CS 447: Networks and Data Communications

Project #01

Assigned Date : Thursday, February 12, 2015
Due Date : **Thursday, February 26, 2015 @ 09:29:59 a.m.**

Overview

For your next project you are going to implement a simple SMTP client and server. This project will be set up to run on DeterLab similar to project #00. This time however you will create your own .NS file consisting of three nodes. These three nodes will consist of: a sender node from which your e-mails will originate, a sever node that will act as an intermediary to host the SMTP service and act as an e-mail repository, and a receiver node that will be the pull from the server node.

1. For this project students are expected to work independently.
2. This project there are no additional requirements for graduate students.
3. Do *NOT* terminate your experiment on DeterLab when finished.
4. Start early. There will not be any extensions.
5. Students may use the language of their choice provided it runs in the nodes they have set up on Deterlab. The only limitation is that they may not use libraries that abstract away the lower level mechanisms used by sockets. If you have questions contact Dr. Gamage or Dustin via e-mail.

Requirements Solicitation: SIUE-SMTP

For your SMTP client to successfully send mail to the server each other you are to follow the protocol set out in RFC 821 using TCP only. For your client to receive mail it should use a technique similar to FTP using UDP only. The examples in the RFC's Appendix F should be useful to you.

E-mails should be saved as files containing header fields and data, at minimum make sure you include: From, Date, and To. The To field should allow for multiple recipients even though only one copy of the e-mail will be saved. More detailed information about what headers contain and their relation to the body can be found in the textbook. These e-mail files should be saved on the server until retrieved by the receiver node and then removed.

The receiver will retrieve these by contacting the server and receiving the number of retrieved messages upon successful connection and then downloading the e-mails using a GET command to the receiver node.

Functionality

At the end of your implementation you should be able to:

1. Log into the sender node on DeterLab
2. Start your client program to send an e-mail.
3. The client program should prompt the user for a SMTP server to connect to.
4. Once successfully connected the client should prompt user to either enter the receivers e-mail(s), the origin e-mail, and the body of the e-mail.
5. The client should then handle the transaction with the server and then inform the user of success or failure.
6. The client should then end gracefully.
7. Log into the receiver node on DeterLab.
8. Run your client program to receive an e-mail.
9. The client program should prompt the user for a SMTP server to connect to.
10. Once successfully connected the program should display the number of unread messages and ask the user if they would like to download them.
11. The user should then download the messages receiving a confirmation notification from the client program and a list of the files (e-mails) downloaded.
12. The client program should end gracefully, and the user should be able to see the files locally in the same directory as the client program.

Detailed Requirements

- All network communication between sender and server must be done using **TCP**.
- All network communication between server and receiver must be done using **UDP**.
- Ports for the client and server should be assigned in the dynamic range.
- Connections should be properly closed and not left open when exchanges are finished, details on whose responsibility this is located in the RFC.
- Your communication when sending should use the codes set out in RFC 821 4.2.
- You should follow the sending sequence set out in RFC 821 4.3.
- Your server and client should support the following commands: HELO, RCPT TO, MAIL FROM, DATA , QUIT, GET.
- Your server and client should support the following replies: 220, 221, 250, 354.
- The server should save all e-mails in files local to that machine until they are retrieved, at which point they should be deleted from the server.
- The end of the data portion of the email should be identified using a new-line, a period, and another new line.
- The header information on the e-mail should be preserved throughout the transfers.
- All programs should terminate gracefully, not leaving corrupted, half-finished or with open files.
- In the topology specified in the .NS file the sender should have no direct contact between it and the receiver.
- For this assignment no form of authentication, reliability, or multi-threading will be required.

Deliverables

The due date for is **Thursday, February 26, 2015 @ 09:29:59 a.m.** . A Moodle dropbox will be opened for your submission, which should include:

- A short readme file **in PDF format** that gives a brief overview of your implementation, the language you used, and how to run your project.
- Your .NS file used to set up the experiment.
- A compressed tarball of the directory containing your source code and the above two items. Do not include executables in this tarball. To create a compressed tarball of the directory source, use the following command: `tar -zcvf name-pr1.tar.gz source/`. Obviously, change the name to your own.

Related Info, Common Pitfalls, and Useful Links

SMTP, or Simple Mail Transfer Protocol, is a very commonly used protocol that serves to standardize e-mail communication. The protocol uses a push/pull model where a user will push mail from their local machines to a server hosting the SMTP service where the messages waits to be pulled by the receiver, who may use other protocols such as POP3 or IMAP to actually retrieve the file. The typical SMTP port number is 25. While the original protocol is specified in RFC 821 updates were later made in RFC 2821 as well as RFC 5321 which resulted in 'Extended SMTP'.

Common Pitfalls

For your convenience here are some of the more common pitfalls that students typically run into when beginning socket programming. Avoiding these will make your development easier, quicker and easier to troubleshoot.

- Hard coding IP addresses and ports.
- Failing to bind properly before trying to use a port.
- Forgetting to save the new socket returned by accept.
- Forgetting to close ports after finishing a transaction.
- Having difficulties parsing incoming packets. Remember that a socket can be thought of as a stream of incoming data similar to stdin or a file. Once you've filled a buffer with data you can use the same string parsing techniques that you've always used to interpret and manipulate that data.

You may find the following links helpful:

- Beej's Guide to Network Programming
  http://beej.us/guide/bgnet/output/html/multipage/index.html

- RFC 821: SIMPLE MAIL TRANSFER PROTOCOL
  https://tools.ietf.org/html/rfc821