Group 6 Task Force
INTERNET-DRAFT

Felix Gust, Musfira Naqvi, Maximilian Wiesholler
TU Munich - Protocol Design
May, 2021

Robust File Transfer Protocol

Abstract

This document describes the Robust File Transfer Protocol (RFTP), which enables reliable file transfer from a server to a client and lies on the User Datagram Protocol (UDP).

Status of this Memo

This is an Internet Draft document for the first assignment of the course Protocol Design @ TU Munich.

Copyright Notice

Contents

1 Introduction

Establishing connection with the help of transmission control protocol (TCP) has proven to be very reliable - especially when it comes to retransmitting lost segment to the client's side. However, due to the connection establishment, retransmits as well as reordering of the segments, the user is often confronted with significant delays until they can use the application itself. Using UDP would solve the connection establishment delay problem - as it directly starts sending the file after given the socket and IP address of the corresponding endpoint - however its quality of service is greatly degraded because of the arbitrarily high error rate which is mostly tolerated. What's more: UDP does not offer any mechanisms regarding congestion and flow control.

Here is where RFTP comes into play:

RFTP is a connection-oriented, end-to-end file transfer protocol. While it ensures that no unnecessary delays are happening during the connection between client and server, it also handles congestion control as well as retransmits of lost packets - mechanisms which are neglected by UDP.

2 Conventions

2.1 Terminology

RFTP	Robust File Transfer Protocol
UDP	User Datagram Protocol
RTT	Round Trip Time
IP	Internet Protocol
JSON	JavaScript Object Notation
MPS	Maximum Packet Size

2.2 Terms

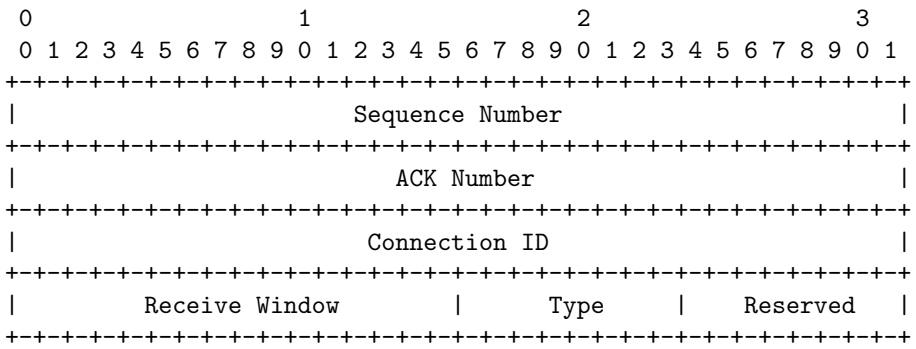
Server	Communication instance of the RFTP that make files available
Administrator	Person that setups and maintains the server
Client	Communication instance of the RFTP that retrieves files
User	Person that runs the client

2.3 Requirements Language

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 [2][RFC2119].

3 Packet Type Concept

All packets are sent as the payload of a UDP packet and share the same header:

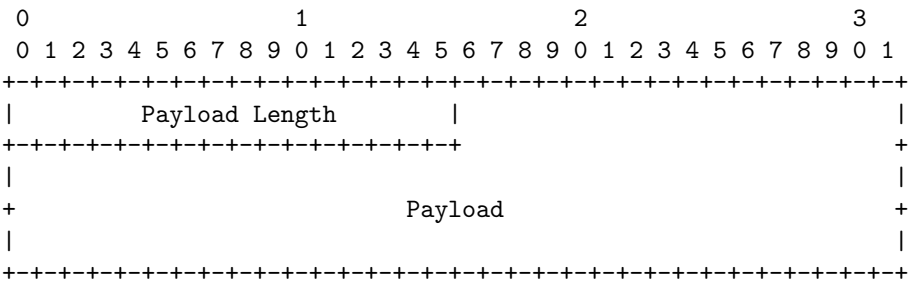


There are different packet types, each belonging to one of three packet kinds: packets with variable length payload, packets with fixed length payload, and packets without payload. The following lists all packet types and their corresponding type value in the header.

- FileRequest (0)
- MoreToCome (1)
- EndOfFile (2)
- FileMeta (3)
- FileSend (4)
- ConnectionFIN (5)
- ConnectionACK (6)
- Reconnect (7)
- BadConnection (8)
- MTUDiscovery (9)
- FileNotFound (10)
- FileBadInput (11)
- FilePointerInvalid (12)

3.1 Packets with Variable Length Payload

All packets with variable length payload specify the payload length in bytes in the first 2 bytes after the header. This length does not include the size of the payload length field itself. As an example, a packet with a 10 byte payload would consist of the header shown above and the following data. The payload length would be set to 10.

