Network Working Group
Document Type: Protocol Specification
Status: Draft

Gian Lorenzo C. Ortha

February 26, 2026

Somewhat Simple File Transfer Protocol

Summary
SSFTP is a Somewhat Simple File Transfer Protocol for transferring relatively small files between two hosts – a server and a client. It is based on the design of TFTP with an additional connection management functionality inspired by TCP. It is designed under UDP, which is inherently unreliable, and thus implements reliability mechanisms on the application layer for data transfer.

Protocol Overview

An SSFTP connection starts with a server that will begin listening for connections on one particular port. This port may vary between implementations and shall be indicated by the application or implementation of this protocol. This port must only be used for listening for connections. A server may have more than one connection. Once the server receives a SYN message through this port, it determines whether it can accept the connection. It may choose not to respond to the application, for instance, if it has reached its maximum number of connections. The server shall also ignore the SYN message if the connection already exists.

For the server to accept the connection, it sends a SYNACK message back to the client to acknowledge its connection request. This message must include a port number to which the client will direct all succeeding traffic to throughout the connection. The server will begin listening on this port for the messages that shall come from the client. The server shall only accept messages on this port from the client which it is connected to. In the cases of multiple connections, the server shall manage each connection, such that each connection is mapped to a respective port, and only accept a message from the corresponding client.

File Operations

Once a connection has been established, the client may initiate one of two operations: a download request (DWN) or an upload request (UPL). These two operations are analogous to the write request and read request of TFTP [RFC1350]. These operations indicate the path or name of the file to be transferred and certain options, which are largely optional. The server, on receipt of either one of these messages, will process the request. If the DWN or UPL request contains options, the server may decide to use these options or use its own options.

The server sends back an OACK message to acknowledge the request. This OACK message must include the options decided upon by the server and the client, must adopt these options. This message marks the beginning of the transaction, whether it be a DWN or UPL request. Only one transaction per connection may be ongoing any any given time. If the client does not receive an OACK message, it must assume that the server rejected the request.

Data, Acks and Sequence

Once a DWN or UPL transaction begins, the client will always be the one to send the first message following the OACK. In an UPL transaction, the client begins by sending the first block of DATA with a sequence number of one (1). In a DWN transaction, the client begins by sending an ACK with a sequence number of one (1) to trigger the server to send the first block of data.

The client or server, on receipt of a DATA message, acknowledges it and requests for the following block through an ACK message with a sequence number of the next block to receive. For instance, if DATA is received with a sequence number of (1), an ACK is sent with a sequence number of two (2). On the other end, when the client or server receives an ACK message, it shall send the DATA of the corresponding to the sequence number of the ACK. Therefore, if an ACK is received with a sequence number of two (2), DATA is sent with a sequence number of two (2). The client and server must keep track of the current sequence number. Data blocks must be sent and received in-sequence. If an out-of-order segment is received, it is discarded. An out-of-order DATA must not be acknowledged, and an out-of-order ACK must not trigger the next block of data to be sent.

The end of a file is marked by a DATA message with a block of data less than the negotiated blksize. If the final block of data happens to have the same size as the blksize, an extra DATA message must be sent with 0 bytes of data. Once the end of the file is reached, the transaction is complete.

Retransmission

It is possible for packets to be delayed or dropped in transfer, which is why it may be necessary to retransmit certain messages. The DATA and ACK messages work with a sort of call-and-response mechanism, that is to say, the receipt of a DATA may trigger the sending of an ACK and vice-versa. As such, if a client or server does not receive a response, after a certain timeout period, it can attempt to retransmit the DATA or ACK to trigger the response from the server. This method may cause duplicate DATA or ACK messages to be received; thus, in the case of duplicate DATA or ACK messages, they are be discarded. Only the original DATA or ACK messages must be processed and acknowledged by the application.

For example, in an UPL request, the client sends the first DATA message, expecting an ACK from the server, but it does not receive anything. After a timeout period, it sends another DATA message. The server finally receives the first DATA message and sends an ACK. Shortly after, it receives the duplicate DATA message, and ignores it. Finally, the client receives the ACK and proceeds to the following block, sending the next DATA message.

