Iteration Test Plan

This document contains the test plan used to ensure compliance with the TFTP standard (RFC 1350). A description of the command line arguments and CLI commands available to each component is also included. Test plans from all iterations completed so far are included.

Description of Command Line Arguments and CLI commands

Client command line arguments

-t Enable test mode (transfers pass through the error simulator)
-q Enable quiet logging mode

Client CLI commands

```
  help         Prints this message
  cd     <directory> Change the directory that files will be written to or read from in the client
  read  <file>  Reads a file from a tftp server to the current working directory.
                 Optionally, the local destination file path may be specified.
  write <file>  Writes a file from the current working directory to a tftp server
 server <address> changes the ip address that the client will send packets to
 shutdown      Exits the client
```

Server CLI commands

```
  help         Prints this message
  cd   <directory> Change the directory that files will be written to or read from on the server
  shutdown      Exits the server
```

Error Simulator CLI commands

help	Prints a help message
shutdown	Exits the simulator
norm	Forward packets through without alteration
rend	Removes the end byte of the next request packet. Ie Removes the 0 Byte after Mode
rrs	Removes the Request Seperator of the next request packet. ie Removes 0 Byte after Filename
mode <mode>	Changes the mode of the next request packet, <mode> should be a string to use as a replacement
csa <type> <packetNum>	Changes the sender TID of a specified packet
op <type><packetNum><opCode>	Changes the opcode of the specified packet

cl <type><packetNum><packetLen>	Changes the length of a specified packet
delay <type><packetNum><numTimeouts>	Delays the specified packet by a number of timeouts
duplicate <type><packetNum>	Sends a duplicate of the specified packet
drop <type> <packetNum>	Drops the specified packet

Basic Transfer Tests (Iteration 1)

File Sizes:
0 byte
200 byte
512 byte
2048 byte
100 000 byte

After each transfer the following checks will be performed:
1.Ensure that the MD5 checksum of the file at the source and destination match
2.Attempt to move the file out of the source and destination directories to ensure the file is not in use

Test Set Up

Ensure that the server and client run in different directories. This is the case in Eclipse by d efault
1. Start the server
2. Start the client
3. Create a file in the server directory corresponding to each of the file sizes listed below

Test Steps

For each file:
1. Read the file using the client.
2. Write the file using the client.

Concurrent Connections Tests

The following will be repeated for both read and write
1. Start the server
2. Start one instance of the client
3. Start a second instance of the client
4. Start the transfer of a 1 MB file using the first client instance
5. Start the transfer of a small (200 byte) file using the second client instance while the first client is still running the transfer
6. Ensure that both transfers run to completion

Independent Implementation Tests

The purpose of these tests is to verify that the TFTP specification is followed by reading and writing files using an independent TFTP client and server

Server

1. Using an independent TFTP client, read a 2048 byte file
2. Using an independent TFTP client, write a 2048 byte file

Client

1. Using an independent TFTP server, read a 2048 byte file
2. Using an independent TFTP server, write a 2048 byte file

Iteration 2

1. Start the server
2. Start the error simulator
3. Start the client, providing it the -t command line option
4. Use the client to write a 2048 byte file
5. Use the client to read a 2048 byte file
6. Start the transfer of a file with the 'rend' command enabled
    - This should be an error 4
7. Start the transfer of a file with the 'norm' command enabled
8. Start the transfer of a file with the 'rrs' command enabled
    - This should be an error 4
9. For transfers with the 'mode' command enabled
    1. Set the mode to '.' –this should be an error 4
    2. Set the mode to 'netascii' – this should work
10. Using the *csa* command enter the following parameters for a write transfer:
    1. *req 1* – The second transfer on server should timeout
    2. Data 2 –Server will log an error 5
    3. Ack 2 -client will log an error 5

11. Using the *csa* command enter the following parameters for a read transfer:
    1. *req 1* – The second transfer on server should timeout
    2. Data 2 – Client will log an error 5
    3. Ack 2 - Server will log an error 5

12. Using the 'op' command enter the following parameters for a write transfer:
    1. req 1 10 – The client should receive an error 4
    2. data 2 4 – The client should receive an error 4
    3. ack 2 3 – The server should receive an error 4

13. Using the 'op' command enter the following parameters for a read transfer:
    1. req 1 10 – The client should receive an error 4
    2. data 2 4 – The server should receive an error 4
    3. ack 2 3 – The client should receive an error 4