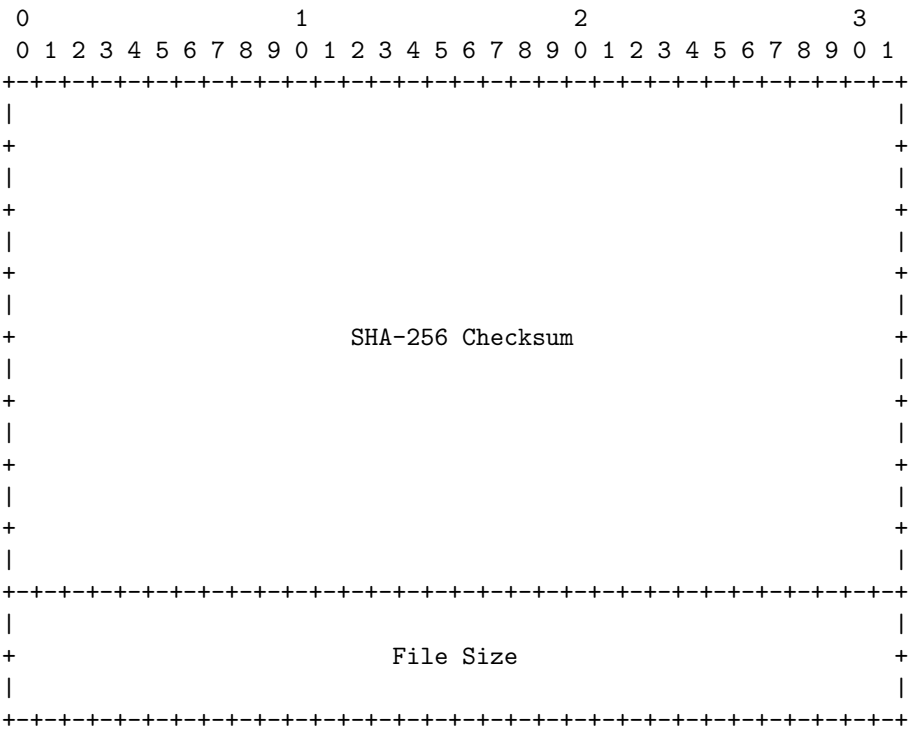


There are three packet types with a variable lenght payload. A FileRequest packet is used by the client to request a file to be sent by the server. The payload consists of the filename. A MoreToCome packet contains a chunk of the file that is currently being sent and indicates that more file chunks are coming. An EndOfFile packet contains the last file chunk. When receiving it, the client knows that the transmission is finished.

3.2 Packets with Fixed Length Payload

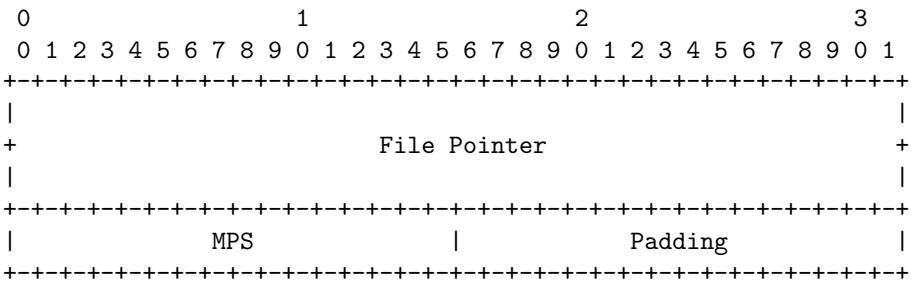
There are two packets that always have the same payload size and therefore do not need the payload length field: FileMeta and FileSend.

3.2.1 FileMeta



A FileMeta packet contains the SHA-256 checksum (32 bytes) and size (8 bytes) of a file.

3.2.2 FileSend



A FileSend packet is used to set up the file download. Its payload consists of a file pointer (8 bytes) that determines the offset from the beginning of the file, the maximum packet size MPS (2 bytes) and padding (2 bytes).

3.3 Packets without Payload

Packets that are used to control the connection and communicate errors do not require a payload and therefore only consist of the header.

