CSE 5462: Project1

1 Overview

The goal of this project is to implement a TCP-like reliable transport layer protocol using the unreliable service provided by UDP, and then write a simple file transfer application to demonstrate its operation.

1. *Function Calls to be implemented*

   The arguments and return values of these function calls must exactly match the ones for the corresponding function calls of UNIX socket implementation. To implement these functions, you can use any UDP related function calls.

   (a) *SOCKET*
   (b) *BIND*
   (c) *ACCEPT*
   (d) *CONNECT*
   (e) *SEND*
   (f) *RECV*
   (g) *CLOSE*

   The focus of the project is on data transfer. Most of the TCP functionality will be implemented in the TCPD (TCP daemon) process which is equivalent to the TCP in the OS that runs in the background. These function calls will require communicating with the local TCPD process. The communication between the application process and the local TCPD process can be implemented with any inter-process communication mechanism such as UDP sockets. UDP communication within a machine can be assumed to be reliable.

   Write a simple file-transfer application that uses your TCP implementation. Note that you will need this program for testing your TCP implementation. The file-transfer protocol will include a server called *ftps* and a client called *ftpc*. Start the server using the command

   ftps <local-port>

   Start *ftpc* with the command

   ftpc <remote-IP> <remote-port> <local-file-to-transfer>

   The *ftpc* client will send all the bytes of that local file using your implementation of TCP. The *ftps* server should receive the file and then store it. Make sure that after receiving the file at the *ftps* server you either give the file a different name or store it in a different directory than the original since all the CSE machines have your root directory mounted. Otherwise you will end up overwriting the original file.

   The file-transfer application will use a simple format. The first 4 bytes (in network byte order) will contain the number of bytes in the file to follow. The next 20 bytes will contain the name of the file. The rest of the bytes to follow will contain the data in the file.

   To simulate real network behavior, all communication between the two machines will go through local *troll* processes. *troll* is a utility that allows you to introduce network losses and delay. More details on *troll* is provided later.

   The steps for transferring a file from machine M2 (client machine) to machine M1 (server machine) are as follows:

(a) Start the *troll* process and the TCPD process on machines M1 and M2.

(b) On machine M1, start the file-transfer server *ftps*. It will make the function calls SOCKET(), BIND() and ACCEPT(). ACCEPT() is a null function. So it will return immediately. *ftps* will then block in the first call to RECV().

(c) On machine M2, start the file-transfer client, *ftpc*. It will make the function calls SOCKET(), BIND(), and CONNECT().

(d) Normally the CONNECT() should initiate TCP handshaking between the two TCPD processes. But in this project you are not implementing TCP handshaking. So CONNECT() is a null function.

(e) The buffer management for this connection will be done in $TCPD_{M2}$.

(f) *ftpc* will read bytes from the file and use the function SEND() to send data to *ftps*. The SEND() function call will need to send these bytes to the local TCPD process. The TCPD process will then store these bytes in a wrap-around buffer.

SEND() should be implemented as a blocking function call. It should not return until all bytes in the buffer passed in the argument is written in the TCPD buffer.

(g) The buffer management functions will then take bytes from the buffer and create packets.

(h) Upon receiving the first byte the TCPD on M1 will unblock the ACCEPT() call. The *ftps* application will then make calls to RECV() to receive data .

RECV() should not return until at least one byte is read from the TCPD buffer. However, if multiple contiguous bytes are available they should be read to fill up the buffer up to the maximum size specified in the argument.

(i) After sending all the bytes of the file, *ftpc* closes the connection. The CLOSE() function call will initiate closing the connection.

(j) Upon receiving all the bytes of the file, *ftps* will close the connection using the CLOSE() function.

