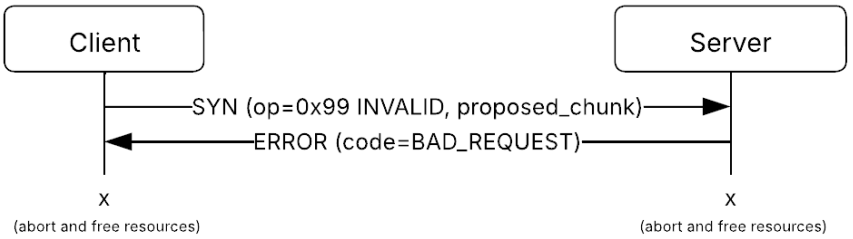

Figure 4. *BAD_REQUEST Error Swimlane Diagram*

3.7. SESSION_MISMATCH (0x03)

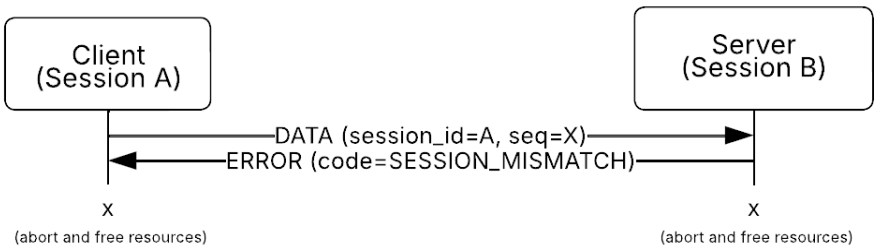

Figure 5. *SESSION_MISMATCH Error Swimlane Diagram*

3.8. TIMEOUT_ABORT (0x04)

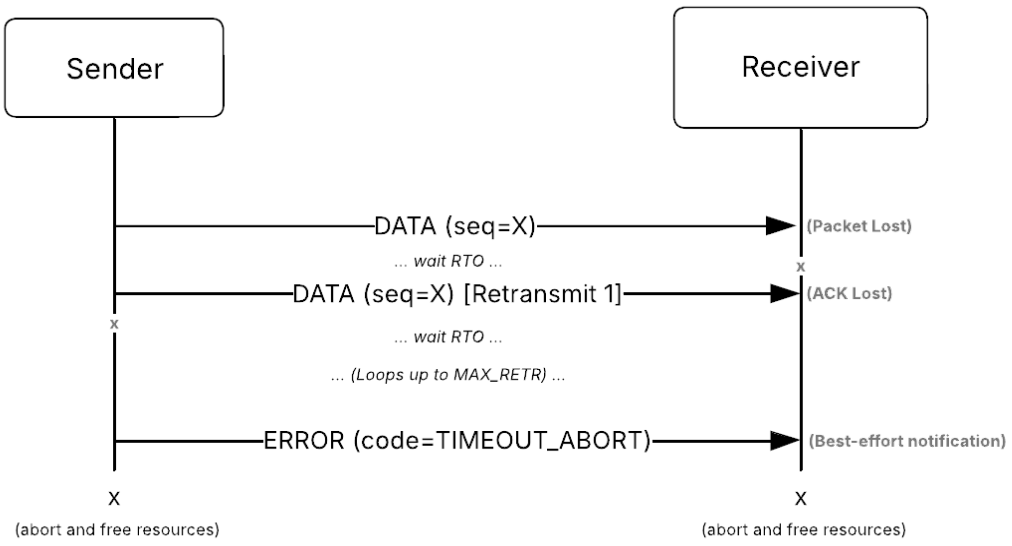

Figure 6. *TIMEOUT_ABORT Error Swimlane Diagram*

3.9. INTERNAL_ERROR (0x05)

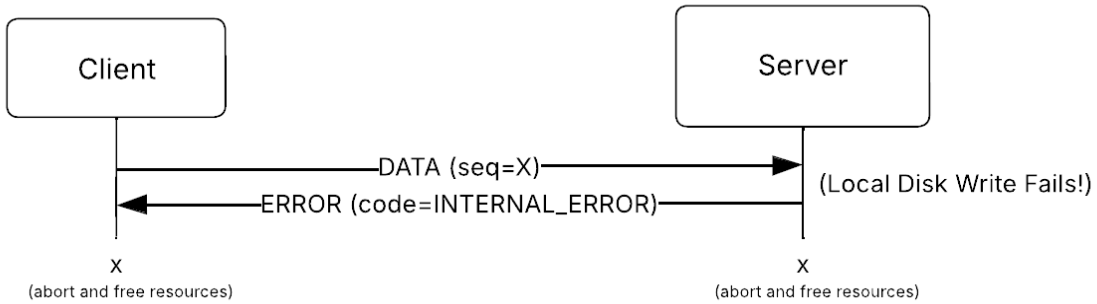

Figure 7. *INTERNAL_ERROR Error Swimlane Diagram*

4. Packet Message Format

4.1. Byte Order

All multi-byte integer fields within the protocol headers and payloads MUST be encoded and transmitted in network byte order (big-endian).