There are only three messages which must have a retransmission mechanism: DATA, ACK, and OACK messages. A retransmission is required for the DATA and ACK messages to ensure any dropped packets of can be sent and received to allow the file transfer to fully succeed. However, an OACK must also have a retransmission mechanism because the server has no way of telling if the client has received it and it may get stuck in a state waiting for the client to do something when the client did not in fact receive anything.

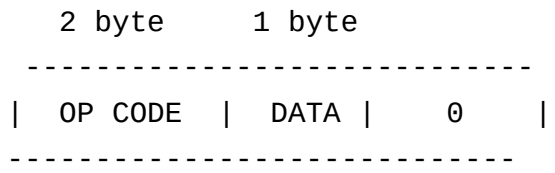
A client or server may attempt to retransmit a message up to a maximum amount of attempts, which, when exceeded, the application should assume that the other end has been disconnected and close the connection.

Connection Termination

When there are no ongoing transmissions, a graceful termination can be initiated by sending a FIN message. A graceful termination should typically be initiated by a client, but a server may also initiate a graceful termination. For a graceful termination to succeed, the sender of the FIN message must receive a FINACK. For as long as the sender of the FIN does not receive a FINACK, it should not terminate the connection. Both the client and the server, however, must be able to initiate a connection termination for errors, application interruptions, or loss of connection. Any form of connection termination that is not a graceful termination may not wait for a FINACK before terminating the connection.

Message Format and Types

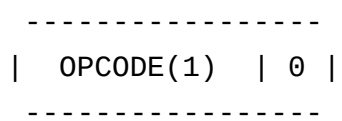
All SSFTP messages must begin with an operation code (OPCODE). This OPCODE may be any number from 1 to 10 each corresponding to a specific operation. Immediately following the OPCODE may be any data included in the message. The contents of this part of the message will vary based on the operation. The format of each message will be elaborated in each subsection below. Finally, the end of the message shall be indicated by a single terminating 0 byte. It is important to note, however, that a 0 byte does not necessarily indicate the end of a message. Certain data may also use a 0 byte in order to separate fields such as two succeeding strings.



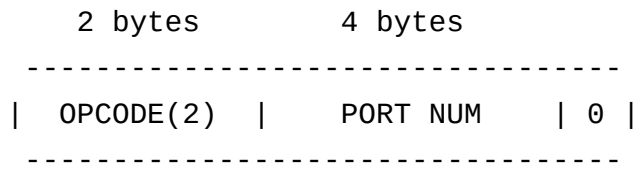
The diagram above shows the general format of an SSFTP message. In all future diagrams, a single 0 will always indicate a 0 byte, thus its size will no longer be explicitly indicated.

SYN and SYNACK

In order for a client to establish a connection to an SSFTP server, it must first send a SYN message. A SYN message is identified by the OPCODE 1 and it contains no data other than the OPCODE and the terminating 0 byte.



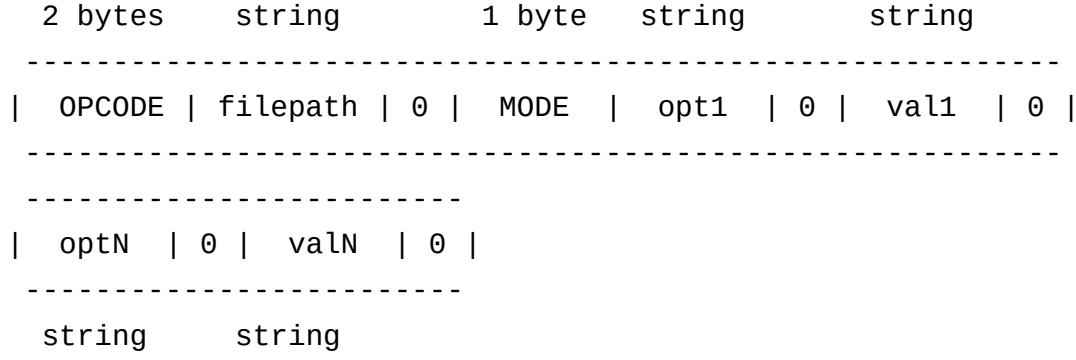
The server, on receipt of a SYN message from the client, acknowledges it by sending a SYNACK message for establishing a logical connection. A SYNACK message is identified by the OPCODE 2. It must include a port number to inform the client which port to send future messages to.



