NSCOM MCO Group INTERNET DRAFT
Request for Comments: 9902 R. Tan
Obsoletes: J. Luna
Category: Experimental February 2026

RFC: Reliable Data Transfer Protocol over UDP (RDTP)

Abstract

This document describes Reliable Data Transfer Protocol (RDTP), a data transfer protocol for low-resource data transfers over UDP, but with reliability and TCP-like features implemented in the application layer. This protocol may prove valuable in use cases where small data transfers are required on low-resource hardware, similar to TFTP, but also require reliable data transfer.

1. Introduction

Reliable Data Transfer Protocol (RDTP) is a custom application-layer protocol built on top of UDP that provides reliable, ordered file transfer between a client and server. The RDTP implements TCP-like features, including session establishment, sequencing, acknowledgements, retransmission, and clean termination, all inside the application layer.

The RDTP is designed to demonstrate how reliability and ordering can be achieved over UDP's connectionless, best-effort delivery model, similar to how protocols like TFTP, DHCP, and DNS implement their own control logic on top of UDP.

The protocol supports two core operations: file download (client requests a file from the server) and file upload (client sends a file to the server).

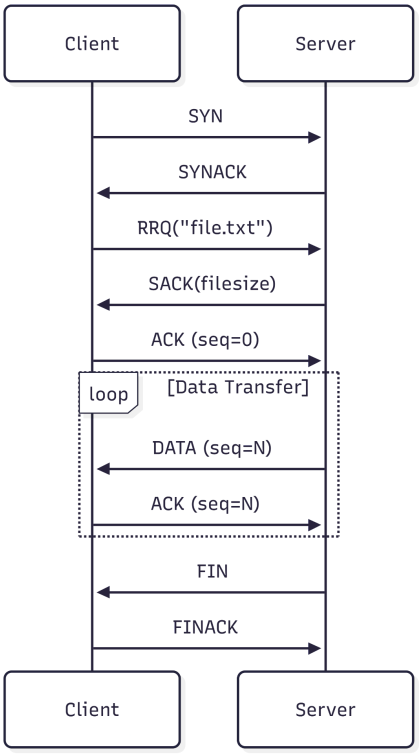
2. Language Conventions

In this document, the key words "MAY", "MUST, "MUST NOT", "OPTIONAL", "RECOMMENDED", "SHOULD", and "SHOULD NOT" are to be interpreted as described in BCP 14, RFC 2119.

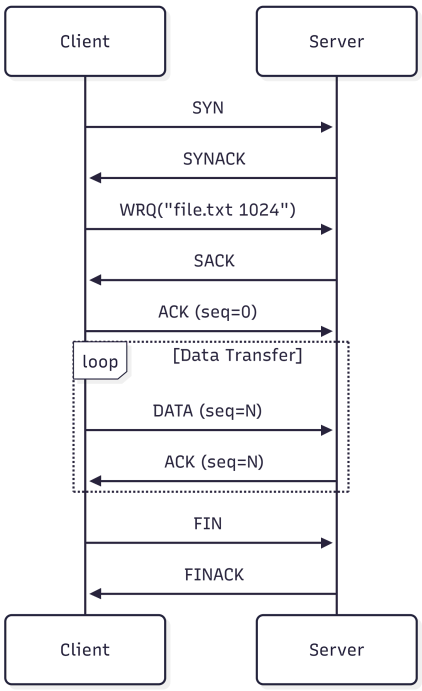
2.1 Session Lifecycle

Every RDTP session follows the following life cycle:

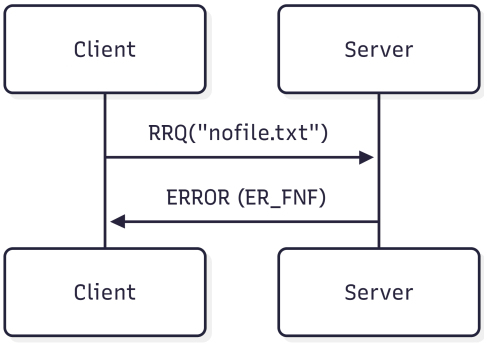
1. **Connection Check** - Client verifies if the server is available via SYN/SYNACK
2. **Session Establishment** - Client sends a read or write request (RRQ/WRQ), server acknowledges
3. **Data Transfer** - File data is transmitted in sequenced, acknowledged chunks
4. **Termination** - Sender signals termination via FIN/FINACK exchange



