COMP312-09A - Trivial File Transfer Protocol (TFTP)

March 4, 2009

1 Introduction

This practical will introduce you to (1) UDP sockets in Java, (2) application-level reliability concepts by implementing TFTP, and (3) the concept of interoperability.

UDP Sockets: UDP sockets provide an unreliable, packet-oriented transport. In 202, you used TCP sockets, which provided a connection-oriented, reliable, byte-stream service. Java provides access to the UDP transport with the DatagramSocket and DatagramPacket classes. Please familiarise yourself with these Java classes.

TFTP: TFTP is a simple protocol for reading and writing files to and from a server. It is defined in RFC 1350, which is available on the Internet, and on COMP312's Moodle webpage. The RFC is fairly hard going at first glance, though combined with the TFTP page on Wikipedia you have all the information necessary to implement the protocol. It took me about 200 lines of Java code to implement the ability to send a file from the server, and about the same number for the client.

Interoperability: An important part of developing and implementing a standard is to test if an implementation works with an implementation developed by someone else. Doing so allows the quality of the standards document to be assessed for abiguities so that it may be refined if necessary. For this practical, you will test your client and server implementations with another COMP312 student's implementation in the lab.

Your assignment will be verified in person in R-block Lab 6, up until Friday March 20th. I will be in the lab to answer questions and give assistance understanding the RFC on Fridays between 1 and 2pm (after Friday's lecture). It is worth 10% of your final grade. While you are required to test interoperability with a class mate, this is an individual assignment.

2 Advice

I recommend you implement your TFTP server using threads, rather than using the DatagramChannel (select) approach, as doing so should be much simpler. A skeleton TftpServer.java is provided, so that you may focus on comprehending and implementing the RFC.

The TFTP RFC says the TFTP server should listen on port 69. As you will not be able to use this port in the lab, I recommend you allow your program to bind to a free port of its choosing (as indicated in the skeleton implementation I have provided) and provide the port number used as an argument to your TftpClient.

The TFTP RFC specifies multiple transfer modes, being “netascii”, “octet”, and “mail”. You are only required to implement the octet (the simplest) method. Octet means you transfer and record the file exactly as is recorded on disk. Your TFTP client MUST send the octet mode string in its RRQ packet, and your TFTP server MUST check the mode string in the received RRQ packet.

The TFTP RFC allows files to be uploaded, using the WRQ method. You MAY implement this if you want to, though you are NOT REQUIRED to.

You need to implement a retransmission method, in case a data packet is lost. You should use the DatagramSocket::setSoTimeout method for this, and allow a single packet to be retransmitted up to three times.

You should check the integrity of received files (and therefore your implementation) using the md5sum utility. Try sending image files rather than just text files, as image files will demonstrate corruption a lot more obviously than a text file might.

3 Questions

To complete the implementations, you will need to know the answers to the following questions. Please write answers in the space provided as this forms part of the verification process.

Question 1: What is the maximum payload permitted when transferring a piece of data? What size byte array, therefore, should you allocate for sending a TFTP data packet?

Question 2: TFTP requests are sent to a pre-defined port, though the server responds, and sends data from, a different port. Why is this?

Question 3: UDP does not have the ability to signal end of file, as there is no disconnection process. How does TFTP signal end-of-file?

Question 4: You are provided with a TftpServer.java file that has store_short and extract_short methods. What does each of these methods do? Why are they necessary?

Question 5: Who did you test interoperability with? Did you find any problems with either of your implementations? Write about your interoperability testing experiences here.

Assignment 1 Mark sheet

Name: __________

Id: ___________

Date: ___________

Questions answered correctly.

(2 marks)

TftpServer implementation. TftpServer works, is well written, does not have serious flaws, and is well commented.

(3 marks)

TftpClient implementation. TftpClient works, is well written, does not have serious flaws, and is well commented.

(3 marks)

Interoperability testing.
Interoperability with a classmate demonstrated in the lab.

(2 marks)