3.3.1 Connection Management

A ConnectionFIN packet is used to terminate an existing connection. A ConnectionACK packet is used to acknowledge a newly established connection or a received ConnectionFIN packet. A Reconnect packet is used by the client to initiate a reconnect to the server. A BadConnection packet is sent by the server if a client's reconnect request is invalid. Finally, an MTUDiscovery packet informs the other party that a Path MTU Discovery, which will take some time, is currently being performed to determine the optimal maximum packet size.

3.3.2 File Errors

The FileNotFound, FilePointerInvalid, and FileBadInput packets are sent from the server to the client if their corresponding file related errors, as described in 7.3, are encountered.

4 Communication

4.1 Acts of the communication

RFTP's connection consists of three parts:

- Connection Establishment
- File Transfer
- Connection Tear-Down

4.1.1 Connection Request

The RFTP connection initiation phase starts with the following: The client first sends a FileRequest packet to the server endpoint. The server answers this with a file meta, which includes the checksum as well as the file size. Note, that with the transmission of the file meta, the server also sends the connection ID (CID) which informs the client socket about the current session ID. The client acknowledges the connection ID by sending the packet type FileSend - which will be explained further in the next section.

4.1.2 Setup of File Download

As mentioned before, the receipt's acknowledgement of the connection ID is sent as a FileSend packet. With this, both endpoints will negotiate the maximum packet size. In the FileSend packet the client notifies the server about its current file pointer, its receiver window and the maximum packet size (MPS) it can support. The server then answers also with a FileSend packet confirming the file pointer and specifying the MPS it supports. Both sides then select the MPS with the lower value.

4.1.3 File Download

The server does not wait for an acknowledgement of its FileSend packet. Instead, it directly starts by sending the first chunks of the requested file to the client.
Within the header of the first file payload, the client MUST gain following information related to the file download:

1. Sequence number of the file data packet
2. PacketType: MoreToCome or EndOfFile
3. Payload length
4. Payload of the actual sent file chunk

With the sequence number the client is able to sort incoming packets correctly so that file chunks are appended in correct order to the file. The client cumulative acknowledges the file data packets. In the ConnectionACK packet the client sets the ack number to the next sequence number of the packet that is sent by the server in the next batch of packets. With the payload length the client knows the size of the whole packet. The packet type indicates whether this is the file data packet with the last file chunk or whether more file data packets will come.

4.1.4 Connection Teardown

When the server reaches the end of the file to be downloaded, it MUST set the packet type to EndOfFile which informs the client about the file chunk to be the last one.
When the client receives the EndOfFile packet the connection teardown starts:
The client sends a ConnectionFIN to the server to implicitly acknowledge the EndOfFile and to initiate the teardown. The server answers with an acknowledgement of the teardown packet and the connection finally terminates.

4.2 Example Communication

CLIENT STATE				**SERVER STATE**
CLOSED				LISTEN
1.	−− >	FileRequest	−− >	
2.	< −−	FileMeta	< −−	
ESTABLISHED				ESTABLISHED
3.	−− >	FileSend	−− >	
4.	< −−	FileSend	< −−	
5.	< −−	MoreToCome	< −−	
6.	−− >	ConnectionACK	−− >	
...				
7.	< −−	EndOfFile	< −−	
8.	−− >	ConnectionFIN	−− >	
9.	< −−	ConnectionACK	< −−	
CLOSED				LISTEN

4.3 Connection Migration

When the client socket changes its IP-address, connection will drop - this is due to the fact that UDP is dependant on the given socket and IP-address at the beginning. In order for the client socket to reconnect with the server, it uses the connection ID with the help of the ReconnectPacket. The server then checks whether the connection ID exists. If the connection ID exists, reconnection is possible on the new client socket. If the connection ID does not exists, the server sends a connection-less BadConnection packet to the client.

Reconnection establishment can happen before the file transfer, during the file transfer or after the file transfer. In any case, after receiving the ReconnectPacket of the client , the server sends

a FileMeta packet of the file. The client answers with a FileSend packet but having received the FileMeta packet on client-side before, two scenarios may happen:

1. Client's previously received stored checksum and the new received checksum are identical: File transfer will be resumed. Therefore the client MUST set the file pointer to the number of bytes of the partial file the client has already stored.

2. Client's previous stored received checksum and the new received checksum are not identical: The file changed server-side and next file chunks from the server will be inconsistent to the client's received file chunks. The client will therefore send a FileSend packet and MUST set file pointer at 0 bytes which informs the server that the file needs to be resend from the beginning.

