CS5310.001/002, Fall 2020, Programming Project

Simulation of Fast Ethernet

Issued: Nov 4, 2020
Due: Midnight of Wednesday, Dec 9, 2020
(Note: that will be two days after the final exam so that you have more time to finish the project. The class home/project submission access will be closed midnight of 12/9/2020).

This project is to simulate the Fast Ethernet. The goal of the project is to practice basic socket API programming through a client/server application. It is a *simulation* in the sense that the fast Ethernet is simulated by multiple processes on multiple machines. Each station in the Ethernet is simulated by a process. The switch of the fast Ethernet is also simulated by a process. Those processes can run on different workstations in the Linux lab.

1 Classical Ethernet vs Fast Ethernet

The IEEE 802.3 protocol, commonly known as the *Ethernet*, is a *1-persistent CSMA/CD* protocol. It provides a maximum data rate of 10 Mbps. Stations are connected to a common communication *bus* through which communications are carried out. The basic ideas are:

- A station wishing to send a message frame will first check if the communication bus before sending out the frame.
- If the bus is busy, the station will wait. If the bus is idle, it will send out the frame.
- A collision occurs when two or more frames from different stations collide on the bus. A collision can occur either during the sending period, or the uncertain period (in which the station has sent out the frame, but is not sure if the frame has arrived safely to its destination station). A station will stop transmission immediately if it detects the collision during the sending period. It then waits for a *random period* of time before trying retransmission.
- The random period mentioned above is determined by the well-known *binary exponential backoff* (BEBO) algorithm:
  - After the first collision, the time is divided into discrete slots whose length is equal to the worst case round-trip propagation time.
  - After the first collision, a station waits for either 0, or 1 time slots before attempting its first retransmission.
  - If the first retransmission collides too, it will wait for either 0, 1, 2, or 3 time slots before the second retransmission.
  - In general, after the i^{th} collision, where $i \leq 10$, a station will wait for a random number of slot times chosen from the interval from 0 to $2^i - 1$. However, after 10 collision, the random interval is fixed between 0 and $2^{10} - 1 = 1023$ time slots.
  - After 16 collisions, the Ethernet controller for a station suspends retransmissions and reports the failure to the data link layer.

The IEEE 802.3u protocol, commonly known as the *fast Ethernet*, is a collision free protocol and provides a maximum data rate of 100 Mbps. Unlike the classical Ethernet where stations compete access of the media independently, access to the communication media in fast Ethernet is controlled by a switch.

2 Simulation of Fast Ethernet

We simulate a fast Ethernet with a collection of processes that execute on a collection of UNIX machines in the CS department Linux lab. The project language is **C/C++**.

All interprocess communication is using UNIX *sockets*. UNIX provides several kinds of sockets and you can choose whichever sockets you like for the project. You also have the freedom to choose either TCP or UDP as the underlying transport protocol.

The simulation is composed of the following processes:

(1) *Station Process* (SP). One station process for each simulated station. For flexibility the number of such station processes allowed should be a variable. But for implementation purpose you can assume in the project there are at most ten station processes, i.e. we are simulating an Ethernet consisting of at most ten stations.

(2) *Communication Switch Process* (CSP). There is only one such process, which simulates the function of a switch in a fast Ethernet.

For every station, the sequence of frames to be sent, together with the destination station number, is specified as simulation input data file that will be read by that particular station (cf. Section 3).

Fig.1 illustrate a fast Ethernet with ten stations.

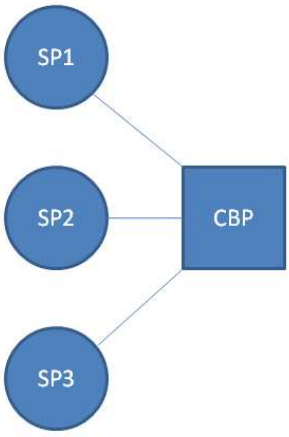

Figure 1: Illustration of the Programming Project

In this case the ten stations cannot communicate to each other directly. A station, say S_1, has to send a message (intended to S_6) to the CSP first, which in turn will send the message to the destination station S_6.

2.1 The Frames

We assume that frames are of fixed size for simplicity. Each frame has a sequence number, a source address, a destination address, and a data section.

(1) There are only two types of frames sent by a SP: request frame; and data frame.

(2) There are three frame types sent by the CSP:

- Data frame (forwarded), which has the same format as the one sent by a SP;
- Positive reply, sent in response to a request (to send a data frame) from a SP;
- Negative reply (i.e. reject), sent in response to a request (to send a data frame frome a SP.

2.2 Communication Switch Process

The main duty of the CSP is to function as the switch of the fast Ethernet. Therefore CSP will give permission to a SP to send a frame, accept frames from a source station and forward each frame received to its destination station.

Every time a request frame is received, the CSP will either send a reply to the SP so that the SP will send a frame, or hold it in its waiting queue if it is busy. When it receives a data frame it will inspect the destination address. It will then send out the frame to the destination station.

The CSP maintains two queues: a data frame queue that contains data frames to be forwarded; and a request frame queue that contains request from SPs for accepting data frames. Each queue has a fixed size.