These two messages simulate a two-way handshake in order for a client and a server to establish a logical connection with one another.

DWN and UPL

After a client connects to a server, the client may initiate one of two operations: a download request (DWN) or an upload request (UPL). A DWN and UPL request are identified by the OPCODEs 3 and 4 respectively. The general message format of a DWL or UPL message is seen in the diagram below.



filepath
: The path to the file to download or upload. This field must be terminated with a single zero byte.

MODE

The mode of transport to use. There are two possible modes of transport: octet and netascii. These values are indicated by the values 1 and 2 respectively.

opt1
The name of the first option. This field is case-insensitive and must be terminated with a single zero byte.

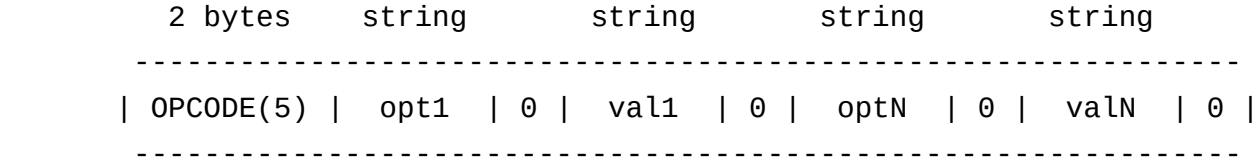
val1
The value of the first option. This field is case-insensitive and must be terminated with a single zero byte.

optN, valN
The name and value of the last option. These fields are case-insensitive and must each be terminated with a single zero byte.

Options are not required for a DWN request, however an UPL request must include a tsize option to inform the server of the size of the file to be uploaded.

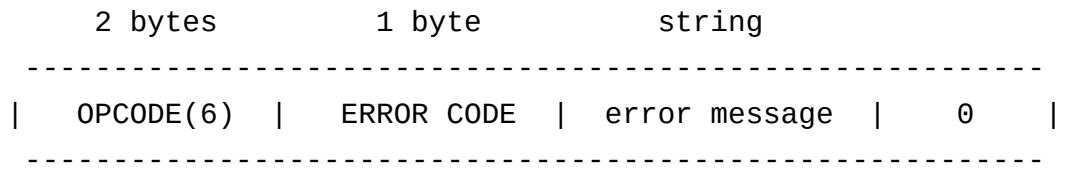
OACK

An OACK message is identified by the OPCODE 5 and is sent by the server to respond to an UPL or DWN request. It includes the list of options that will be used for the transaction. Options that the client does not indicate may appear in an OACK message to inform the client of the final options to use for the transaction. The client must use the values of each option found in the OACK message. The format of an OACK message may be found below.



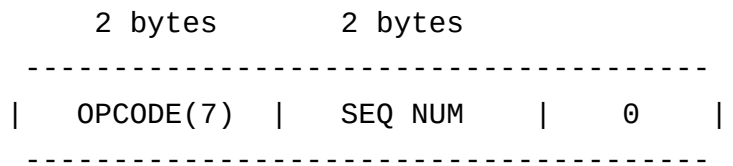
ERR

If the server or client encounters an error, it may send an ERR message. An ERR message is identified the OPCODE 6. Following the opcode is a byte containing the error code, and afterwhich is the error message. The format of an ERR message may be found below:



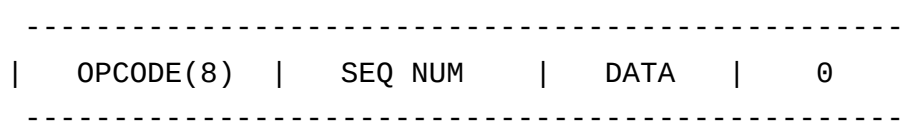
ACK

An ACK message is identified by the OPCODE 7, and is used to acknowledge and request for subsequent DATA messages. It only contains a two-byte sequence number, followed by the terminating zero. The format of an ACK message may be found below:



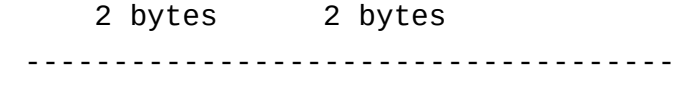
DATA

A DATA message is identified by the OPCODE 8, and is used to transmit the blocks of data for the file transfer. It includes a two-byte sequence number, followed by the data included in the message. The data field may be as long as the negotiated blksize, which is followed by the terminating zero-byte. It is important to note that a zero-byte does not indicate the end of the data field, as it may be possible for the data to include a zeo-byte. The format of a DATA message may be found below:



FIN and FINACK

A FIN message is identified by the OPCODE 9, and is used to signal a disconnection to the connected client or server. It includes an exit code to inform the application of the reason for the disconnection. The format of the FIN message may be found below:



```
|   OPCODE(9)  |  ExitCode  |    0    |
 --------------------------------------
```

On receipt of a FIN message, the client or server should acknowledge it and inform the other end by sending back a FINACK message. A FINACK message is identified by the OPCODE 10 and contains no additional data other than the terminating zero-byte. The format of the FINACK message may be found below:

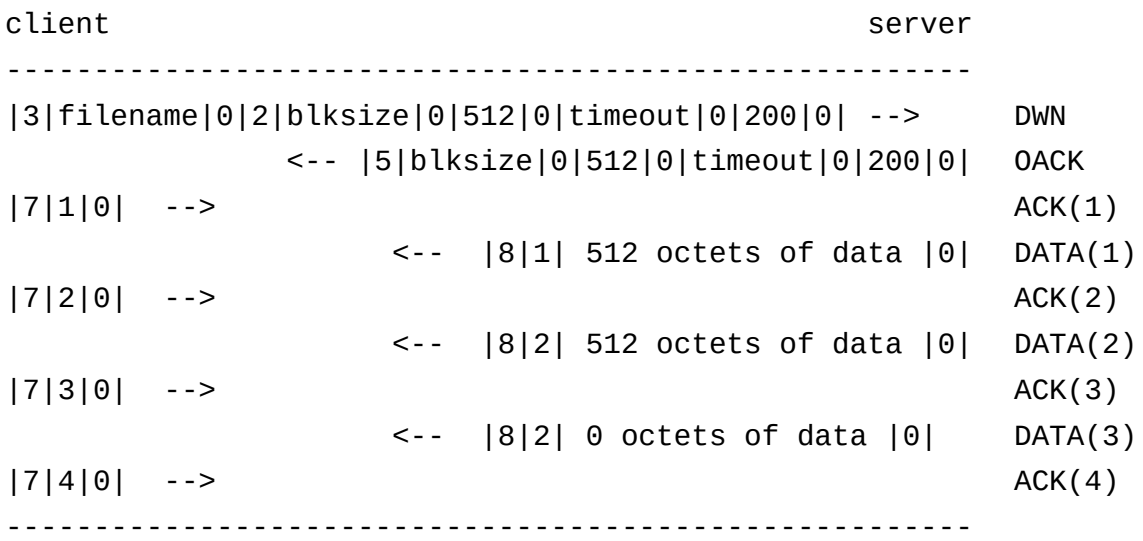
```
     2 bytes
 ------------------------
|   OPCODE(10)  |   0    |
 ------------------------
```

