CSCI 3650 / 5550 - ECE 4245 Feb 10, 2020

Programming Assignment 1

Due on Wed Feb 19 - 12:00 PM (Beginning of Class) *Spring 2020*

Using an Echo Application to Measure TCP Performance

To be completed individually.

In this assignment, you will first write a program that uses the socket interface to send messages between a pair of processes on different machines, namely, a client and a server. This first part will familiarize you with basic socket programming if you have not been exposed to it before. Then, you will use this program to measure the round trip time and throughput of a TCP connection between the client and the server. The second part of the assignment is meant to introduce you to basic network measurements.

Part I: Writing an Echo Client-Server Application

Overview: For this part, you will implement a client and a server that communicate over the network using TCP. The server is essentially an echo server, which simply echoes the message it receives from the client. Here is what the server and client must do:

- The server should accept, as a command line argument, a port number that it will run at. To run your server on our hopper.slu.edu, we have opened up ports 58000-58999 so please pick a port in this range for your server. If you decide to use port 58000, make sure nobody else is currently using your port on hopper, for example using the command `netstat` and grepping for port 58000. `netstat | grep 58000`. After being started, the server should repeatedly accept an input message from a client and send back the same message. You do NOT have to use hopper for both machines for this assignment, but your own machine will work too.
- The client should accept, as command line arguments, a host name (or IP address), as well as a port number for the server. Using this information, it creates a connection (using TCP) with the server, which should be running already. The client program then sends a message (text string) to the server using the connection. When it receives back the message, it prints it and exits.

In Part I, the message is simply a text string with no specific format. In other words, any text message will do. You may use C, C++, Java, or Python to build your client and server programs. In C/C++, you should familiarize yourselves with the following system calls: socket(), bind(), listen(), accept(), connect(), send() and recv(). We outline a number of resources below with additional information on these system calls, as well as their equivalent in several other programming languages.

What to turn in: The programs you submit should work correctly and be well documented with a record of the information exchanged between your client and server. It would be a good idea to test your client program with the server program implemented by a classmate, or vice-versa. This is possible because your programs should adhere to the same exchange protocol described above (and below for the second part of this assignment). This is also an interesting way to see how protocol entities can "interoperate" if they accurately implement the same protocol specification. You must, however, implement both the client and server on your own. The instructor or the grader will use anti-plagiarism software to detect code similarity of different submissions. You need not submit a hard copy of your program listings (code). See instructions below on how to submit your assignment.

Resources:

- You can develop your programs on any machine, however, you may test your programs on our Linux machines *i.e.*, hopper.slu.edu, linuxlab01.mcs.slu.edu - linuxlab13.mcs.slu.edu (RTH 117) or linuxclassroom01.mcs.slu.edu - linuxclassroom24.mcs.slu.edu (RTH 115) to ensure they will be graded correctly.
- This primer (`https://goo.gl/avmait`) is an excellent introduction to BSD sockets. A good book is also available (`http://www.kohala.com/start/unpv12e.html`)
- A short tutorial on socket programming in python `https://docs.python.org/3/howto/sockets.html`
- A short tutorial on socket programming, from the University of Wisconsin: `http://cs.slu.edu/~esposito/teaching/3650/sock.pdf`
- Additional socket programming links:
  - `http://compnetworking.about.com/cs/socketprogramming/`
  - Network Programming with Sockets (`http://www.lowtek.com/sockets/`)
  - Last but not least, if you are using Python, see Section 2.7 of the textbook on socket programming and Python examples.

Part II: Performing RTT and Throughput Measurements

You may use any library you want but I recommend you to check our the SCAPY library to implement this protocol before you start. `https://scapy.net/`.

Overview: In this part, you will extend the echo application implemented in Part I to measure the round trip time (RTT) and throughput of the path connecting the client to the server. To measure RTT, you will use TCP to send and receive messages of size 1, 100, 200, 400, 800 and 1000 bytes. To measure throughput, you will use TCP to send and receive messages of size 1K, 2K, 4K, 8K, 16K and 32K bytes. Note the difference in units for the two sets of message sizes. For each measurement and for each message size, the client will send at least ten probe messages to the server, which will echo back the messages.

Protocol phases: As in Part I, the client first needs to set up a TCP connection to the server using the socket interface. The echo application will be extended, however, by specifying the exact protocol interactions between the client and the server. This entails specifying the exact message formats, as well as the different communication phases, as outlined next.