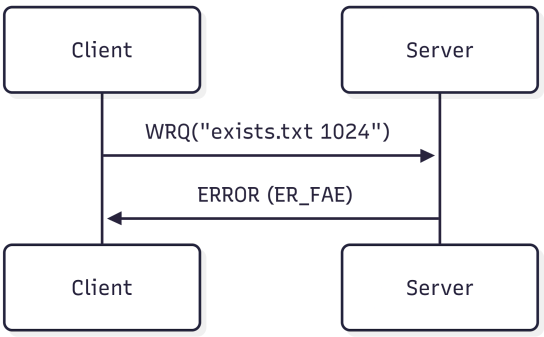
2.2 Swimlane Diagram – File Download (RRQ)



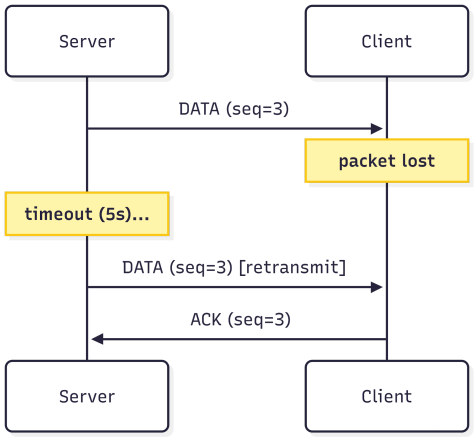
2.3 Swimlane Diagram – File Upload (WRQ)



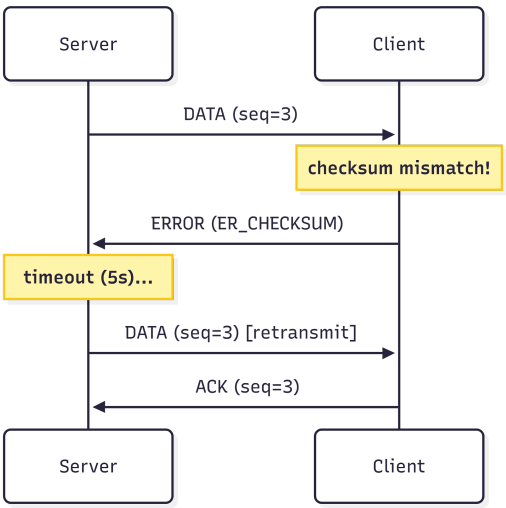
2.4 Swimlane Diagram – File Not Found (Error)



2.5 Swimlane Diagram – File Already Exists (Error)



2.6 Swimlane Diagram – Retransmission on timeout



2.7 Swimlane Diagram – Checksum Mismatch

3. Packet Message Formats

This section of the document describes the types, contents, and use cases of the various packets exchanged throughout the program.

There are a total of ten different packet types utilized throughout the program. Each packet type is designated an operation code as an identifier. As mentioned in 3.1, the operation code is the very first field in the packets. Below is a list of the types alongside their respective operation code:

Op.code	Operation
0	Read Request (RRQ)
1	Write Request (WRQ)
2	Session Acknowledgement (SACK)
3	Data (DATA)
4	Acknowledgement (ACK)
5	Synchronize (SYN)
6	Synchronize Acknowledgement (SYNACK)
7	Finish (FIN)
8	Finish Acknowledgement (FINACK)
9	Error (ERROR)

3.1 Header format

In the code implementation of the program, the packets were read and separated into two sections: operation code, sequence number, and payload. The format and use-cases of some of the fields are edited in some specific packets in accordance with their needs. For packets wherein more than the three fields provided are utilized, an empty byte is placed in the payload to separate them.