4.2. Common Header Format

Every protocol packet MUST begin with a 16-byte common header. The header structure is defined as follows:

```
  0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     type      |     flags     |     session_id (first 16)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   session_id (last 16 bits)   |        seq (first 16 bits)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        seq (last 16 bits)     |        ack (first 16 bits)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        ack (last 16 bits)     |          payload_len          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- type (1 byte, uint8): Specifies the control or data message type.
- flags (1 byte, uint8): Bitmask. Bit 0 represents the EOF marker. All other bits are reserved and MUST be set to 0.
- session_id (4 bytes, uint32): Unique session identifier.
- seq (4 bytes, uint32): Sequence number for DATA packets.
- ack (4 bytes, uint32): Acknowledgement number.
- payload_len (2 bytes, uint16): Number of payload bytes that immediately follow this header.

4.3. Message Type Values

The 'type' field (uint8) MUST use one of the following values:

- 0x00 = SYN
- 0x01 = SYN_ACK
- 0x02 = ACK
- 0x03 = DATA
- 0x04 = FIN
- 0x05 = FIN_ACK
- 0x06 = ERROR

4.4. Field Semantics

- session_id: In the initial SYN packet, the client MUST set the session_id to 0. The server MUST assign a non-zero session_id and include it in the SYN_ACK header. All subsequent packets for the session MUST carry this assigned session_id.
- seq: This field is used for DATA packets only. Sequence numbering MUST
- start at the negotiated ISN and increment by 1 for each DATA chunk. For non-DATA packets, seq MUST be set to 0.

- ack: This field is used in ACK packets to confirm the successful receipt of DATA. The ack field MUST equal the sequence number of the last in-order DATA packet received. For non-ACK packets, the ack field MUST be set to 0.

- payload_len: The number of payload bytes that immediately follow the header. Control packets without a payload MUST set payload_len to 0.

4.5. Payload Formats

4.5.1. SYN Payload (Client to Server)

- op (1 Byte): 0x00 = GET, 0x01 = PUT
- proposed_chunk_size (2 Bytes, uint16)
- filename_len (1 Byte, uint8)
- filename (filename_len Bytes, UTF-8 encoded)

4.5.2. SYN_ACK Payload (Server to Client)

- status (1 Byte): 0x00 = OK, 0x01 = REJECT
- chosen_chunk_size (2 Bytes, uint16)
- ISN (4 Bytes, uint32)

4.5.3. ACK Payload

- No payload. payload_len MUST be 0.

4.5.4. DATA Payload

- Raw file bytes (payload_len Bytes).
- The sender MUST set flags.EOF = 1 on the final DATA chunk.

4.5.5. FIN / FIN_ACK Payload

- No payload. payload_len MUST be 0.

4.5.6. ERROR Payload