14. Using the 'cl' command enter the following parameters for a write transfer:
    1. req 1 1 – The client should receive an error 4

2. data 2 40 – The transfer should end prematurely, client should timeout trying to send
3. ack 2 40 – The server should receive an error 4

15. Using the 'cl' command enter the following parameters for a read transfer:
    1. req 1 1 – The client should receive an error 4
    2. data 2 40 – The transfer should end prematurely, server should timeout trying to send
    3. ack 2 40 – The server should receive an error 4

Perform the Concurrent Connections test with the –t option passed to both client

Iteration 3

Case 1 - Lose a Packet

1. Using the ‘drop’ command, enter the following commands for a write transfer
    1. drop ack 1 – The client will timeout and resend the request, the first server thread will timeout and shut down, and a second server thread will complete the transfer
    2. drop ack <last-block-num + 1> – The client will timeout and resend the last data, the server will resend the dropped ack, and the transfer will complete
    3. drop ack 2 – The client will timeout and resend the data, the server will resend the dropped ack
    4. drop data 1 – The client will timeout and resend the data
    5. drop data <last-block-num> – The client will timeout and resend the data
    6. drop data 2 – The client will timeout and resend the data
    7. drop req 1 – The client will timeout and resend the request

2. Using the ‘drop’ command, enter the following commands for a read transfer
    1. drop ack 1 – The server will timeout and resend the data, and the client will resend the dropped ack
    2. drop ack <last-block-num> – The server will timeout and resend the last data, the client will resend the dropped ack, and the transfer will complete
    3. drop ack 2 – The server will timeout and resend the data, the client will resend the dropped ack
    4. drop data 1 – The server will timeout and resend the data
    5. drop data <last-block-num> – The server will timeout and resend the data
    6. drop data 2 – The server will timeout and resend the data
    7. drop req 1 – The client will timeout and terminate

Case 2 - Delay a Packet

1. Using the ‘delay’ command, enter the following commands for a write transfer
    1. delay ack 1 2 – The client will timeout two times, finally receive the delayed ack, and respond with a data

   2. delay ack 1 7 – The server will timeout five times and end the transfer. The client will timeout five times and resend the request. The server will respond to the request with an ack and the transfer will begin normally. The client will receive the delayed ack and respond with an error 5.
   3. delay ack <last-block-num + 1> 2 – The client will timeout and resend the data. The server will respond by resending the ack. The client will receive the ack and then end the transfer. The delayed ack will be lost since the client is no longer listening
   4. delay ack <last-block-num + 1> 7 – The client will timeout and resend the data. The server will respond by resending the ack. The client will receive the ack and then end the transfer. The delayed ack will be lost since the client is no longer listening
   5. delay ack 2 2 – The client will timeout and resend the data. The server will respond by resending the ack. The client will receive the ack and respond with a data. The delayed ack will be ignored when it is received
   6. delay ack 2 7 – The client will timeout and resend the data. The server will respond by resending the ack. The client will receive the ack and respond with a data. The delayed ack will be ignored when it is received
   7. delay data 1 2 – The client will timeout and resend the data. The server will respond to the data with an ack. The server will receive the delayed data and respond with an ack. The client will respond to the first ack with a data and ignore the second ack.
   8. delay data 1 7 – The client will timeout and resend the data. The server will respond to the data with an ack. The server will receive the delayed data and respond with an ack. The client will respond to the first ack with a data and ignore the second ack.
   9. delay data <last-block-num> 2 – The client will timeout and resend the data. The server will respond to the data with an ack. The client will receive the ack and the transfer will end. The delayed data will be lost because the server is no longer listening.
   10. delay data <last-block-num> 7 – The client will timeout and resend the data. The server will respond to the data with an ack. The client will receive the ack and the transfer will end. The delayed data will be lost because the server is no longer listening.
   11. delay data 2 2 – The client will timeout and resend the data. The server will respond to the data with an ack. The server will receive the delayed data and respond with an ack. The client will respond to the first ack with a data and ignore the second ack.
   12. delay data 2 7 – The client will timeout and resend the data. The server will respond to the data with an ack. The server will receive the delayed data and respond with an ack. The client will respond to the first ack with a data and ignore the second ack.