Examples

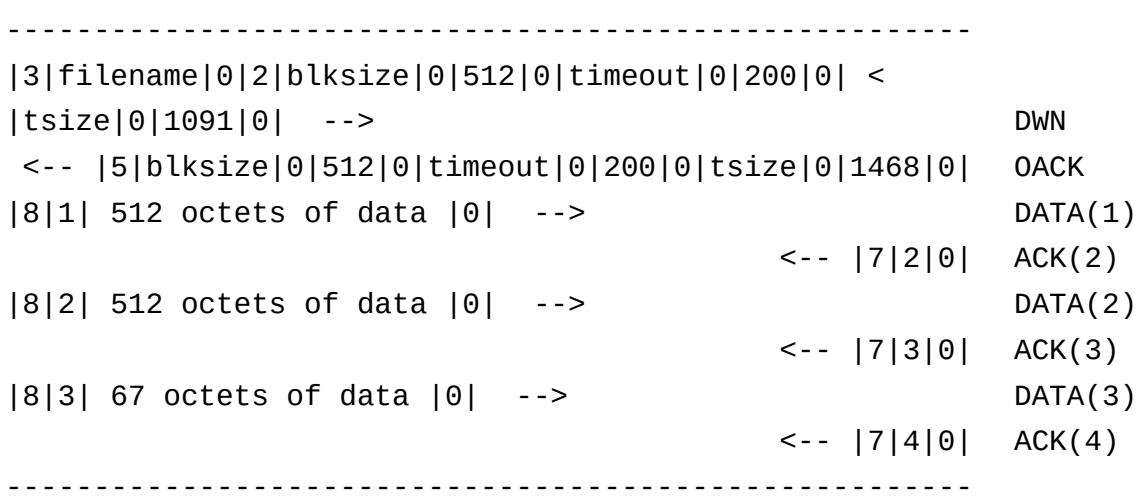
Connection and Disconnection

```
client                                                  server
-------------------------------------------------------------
|1|0|  -->                                                    SYN
                                          <--  |2|38508|0|   SYNACK
                           ...
|9|0|0|  -->                                                   FIN
                                               <--  |10|0|   FINACK
-------------------------------------------------------------
```

DWN Request



UPL Request



DWN Request (With Packet Delay)

```
client                                                server
-------------------------------------------------------
|3|filename|0|2|blksize|0|420|0|timeout|0|200|0| -->     DWN
                <-- |5|blksize|0|420|0|timeout|0|200|0|  OACK
|7|1|0|  -->                                             ACK(1)
                  [No data received]
|7|1|0|  -->                                             ACK(1)
                      <--  |8|1| 420 octets of data |0|   DATA(1)
|7|2|0|  -->                                             ACK(2)
                  [No data received]
|7|2|0|  -->                                             ACK(2)
                      <--  |8|2| 77 octets of data |0|    DATA(2)
|7|3|0|  -->                                             ACK(3)
-------------------------------------------------------
```

UPL Request (With packet dropping, max 3 attempts)

```
client                                                server
-------------------------------------------------------
|3|filename|0|2|blksize|0|1024|0|<
>|timeout|0|200|0|tsize|0|5679|0|  -->                    DWN
                     <-- |5|blksize|0|1024|0|timeout|0|<
                                 >|200|0|tsize|0|1468|0|  OACK
|8|1| 1024 octets of data |0|  -->                        DATA(1)
                  [No ack received]
|8|1| 1024 octets of data |0|  -->                        DATA(1)
                                            <-- |7|2|0|   ACK(2)
|8|2| 1024 octets of data |0|  -->                        DATA(2)
                  [No ack received]
|8|2| 1024 octets of data |0|  -->                        DATA(2)
                  [No ack received]
|8|2| 1024 octets of data |0|  -->                        DATA(2)
                  [No ack received]
|9|2|0|  -->                                             FIN
-------------------------------------------------------
```

APPENDIX

DWN and UPL Options

blksize
: Sets the DATA message’s transfer block size.

tsize
: Reports the size of the file being transferred to manage transfer expectations.

timeout
: Specifies the timeout period for retransmissions.

Error Codes

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	File already exists.
4	Disk full or allocation exceeded.
5	Invalid/incomplete options.
6	Ongoing Operation
7	Illegal operation.

Fin Message Exit Codes

Value	Meaning
0	No errors. Graceful exit.
1	Manual forceful termination.
2	Loss of connection.
255	Critical error(s) occurred.

REFERENCES

[RFC1350] Sollins, K., "The TFTP Protocol (Revision 2)", STD 33, RFC 1350, DOI 10.17487/RFC1350, July 1992, https://www.rfc-editor.org/info/rfc1350.

[RFC1350] Malkin, G. and A. Harkin, "TFTP Option Extension", RFC 2347, DOI 10.17487/RFC2347, May 1998, https://www.rfc-editor.org/info/rfc2347.