TFTP — Trivial File Transfer Protocol
Team 3000000

Shasthra Ranasinghe
#100867803

Aritra Sengupta
#100921432

Ismail Syed
#100923110

Mohamed Zalat
#100968390

Kunall Banerjee
#100978717

December 6, 2016

Contents

1 Summary

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each non-terminal packet is acknowledged separately. TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP.

1.1 Purpose

TFTP is implemented on top of the Internet User Datagram protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP. (This should not exclude the possibility of implementing TFTP on top of other datagram protocols.) It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data. Our implementation of TFTP does not support any modes of transfer, however, according to the spec, two modes of transfer are currently supported: `netascii` and `octet`.

1.2 Design — Where to run the error simulator

After careful consideration, we decided to run the error simulator on the same PC as the server. This way, we need to only specify the IP of the client every time we ran the server.

2 Setup instructions

2.1 Importing the Project into Eclipse

1. Open the project in `Eclipse`
2. Select File → Import then General → Existing Projects into your workspace
3. For the root directory select the submitted folder `SYSC3303_Iteration5`

2.2 Running

1. Run `TFTPClient.java`
   (a) Type `DEFAULT` to choose the directory the program is running in or type in a path to a directory (the directory can be changed by typing "cd"). The directory chosen in this step will be where files transferred from the server to the client will be stored.
   (b) Choose to keep `verbose` mode on or off by typing `Y` or `N`
   (c) To use test mode type `TEST` to use normal mode type `NORMAL`
   (d) Choose between Read or Write request by typing `R` or `W`
   (e) Type file name of a file existing in the directory chosen in step (a)
2. Run `TFTPSim.java`
   (a) Choose between normal mode, network error or illegal TFTP operation mode
   (b) Follow the instructions on the screen that follow
3. Run `TFTPServer.java`
   (a) Type `DEFAULT` to choose the directory the program is running in or type in a path to a directory (the directory can be changed by typing "cd"). The directory chosen in this step will be where files transferred from the server to the client will be stored.
   (b) Choose to keep `verbose` mode on or off by typing `Y` or `N`
4. To close the server or the client type `QUIT`

Note: Files can **NOT** be overwritten on the server *or* client

3 Breakdown of responsibilities

3.1 Shasthra Ranasinghe

3.1.1 Iteration 1

- Server side implementation
- Writing the `DataPacket`, `Reader` and `Writer` classes
- Implementing steady-state file transfer from the server
- Implementing threading on the server
- Modifying the Error Simulator to work with `DATA` and `ACK` packets with a multi-threaded server

3.1.2 Iteration 2

- Client side implementation
- Error checking on the client-side
- Error simulator modifications
- Fixed issues and bugs from `iteration 1`

3.1.3 Iteration 3

- Project management and coordination
- UML class diagram
- Timing diagrams for all possible cases

3.1.4 Iteration 4

- Error simulator
  - Refactored the error simulator
  - Added support for error packets 4 & 5

3.1.5 Iteration 5

- Testing
- Bug fixes in the error simulator

3.2 Aritra Sengupta

3.2.1 Iteration 1

- Writing the `Exceptions`, `FileIO` and `TFTPPackets` packages
- Implementing steady-state file transfer from the server
- Implementing threading on the server

3.2.2 Iteration 2

- Project management and coordination
- UML class diagram
- Timing diagrams for error codes `01, 02, 03` and `06`
- Updated UCMs for `READ` and `WRITE` file transfers

3.2.3 Iteration 3

- Server side implementation
- Refactored server code
- Modified the server so it drops duplicate packets
- Modified the server to resend `DATA` packets if it does not receive an `ACK` from the client

3.2.4 Iteration 4

- Server side implementation
- Refactored server-side code

3.2.5 Iteration 5

- Bug fixes on the server-side

3.3 Ismail Syed

3.3.1 Iteration 1

- Project management and coordination
- UML class diagram
- UCM diagrams

3.3.2 Iteration 2

- Client side implementation
- Fixed issues and bugs from `iteration 1`

3.3.3 Iteration 3

- Error Simulator implementation
- Refactoring of the entire simulator
- Thread creation implementation
- Designed, implemented and added the ability to simulate network errors

3.3.4 Iteration 4

- Error simulator
  - Refactored the error simulator
  - Added support for error packets 4 & 5

3.3.5 Iteration 5

- Testing
- Bug fixes in the error simulator

3.4 Mohamed Zalat

3.4.1 Iteration 1

- Client side implementation
- Allow client to read and write to files
- Added verbose and quite mode
- Added test and normal mode
- Client responds to data and acknowledgment packets appropriately

3.4.2 Iteration 2

- Server side implementation
- Created the whole class structure for the `ErrorPacket` class
- Exceptions are caught gracefully & translated to proper TFTP error codes

3.4.3 Iteration 3

- Client side implementation
- Added re-sending of data packets and request(s) to client
- Added dropping for duplicate `ACK` to client
- Client responds to duplicate and old `DATA` packets with `ACK` while not writing to client
- Added proper file handling to client