- error_code (1 Byte)
- msg_len (1 Byte)
- msg (msg_len Bytes, UTF-8 encoded)

The following error_code values are defined:

- 0x01 FILE_NOT_FOUND
- 0x02 BAD_REQUEST
- 0x03 SESSION_MISMATCH
- 0x04 TIMEOUT_ABORT
- 0x05 INTERNAL_ERROR

5. State Machines

5.1. Client State Machine

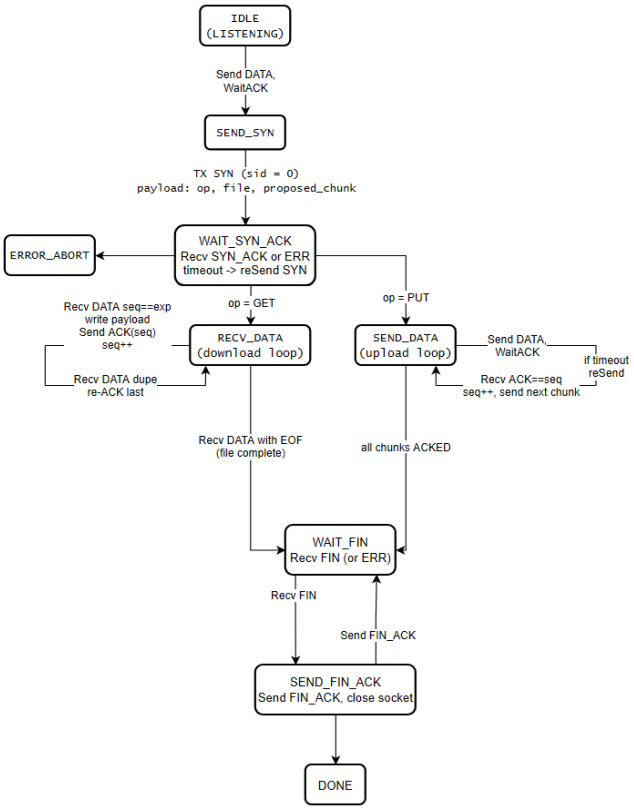

Figure 8. *Client State Machine Diagram*

5.2. Server State Machine

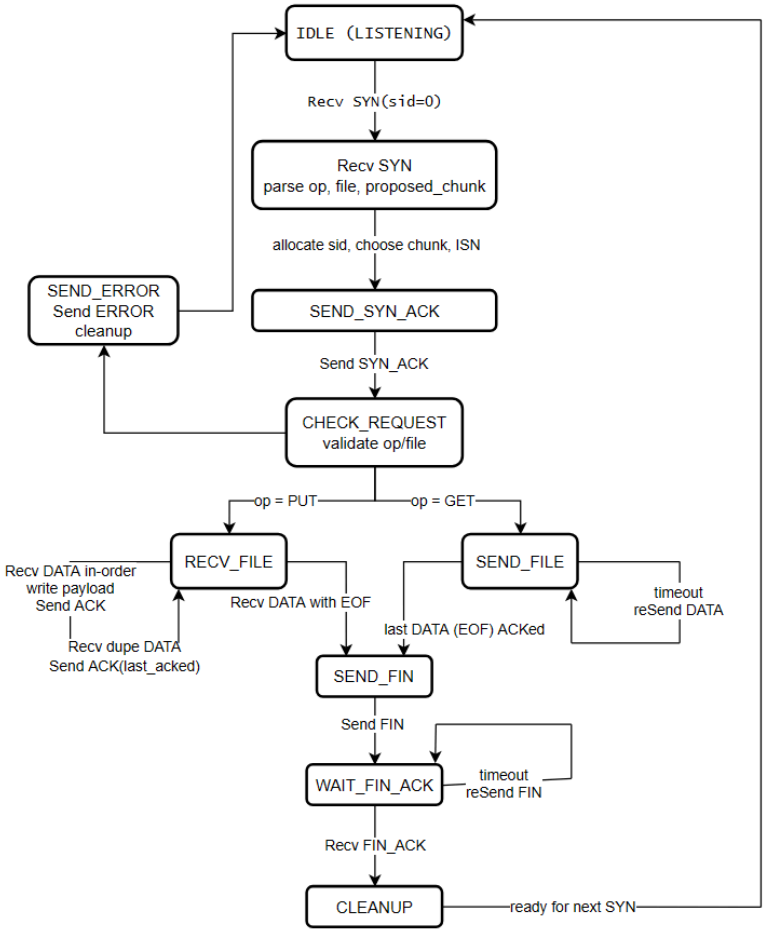

Figure 9. *Server State Machine Diagram*

6. Reliability Mechanisms (Stop and Wait ARQ)

6.1. Core Operation

- The sender MUST transmit exactly one DATA packet at a time and MUST wait for the corresponding ACK before transmitting the next packet.

- The receiver MUST ONLY accept the expected sequence number to guarantee in-order delivery.

- If the receiver receives a duplicate DATA packet (e.g., due to a delayed or lost ACK), it MUST re-ACK the sequence number but MUST NOT write the duplicate payload to the file.

6.2. Timeout and Retries

- The sender MUST utilize a fixed Retransmission TimeOut (RTO) of 500ms.

- Upon transmitting a packet, the sender MUST start a timer. If the timer expires before a valid ACK is received, the sender MUST retransmit the identical DATA packet.

- The sender MUST NOT exceed 10 retransmission attempts (MAX_RETR) for a single packet.

- If no ACK is received after the maximum retransmission attempts are exhausted, the sender MUST abort the session and SHOULD transmit an ERROR packet (TIMEOUT_ABORT) to the unresponsive peer.

7. Error Handling

- The protocol MUST handle fatal exceptions using the ERROR message type. Upon sending an ERROR packet, the sender MUST immediately abort the current session and MUST free all associated resources, including file handles, session states, and sockets.

- FILE_NOT_FOUND (0x01): The server MUST send this ERROR when a client requests a GET operation for a filename that does not exist or is not accessible.

- BAD_REQUEST (0x02): A peer MUST send this ERROR when it receives a malformed request. This includes an invalid operation code, an invalid filename length, or a payload length mismatch.

- SESSION_MISMATCH (0x03): If a peer receives a packet containing an unexpected or unrecognized session_id, the receiver MUST send this ERROR and MUST immediately abort the transfer.

- TIMEOUT_ABORT (0x04): If retransmission attempts exceed the MAX_RETR limit, the sender MUST abort the session and SHOULD send this ERROR to the unresponsive peer as a best-effort notification.

- INTERNAL_ERROR (0x05): A peer MUST send this ERROR upon encountering any unexpected failure during local processing, such as a file I/O fault or memory allocation error.

8. File Transfer Operations

8.1. GET (Download)