1. **Connection Setup Phase (CSP).** This is the first phase in the protocol where the client informs the server that it wants to conduct active network measurements in order to compute the RTT and throughput of its path to the server. We will outline the expected behavior from both the client and the server next.

   CSP: Client. After setting up a TCP connection to the server, the client must send a single message to the server having the following format:

   <PROTOCOL PHASE><WS><M-TYPE><WS><MSG SIZE><WS><PROBES><WS><SERVER DELAY>\n

   - PROTOCOL PHASE: The protocol phase during the initial setup will be denoted by the lower case character 's'. This allows the server to differentiate between the different protocol phases that the client can be operating at, as we will see.
   - M-TYPE (MEASUREMENT TYPE): Allows the client to specify whether it wants to compute the RTT, denoted by "rtt", or the throughput, denoted by "tput".
   - MSG SIZE (MESSAGE SIZE): Specifies the number of bytes in the probe's payload.

- PROBES: Allows the client to specify the number of measurement probes that the server should expect to receive. Once all the probe messages have been echoed back and a sample measurement is taken for each one, the client should compute an estimate of the mean (average) RTT or mean throughput, depending on the type of measurement being performed. A detailed description of the probe message's format is provided in the description of the Measurement Phase.
- SERVER DELAY: Specifies the amount of time that the server should wait before echoing the message back to the client. The default value should be 0. You will vary this value later to emulate paths with longer propagation delays. Even though increasing the server delay merely increases the processing time at the server, it nevertheless causes the feedback delay, observed by the sender, to increase, which has an effect somewhat similar to increasing the path's propagation delay.
- WS: A single white space to separate the different fields in the message. The white space could serve as a delimiter for the server when parsing or tokenizing the received message.
- The last is a new line character that indicates the end of the message.

CSP: Server. The server should parse the connection setup message to log the values of all the variables therein since they will be needed for error checking purposes. Upon the reception of a valid connection setup message, the server should respond with a text message containing the string "200 OK: Ready" informing the client that it can proceed to the next phase. On the other hand, if the connection setup message is incomplete or invalid, the server should respond with a text message containing the string "404 ERROR: Invalid Connection Setup Message" and then terminate the connection.

CSP: Summary. During correct operation, after setting up a TCP connection to the server, the client sends a single connection setup message to the server. The server parses and logs the information in the message and responds with a "200 OK: Ready" text message informing the client to proceed to the next phase.

2. **Measurement Phase (MP)**. In this phase, the client starts sending probe messages to the server in order to make the appropriate measurements required for computing the mean RTT or the mean throughput of the path connecting it to the server. We will outline the expected behavior from both the client and the server next.

MP: Client. The client should send the specified number of probe messages to the server with an increasing sequence number starting from 1. More specifically, the message format is as follows:

<PROTOCOL PHASE><WS><PAYLOAD><WS><PROBE SEQUENCE NUMBER>\n

- PROTOCOL PHASE: The protocol phase when conducting the measurements will be denoted by the lower case character 'm'.
- PAYLOAD: This is the probe's payload and can be any arbitrary text whose size was specified in the connection setup message using the MESSAGE SIZE variable.
- PROBE SEQUENCE NUMBER: The probe messages should have increasing sequence numbers starting from 1 up to the number of probes specified in the connection setup message using the NUMBER OF PROBES variable.

MP: Server. The server should echo back every probe message received. It should also keep track of the probe sequence numbers to make sure they are indeed being incremented by 1 each time and do not exceed the number of probes specified in the connection setup phase. If the probe message is incomplete or invalid (contains an incorrect sequence number for example) the server should not echo the message back. Instead, the server should respond with a text message containing the string "404 ERROR: Invalid Measurement Message" and then terminate the connection.

MP: Summary. The client repeatedly sends measurement messages to the server in an attempt to compute the mean RTT and/or mean throughput. A sample measurement is taken for each probe sent out. The server repeatedly echoes messages back to the client unless it detects erroneous behavior in which case it sends an error message and terminates the connection.

3. **Connection Termination Phase (CTP)**. In this phase, the client and the server attempt to gracefully terminate the connection. We will outline the expected behavior from both the client and the server next.