Reconnection may happen before even the file transfer started. Then scenario 1 can only happen if there had been an incomplete file transfer for this file before with another connection. See 7.7 for further details.

Reconnection may happen after the file transfer completed when the client tries to send a ConnectionFIN packet for the teardown of the connection. In this case successful reconnection is also followed by a FileSend packet and both previous mentioned scenarios are possible. In scenario 1 the client sets the file pointer to the number of bytes of the stored file. After the setup of the file transfer by sending the FileSend packet to each other, the server will not need to send further file bytes. The server then immediately sends a EndOfFile packet with an empty payload.

4.4 Connection Drop

In the understanding of the RFTP a connection drop is a loss of a packet or several packets. If packets are lost the sender retransmits lost packets. If the client's packet retransmission does not continue the communication, the client may entered the state of a connection migration and sends a Reconnect packet. See 4.3 for further details.

5 Traffic Control

UDP has no flow control and congestion control mechanisms. These must be implemented by the RFTP. Moreover, RFTP must handle different kind of timeouts and consider fragmentation issues.

To ensure correct packets RFTP makes implicitly use of UDP checksums. If UDP gets a UDP packet for which the checksum in the packet is not identical to the checksum UDP computes over the packet, UDP drops silently the packet. This drop MUST be then handled by RFTP. The RFTP sender will retransmit the RFTP packet after the timeout (See 5.4 for further information regarding timeout-control).

5.1 ACK Strategies

5.1.1 Outside of File Transfer

For communication which is not involved with the file transfer, Stop-And-Wait is used as the ARQ protocol. In this case, the server transmits a RFTP packet and then waits for the corresponding acknowledgment which is sent by the client. Should the server not receive an ACK within a set time interval, it will interpret this as the packet being lost and retransmits it again. See 5.4 for further information regarding timeout-control.

5.1.2 During File-Transfer

When it comes to the actual file-transfer, Go-back-N is used. It is possible for the server to send many file packets at once without expecting single acknowledgements for each transmitted packet. It is important to note that the number of sent file packets is dependant on the client's receiver window. The client will not accept the receipt of packets out of order.
After receiving multiple file data packets, that can be ordered without having a gap between them, the client will then answer with a cumulative acknowledgement. It does so by acknowledging the packet with the sequence number n+i to indicate the successful receipt of the previous files with the sequence number n to n+i excluded. The client sets in the acknowledgment response the ACK number to n + i + 1.
In the case of a not received segment which may lead to a timeout, the server will retransmit all of the client's expected file packets which are supposed to fall into the client'S receiver window.

5.2 Flow Control

To prevent the sender from overwhelming the receiver with too many packets, RFTP limits the number of packets that can be sent between two successfully received acknowledgement packets. The receiver MUST include the number of bytes currently available in its receive window in the corresponding field of every ACK packet. Between the reception of the current and the next ACK packet, the sender MUST NOT send more bytes than the receive window field in the current ACK packet specifies. If this field is zero, the sender stops sending until the next ACK packet containing a non-zero value for the receive window is received or one of the timeouts (see 5.4) occurs.

5.3 Congestion Control

The congestion control of RFTP aims to be similar to TCP-Reno [1][RFC5681]. The sender maintains two variables:

- Congestion window (cwnd): the maximum number of bytes the sender can transmit before receiving an acknowledgement
- Slow start threshold (ssthresh): limit for cwnd during the slow start phase

The congestion control algorithm consists of two phases: slow start and congestion avoidance. In both phases the sender MUST NOT send more bytes than the sum of the highest currently acknowledged sequence number and cwnd.

5.3.1 Slow Start

Congestion control always begins in the slow start phase. In the beginning the value of sstresh SHOULD be set arbitrarily high (e.g. to the largest possible receive window size) and the value of cwnd MUST be set to 1 MPS. The sender starts transmitting data and increases cwnd by 1 MPS for each received ACK packet. If cwnd exceeds sshtresh or a packet timeout occurs (see 5.4), which indicated packet loss, cwnd is halved, sstresh is set to equal cwnd, and the algorithm transitions to the congestion avoidance phase.

5.3.2 Congestion Avoidance

For every received ACK packet, the sender increases cwnd by 1 MPS/cwnd. If a packet timeout occurs, which indicated packet loss, cwnd is halved and sstresh is set to equal cwnd. If a connection migration timeout occurs, the algorithm transitions back to the slow start phase.