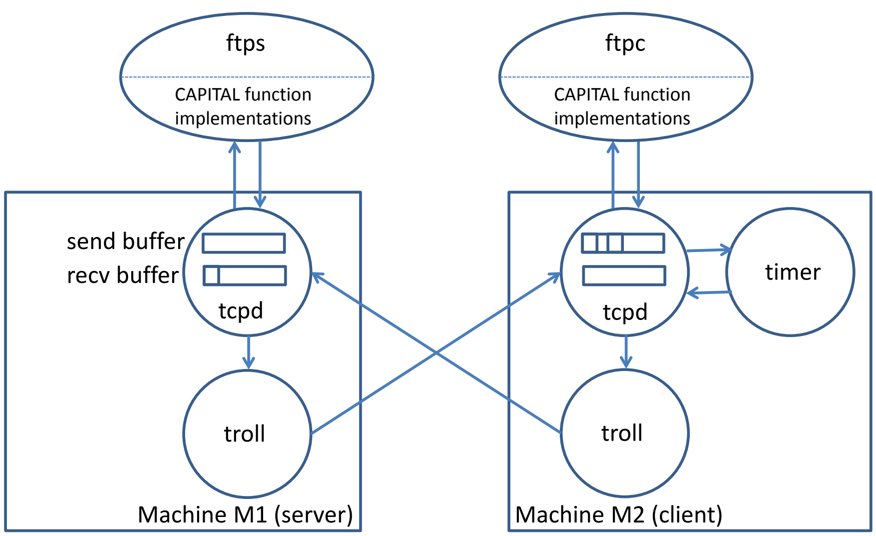

Figure 1: Connection Setup

2. *RTT Computation*

Implement the Jacobson's algorithm for computing RTT and RTO.

3. *Checksum Computation*

   The CRC (Cyclic Redundancy Code) checksumming technique should be used for computing the checksum.

4. *Packet Formats*

   The TCP packet structure should be strictly followed for both TCP and ACK packets. Instead of TCP's cumulative ACK, the ACK packet will acknowledge the data packet just received. Note that each packet will be ACKed.

5. *Timer Implementation*

   Each data packet after transmission will require a timer to be started. When the timer runs out, the packet will need to be re-transmitted. Since a large number of packets may be in transit at any given time, a large number of timers may be simultaneously running.

   Instead of using explicit timers for each packet, you will implement the timers using a delta-list. More details on delta-list is available on the project web-site.

   The delta list must be maintained in a separate process called the "timer-process". When a new timer needs to be started, a message is sent to the local timer process, indicating how long the timer needs to run for, which port the timer process should send notification upon expiry, and the byte sequence number of the packet for which this timer is being started.

6. *Buffer Management and Sliding Window Protocol*

   Implement the selective repeat algorithm. Use a fixed window size of 20. You are not required to implement slow-start, congestion control, or flow control algorithms.

   The send and receive buffers will be wrap-around (or circular) buffers. Use a buffer of 64 KB for sending and 64 KB for receiving and MSS of 1000 Bytes.

   Buffers/arrays in other processes should not store more than 1 MSS worth of data.

7. *Connection Shutdown* All the data structures related to the socket will be deallocated. However the buffer management function should make sure that all data has been acknowledged before deallocating the data structures. Implementing the state diagram for shutdown of a TCP connection is not required.

2 Troll

In the CSE network it is hard to artificially create real network scenarios (lossy links, packet garbling etc.) Use the *troll* utility to control the rate of garbling, discarding, delaying or duplication of packets. All packets will first go through a local troll process running on the same machine, where they will be subject to delay, garbling and/or drops. The packets will then be forwarded to the intended destination. The final destination should be marked in the first 16-bytes of the packet sent to the troll process. (Read the *troll* manual)

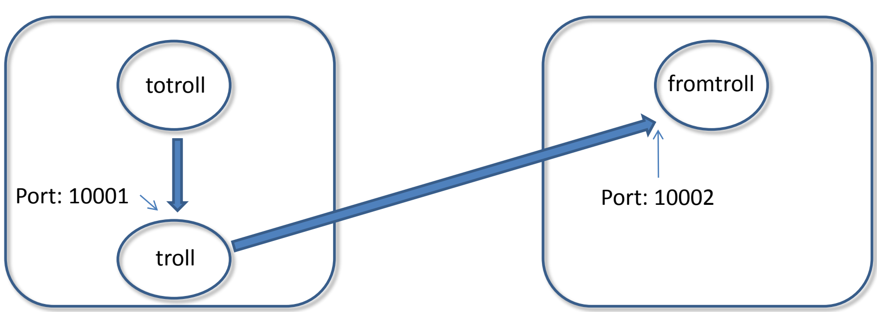

Figure 2: Troll

Test the *troll* program using the *totroll* and *fromtroll* programs. Source code for the two programs is available on the project web page. Here is an example of how you can test the troll functionality:

- On machine A, start *troll* to communicate on port 10001 using the command:

  troll 10001

- On machine B, start *fromtroll* to listen on port 10002 using the command

  fromtroll 10002

- On machine A, use *totroll* to send a short message via *troll* to *fromtroll* on machine B using the following command.

  totroll A 10001 B 10002

A and B have to be replaced by the IP addresses of the corresponding machines. You can use a combination of the following commands to find the IP address of your machine: *nslookup*, *hostname*, *ifconfig*. You can choose any two machines to run the *fromtroll* and *totroll* programs. Note that the particular port numbers used in the example above may be unavailable if some other process is using it.

3 Testing and Developing the Project

This project has several components. Debug and test each function carefully before integrating it into your code. Start your work very early as certain components such as the sliding window protocol may take a long time to debug.

4 Milestones

- **Feb 18th (Thursday) Checkpoint Demo and Code Submission:** Be creative in how you demonstrate the correct functionality of the modules (see table).

  In addition to the demo, you need to submit your code using the *submit* utility by 9pm. Use the following command for submitting the code :

  submit c5462aa lab7 <code-directory-name>

  Your code directory must contain a README file that describes all the C files in your directory. Also indicate how to test the functionality of each module. It must contain a Makefile. If you resubmit the files, submit both the code and the final report again, as each invocation of *submit* deletes all files of the previous submission for the same lab. Read *man submit* for clarification. Only one person from each group must submit the project.

- **Mar 10 (Thursday), Final Demo, Final Report and Code Submission:** Submit the final version of your code and the final project report by this deadline. The final project report should include details on the implementation, discussion on all the features of your program, optimizations, potential enhancements for the future, and the list of components that each student worked on. Do not include code fragments in the report.

  The report must be typeset using a formatting tool, such as Word, Latex etc. Use 11pt font with single spacing. The report must be 6-10 pages long.

  Submit your code and final report using the *submit* utility. Use the following command for submitting the final project code and the final report:

  submit c5462aa lab8 <code-directory-name> <final-report-file-name>

Your code directory must contain a README file that describes all the C files in your directory. Also indicate how to run your program. It must contain a Makefile. If you resubmit the files, submit both the code and the final report again, as each invocation of *submit* deletes all files of the previous submission for the same lab. Read *man submit* for clarification. Only one person from each group must submit the project.

Deadline	Item	Point details	Points
In CL112 4-7pm, Feb 18	**Checkpoint Demonstration**		**16**
	Delta-timer	8	
	Checksumming	8	
Submit 9pm, Feb 18	**Well-documented code**		**2**
In CL112 4-7pm Mar 10	**Final Demonstration**		**22**
	RTT and RTO Computation	4	
	Circular Buffer management	8	
	Successful execution with troll on both machines (with garble 25%, destroy 25%, duplicate 25%, reorder, exponential delay with mean 10ms)	10	
Submit 9pm Mar 10	**Final Report**		**4**
	Project overview and details on each process	1	
	Detailed description of each project component (Wraparound buffer, TIMER, checksumming, RTT/RTO, Packet formats, Implementation of SOCKET, BIND, SEND, RECV, CLOSE)	1	
	Detailed description of how to compile, run and test the program	1	
	Possible future extensions to make this program more efficient and to add more features	1	
Submit 9pm Mar 10	**Well-documented code**		**2**
	TOTAL		**46**

5 Miscellaneous

- **Platform:** Use the *stdlinux* system for all your implementation.
- **Delayed Submissions:** Late demonstrations, code submissions or report submissions are not eligible for any points.