Problem Statement

Implement 1-persistent, non-persistent and p-persistent CSMA techniques.

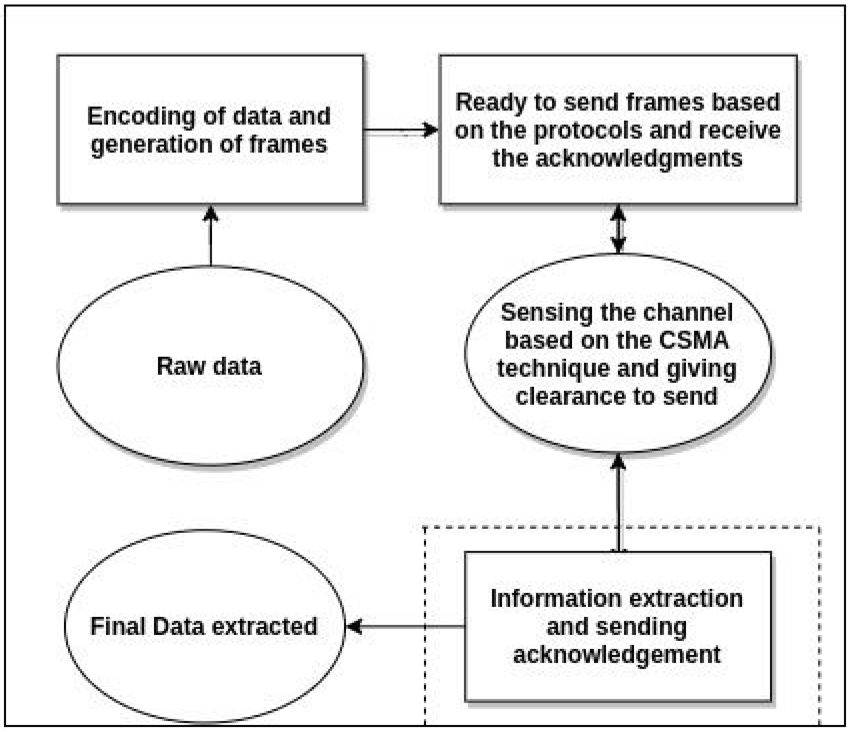
Design description

Purpose of the program: The program tries to simulate the environment of data communication at the Media Access Control layer for analysing the performances of various channel sensing modules namely 1-persistent, non-persistent and p-persistent. The key points are -

1. Encoding Data using error handling modules.
2. Sensing the channel before sending the data onto the channel to minimize collisions.
3. Sending data over the channel as per the protocol.
4. Receiving data from the channel.
5. Sensing the channel again.
6. Sending Acknowledgement through the channel.
7. Receiving the acknowledgement and repeating the above steps until all the frames are sent successfully.

In the later sections of the report, a complete analysis of the results over various possible test cases is shown for further examination of different methods used.

Structural design:


Encoding of data and generation of frames
Ready to send frames based on the protocols and receive the acknowledgments
Raw data
Sensing the channel based on the CSMA technique and giving clearance to send
Final Data extracted
Information extraction and sending acknowledgement

1. The raw data is read from the input file.
2. CRC is used to encode the data.
3. The program sends the data onto the buffer and waits for clearance from the sensing module.
4. The data is being sent onto the channel after getting clearance.
5. Receiver module receives the data from the channel.
6. An acknowledgement is sent to the sender via channel after getting clearance from the sensing module.
7. The sender module receives the acknowledgement and sends the next frame.

Input and output format:

1. The input is read from the file “input.txt”.
2. The logs are shown in a file named “temp_file.txt”.
3. The received message is stored in the file “output.txt”.

Code Snippet

The sender opens a socket and binds to localhost and predefined port.
Input is taken from the user and using the CRC module in Ass1.

