CprE 489 Fall 2016 (Section XE)
Programming Assignment #3
TCP Congestion Control

Due Date: 12/11/2016 (Sun) by 11:59 PM
Submit on Bb Learn

Objective

To write two programs to implement the TCP Tahoe congestion control mechanism, one program for the Sender, and another for the Receiver.

Notes

- If you feel there are ambiguities in the specifications, feel free to document them and state your assumptions in the report.
- Please ensure that your code is well commented.

Problem Description

In this programming assignment you are required to design and develop two programs to implement the TCP Tahoe congestion control mechanism: one program for the Sender and the other for the Receiver. You will implement the three major techniques that are used in TCP Tahoe: **Slow Start, Congestion Avoidance, and Fast Retransmit.**

You will be provided incomplete Sender and Receiver programs. These incomplete Sender and Receiver programs establish a TCP connection between Sender and Receiver, and show an example of two-way communication with comments explaining the different parameters necessary to send and receive data. To simulate network congestion, please use the `AddCongestion.c` routine to introduce errors to the transmitted packets. For CRC generation and error detection, you should use the routines developed in Programming Assignment #1. If you were not able to produce a working implementation of these routines, you may use the provided object file. See the CRC notes below for more information.

Sender Program

Write a program to implement the Sender, which shall

- Establish a TCP connection to the Receiver (Has been done) and send packets over that connection.
- Read the provided `input.txt` file into the memory as an array of *char* and send the content to the Receiver.
- Form each packet according to the following format:

Sequence Number	Data	CRC
2 bytes	2 bytes	2 bytes

- **Sequence Number**
    - Byte index, starting from 1000.
- **Data**
    - Two characters (sent from Sender to Receiver)
    - No data is sent from Receiver to Sender

- **CRC**
    - CRC generated for this entire packet, including Sequence Number and Data fields (see the section on CRC)

- Apply the `AddCongestion.c` routine to the entire packet. It has two arguments:
    - a null terminated version of the packet above
    - a bit error rate (BER), between 0.00001 and 1

  This step introduces random errors to the packet (with the provided BER), which simulates congestion in the network.
- Implement an RTO timer, with RTO = 3 seconds.
- Implement the congestion window, with MSS = 2 bytes. For this lab, assume that the advertised receiver window (rwnd) is always larger than the congestion window (cwnd).
- Implement the TCP Tahoe congestion control mechanism:
    - Slow Start: start with cwnd = 1 (MSS), and increase it by one when a non-duplicate ACK is received.
    - Congestion Avoidance (with the initial ssthresh = 16 MSS): increase cwnd by one only after the sender has received cwnd non-duplicate ACKs.
    - Fast Retransmit: retransmit the packet when three duplicate ACKs are received.
- Record cwnd after each RTT.

Receiver Program

Write a program to implement the Receiver, which shall

- Accept connections from the Sender (Has been done).
- Accept data packets from the Sender.
- Run the CRC check:
    - If the packet is received error free, then **delay for 1 second** and sends back an ACK with the following format:

Acknowledgment Number
2 bytes

    - If the packet is received in error, do nothing.
    - ACK packets are not corrupted.

Procedure

- Complete the Sender and Receiver programs as described above.
- Transmit the provided input.txt file from the Sender to the Receiver and record cwnd after each RTT.
- Plot the *congestion window size* vs. *RTT* curve with the X-axis ranging from 0 to 40 RTTs.
- Submit your figure and code to the TA for grading.

Exercises

1) Run your program with each of the following BER values: 0.0001, 0.001, 0.005, 0.01, and plot the *congestion window size* vs. *RTT* curves for each BER value.

Compiling and Running “sender” and “receiver” Programs

Receiver

Compile `receiver.c` to create an executable,

```
gcc -o receiver receiver.c
```

Then, run `receiver`, and have it listen on some port number,

```
./receiver 127.0.0.1 50404
```

Note 127.0.0.1 is the IP address of the localhost. In order to avoid contentions and confusions, use a port number that is equal to the rightmost five digits of your student ID number. For example, if your Student ID number is 123456789, then the port number should be 56789. If this number is less than 1023, greater than 65535, or equal to any reserved port number (see the `/etc/services` file) you may use any random port number.

Sender

Compile `sender.c` to create an executable,

```
gcc -o sender sender.c
```

Then, run `sender`, and have it connect to the receiver program on a given IP and port number (that should match the receiver’s). You are also required to pass the BER value.

```
./sender 127.0.0.1 50404 0.001
```

CRC Generation and Checking

If you were unable to produce a working implementation of CRC generation and error detection in Programming Assignment #1, you may use the provided object file for this purpose.

Compiling

An object file, `ccitt16.o`, is provided that will generate and check CRC for you. In order to use the `.o` file, you first need to include `ccitt16.h` with your file (e.g., `sender.c`):

```
#include "ccitt16.h"
```

Now, compile your file with the **–c** option to generate a `.o` file:

```
gcc -c sender.c
```

Finally, compile both `.o` files together to create an executable:

```
gcc -o sender sender.o ccitt16.o
```

Usage

The function provided by `ccitt16.o` has the following prototype:

```
short int calculate_CCITT16(unsigned char cData[], unsigned int iLen,
                            unsigned int iAction);
```

`iAction` is defined as either `GENERATE_CRC` or `CHECK_CRC` in the `ccitt16.h` header file:

```
#define GENERATE_CRC    1
```

- Returns the checksum of `cData[]` with length `iLen` as a short `int`

```
#define CHECK_CRC       2
```

- Uses the last two bytes of `cData[]` as CRC check bits to check `cData[]`; returns either 0 or 1:
    - `#define CRC_CHECK_SUCCESSFUL  0`
    - `#define CRC_CHECK_FAILURE     1`

How to Write and Run Your Program

All programs shall be written in C, under the Linux environment. If you don't have a Linux machine with `gcc` installed, you may remote log into one of our Linux servers. For more information, please refer to http://it.engineering.iastate.edu/remote and http://it.engineering.iastate.edu/remote-access.

You should follow these steps:
- Edit your program using any of the editors available under Linux.
- Compile your program using `gcc`.
- For programs using sockets, use the following command to compile:
  ```
  gcc –o file file.c
  ```
  where `file.c` is your code, and file is the required executable file. Note that you can link to other libraries as needed, such as the math library using `–lm`. Also, please note that under other versions of UNIX besides Linux (e.g., Solaris), you need the `–lsocket` and `–lnsl` options for socket programs.

- You may run your program by typing the full path to your compiled executable:
  ```
  ./file
  ```

What to Turn In

1. Create a pdf file to report what you learned from this assignment, and your answers to the exercises, and then name it **pa3_report.pdf**.
2. Create a **README** file that describes your program, as well as how to demo your program (i.e., which files to compile, how to compile, which files to execute, how to execute, what is the input to the program, what is the expected output, etc.). Compress your **README** file, well-commented source code, and executable files to an archive named **pa3_code.tar.gz** (or rar, zip).

Grading Policy

The TA will examine your source code and test it on Linux machines. If your program cannot be successfully compiled but your executable files work, you will receive partial credit.