CSMA / CD

Developer: Pranay Sarkar <pranay.sarkar@stud.tu-darmstadt.de>
Matrikulation Number: 2337328
Group:: A:CSMA/CD

Supervisor: Shadi Shahoud <shadi.shahoud@stud.tu-darmstadt.de>

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Communication Networks 1
Bonus task
Group: A:CSMA/CD
Summer 2015
Fachbereich Informatik (FB 20)

Contents

1 Preface

OSI model of computer networks consists of 7 layers, namely:

- Physical layer
- Data Link layer
- Network Layer
- Transport Layer
- Session Layer
- Presentation layer
- Application Layer

CSMA/CD is a media access control, used in *data link layer* (mostly in local area networking and Ethernet topology).

It should be noted that the *data link layer* of local to a network and have a single broadcast domain. That is one of main reason for which media access protocols are used.

2 Media Access Control

Data communication protocol, **Media Access Control (MAC)**, is a sublayer of data link layer. This MAC sublayer provide channel access control and addressing mechanisms for terminals in a multiple access network which enables those terminals to communicate over the network, which incorporates a shared medium.

The hardware used for implementing this MAC sublayer is called **Medium Access Controller**.

2.1 Multiple Access Protocols

Some of the commonly used multiple access protocols are:

- CSMA/CD
- Token Ring
- Token bus
- CSMA/CA
- Slotted ALOHA
- CDMA

Here we will be working with and discussing CSMA/CD.

}

3 CSMA/CD

3.1 Algorithm

CSMA/CD Algorithm works as follows:

1. Any host (network adapter) attached to the shared medium (bus) can start transmitting any time. No fixed time-slots are used.
2. Any host (network adapter) never transmits a frame when it senses that some other host (network adapter) is transmitting. It uses carrier sensing for that purpose.
3. Any host (network adapter) aborts its frame transmission instantly when it detects that another host is transmitting. It uses collision detection for that purpose.
4. The host (network adapter), waits for a random amount of time before trying to retransmit the data. The wait time is comparatively small than the time to transmit the frame. This time period is called **backoff phase**.
   It should be noted that, amount of retransmission can vary in different environments as per the need.

3.2 Implementation

3.2.1 Language Used

We have used Java as our main development language.

3.2.2 Host Enumeration

We have emulated host as thread inside the program. Each host will have its own identifier number.

Thread handling the virtual hosts:

```
class virtualHost extends Thread {
        ....
        /**
        * Default constructor, for hosts, running a common bus (channel)
        * @param name: Hostname
        * @param total: Total sending number
        * @param propDelay: Propagation delay
        */
        public virtualHost(String name, int total, int propDelay) {
                ...
        }
}
```

We managed the virtual hosts as an array of **virtualHost object** (in *CSMACD_Main_GroupA.java*):

```
//Array container of virtual hosts
virtualHost[] virtualHosts=new virtualHost[hostNumber];
```

Each virtual host have its own number:

```
//Assign each virtual host thread to each of the virtual hosts
for(int j=0;j<hostNumber;j++){
virtualHosts[j]=new virtualHost("EndHost"+j,transNumber,delayTime);
}
```

To start virtual host, just start the threads:

```
//Starting host threads (start sending)
for (int i = 0; i < hostNumber; i++) {
virtualHosts[i].start();
}
```

3.2.3 Shared Medium

We have emulated the shared medium as a static integer variable **bus**. There are 3 states of the *bus*:

- **State = 0** :: Means the shared bus is not in idle state. Some host is trying to send some frame(s).
- **State = 1** :: Means any packet is sent successfully and te bus has just become idle for another transmission.
- **State = 2** :: Means a collision have just happened in the shared bus when one of the hosts tried to send a frame.