5.4 Timeout Control

The RFTP distinguishes different timeouts: Packet timeout, connection migration timeout, RFTP connection timeout.

RFTP packets may get lost. Loss detection is implicit by timeout: If there is no response from the receiver until the timeout, the sender MUST retransmit the packet. The timeout SHOULD be set to the RTT multiplied with two.

If the client retransmits an RFTP packet multiple times without any response from the server, a connection migration might happened and the client MUST send a Reconnect packet. The timeout after which the client sends the Reconnect packet SHOULD be set to the RTT multiplied with 10. If the server does not respond to the Reconnect packet, the client MUST send further Reconnect packets.

Timeout for RFTP packets must be increased when the communication partner conducts a path MTU discovery (see 5.5.1) . Considering the varying number of nodes between both communication endpoints the timeout SHOULD be set to the RTT multiplied with 15. After this timeout, retransmission phase starts and may be followed later by a reconnection attempt.

Client and server MUST run independent stopwatches for the RFTP connection. For an arriving RFTP packet the receiver resets its stopwatch. If the stopwatch's time expires at one side the connection terminates and resources are deallocated. The stopwatch SHOULD be set to 45 seconds and reset to 45 seconds.

5.5 Fragmentation Control

Most of the files are too big to be sent within one RFTP packet. The server sends chunks of a file within MoreToCome or EndOfFile packets that are then ordered by the client so that the chunks will be appended correctly to the file output.

5.5.1 Avoiding Fragmentation on the IP layer

Fragmentation on the IP layer MUST be avoided. If one fragment on the IP layer would be lost, the assembled UDP packet is incomplete and UDP drops the packet after checksum computation. The documentation presents in section 3 three types of RFTP packets. For RFTP packets without payload and RFTP packets with fixed payload there will not be any fragmentation on the IP layer because byte sizes of these packets. For instance, FileMeta with fixed payload has a size of 56 Bytes. Together with the UDP header of 8 bytes the UDP packet has 64 Bytes. Assuming an IPv4 header with 60 bytes including IP Options [5][RFC791] or an IPv6 header with 40 bytes [3][RFC2460], the resulting IP packet will be 124 bytes or 104 bytes respectively. Referring to IP that requires hosts to be able to deliver IP packets with at least 576 bytes for IPv4 [5][RFC791] and 1280 bytes for IPv6 [3][RFC2460], fragmentation of the RFTP packet does not happen.

However, RFTP packets with variable payloads have variable byte sizes. In the RFTP this covers the MoreToCome packet type and EndOfFile packet type. The maximum packet size (MPS) is negotiated with the FileSend packet. The client sends a FileSend packet to the server with the MPS the client supports and the server sends a FileSend packet to the client with the MPS the server supports. Both sides then take the smaller one for the following file transfer.

The MPS SHOULD be computed by taking the MTU and subtracting the size of the UDP header and IP header. For IPv4 including IP options the MPS in bytes will be MTU - 60 - 20. For IPv6 the MPS in bytes will be MTU - 40 - 20. If MTU is not used as reference the MPS must be defined by the minimal IP packet size requirements for network hosts. For IPv4 the MPS in bytes is then 576 - 60 - 20. For IPv6 the MPS in bytes is then 1280 - 40 - 20.

How each communication endpoint (server and client) gets the MTU for MPS computation is independent from the RFTP connection. Two options are possible: A communication endpoint reads the MTU from the link layer or conducts a Path MTU Discovery [4][RFC1191]. While the former does not need further considerations the latter needs support by the RFTP. Since a Path MTU Discovery takes time and creates a delay in the RFTP communication, an instance that conducts a discovery sends a RFTP packet of type MTUDiscovery to inform the other side about the delay. The

other side MUST acknowledges this packet with a ConnectionACK. This process ensures that the other side does not start to retransmit packets early. Moreover this avoids an early ReconnectPacket when the server makes a Path MTU Discovery. When the discovery finishes the instance sends the FileSend packet with the MPS it has computed from the discovered MTU.

6 Server-side File Handling

The server stores a list of files in a potentially nested directory structure. Each of the files can be requested by the client in a FileRequest packet. With the start of the server process the administrator MUST specify the main directory of the files.

