SC-601 SC601 : Network Concepts

B. W. Gore

25 Aug 2025

Department of Scientific Computing, Modeling & Simulation
SPPU, PUNE 411 007

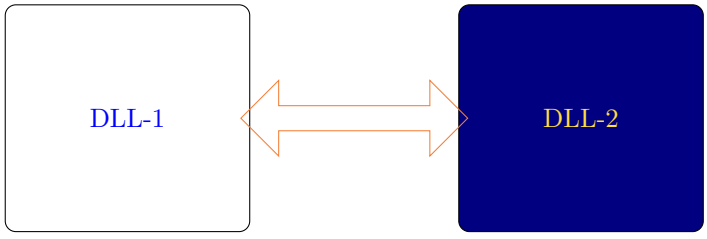
Project Assignment: Network Simulator

1 What should be submitted?

The idea is to implement concurrency. Consider two identical copies of a piece of software, programmed to act as protocol processors at the data link. We simulate their concurrent execution, using a computer with a single CPU and a language such as C. The idea will be to share the CPU time using a round robin scheduler. The simulation program will be divided into a number of units named **Strips**. We define strip as a routine called by the scheduler. It need not be named as "strip", but could be named "DLL1" or "LAN", or "Line1". Each of the strips will be a collection of an application program, a protocol processor, a data link, a LAN system or a part of any one of these. In other words, one or more strips will simulate a protocol processor, or other network element.

Think of each strip as a routine being called by the main executer, which is the scheduler, running an endless loop. Avoid the problem of endless looping by demanding that no strip will execute more than 300 statements before exiting. The programmer should use only Conditional & Unconditional Statements, for loops

There is a special strip named "connectors". Its job is to simulate datalinks of all kinds. Each connector merely transfers data from one strip to another.



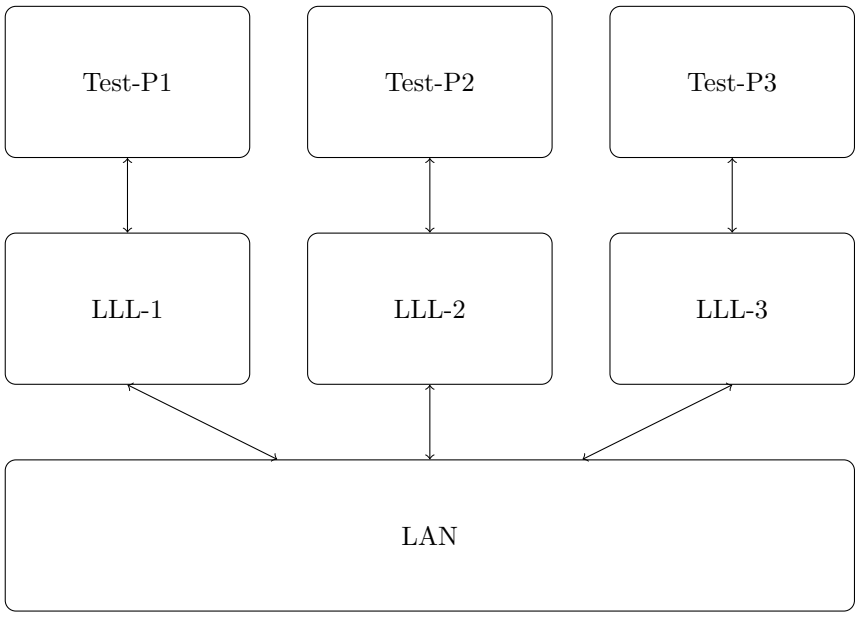


Figure 1: LAN setup

LLL1, LLL2 and LLL3 run identical copies of the logical link layer protocol on three different machines. TestP1, TestP2 and TestP3 are identical copies of a test program running on those three machines. We name ports in an obvious way as TestP1-1, TestP2-1 and TestP3-1. The upper ports on the LLLs are named as LLL1-1, LLL2-1 and LLL3-1. The three ports below the boxes marked LLL are named as LLL1-2, LLL2-2 and LLL3-3. The three ports of the LAN are LAN-1, LAN-2 and LAN-3.