3.2 Message Types

RRQ and WRQ are sent by the client to indicate the kind of operation they want to perform during the session. The SACK packet is sent by the server in response to acknowledge the session request.

The DATA packet contains the chunk of data for transfer. The ACK is sent when a DATA packet is received to indicate that the DATA packet was received successfully.

SYN is sent by the client to check if a connection can be established with the server, and mostly to check whether or not the server is online. SYNACK is sent in response to signify that the server is available.

FIN is sent when a WRQ or RRQ session is complete, and signifies the end of the data transfer process. FINACK is sent in response to acknowledge the FIN packet.

3.3 Message-Specific Payloads

The payloads of the RRQ and the WRQ both contain filenames, for the file to be downloaded and for the file to be uploaded, respectively. Additionally, the WRQ contains an empty byte as a spacer in its payload, as after the filename, it also contains the total size of the file. When the session is of an RRQ type, the SACK contains the total size of the file in its payload; when it is of a WRQ type, the SACK does not contain anything in its payload.

The DATA packet's payload contains the data of the file’s chunk being sent over. Additionally, the payload contains a checksum used for error detection and is encrypted. The ACK packet in response does not have any payload.

SYN, SYNACK, FIN, and FINACK do not contain anything in their payloads. The ERROR packet utilizes the sequence number field in the packet as an error code, which identifies the kind of error encountered. Its payload is empty.

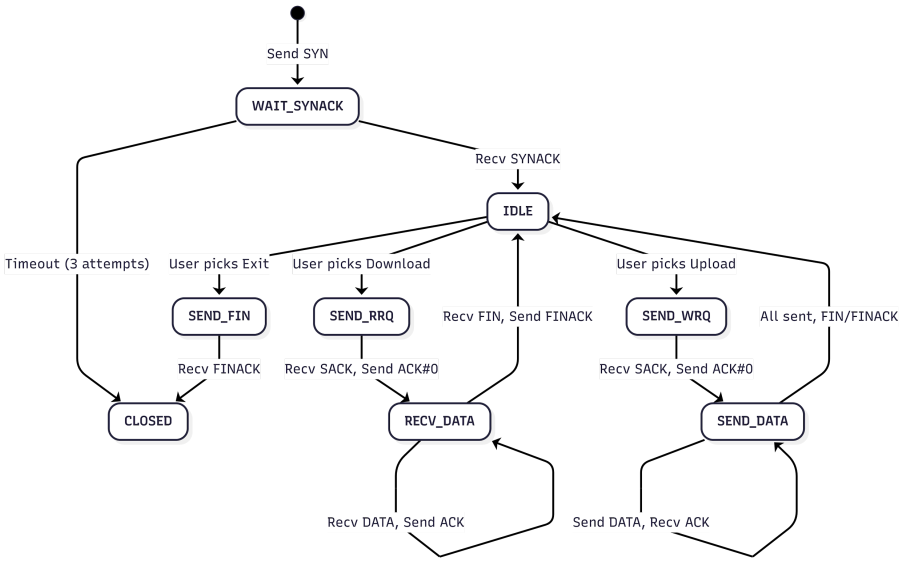
4. State Machines

RDTP utilizes a client-server network architecture. The main motivation for this is to allow multiple clients to communicate with a server. An RDTC server contains all the files uploaded by the client, and in turn allows clients to download from a selection of all the available files in the server.

