Assignment No 2 version 1.0
Release Date: 16th Nov 2014
Due date: [various]
Midnight 14th Dec 2014 and 04th Jan 2015

1.0 Objective

This assignment should provide you with experience in network programming with sockets, client/server systems, and testing a network system. You will build a client/server pair that will execute on the SEECS lab machines running Ubuntu Linux.

2.0 General

This programming assignment is to be completed in groups of two students (same group from Assignment 1). You are required to use Python as your programming language.

The assignment would be graded in incremental steps (see Section 4 for details) and carries a total of 30 marks. Your group demo would be held in last week of the semester, followed by individual viva. You would be asked questions about your implementation to ascertain the amount of work done by each group member. Each group member would be graded separately.

Queries regarding the assignment specifications should be posted on the forum at Piazza. Please do not post your questions to the email addresses of the course faculty/staff.

3.0 Specifications

Your task is to code a client/server pair that allows the client to transfer a single file to a server's local directory. The client/server communication is to run over UDP on a _VERY_ unreliable network and one of your main goals is to deliver the files successfully despite this unreliability.

Specifically, this network exhibits a tendency to:

- Drop packets
- Duplicate packets
- Create bit errors within packets (a single bit error)
- Delivers out of order packets

You should be able to overcome this un-reliability using standard techniques (such as simple retransmissions). Consider your options and start simple then work up. Before you even attempt to combat unreliability, you should have implemented basic communications over a reliable network first.

The client and server must run as 2 independent applications under Linux. The Server program will be named "MyServer" and the client will be called the "MyClient", ie: you will write Python files named MyServer.py and a client file named MyClient.py.

- **The client and server must communicate using UDP:** In Python, network programing support is accessed through the socket module. Unfortunately the regular UDP sendTo( ) call is too reliable

for our testing *(the irony!);* so in order to simulate a ***_VERY_ unreliable*** network conditions, you will need to implement simulated bad network conditions in software. You will be provided with examples of a BadNet Python classes to do just this -- a BadNet class will have **one** method of importance: a **"transmit"** method that you must use to send your packets between the client and server instead of the usual sendTo( ) call. This BadNet transmit method will then be tweaked to exhibit the desired network behavior. This way we can test your code. Having said that, it may be worthwhile to begin by using regular socket sendTo( ) in your basic coding and removing it later when you are ready to start handling network unreliability. Alternatively it should be just as easy to start the assignment with a BadNet.transmit method that does nothing but uses regular sendTo( ) call.

The transmit method is declared as:
def transmit (csocket, message, serverName, serverPort)
Note that the transmit method does not return anything. If you need any kind of feedback, implement it in your client and/or server.

The method is static, so can be called using the Class name in your Client and Server:
BadNet.transmit(clientSocket, partMessage, serverName, serverPort)

- **Classes for your testing:** 6 python classes are being released for your testing purposes. Each class implements one or more of the BadNet behavior. The details of these classes are;

  - **BadNet0:** (rather a GoodNet) Does not cause any packet drop, duplication or error
  - **BadNet1:** Drops every 5th packet
  - **BadNet2:** Errors every 5th Packet
  - **BadNet3:** Duplicates every 5th Packet
  - **BadNet4:** Re-orders packets, every 5th packet
  - **BadNet5:** Mix of errors, duplicates, drops and re-ordering– every 5th packet

  Each BadNet class has different behavior, which can be used to test your program. You can use them by placing them one at a time into your working directory with the client and/or server and using proper ***imports*** to use them. Note that both client and server will be subject to the same network conditions, both have to import and use the Badnets.

- **Your Program should successfully demonstrate:**
  - Transporting a range of file sizes up to 2MB.
  - Your implementation efficiency does count.
  - Transferring a file across a reliable network (no errors, drops, duplicates etc)
  - Dealing with varying probabilities of:
    a) Dropped packets
       -- by retransmission
    b) Duplicate packets.
       -- by discarding packets
    c) Errors in packets
       -- by retransmission
    d) Out-of-order arrivals
       -- by re-ordering
    e) Any combination of a), b), c) and d) above

- **Testing class would be different:** During demo and your viva, yet another BadNet class, different to the ones for your testing, would be provided and used. Thus, no BadNet class should be present in your final code, and you must use the one supplied on the day. We advise you to think of ways to improve the provided BadNet testing classes, in order that you are not disappointed with your results after your performance is tested with a more thorough BadNet.

- **Transmitted packets must be <= 1KB:** This means that if you are transferring a file of size 100KB and select 1 KB as the packet size, you would chop the total file in approx. 100 pieces and try to send theses pieces one by one. The packet formats and higher-level protocols you choose to use between the client and server are completely up to you, they should be designed to allow you to ***effectively*** and ***efficiently*** combat the problems in the network.

- **The Server:** The Server would receive the file sent by the client and it should store it in it's local directory. You should therefore start the client and server from different directories. The Server starts with a call to MyServer.py, this call is provided with single command line argument specifying the Server port No. For example,

  python MyServer.py 16000

  It is suggested that you use the port Nos within the range 10000-32000 for your code testing. Above and below this range, there is a greater chance of colliding with a port already in use.

  For measuring the performance of your implementation, you should start a timer at the server as soon as you receive the connection request from the client (first thing after the accept call) and stop it when you finally close the connection (first thing after the close call). Difference between these two values would show how fast your implementation is. A sample pdf file (PLT.pdf) is being issued that you can use to benchmark the performance of your implementation with successive versions of your software and with your peers. Obviously an efficient and fast implementation would get more credit.

- **The Client:** The Client starts with MyClient.py, this call is provided with several command line arguments:

  1. The port number for the server
  2. The file name to be transferred to the server

  eg: ***python MyClient.py 16000 testfile***

- **Stop and wait vs pipelined implementation:** As announced earlier, I would accept the stop and wait implementation as well but there would be marks deduction for this implementation. I would encourage you to model a pipelined protocol.

4.0 Submission & Report:

To keep you involved with this assignment, there are two deadlines. The details of the deadlines and the respective deliverables are given below.

Deadline	Deliverable	Grading
14th Dec 2014	BadNet0+BadNet1+BadNet2+ Report (including your FSM)	Max of 10
04th Jan, 2015	Complete (All BadNets + Final report)	Max of 20

You will be required to submit your source code and a short report to an upload link that would be made available on the LMS on each incremental deadline. Zip your source code files and your report document in a file named assign2_Surname_Surname (Surname of both Group members). Nominate one of the group members to submit on behalf of the group. Please note that you must upload your submission BEFORE the deadline. The LMS would stop accepting submissions after the due date. Late submissions (emailed to your instructor) would carry 15% marks penalty per day with maximum of 2 days late submission allowed. Students who miss the final grading lab would not be allowed to retake the viva.

Your submitted code would be checked for similarities and any instances of plagiarism would result in award of ZERO marks for all such students, irrespective of who has shared with whom. No exceptions!!

You must write down group members' Registration No's and Names at the beginning of the report **!!** All reports will be read for marking purposes. Your report should invariably have the FSM model of your implementation.

The size of your report **MUST be under 2 pages**. Your report should briefly document your techniques and methodology used to combat the relevant problems in the network. Treat it as a summary document only (point form is acceptable). It should act a reference for your instructor to quickly figure out what you have and haven't completed, how you did it, and it should mention anything you think is special about your system. You will be asked to demonstrate your program during your viva.