6.1 File Checksum Preparation

Part of the server's FileMeta packet includes the checksum of the requested file. Since the computation of the checksum requires computation resources, the server SHOULD first check if a stored checksum exists. There may be files that are not changed and files which will be changed frequently. For the former the administrator MAY store checksums next to the files to avoid checksum computation. If the server cannot find a stored file checksum, it generates the checksum of the file.

6.2 Detecting File related Errors

The server MUST detect the following file related errors.

6.2.1 File Not Found

The server receives a FileRequest packet with a file name that does not exist on the server. The server sends a FileNotFound packet to initiate a connection tear-down and waits for the acknowledgment of the client to close the connection.

6.2.2 File Pointer Invalid

The server receives a FileSend packet with a file pointer that is out of the range of the file's possible indices. The server reacts with a FilePointerInvalid packet to initiate a connection teardown and waits for the acknowledgment of the client to close the connection.

6.2.3 File with Bad Characters

The server receives a FileRequest packet with a file name that contains bad characters or bad character sequences. The server reacts with a FileBadInput packet to initiate a connection teardown and waits for the acknowledgment of the client to close the connection.

Policies for bad characters and bad character sequences MUST be extensible. The authors recommend to allow only relative paths from the server's RFTP file directory which remain in the directory or lead into subdirectories. Thus, bad characters that would lead to the main root of the server or to parent directories SHOULD be avoided.

6.3 Receiving File related Errors

In the current standard of the RFTP there are no file related error messages from the client to the server. If the client detects file related errors, it starts a connection teardown by sending a generic ConnectionFIN packet to the server. When the server receives a ConnectionFIN packet, it MUST respond with a ConnectionACK packet and terminate the connection.

6.4 Handling of Multiple File Downloads

The server can have multiple RFTP connections to a single client as well as multiple connections to different clients. The server can identify each RFTP connection by a connection ID and has for each connection a separate socket. RFTP does not support multiple downloads per connection ID. The server will have multiple RFTP connections to the same client if the client requests further files when previous requested ones are still downloaded.

7 Client-side File Handling

7.1 File Types

At the beginning of the file download, the client MUST store three files:

- The file itself with the previous specified name. It is identified by the correct file name. Initially, the file is stored as a 0 bytes file and acts as a placeholder.
- A so-called progress file to which the file chunks will be written to with extension .PRGS.
- A simple meta file that contains file relevant information. This MUST include the file checksum which the client receives from the server.

All three files are stored next to each other in the same directory. For instance, for a file that is downloaded from the server with the file name 'example.txt' the client generates the files:

- example.txt
- example.txt.PRGS
- example.txt.json

In this example the meta file would be stored as a JSON file.

All of the three files are saved when the client receives the FileSend packet from the server which indicates the beginning of the download. It is important to note, that the download has to start from scratch. If the client is able to resume the download (see 7.7) these three files will already exist.

After the download completes, the client MUST do the following actions in the order they are mentioned:

- Remove the file with the correct file name.
- Rename the .PRGS file to the correct file name by removing the .PRGS extension.
- Remove the meta file.

If the three files should be removed before the download completes, the files MUST be moved to a logically different directory where the files get deleted eventually. Moving these files has the reason for not letting the server wait when the client has to delete a big .PRGS file. The directory, to which the files get moved is called volatile directory in the following sections - in order to emphasize that these files in this directory will eventually be deleted.

7.2 Detecting File related Errors

The client MUST detect the following file related errors:

7.2.1 File Checksum Invalid

The client received a checksum c' by the server for a file for which the client has stored a checksum c before. However, the client detects that c != c'. Since the checksum has changed the client MUST move the previously stored files to a directory that gets deleted eventually. This logical movement of files is necessary for not letting the server wait when the client has to delete a big .PRGS file. As a response to the server the client MUST set the file pointer to 0.

7.2.2 File Too Big

The client received a file size that would exceed client's memory capacity. The client MUST send a ConnectionFIN to initiate a teardown. The client then waits for the ConnectionACK by the server to terminate the connection.

7.3 Receiving File related Errors

The client can receive the following file related errors, each encoded as packet type in a payload free packet:

7.3.1 File Not Found

The server could not find the file by the file name the client has sent in its request. The servers sends a packet with the packet type FileNotFound. This packet type implicitly indicates a teardown which the server starts. The client MUST acknowledges this by a ConnectionACK packet and terminates the connection.

7.3.2 File Pointer Invalid

The server could not use the given file pointer in the FileSend packet. The server sends a FilePointerInvalid packet. This packet type implicitly indicates a teardown which is initiated by the server. The client MUST acknowledges this by a ConnectionACK packet and terminates the connection.