2. Using the 'delay' command, enter the following commands for a read transfer
   1. delay ack 1 2 – The server will timeout and resend the data. The client will resend the ack. The server will respond to the resent ack with a data, and the delayed ack will be ignored.
   2. delay ack 1 7 – The server will timeout and resend the data. The client will resend the ack. The server will respond to the resent ack with a data, and the delayed ack will be ignored.
   3. delay ack <last-block-num> 2 – The server will timeout and resend the data. The client will respond by resending the ack. The server will receive the ack and then end the transfer. The delayed ack will be lost since the server is no longer listening
   4. delay ack <last-block-num> 7 – The server will timeout and resend the data. The client will respond by resending the ack. The server will receive the ack and then end the transfer. The delayed ack will be lost since the server is no longer listening

5. delay ack 2 2 – The server will timeout and resend the data. The client will respond by resending the ack. The server will receive the ack and respond with a data. The delayed ack will be ignored when it is received
6. delay ack 2 7 – The server will timeout and resend the data. The client will respond by resending the ack. The server will receive the ack and respond with a data. The delayed ack will be ignored when it is received
7. delay data 1 2 – The server will timeout and resend the data. The client will respond to the data with an ack. The client will receive the delayed data and respond with an ack. The server will respond to the first ack with a data and ignore the second ack.
8. delay data 1 7 – The server will timeout and resend the data. The client will respond to the data with an ack. The client will receive the delayed data and respond with an ack. The server will respond to the first ack with a data and ignore the second ack.
9. delay data <last-block-num> 2 – The server will timeout and resend the data. The client will respond to the data with an ack. The server will receive the ack and the transfer will end. The delayed data will be lost because the client is no longer listening.
10. delay data <last-block-num> 7 – The server will timeout and resend the data. The client will respond to the data with an ack. The server will receive the ack and the transfer will end. The delayed data will be lost because the client is no longer listening.
11. delay data 2 2 – The server will timeout and resend the data. The client will respond to the data with an ack. The client will receive the delayed data and respond with an ack. The server will respond to the first ack with a data and ignore the second ack.
12. delay data 2 7 – The server will timeout and resend the data. The client will respond to the data with an ack. The client will receive the delayed data and respond with an ack. The server will respond to the first ack with a data and ignore the second ack.

Case 3 - Duplicate a Packet

1. Using the ‘duplicate’ command, enter the following commands for a write transfer
    1. duplicate ack 1 – The client will respond to the original ack with a data, and will ignore the duplicate ack
    2. duplicate ack <last-block-num + 1> – The client will end the transfer after receiving the original ack, and the duplicate will get lost since the client is no longer listening
    3. duplicate ack 2 – The client will respond to the original ack with a data, and will ignore the duplicate ack
    4. duplicate data 1 – The server will respond to the original data with an ack, and will respond to the duplicate data with the same ack. The client will respond to the first ack with a data, and ignore the second ack
    5. duplicate data <last-block-num> – The server will respond to the original data with an ack, and will respond to the duplicate data with the same ack. The client will respond to the first ack with a data, and ignore the second ack
    6. duplicate data 2 – The server will respond to the original data with an ack, and will respond to the duplicate data with the same ack. The client will respond to the first ack with a data, and ignore the second ack
    7. duplicate req 1 – The server will respond to the first request with an ack and respond to the duplicated request with an error 6, since the file already exists

2. Using the ‘duplicate’ command, enter the following commands for a read transfer
   1. duplicate ack 1 – The server will respond to the original ack with a data, and will ignore the duplicate ack
   2. duplicate ack <last-block-num> – The server will end the transfer after receiving the original ack, and the duplicate will get lost since the server is no longer listening
   3. duplicate ack 2 – The server will respond to the original ack with a data, and will ignore the duplicate ack
   4. duplicate data 1 – The client will respond to the original data with an ack, and will respond to the duplicate data with the same ack. The server will respond to the first ack with a data, and ignore the second ack
   5. duplicate data <last-block-num> – The client will respond to the original data with an ack, and will respond to the duplicate data with the same ack. The server will respond to the first ack with a data, and ignore the second ack
   6. duplicate data 2 – The client will respond to the original data with an ack, and will respond to the duplicate data with the same ack. The server will respond to the first ack with a data, and ignore the second ack
   7. duplicate req 1 – The server will respond to both requests by creating two transfer threads and both will send a data from different ports. The client will respond to the first data it receives with an ack, and will respond to the data from a different port with an error 5.

Iteration 4

For each of the following test cases, the client and server *cd* commands may be used to specify the source and destination directory of both the client and the server. The command “cd .” may be used to return the source and destination directory to the current working directory of the program.