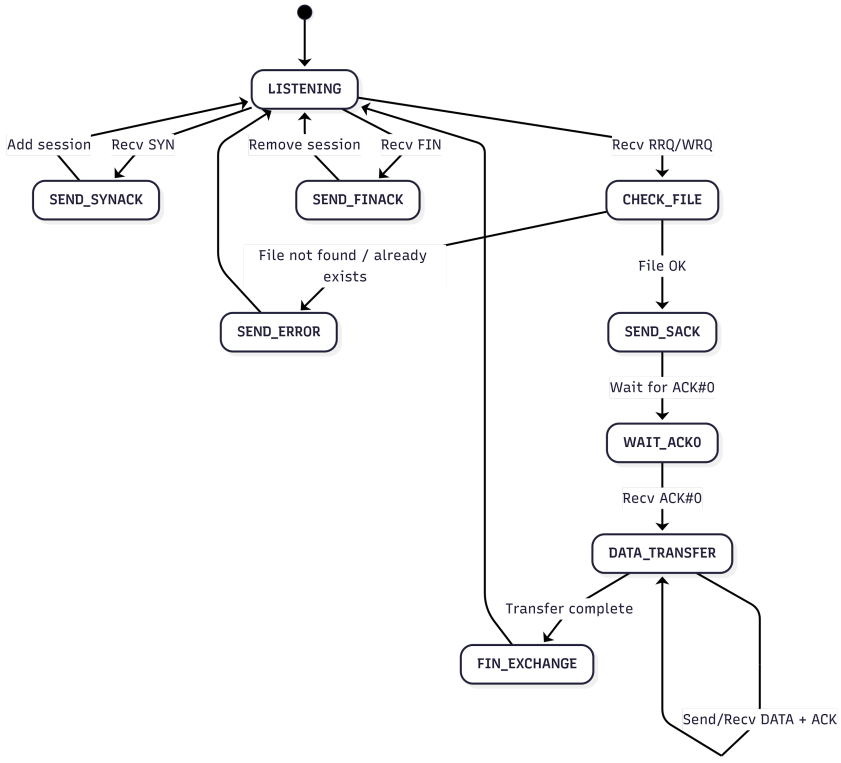
The client side of the program provides a console interface for easier navigation of RDTP’s functionalities.

The server side of the program provides regular updates to the current state of the server, printing notifications when a new packet arrives

or when a new client accesses the server. It’s less user-friendly in comparison to the client; it is justified as the server is not necessarily made to be directly used by the client.



4.1 Client State Machine



4.2 Server State Machine

5. Reliability Mechanisms

5.1 Sequencing

Each DATA packet has a sequence number starting from 1, incrementing by 1 for each subsequent chunk. This allows the receiver to detect missing, duplicate, or out-of-order packets.

5.2 Acknowledgements

The receiver sends an ACK packet for every DATA packet received. The ACK carries the same sequence number as the DATA packet it acknowledges, confirming successful transmission.

An ACK with a sequence number of 0 is used during session establishment to signal the start of the data transfer after receiving a SACK. The ACK#0 is used in both download (RRQ) and upload (WRQ) flows.

5.3 Stop-and-Wait

RDTP uses a stop-and-wait protocol wherein the sender transmits one DATA packet and waits for the corresponding ACK before sending the next.

5.4 Timeout and Retransmission

If the sender does not receive an ACK within 5 seconds, it retransmits the DATA packet. The sender will retry up to 5 times before sending an ER_TIMEOUT error and aborting the transfer. Retransmission is applied to:

- DATA packets (5 attempts)
- FIN packets during termination (5 attempts)
- SYN packets during connection check (3 attempts)

5.5 Checksum Verification

Each packet includes a 16-bit checksum computed over the encrypted payload. The checksum is the sum of all bytes in the encrypted payload, masked to 16 bits.

The receiver recomputes the checksum on the received encrypted payload and compares it to the checksum in the header. If they do not match, the receiver sends an ER_CHECKSUM error packet, and the sender retransmits after a timeout.

5.6 Encryption

All payloads are encrypted using a combination of AES-128 and HMAC-SHA256, using Fernet symmetric encryption from a Python library called cryptography.