7.3.3 File with Bad Characters

The FileRequest packet by the client contains a file name with bad characters from the server's perspective. The server sends a FileBadInput packet. This packet type implicitly indicates a teardown which is initiated by the server. The client MUST acknowledges this by a ConnectionACK packet and terminates the connection.

7.4 User related Errors

The user of the client may deliver bad input that MUST be handled by the client before even a connection request starts:

Unknown File Path
The user provides a file path where the file would be stored. If the client observes that this path does not exist in the user's file system, the client notifies the user with an error message.

File with Bad Characters
The user SHOULD have the option to specify a different file name to which the three files get stored when the download begins. If the specified file name however contains characters which are not accepted by the operating system, the client notifies the user with an error message.

7.5 Pause Download

The user can always pause the download. By doing so the client MUST send a ConnectionFIN packet to the server and wait for the corresponding acknowledgment. All three files will not be deleted.

7.6 Abort Download

The user can always abort the download. Doing so the client SHOULD abort the communication appropriately. Therefore, the client sends a ConnectionFIN packet to the server for which acknowledgment the client waits. Then the client moves the three files to a volatile directory.

7.7 Resume Download

With the checksum and already stored chunks in the .PRGS file the user has the possibility to resume the download with the client. RFTP distinguishes two situations of resuming:

7.7.1 During Valid Connection

The connection can be temporarily broken because of a connection migration or drop. If reconnection happens before the connection ID has expired, the download can be resumed without starting a new connection. For detailed info about resuming the download in a non-expired RFTP connection see 4.3.

7.7.2 After Connection Expiration

An RFTP connection is expired when the connection ID is not valid anymore. This can have different reasons:

- The client paused the download.
- The client had lost the connection to the server and finally the timeout for the RFTP connection was exceeded.
- The client process crashed.
- The server process crashed.

In all of the cases the client sends a FileRequest to initiate a new RFTP connection with the server. When receiving the server's FileMeta response, the client MUST compare the new checksum c' with the previous stored checksum c in the meta file. If both checksums are equal, i.e. c = c', the client MUST set the file pointer in the FileSend packet to the number of bytes of the .PRGS file so that the download can continue with the next byte that will be appended to the .PRGS file. If both checksums are not equal, the client moves the previous stored files to a volatile directory and sets the file pointer in the FileSend packet to 0.

7.8 Handling of Multiple File Downloads

RFTP does not support multiple file download within one connection. To download another file the client MUST establish a new connection to the server.

8 Operational Considerations

The port number on which the server listens for incoming requests should be in the range of registered ports that is not in use or reserved. Since this draft is part of an exercise there is no need to make a reservation request to the IANA.

The volatile directory to move outdated file data to is mentioned in 7.1. Such a directory may be within a parent directory that is cleaned at a regular basis, e.g. a directory that is cleaned at every boot. Moreover, the application of the RFTP may provide a command for the user to directly remove the outdated file data.

9 IANA Considerations

This document has no IANA actions.

10 Security Considerations

Since the RFTP does not require security mechanisms this document does not contain any security considerations.

11 References

[1] M. Allman, V. Paxson, and E. Blanton. *TCP Congestion Control*. RFC 5681. http://www.rfc-editor.org/rfc/rfc5681.txt. RFC Editor, Sept. 2009. URL: http://www.rfc-editor.org/rfc/rfc5681.txt.

[2] Scott Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. BCP 14. http://www.rfc-editor.org/rfc/rfc2119.txt. RFC Editor, Mar. 1997. URL: http://www.rfc-editor.org/rfc/rfc2119.txt.

[3] Stephen E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. http://www.rfc-editor.org/rfc/rfc2460.txt. RFC Editor, Dec. 1998. URL: http://www.rfc-editor.org/rfc/rfc2460.txt.

[4] Jeffrey Mogul and Steve Deering. *Path MTU discovery*. RFC 1191. http://www.rfc-editor.org/rfc/rfc1191.txt. RFC Editor, Nov. 1990. URL: http://www.rfc-editor.org/rfc/rfc1191.txt.

[5] Jon Postel. *Internet Protocol*. STD 5. http://www.rfc-editor.org/rfc/rfc791.txt. RFC Editor, Sept. 1981. URL: http://www.rfc-editor.org/rfc/rfc791.txt.