Each port is an array of the form $d[2][256]$.

$d[0][*]$ represents the "out buffer" element of the port from which a connector may send data to some other port.

$d[1][*]$ represents the "in buffer" element of the port into which a connector may fetch data from some other port.

We will use the convention of using $d[*][0]$ to be a byte count, *i.e.* how many bytes are there in the buffer. If $d[*][0]$ is zero, then there is no data in the buffer. We will use this representation to mean that there is nothing to process. If $d[*][0]$ is 72, it means that $d[*][1 \cdots 72]$ is the real data.

In the real world, various units run at various speeds. One host could have a powerful CPU and another a much less powerful. If one is a producer of data and another is a consumer, there will be a mismatch of speeds. How is the synchronization between producer & consumer is to be achieved; how is the

speed mismatch going to be simulated?

Here is a simple solution. Consider a C program that runs a loop 100 times, and prints out 1's with probability of 5 percent during each iteration. Suppose that k is a random number in the range of 0 to 19999. j is upto 5% of 20000, *i.e.* 1000. We create k, j randomly with these restrictions 100 times.

```
boolean prob(int P)
{ int k, j;
  const int N = 20000;
  k = random(N);
  j = N * P/100;
  if(k < j) return false;
  else return true;
}
```

We can now write a strip TestP1 with the associated data structure TestP1-1[2][256]. Defining the out-buffer TestP1-1[0]|256] and the in-buffer TestP1-1[1][256]. The routine TestP1 has two jobs to perform. It alvways checks its in-buffer and prints it out in the format

```
TestP1 received 234567122.
```

It also creates a packet with random contents roughly 10% of the time it is allowed to run. The data content of this packet will be in the range of 13 to 20 bytes. It selects the destination randomly, say, TestP3. Whenever it creates such a packet, it prints out a message in the format

```
TestP1 sent the message 14571 to TestP3
```

It also creates a packet with random contents roughly 10% of the time it is allowed to run. The data content of this packet will be in the range of 13 to 20 bytes. It selects the destination randomly, say, TestP3. Whenever it creates such a packet, it prints out a message in the format

```
TestP1 sent the message 14571 to TestP3
```

If TestP1-1[1]|[0] is non-zero, print out the packet received in the format mentioned above.

If prob(10) is 0, return

The statement d = 1+random() mod N assigns d a value from the set $\{1, 2, 3, \cdots, N\}$. Here, the funtion random() is supposed to produce a large random integer value. Using this randomly selected destination number, you can send the packet to the corresponding destination.

Set bytecount TestP1-1[0] to be a random number in the range of 13 to 20 by computing (13 + random() mod 8). Let this field be called bytecount, or *bc*. Fill

up the buffer TestP1-1$[1 \cdots bc]$ with random numbers in the range of 0 to 255, to simulate the packet generated.

Send out the message to the selected destination

end.

See the following strip which acts as **connectors** for the set up shown in Fig 1. The statement "*Connect X and Y*" means the following: If the byte count of $X[O]$ *i.e.* the byte count of out-buffer of X, is non-zero, and the byte count of $Y[1]$, *i.e.* the byte count of the in-buffer of Y[1], is 0, then, copy bytes of $X[0]$ to $Y[1]$ including the byte count. Then if the bytecount of $Y[O]$ is non-zero and if the bytecount of X[1] is zero, transfer the bytecount and contents of $Y[0][*]$ into the buffer $X[1][*]$. Please note that the connection is a symmetrical, two-way activity. Establish all the connections shown in Fig. 1.

We will now define the sections of the project.

1. **Packet structure and Addressing Conventions**: We adopt the following addressing conventions: Simulated IP address (called SIP): We will use one byte $(0to9)$ for Network Address, and one byte $(0to9)$ for the machine address. Hence, $(7, 4)$ denotes the 7^{th} network and the 4^{th} machine on it.

   Simulated MAC Address (SMAC): We will use one byte (A to Z) to simulate the MAC address. Note that this allows only for 26 LAN ports over the whole network, as the SMAC address is unique to every LAN interface.

