CSC 270H Assignment 3, Fall 2002
Ethernet simulation

Due at the end of Thursday November 21, 2002; no late assignments without written explanation.

Introduction

The vast majority of modern local-area networks use a protocol called "Ethernet". The CDF computers are connected to each other by ethernet.

The ethernet standard, IEEE 802, states how computers should transmit data "packets" on a ethernet network segment. In this assignment, we will be concerned with only a single ethernet segment. (A complete ethernet consists of one or more segments with "packet switching" or "repeating" between them. Further-connecting these local-area networks together can result in larger-area networks, such as the internet.)

Transmitting on an ethernet segment is like shouting into a room in that only one person in the room (on a segment) can transmit a message at the same time, and everyone shouting into the same room (transmitting on the same segment) can hear everything which is transmitted (so you have to ignore messages which are not for you).

What prevents two computers on the same ethernet segment from transmitting at the same time?

• Before a computer transmits, it listens to the network and makes sure it is silent.

• Nevertheless, sometimes two computers do transmit at the same time—they both listen to the network and hear that it is silent, then they both transmit on the network. This is called a "collision".

How do we deal with collisions? In this assignment, we will simulate an ethernet network and experiment with different collision formulas. (These delay formulas will also be applied in the case where the network is found not to be silent when we want to transmit.) Since we are only interested in the timing and collision behaviour, we will not worry about the data content of the packets, only about their size and whether or not there is a collision.

Ethernet details—fixed

Ethernet packets have a minimum size of 64 bytes and a maximum size of 1518 bytes. (If you have a longer message to send, you break it up into multiple packets; this is the origin of the name "packet": a little glob of data all wrapped up for transmission.)

This 64 to 1518 bytes includes information about which host (computer) the data is from and to and some other "header" information. This won't matter for this simulation, nor will the data itself.

Data communications speed is generally measured in bits, but we will say that the speed of our ethernet works out to exactly one million data bytes a second. For example, a 345-byte packet will take 345 μs to transmit. So the *amount* of data *will* matter for this simulation.

As mentioned above, hosts listen on the network before transmitting. However, there is invariably some lag between when they listen and when they start transmitting (and this is one of the sources of the collision problem). For this assignment, this lag will be fixed at 2 μs. That is, if a host starts trying to transmit starting at time 1.234567 seconds, then if the network is clear at time 1.234567, that host will start its transmission at time 1.234569. On the other hand, if the network is busy at time 1.234567, then the host will wait for an amount of time according to one of the formulas below, then try again.

Ethernet details—parameters

The above values will be constant for this assignment. The following parameters will be varied to learn about ethernet behaviour, which is the purpose of the simulation.

For simplicity, we will assume that there is only one program running at one time on each host (computer). The number of computers and their behaviour are parameters. We will identify three types of communications behaviour:

1. Repeated transmission of a uniformly-randomly-distributed number of bytes (between 64 and 1518 inclusive, and each packet size is an independent selection). The delay between the end of the successful transmission of one packet and the beginning of the first attempt to transmit the next is a uniformly-randomly chosen number of microseconds between 50 and 500 000. In this case, we don't worry about the other side of the conversation; just transmit the data. For the purposes of this assignment, this kind of host will be called a "spewer".

2. Communication between two hosts we think of as peers. One of the hosts transmits a uniformly-randomly-distributed number of bytes (between 64 and 1518 inclusive) to the other. From then on, after each successful transmission, the other host sends a message back, of a uniformly-randomly-distributed size between 10 bytes less and 10 bytes more than the message it just received, except not exceeding 1518 bytes. In this back-and-forth transmission, the interval between receipt of one message and transmission of the next is also a uniformly-randomly chosen number of microseconds between 50 and 500 000.

3. Communication of a "client-server" nature. One of the two hosts is the "client" and the other is the "server". The client sends an exponentially-randomly-distributed number of bytes to the server to initiate the communication; it has minimum size 64 and a mean of 100 bytes. The server responds with a total of a uniformly-randomly selected number of bytes between 100 and 100 000. Each of these communication streams is broken down into multiple packets such that all but the last packet are of size 1518, and the total number of bytes is the total message size as randomly selected. (This is a substantial simplification because in real life, there is "overhead" in the packets such as the "header" items, but we won't worry about that for this assignment.) If the last packet would be smaller than 64 bytes, it is padded out to 64 bytes. The wait between the completion of the client request and the attempted start of the server reply is a uniformly-randomly chosen number of microseconds between 1000 and 2000. The wait between multiple packets in the client request or in the server reply is 3 μs. The wait between the completion of the server reply and the initiation of a subsequent client request is an exponentially-distributed random time with a mean of 2 000 000 μs (i.e. two seconds).

The parameterization of this consists of specifying a number of hosts (for type 1) or pairs of hosts (for types 2 and 3) for each type of behaviour. Some of these values can be zero, e.g. it is possible to have only server-client pairs and no peers or spewers.