Code fragment for implementing this (in *virtualHost.hava*):

```
public static int bus = 0;          /**< status indicator of bus whether
           that is occupied or not 0= bus not idle, 1= sent, 2=collission*/
```

3.2.4 Number of retransmission tries

There have to a count of number of tries for retransmission after which the host will not be trying to send the frame anymore.
In our implementation we have made this retransmission try count to 10.
Code fragment (in *virtualHost.hava*):

```
//try to send: 10 times..
while (count < 10) {
        //According to the situation of the bus, do transmission—retransmission
        ...
}
//Retransmission failed for last 10 times? Transmission failure.
if (count >= 10){
        System.out.println("Host computer[" + name + "]Packet " +n+ "
         transmission failure.");
}
```

3.2.5 Backoff timer

The **random backoff timer** As per the CSMA/CD algorithm there exists a random backoff timer in case of collision while sending any packet.

In the Emulation that is done using a random number generator and the back-off time increases as the number of failed transmission gets higher.

Code fragment (in *virtualHost.hava*):

```
//bus = 2 means:: collision between packets while trying to sent them
if(bus==2) {
        count++;
        System.out.println("Host computer["+name+"]Packet " +count+ "
         Collision!");
        bus=0;
        //increase idle time after each failed transmission
        try {
                backoffTimer timer = new backoffTimer();
                Thread.sleep(2*propDelay*timer.backoffTime(count));
        } catch (InterruptedException e) {
        System.err.println("Inturrupted: Interrupt exception");
        }
        continue;
}
```

The function: **backoffTime()** generates a random backoff time incrementing with failed transmission number.

Working mechanism of backoffTime() in *backoffTimer.java*:

```
public class backoffTimer {
/**
*
* Randomly selected back-off time,
* Calculated according to the retransmission number
* random multiples by k times for k-th retransmission
* @param transNum  : number of rretransmission
* @return Random multiples
*/
        public int backoffTime(int transNum) {
                int rndom;
                int temp;
                temp=Math.min(transNum,10);
                rndom=(int)(Math.random()*(Math.pow(2,temp)-1));
        return rndom;
        }
}
```

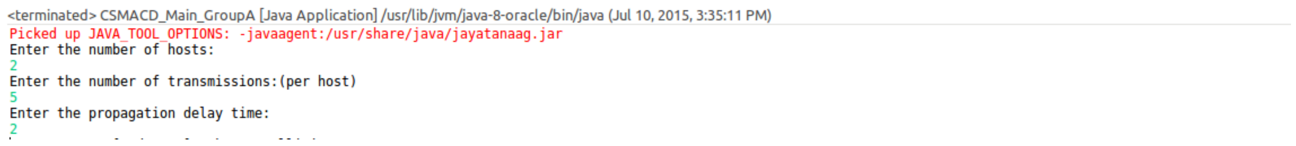
3.3 Working Program

3.3.1 Initial Inputs

When the program is started it will have 3 input sections:

1. Number of hosts to be emulated
2. Number of transmissions (per host)
3. Propagation delay time to emulate

Screenshot from working program:

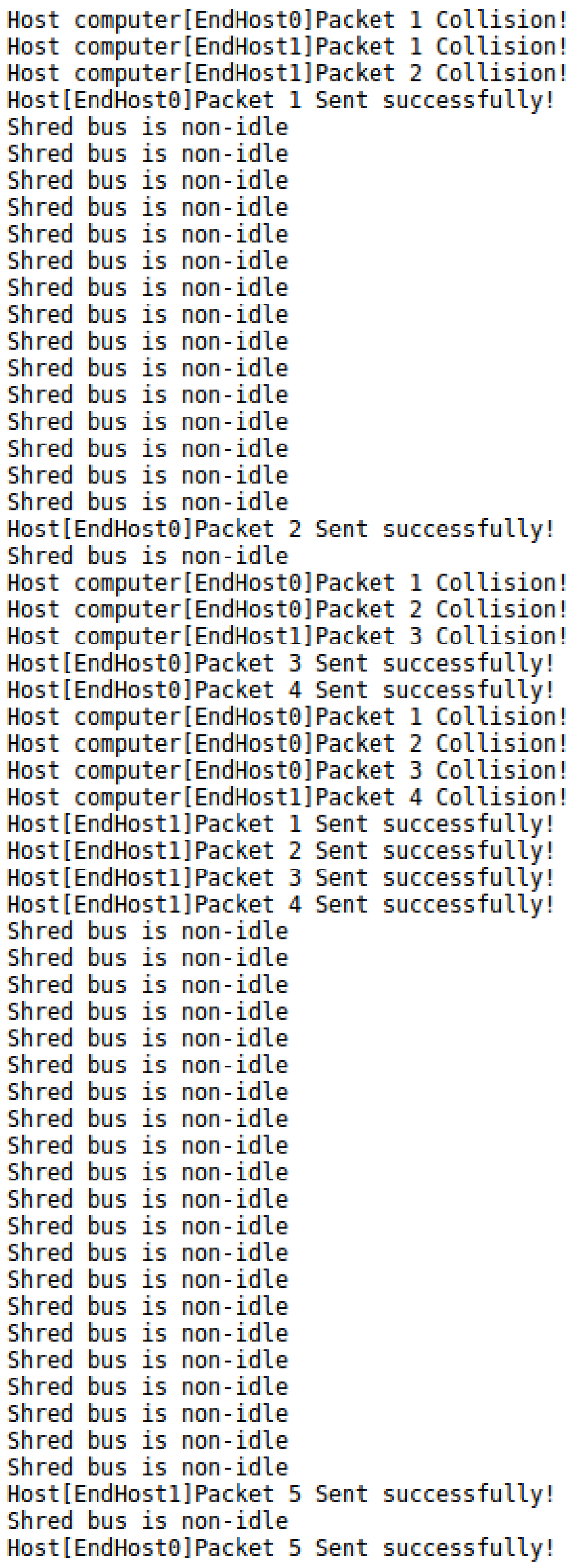


```
<terminated> CSMACD_Main_GroupA [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Jul 10, 2015, 3:35:11 PM)
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Enter the number of hosts:
2
Enter the number of transmissions:(per host)
5
Enter the propagation delay time:
2
```

Figure 3.1: Input parameter screen from running program

3.3.2 Result of emunation of CSMA/CD

Trying with 2 hosts, trying to send 5 frames and propagation delay of 1 unit, Screenshot of output:



```
Host computer[EndHost0]Packet 1 Collision!
Host computer[EndHost1]Packet 1 Collision!
Host computer[EndHost1]Packet 2 Collision!
Host[EndHost0]Packet 1 Sent successfully!
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Host[EndHost0]Packet 2 Sent successfully!
Shred bus is non-idle
Host computer[EndHost0]Packet 1 Collision!
Host computer[EndHost0]Packet 2 Collision!
Host computer[EndHost1]Packet 3 Collision!
Host[EndHost0]Packet 3 Sent successfully!
Host[EndHost0]Packet 4 Sent successfully!
Host computer[EndHost0]Packet 1 Collision!
Host computer[EndHost0]Packet 2 Collision!
Host computer[EndHost0]Packet 3 Collision!
Host computer[EndHost1]Packet 4 Collision!
Host[EndHost1]Packet 1 Sent successfully!
Host[EndHost1]Packet 2 Sent successfully!
Host[EndHost1]Packet 3 Sent successfully!
Host[EndHost1]Packet 4 Sent successfully!
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Shred bus is non-idle
Host[EndHost1]Packet 5 Sent successfully!
Shred bus is non-idle
Host[EndHost0]Packet 5 Sent successfully!
```

Figure 3.2: Output screen from running program

4 Conclusion

There are several channel access control mechanism available. CSMA/CD being one of them. It is also the most widespread protocol. This protocol is used within a network collision domain i.e. Ethernet and hub based networks.

Multiple access protocols are not required in switched full-duplex network, such as the Ethernet networks of today. Normally it is available mainly for compatibility reasons.

5 Acknowledgement

I would like to thank **KOM - Multimedia Communications Lab** department (FB 20) and Communication Networks 1 group for proving me the opportunity of working with this project. I would also like to thank the instructor of the project for giving the initial idea to our group.

Although the other 2 members left the group afterwards, the initial idea helped me a lot to understand the scope of the project and to implement it.

6 References

1. https://en.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_detection
2. https://en.wikipedia.org/wiki/Media_access_control
3. http://www.engr.uconn.edu/~bing/cse330/papers/ethernet_552.pdf