1. The client MUST initiate the transfer by sending a SYN packet specifying op=GET, the target filename, and the proposed_chunk_size.

2. The server MUST respond with a SYN_ACK packet containing status=OK, the chosen_chunk_size, the Initial Sequence Number (ISN), and a new session_id.

3. The server MUST read the requested file in binary mode, split it into chunks of at most CHUNK_SIZE, and send each chunk in a DATA packet using Stop-and-Wait ARQ.

4. The client MUST write the received chunks to disk in order and MUST acknowledge each successfully received chunk by transmitting an ACK packet where ack=seq.

5. After the final chunk where flags.EOF=1 is acknowledged, the server MUST send a FIN packet and the client MUST respond with a FIN_ACK packet.

8.2. PUT (Upload)

1. The client MUST initiate the transfer by sending a SYN packet specifying op=PUT, the target filename, and the proposed_chunk_size.

2. The server MUST respond with a SYN_ACK packet containing status=OK, the chosen_chunk_size, the Initial Sequence Number (ISN), and a new session_id.

3. The client MUST open the local file in binary mode and MUST send the file chunks as DATA packets using Stop-and-Wait ARQ.

4. The server MUST write the received chunks to disk in order and MUST acknowledge each successfully received chunk by transmitting an ACK packet where ack=seq.

5. After the final chunk where flags.EOF=1 is acknowledged, the server MUST send a FIN packet to initiate teardown and the client MUST respond with a FIN_ACK packet.

9. End-of-file Signaling

1. End-of-file MUST be signaled using the EOF bit in the header flags field.

2. The sender MUST set flags.EOF=1 on the very last DATA packet of the file transfer.

3. The receiver MUST treat this specific packet as the final chunk, MUST write the payload exactly once, and MUST acknowledge it normally using an ACK packet where ack=seq.

4. After the final DATA packet is acknowledged, the sender MUST transmit a FIN packet to formally terminate the session, and the receiver MUST reply with a FIN_ACK packet.

10. Security Considerations

This protocol does not provide encryption, message authentication, or data integrity checks beyond standard UDP checksums. Payloads are transmitted in plaintext, making them vulnerable to eavesdropping, interception, and modification. Furthermore, the protocol does not authenticate the client or server, making sessions susceptible to hijacking or spoofing attacks. This protocol MUST NOT be used to transfer sensitive or confidential information over untrusted networks.

Authors' Addresses

Rainer Gonzaga
De La Salle University
Manila, Philippines
Email: rainer_gonzaga@dlsu.edu.ph

Kristopher Lance Chiu
De La Salle University
Manila, Philippines
Email: kristopher_chiu@dlsu.edu.ph