When a request frame is received, and if data frame queue is not full, the CSP will send a reply to allow the SP to send its data frame. If the data frame queue is full, and the request frame queue is not full, then the CSP will place that request frame into the request frame queue. If the request frame queue is full as well, a *reject* reply will to send back the SP.

2.3 Station Processes

The operations of an SP is not complicated. Here is a brief summary:

(0) If there is a incoming data frame, receive it.

(1) It reads a line from its simulation input data file and format it as a data frame.

(2) It then sends a request frame to the CSP asking permission to send its data frame just formatted and wait for reply.

(3) If it a *reject* reply from theh CSP, it has to retransmit the request frame. Your implementation has to allow a total 3 retransmissions.

(4) If it a positive reply from theh CSP, it will then proceed to send data frame to the CSP.

2.4 Multiplexing

Both the CSP and SP have to deal with multiple input events simultaneously. The CSP must be able to receive request/data frames from multiple SP, send replies, and forward data frames, at the samem time. The SP on the other hand must be able to send data/request frames, read from input file, and receive data frames sent by other SPs, at the same time.

2.5 Inter-machine Communications

The *socket* system calls are capable of providing inter-machine communications. You must use TCP APIs to implement the project. The material presented in class and examples from the textbook should be sufficient for this project.

3 Controlling the Simulation Using Events

In order for us to be able to create different communication scenarios, we need to control the progress of the simulation. As stated before, *simulation input data files* are used for this purpose. Each SP has an associated simulation data file that will read only by that process. The simulation file is real simple. Here are several sample lines that could be in the simulation file for SP 1:

Frame 1, To SP 3
Wait for receiving 2 frames
Frame 2, To SP 8
Wait for receiving 1 frame
Frame 3, To SP 6
Frame 4, To SP 2

The line *Wait for receiving n frames* means that the SP will not send send next frame until it receives n frames from other stations.

4 Simulation Result

The result of the simulation is the activity log files maintained by each sP and the CSP. For example, an SP may record the following information in a its log file:

1. "Send request to CSP to send data frame 1 to SP 3"
2. "Receive positive reply (permission) from CSP to send data frame 1 to SP 3"
3. "Send (via CSP) data frame 1 to SP 3"
4. "Receive (via CSP) a data frame from SP 2"
5. "Send request to CSP to send data frame 1 to SP 4"
6. "Receive reject reply from CSP to send data frame 2 to SP 4"
7. "Retransmit request to CSP to send data frame 2 to SP 4"
8. "Receive positive reply (permission) from CSP to send data frame 2 to SP 4"
9. "Send (via CSP) data frame 2 to SP 4"
11. "Send request to CSP to send data frame 3 to SP 6"
12. "Receive positive reply (permission) from CSP to send data frame 3 to SP 6"
13. "Send (via CSP) data frame 3 to SP 6"

The CSP may record the following kind of information:

1. "Receive request from SP 1"
2. "Receive request from SP 2"
3. "Send positive reply to SP 1"
4. "Send positive reply to SP 2"
5. "Receive data frame from SP 1 (to SP 3)"
6. "Forward data frame (from SP 1) to SP 3"
7. "Receive data frame from SP 2 (to SP 4)"
8. "Receive request from SP 1"
9. "Forward data frame (from SP 2) to SP 4"
10. "Send positive reply to SP 1"

Please note that the CSP doesn't care the destination information of a request. Both the CSP and SPs don't keep a copy of data frame sent out.

Total number of SPs: you shouldn't hard code the limit the number of SPs. Your implementation should allow up to 10 SPs. Those SPs should share the same code. You shouldn't have separate code for each SP.

5 Project Report

The whole project must be implemented on CS department Linux lab computers. Implementations on Windows will not be accepted.

Your project report must include the following:

(1) Project description file. This file must be in Word or PDF format.

- The description file must contain an itemized description about what your have implemented and what you have not been able to complete. For each implemented feature in your project, a brief description about the method you use. For instance, how have you implemented the multiplexing?
- The whole program must be well-commented. For instance, important functions must be concisely commented about the parameters needed, the main duty performed, values returned, and etc.

(2) A README file that clearly describes how to compile and test your program.

(3) Source code:

- The entire source code must be uploaded. The whole program must be self-compilable. Be sure to include all files needed to compile your program on any department Linux machines.
- You should provide complete information in your README file of the directory of your project in your department Linux account so that I can verify it if needed. You are not allowed to change the code after the close of the project submission.

(4) Simulation results. Please supply your input and corresponding output. Your project may be tested using your own input and may also be tested using my own input data.

The project should be submitted through the class forum homework/project panel. Please follow the upload instructions.

The upload page will be closed after the due date so that I will have time to grade it. No email submission is accepted.

Each project will be tested during grading. Independence of project implementation will be rigoriously verified. The ability for the clients and server to be able to run on different computers, not just on a single computer, is important. Failing to complete essential features of the project, non-independence of project implementation, and not following the stated project submission instruction will result in low scores and will affect your final grade proportionally.