The final parameter concerns the purpose of this simulation, which is to test various collision algorithms. The basic algorithm is to retry the transmission later—and since the simultaneous transmission messes up the ethernet, *both* hosts must retransmit. The difference between the strategies is how long to wait before retrying, as follows. (If it then collides again, you do the waiting strategy again, and so on.)

1. Wait for 20 μs before retrying.
   (This turns out to be *extremely* bad, thus justifying the following more complex strategies.)

2. Each colliding host selects a uniformly-randomly distributed value from 3 to 50 and waits that many μs.

3. Each colliding host selects an exponentially-distributed random value with minimum 3 and mean 7 and waits that many μs.

4. Each host maintains a value, which for discussion here we can call x, whose initial value is chosen randomly between 5 and 15 μs (independently for each host). When a collision or conflict occurs, each computer transmitting waits for its x μs before retrying, then changes its x value to $2x+3$. Every time a packet is successfully transmitted, the transmitter subtracts 2 from its x, but never going below 3 (i.e. if $x <= 4$, set x to 3 rather than to $x - 2$).

5. Whichever of the above strategies seems to be best, add a further fifth strategy which is the same basic algorithm but with different numbers (replacing the numbers such as 20, 3, 50, etc above) to try to improve the performance further.

Your program

Write a program in C++ (in an object-oriented style, using classes) which simulates the operation of this ethernet with computer programs transmitting on it as discussed above.

It will take four command-line arguments, specifying the above parameters in order: the number of spewers, the number of peer pairs, the number of client-server pairs, and the strategy number from 1 to 5. We strongly recommend that you base your main() on the example code on the web page for this assignment, or test its argument-handling carefully, as it will be run by an automated process after assignment submission.

Your program's output

The main answer is the total number of bytes of data successfully transmitted during a certain fixed time. This one number is the "throughput" for a particular run of your program.

However, we need to collect more statistics than this. For one, it is difficult to gain confidence that your program is working if it does not output some further information about what it did.

More than this, though, you will want to be able to figure out *why* your simulation performs better and worse in the various situations. So you will want to collect further statistics about what it is doing, such as the number of collisions, how long various hosts are waiting for the network to be clear, for how much time the network was silent, etc.

It is up to you to determine what statistics to collect. Your need for statistics is driven first by making sure that your program is working, and then for the purposes of writing your report.

Your report

Write a short report (less than 60 lines, 80 columns width) which cites typical output from simulations using each of the collision algorithms, and specifies which seems to be best. If different strategies are better for different purposes (that is, different mixes of spewers, peers, and client-server pairs), try to characterize the purposes for which a particular strategy is better. Then briefly state your fifth strategy, and how it compares to the others.

Your report might be much shorter than a page.

If you are unable to determine differences in the various algorithms in terms of performance, state this, and conjecture as to why.

Other notes

You will have to decide upon an appropriate length for the simulation run. If you are not getting sufficiently useful data, you may have to lengthen the run.

You are to write an event-driven simulation, not a time-driven simulation. That is, your program will constantly be skipping ahead to the next event. It will *not* simulate the passage of time explicitly when events are not occurring. Please ask if the distinction between an event-driven simulation and a time-driven simulation is still not clear.

There are several program components above, and you have to write them all. Furthermore, the planning of your module breakdown is a part of this assignment. You should begin by going through this assignment handout and taking notes about the different modules (.cpp files), including listing their exported functions and rough parameter lists. Create reusable classes.

You may name your files and classes what you like, but you should work on providing good names, and organizing your program well. All of your submitted files will "count" unless they are of zero length; thus, to cancel a file submission, you can resubmit under that file name with a zero-length file (make sure that an "ls –l" of that file says exactly zero for the file size). No files should be submitted which do not comprise part of your assignment.

It is your responsibility to make sure that your code works on the CDF Unix system. We will compile it with

```
g++ -Wall -ansi -pedantic *.cpp -lm
```

(after first deleting all zero-length files) and we will execute it with the four command-line arguments discussed above.

As a rough guideline, the statistical output should be under 25 lines long (80 columns wide), at the most. In many cases just several lines of output will suffice.

Please see the assignment web page for some hints and frequently-asked questions.

Submission details

To make sure that your files are in plain text, display them on the screen at CDF with the *cat* command. If you are having trouble with ˆM characters, try *cat –v* to show control characters.

Once you are satisfied with your files, you can submit them for grading with the command

```
submit -c csc270h -a a3 file1 file2 file3 ...
```

You do not submit your compiled program, just the .cpp and .h files and your report (called "report").

You may still change your files and resubmit them any time up to the due time. You can check that your assignment has been submitted with the command

```
submit -l -c csc270h -a a3
```

You can resubmit a particular file which was already submitted by using the –f option, e.g.

```
submit -c csc270h -a a3 -f event.cpp
```

and if necessary, you can use this to remove a file from consideration by resubmitting a zero-length file with the same name.

This is the only submission method; you do not turn in any paper.

Late assignments are not ordinarily accepted and *always* require a written explanation (e-mail preferred).