Iterate over the frames, and for each frame, **send** the frame to the server using the method.

```
void saw_send(String[] frames) throws IOException
   {
       for(int i=0;i<frames.length;i++)
       {
           System.out.println("Sending frame: " + frames[i]);
           out.writeUTF(frames[i]);
           String ack=input2.readUTF();
           //while(ack.substring(3,ack.length()-3).equals("Received")&&
!ack.substring(0,3).equals(stnID)) ack=input2.readUTF();
           while(!ack.equals(stnID+"Received"+rstnID))
           {
               /*
               System.out.println("Sending Frame again...");
               out.writeUTF(frames[i]);*/
               ack=input2.readUTF();
           }
           System.out.println("Sending next frame");
       }
       String ack="000";
       while(!ack.equals(stnID+"Received"+rstnID))
       {
           // sense();
           System.out.println("Sending frame: 010101110");
           out.writeUTF(rstnID+"010101110"+stnID);  // 010101110 denotes
the message has ended
           while(!ack.substring(0,3).equals(stnID)) ack=input2.readUTF();
       }
       System.out.println("Message Sent.");
   }
```

The corresponding sensing of the channel is done as follows:

```
void sense() throws InterruptedException
   {
       System.out.println(IP+" Sensing Channel...");
       int i=0;
       int rand_int=1;
       // rand_int=rand.nextInt(10);                   // uncomment this
line for non-persistent
       while(ServerSide.busy!=0){sleep((rand_int++)*100);i++;}
      // System.out.println(IP+" Clear to Send after
"+Double.toString(i*rand_int*0.1)+"s");
      ServerSide.busy=1;  //comment this line for p-persistent
   }
```

The busy variable defines the status of the channel. It can acquire 2 values: 0 (channel is free) & 1 (channel is busy).

Once, the channel is free, the server sends the data over to the receiving station.

The server then clears the busy variable.

```
void clear() {
ServerSide.busy=0;
}
```

In case of p-persistent, the channel waits with a probability of p for 1 second and checks again if the channel is free or not. If the channel is free, the data is sent else back-off procedure is applied.

```
boolean after_sense(double p) throws InterruptedException //p-persistent
protocol after sensing the channel free
    {
        double R=rand.nextDouble();
        if(R<=p) return true;
        sleep(1000);
        if(ServerSide.busy==1) return false;
        ServerSide.busy=1;
        return true;
    }

void back_off() throws InterruptedException
    {
            int r=rand.nextInt((int)java.lang.Math.pow(2,K)-1);
            sleep(r*100);
    }
```

The workflow of p-persistent sensing. The parameter of after_sense(p) defines the probability. If p=0, non-persistent method is being implemented.

```
boolean ok2=true;
while(true){
sense();
boolean ok=after_sense(0);
       if(!ok)
       {
             K--;
            if(K!=0) back_off();
            else {ok2=false;K=8;break;}
        }
        else {K=8;break;}
}
```

Analysis of Algorithm

	Stations Involved		Throughput (bits/s)		
Bits transferred	**No of Senders**	**No of Receivers**	**1-persistent**	**non-persistent**	**p-persistent**
232	1	1	11.557238	9.1439382	7.494508334
464	2	2	6.722494277	6.244532669	6.558396585
696	3	3	4.555927943	2.963631641	3.97841595

For p-persistent	
p	**Throughput(bits/s)**
0.2	3.546505851
0.4	4.68947395
0.6	6.802621355
0.8	9.502938948

Discussion

Theoretically, non-persistent is able to provide a maximum throughput of 90% while 1-persistent is able to provide a maximum throughput of 50% only. On the contrary, the results above represent a completely different scenario. Few points are to be noted here. First, the flow control used here is Stop and Wait being the simplest of all. Hence, the contemporary receiver and sender are not competing with each other rather with other senders and receivers. Second, the sample space of stations used is very low due to machine and simulation constraints. These conditions together with the possibility of manual error(I hope this is not the case, though) leads to the above results. Evidently, I am not providing here a pictorial view of change of throughput through graphs. Similar arguments can be made for unexpected results of p-persistent protocol as well.

Comments

- ❖ The main issue was to synchronise different threads fighting for the channel to get free leading to various collisions. The code often broke due to this problem.
- ❖ However, I could have made this easier by not implementing a complete data transfer procedure. But in the end, I wanted to build a system integrable enough to make a complete simulation of data transfer in a network.
- ❖ The assignment was not too hard nor too easy from a student's point of view. This task helped understand the concepts of socket programming. The concepts of threading and multiprocessing were tested as well.