3.4.4 Iteration 4

- Client side implementation
- Refactored client-side code

3.4.5 Iteration 5

- Timing diagrams
- Bug fixes on client-side

3.5 Kunall Banerjee

3.5.1 Iteration 1

- Client side implementation
- Client is able to read and write to files
- Added verbose and quite mode
- Added test and normal mode
- Client responds to data and acknowledgment packets appropriately

3.5.2 Iteration 2

- Server side implementation
- Created the whole class structure for the `ErrorPacket` class
- Exceptions are caught gracefully & translated to proper TFTP error codes
- Fixed issues raised in feedback obtained for `iteration 1`

3.5.3 Iteration 3

- Error Simulator implementation
- Refactoring of the entire simulator
- Thread creation implementation
- Designed, implemented and added the ability to simulate network errors

3.5.4 Iteration 4

- Project management and coordination
- Worked briefly on the error simulator
- Added TFTP Packet format errors 4 & 5
- Timing Diagrams

3.5.5 Iteration 5

- Project management and coordination
- Bug fixes
- Testing

4 Testing

4.1 Preface

Complete set-up instructions above before continuing with the tests. **NOTE**: The numbers in the files denote how big the files are in bytes.

4.2 Testing the `Client`

1. Enter `DEFAULT` to use the default directory or type in a complete path to use that directory instead
2. Enter `DEFAULT` to use the local `IP` or enter the `IP` of the server/simulator
3. Choose between `verbose` or `quiet` mode
4. Choose between `NORMAL` or `TEST` mode
5. Choose between a read (`R`) or write (`W`) request
6. Enter a file name (along with its extension)
7. Press enter to start transfer!

NOTE: You can change the directory at any point by typing in `cd` followed by the path name. You can change the `IP` the same way by typing in `cip` followed by the full `IP` address.
Also **NOTE**: The simulator needs to be running in `normal` mode for it to act as a proxy.

4.3 Testing the `Simulator`

4.3.1 Choosing a mode

1. Choose between `normal` mode, `network error` mode or `invalid TFTP` mode

4.3.2 Simulator running in `normal` mode

1. If you chose `normal` mode, the simulator just acts as a proxy and just passes all requests that come its way to either end

4.3.3 Simulator running in `network error` mode

Choose between losing a packet, delaying a packet & duplicating a packet.

1. If you chose to lose a packet, then
    (a) Choose which packet to lose from the following: `DATA, ACK, RRQ, WRQ`

i. DATA
Will prompt the user with the block number for which the error simulator should simulate the lost packet on

ii. ACK
will prompt the user with the block number for which the error simulator should simulate the lost packet on

iii. RRQ
Selecting RRQ Packet will lose read requests being sent from the client

iv. WRQ
Selecting WRQ Packet will lose write requests being sent from the client

2. If you chose to delay a packet, then

   (a) Choose which packet to delay from the following: `DATA, ACK, RRQ, WRQ`

      i. DATA
      Will prompt the user with the block number for which the error simulator should simulate the delay packet on. The simulator will then prompt the user with the delay length for the packet

      ii. ACK
      Will prompt the user with the block number for which the error simulator should simulate the delay packet on. The simulator will then prompt the user with the delay length for the packet

      iii. RRQ
      Will delay read requests being sent from the client. The simulator will then prompt the user with the delay length for the packet

      iv. WRQ
      Will delay write requests being sent from the client. The simulator will then prompt the user with the delay length for the packet

3. If you chose to duplicate a packet, then

   (a) Choose which packet to duplicate from the following: `DATA, ACK, RRQ, WRQ`

      i. DATA
      Will prompt the user with the block number on which the error simulator should simulate the duplicate packet on

      ii. ACK
      Will prompt the user with the block number on which the error simulator should simulate the duplicate packet on

iii. RRQ
Will duplicate read requests being sent from the client

iv. WRQ
Will duplicate write requests being sent from the client

4.3.4 Simulator running in `invalid TFTP` mode

1. Choose to generate an `illegal TFTP op`, i.e., corrupt a `TFTP` packet or generate an `unknown TID`
2. If you chose to generate an `unknown TID` error, then
   (a) Choose a packet from the following: `DATA, ACK, RRQ, WRQ, ERROR`
      i. The simulator will proceed to generate an unknown TID at the specified `TFTP` packet
3. If you chose to invalidate a `TFTP` packet, i.e, to corrupt it, then
   (a) Choose a packet from the following: `DATA, ACK, RRQ, WRQ, ERROR` Choose:
      i. Generate an invalid opcode
      ii. Send extra data (than is permitted)
      iii. Missing file name
      iv. Missing first `0`
      v. Missing mode
      vi. Corrupt mode
      vii. Missing second `0`
      viii. Add a zero anywhere between the `byte` array
   (b) The simulator will then proceed to corrupt the packet accordingly

4.4 Testing the `Server`

1. Enter `DEFAULT` to use the default directory or type in a complete path to use that directory instead
2. Choose between `verbose` or `quiet` mode
3. At this point, the server is ready to accept new connections and is waiting for a transfer

NOTE: You can change the directory at any point by typing in `cd` followed by the path name. You can shut-down the server by typing in `QUIT` at any point during a transfer (or even otherwise)