   CTP: Client. The client should send a termination request to the server and then wait for a response (unless of course the server already terminated the connection due to an error in the Measurement Phase). Once a response is received, the client should terminate the connection. The message format is as follows:

   <PROTOCOL PHASE>\n

- PROTOCOL PHASE: The protocol phase when terminating the connection will be denoted by the lower case character 't'.

CTP: Server. If the message format is correct, the server should respond with a text message containing the string "200 OK: Closing Connection". Otherwise, the server should respond with a text message containing the string "404 ERROR: Invalid Connection Termination Message". Either way the server should terminate the connection.

CTP: Summary During correct operation, the client sends a termination message, the server responds with "200 OK: Closing Connection" text message and then both terminate the connection.

What to submit

Turn in a brief (1-2 page) description of your experiments and a summary of the results, along with two graphs: one for TCP's round trip time as a function of message (probe's payload) size, and one for TCP throughput as a function of message size. When generating these plots, the SERVER DELAY should be set to the default value 0. Then, vary the SERVER DELAY parameter, and comment on how/why varying the SERVER DELAY parameter affects the shape of the plots. You may include up to two additional plots, for different SERVER DELAY values, to support your comments/conclusions.

Notes

- Make sure to report throughput numbers (e.g., 500 kbps) and not just the amount of time it takes to exchange some number of bytes.
- Give a thorough description of your experiments, including what kind of machines (e.g. the names of the CS Linux machines you used), operating systems, network, statistics collection method (including how many times an experiment was repeated, i.e. number of probes for a certain payload?s size, before the average is taken), etc.
- You may use the *gettimeofday()* system call or something similar to get the time.
- To draw graphs, you can use any plotting program you like (Excel, MATLAB, R, Gnuplot, etc).

Graduate Students / Extra Credit for Undergraduate

You should have received an email invitation to join the GENI testbed. Graduate students are required to use the GENI testbed for their tests. Undergraduate may use it for 20 extra credit points. If you need or want to use GENI (portal.geni.net), remember to pick the NATIONAL CENTER FOR SUPERCOMPUTING APPLICATIONS (NCSA) option when you login. Two virtual machines should be picked from two different geographical locations. You should not need to connect the virtual machines with a (virtual) link, but simply upload your code on the VMs and run your client and server.

There are many tutorials online on how to use the GENI testbed. Please start from this page `https://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials`.

How to Submit

Please save your report as a PDF and included it in a zip (or tar) file together with your source code before the due date specified on top of this document. At the top of the first page of your document (together with your name), please write how long it took you to complete this assignment and when you started working on it; for example:
John Smith: 18 hours. Started on Feb 16th. Your compressed file should be named as: `LAST_FIRST_PA1_Spring2020.zip`, for example, `Smith_John_PA1_Spring2019.zip`.

A submission link is provided on the course assignment webpage:
`http://cs.slu.edu/~esposito/teaching/3550/assignments/`
You may update your submission before the deadline by resubmitting your zip or tar file with the same name.
Please review the grading criteria on the next page before starting your assignment.

PA1 Grading Criteria (Total Score is out of 100)

Student Name _____ Total Score _____

Getting The Client and Server Working (Score is out of 45)

Logistics (20 pts)

Designed well, e.g. allows for address/port specification (4 pts)
Code is readable, i.e. commented and formatted (4 pts)
The overall design and how it works described (4 pts)
Possible tradeoffs and extensions discussed (3 pts)
Has (compilation and) running instructions (3 pts)
E-copy submitted and complete (2 pts)

Echoing Works (15 pts)

Client correctly sends a text message (5 pts)
Server correctly echoes the text message (5 pts)
Client interoperates with given remote servers (5 pts)

Error Conditions (10 pts)

Well tested, e.g. catches socket errors correctly, server catches client misbehavior, etc.

Score _____ out of 45.

Performance and Results (Score is out of 55)

Methodology (20 pts)

Client runs on a different machine than server (5 pts)
Code calculates round-trip time for different message sizes (5 pts)
Code calculates throughput for different message sizes (5 pts)
Describes evaluation environment (5 pts)

Graphs (how well each shows expected behavior) (15 pts)

Latency vs. message size (5 pts)
Throughput vs. message size (5 pts)
Each point is averaged over many experiments (5 pts)

Discussion (20 pts)

How well 1-2 page paper is written (10 pts)
Explains graphs in detail (10 pts)

Score _____ out of 55

Total Score _____ out of 100