   We will write $[12, 7, 4, ..]$ to refer to the data in a packet which contains the byte count 12, and the SIP address $(7, 4)$. When this packet is sent over the LAN, it will be enveloped in a MAC data unit and will look like $[14, S, 7, 4, ...]$. The 14 and S have been added as the MAC header. SIP packets will have the following structure: $[12, 7, 4, \text{type...}]$, where type is a one byte constant from the set $\{A, D, B, C, L, R\}$, where A means ack; D means data; B means WNAN broadcast; C means control packet; L is ARP Request packet broadcast on the LAN; and R is the ARP reply packet being sent to a given MAC address which first sent out an ARP query **NOTE: please make sure that outputs are neatly generated and printed out on screen in highly readbale forms.**

2. **Collision Detection**: Simulate the simple LAN shown in Fig 1. We will not simulate the carrier sensing operation here. The collision detect operations of the LAN are specified as follows:

   - ☞ If no input port of the LAN has data, exit
   - ☞ If there is more than one input port with data, exit, simulating loss of information due to collision.
   - ☞ Otherwise, copy the relevant input buffer to each of the output buffers.

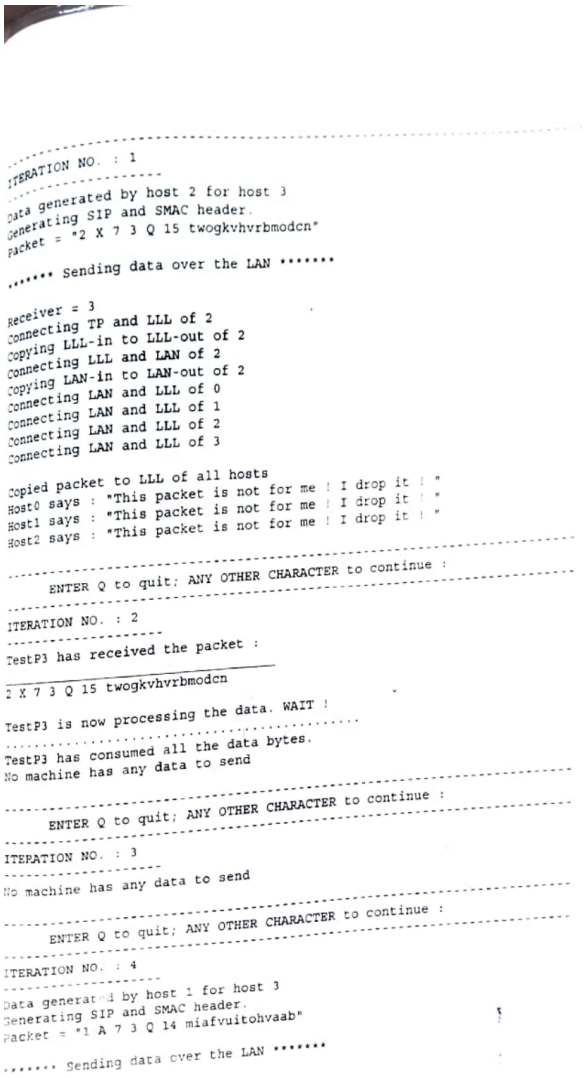
The LAN delivers copies of all packets received to each LLL processor. The LLL processor discards any packet not having the corect SMAC address of the receiving unit.

2 Further enhancements

These ideas are not for the current submission, but for you to enhance this project in the future. You may add the following functionalities to your code later:

- ☆ Routing Table and address resolution protocol (ARP) table.
- ☆ Flow control
- ☆ A new device is connected to the LAN and the router does not have its knowledge. A packect arrives to the router for that newly added host. Use address resolution protocol (ARP) to find the route mapping. The Routing Table will then get an update with the entry of this new device.