Both the client and server share the same pre-configured symmetric key. The encryption flow is:

1. Sending: Plaintext payload → Encrypt → Compute checksum on ciphertext → Pack header and ciphertext
2. Receiving: Unpack header → verify checksum on ciphertext → Decrypt → return plaintext payload

6. Error-handling

There are five errors that are identified throughout the process of the data transfer. Each error has its respective error code that signifies the type of error that was encountered; this error code is placed on the sequence number field of the header (as discussed in section 3.3). Below are the different error codes and their respective errors:

Er.Code	Error Type
0	Timeout (TIMEOUT)
1	File Not Found (FNF)
2	File Already Exists (FAE)
3	Unexpected Packet (UNEXPECTED)
4	Checksum Mismatch (CHECKSUM)
5	Not enough disk space (SPACE)

6.1 Error Types

A TIMEOUT error occurs when, after five attempts at retransmission, no reply is received from either the client or the server. The assumption

when this error occurs is that either the client or the server may be down, resulting in the end of the session.

An FNF error occurs when the client attempts to send an RRQ, but the file requested is not in the current server's data. A FAE error occurs when the client attempts to send a WRQ, but there is already a file with the same name in the server.

An UNEXPECTED error occurs when the client or server receives a packet that is declared as 'unexpected.' This occurs when the packet that arrives is of a different type than what was expected, like receiving an RRQ packet in the middle of a data transfer, wherein the expected packet might be a DATA or an ACK packet.

A CHECKSUM error occurs during the data transfer process in the event that the computed checksum and the checksum included in the DATA packet are a mismatch. Receiving this packet implies an error might have occurred while the DATA packet was in the middle of a transmission, and the received DATA packet is thus discarded.

A SPACE error occurs when either the client does not have enough free space to download the file, or the server does not have enough free space to upload the file.

7. File Transfer Operations

All file operations use binary mode ("rb" for reading, "wb" for writing). Data is sent in chunks, and the chunks have a fixed block size of 512 bytes. A fixed block size was chosen in order to maintain simplicity and provide clarity and predictability regarding the protocol's behavior throughout the process, and maintain reliability. 512-bytes was chosen as it is the default in TFTP, and is a large enough size ensuring quick transfer, but not too large that it goes against the philosophy of RDTP to be used primarily for trivial files.

7.1 Download (RRQ)

1. Client sends RRQ with their desired filename in the payload
2. Server checks if the file exists
3. If not found: Server sends ER_FNF error packet. Client displays message and returns to menu

4. If **found:** Server sends SACK with file size in payload
5. Client sends ACK with seq=0 (ACK#0) to confirm transfer start
6. The server reads the file in 512-byte chunks and sends each as an encrypted DATA packet with incrementing sequence numbers
7. Client verifies checksum on encrypted payload, decrypts, and sends ACK for each DATA packet
8. If **checksum fails:** Client sends ERROR (ER_CHECKSUM); server retransmits
9. After all chunks are sent, the server sends FIN
10. Client saves the reassembled data to the downloads/ directory
11. Client sends FINACK to confirm transfer completion

7.2 Upload (WRQ)

1. Client checks that the file exists in the directory and the filename contains no spaces
2. Client sends WRQ with filename and filesize in the payload
3. Server checks if the file already exists
4. If **exists:** Server sends ERROR (ER_FAE); client displays message and returns to menu
5. If **not:** Server sends SACK to accept the upload
6. Client sends ACK with seq=0 to confirm transfer start
7. Server waits for ACK#0 before entering the data receive loop
8. Client reads the file in 512-byte chunks and sends each as an encrypted DATA packet with incrementing sequence numbers
9. Server verifies checksum on encrypted payload, decrypts, and sends ACK for each DATA packet
10. If checksum fails: Server sends ERROR (ER_CHECKSUM); client retransmits
11. After all chunks are sent, client sends FIN
12. Server saves the reassembled data
13. Server sends FINACK to confirm transfer completion

8. End-of-file Signaling

RDTP uses an explicit FIN packet to signal that all data has been transmitted. This differs from TFTP’s approach of using a short block to signify the end of a transfer.

The FIN/FINACK exchange works as follows:

1. The sender finishes sending all DATA packets
2. The sender transmits a FIN packet

3. The receiver acknowledges with a FINACK packet
4. Both sides now agree that the transfer is complete

To ensure reliable transmission even if packets are lost, the FIN packets are sent with retransmission support (with up to 5 attempts).