Error Code 1 Scenarios - File not Found

Case 1 - WRQ File not present in client directory

1. Attempt to write a file from the client which does not exist in the client’s working directory
2. Ensure that the client CLI displays a message informing the user that the file could not be found
3. Ensure that the server did not receive a WRQ from the client

Case 2 - RRQ File not present in server directory

1. Attempt to read a file which does not exist in the server’s working directory
2. Ensure that the client CLI displays an error message informing the user that the file could not be found
3. Check the client logs to ensure that an Error packet with code 1 was received by the client

Error Code 2 Scenarios - Access Violation

Case 3 - WRQ Write to a read-only folder on server

1. Set up a directory which the current user does not have permissions to write to
2. Use the server *cd* command to change to the directory created in step 1.
3. Start a write transfer from the client.
4. Check the server log to ensure that the transfe r has stopped.
5. Ensure that the client CLI displays an appropriate error message (Access violation)
6. Check the client logs to ensure that an Error packet with code 2 was received by the client

Case 4 - WRQ Write from a client directory without read permission

1. Set up a directory which the current user does not have permissions to read from
2. Use the client *cd* command to change to the directory created in step 1
3. Start a read transfer from the client. In the CLI command to start this transfer, specify a filename in the directory from step 3 as the source file name.
4. Check the server log to ensure that the server did not receive a WRQ packet
5. Ensure that the client CLI displays an appropriate error message (Access violation)

Case 5 - RRQ Read from server directory without read permission

1. Set up a directory which the current user does not have permissions to read from
2. Use the server *cd* command to change to the directory created in step 1.
3. Start a read transfer from the client.
4. Ensure that the transfer terminates on the server side
5. Ensure that the client displays an appropriate error message (Access violation)
6. Check the client log to ensure that an Error packet with code 2 was received by the client

Case 6 - RRQ Read to client directory without write permission

1. Set up a directory which the current user does not have permissions to write to
2. Use the client *cd* command to switch to the directory created in step 1.
3. Start a read transfer from the client.
4. Ensure that the client displays an appropriate error message (Access violation)
5. Check the server log to ensure that the server did not receive a RRQ packet

Error Code 3 Scenarios - Disk full or allocation exceeded

Each of the following scenarios require the use of a disk which has very little space remaining. Such a disk can be created quickly using a small USB stick. Use fsutil to create a large empty file to occupy most remaining space.

Case 7 - WRQ disk full

1. Use the server's *cd* command to set the output directory to a disk which is full
2. Start a write transfer from the client
3. Check the server log to ensure that the transfer has terminated with an appropriate error message
4. Ensure that an appropriate error message is displayed by the client
5. Check the client logs to ensure that an Error packet with code 3 was received by the client
6. Ensure that no empty or incomplete file exists in the server's output directory

Case 8 - RRQ disk full

1. Use the client *cd* command to change the output directory to a disk which is full
2. Start a read transfer.
3. Ensure that the transfer on the client side has terminated with an appropriate error message
4. Ensure that the transfer on the server side has terminated with an appropriate error message
5. Check the server logs to ensure that an Error packet with code 3 was received
6. Check the destination path of the Client to ensure that no empty or incomplete file exists there

Error Code 6 Scenarios - File already exists

Case 9 - WRQ file already exists on server

1. Start a write transfer. Specify a file which already exists in the server's output directory.
2. Ensure that the transfer on the server side has terminated with an appropriate error message
3. Ensure that the transfer on the client side has terminated with an appropriate error message
4. Check the client log to ensure that the client received an error message with code 6

Case 10 - RRQ file already exists on client

1. Start a read transfer. Specify a file which already exists in the client's working directory.
2. Check the server log to ensure that no RRQ was received by the server
3. Ensure that the transfer on the client side has terminated with an appropriate error message.

Iteration 5

Each of the following scenarios require that the client and server be on separate computers and that the error simulator is on the same computer as the server. All of the following use the *server <address>* command

Case 1- no error simulator
1. Start the server
2. Start the client
3. Use the *server* command and specify the server's ip address
4. Start a transfer

Case 2- with error simulator
1. Start the server
2. Start the errorSimulator
3. Start the client with –t flag
4. Use the *server* command and specify the server's ip address
5. Start a transfer

Case 3- concurrent transfers

1. Start the server
2. Start the errorSimulator
3. Start a client with –t flag
4. Star a client with no –t flag
4. Use the *server* command and specify the server's ip address
5. Start a transfer from each client with files large enough that the transfers run concurrently