3 Sample outoput



```
------------------------------------------------------------------
ITERATION NO. : 1
------------------
Data generated by host 2 for host 3
Generating SIP and SMAC header.
Packet = "2 X 7 3 Q 15 twogkvhvrbmodcn"

******* Sending data over the LAN *******

Receiver = 3
Connecting TP and LLL of 2
Copying LLL-in to LLL-out of 2
Connecting LLL and LAN of 2
Copying LAN-in to LAN-out of 2
Connecting LAN and LLL of 0
Connecting LAN and LLL of 1
Connecting LAN and LLL of 2
Connecting LAN and LLL of 3

Copied packet to LLL of all hosts
Host0 says : "This packet is not for me ! I drop it ! "
Host1 says : "This packet is not for me ! I drop it ! "
Host2 says : "This packet is not for me ! I drop it ! "

------------------------------------------------------------------
     ENTER Q to quit; ANY OTHER CHARACTER to continue :
------------------------------------------------------------------
ITERATION NO. : 2
------------------
TestP3 has received the packet :
--------------------------------
2 X 7 3 Q 15 twogkvhvrbmodcn

TestP3 is now processing the data. WAIT !
.........................................
TestP3 has consumed all the data bytes.
No machine has any data to send

------------------------------------------------------------------
     ENTER Q to quit; ANY OTHER CHARACTER to continue :
------------------------------------------------------------------
ITERATION NO. : 3
------------------
No machine has any data to send

------------------------------------------------------------------
     ENTER Q to quit; ANY OTHER CHARACTER to continue :
------------------------------------------------------------------
ITERATION NO. : 4
------------------
Data generated by host 1 for host 3
Generating SIP and SMAC header.
Packet = "1 A 7 3 Q 14 miafvuitohvaab"

******* Sending data over the LAN *******
```

```
Receiver = 3
Connecting TP and LLL of 1
Copying LLL-in to LLL-out of 1
Connecting LLL and LAN of 1
Copying LAN-in to LAN-out of 1
Connecting LAN and LLL of 0
Connecting LAN and LLL of 1
Connecting LAN and LLL of 2
Connecting LAN and LLL of 3

Copied packet to LLL of all hosts
Host0 says : "This packet is not for me ! I drop it ! "
Host1 says : "This packet is not for me ! I drop it ! "
Host2 says : "This packet is not for me ! I drop it ! "

------------------------------------------------------------------------
     ENTER Q to quit; ANY OTHER CHARACTER to continue :
------------------------------------------------------------------------
ITERATION NO. : 5
-------------------
Data generated by host 0 for host 3
Generating SIP and SMAC header.
Packet = "0 M 7 3 Q 18 vtfuhqyklqqjtamqan"
Data generated by host 1 for host 0
Generating SIP and SMAC header.
Packet = "1 A 7 0 M 17 bztjazuamdziphcxt"
TestP3 has received the packet :
_____
1 A 7 3 Q 14 miafvuitohvaab

TestP3 is now processing the data. WAIT !
.....
TestP3 has consumed all the data bytes.
```

DATA LOST DUE TO COLLISION

```
------------------------------------------------------------------------
     ENTER Q to quit; ANY OTHER CHARACTER to continue :
------------------------------------------------------------------------
ITERATION NO. : 6
-------------------
No machine has any data to send
------------------------------------------------------------------------
     ENTER Q to quit; ANY OTHER CHARACTER to continue :
------------------------------------------------------------------------
ITERATION NO. : 7
-------------------
```

```
******* Sending data over the LAN *******

Receiver = 3
Connecting TP and LLL of 2
Copying LLL-in to LLL-out of 2
Connecting LLL and LAN of 2
Copying LAN-in to LAN-out of 2
Connecting LAN and LLL of 0
Connecting LAN and LLL of 1
Connecting LAN and LLL of 2
Connecting LAN and LLL of 3

Copied packet to LLL of all hosts
Host0 says : "This packet is not for me ! I drop it ! "
Host1 says : "This packet is not for me ! I drop it ! "
Host2 says : "This packet is not for me ! I drop it ! "

-------------------------------------------------------------------------
     ENTER Q to quit; ANY OTHER CHARACTER to continue :


                USER REQUESTED TERMINATION

-------------------------------------------------------------------------
```