Florida Institute of Technology

CSE4510 – CSE5231

Fall 2015

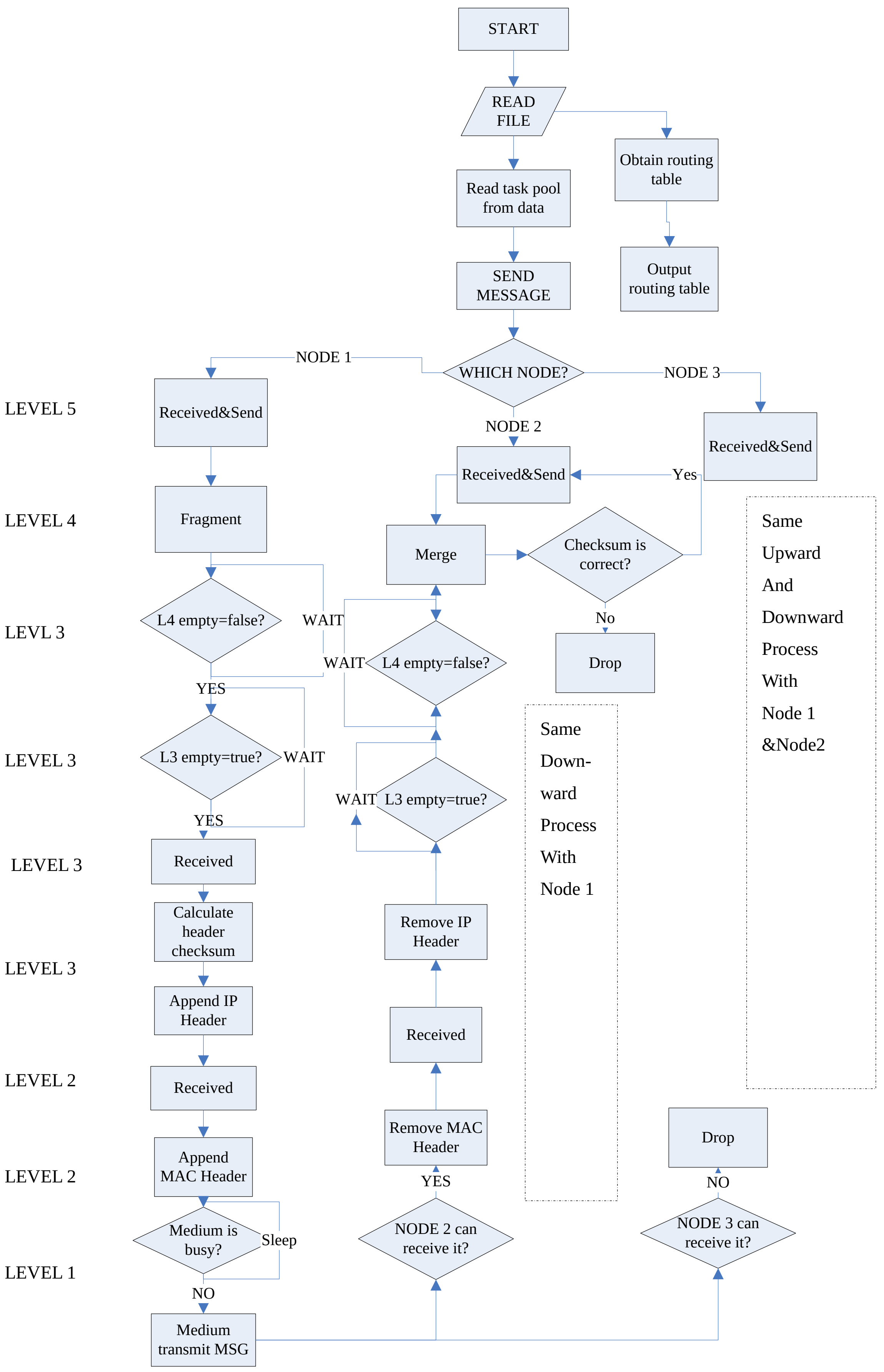
Class Final Project

GROUP Number: _____ 5______

Student: __________Lun Jin_________________

Student: __________Dongning Li_____________

Student: __________Mengjin Ye______________


START
READ FILE
Obtain routing table
Read task pool from data
Output routing table
SEND MESSAGE
WHICH NODE?
NODE 1
NODE 2
NODE 3
LEVEL 5
Received&Send
Received&Send
Received&Send
Yes
LEVEL 4
Fragment
Merge
Checksum is correct?
No
Drop
Same Upward And Downward Process With Node 1 &Node2
LEVL 3
L4 empty=false?
WAIT
WAIT
L4 empty=false?
YES
LEVEL 3
L3 empty=true?
WAIT
WAIT
L3 empty=true?
Same Down-ward Process With Node 1
YES
LEVEL 3
Received
Calculate header checksum
Remove IP Header
LEVEL 3
Append IP Header
Received
LEVEL 2
Received
Remove MAC Header
Drop
LEVEL 2
Append MAC Header
YES
NO
Medium is busy?
Sleep
NODE 2 can receive it?
NODE 3 can receive it?
LEVEL 1
NO
Medium transmit MSG

1. Description

(1) (1)At the beginning of the program, read file from the txt file, and read “task pool” from data, at the same time, we can obtain the routing table on each node based on the information in the file.

(2) User starts to send messages to node. Message reaches level 5, based on the requirement of the Assignment 2, level 5 only receives the sends the message to the level 4;

(3) Message reaches level 4; level 4 starts to fragment the frame into some same length packets to make the frame to be transmitted between the layers. And the maximum number of piece N could use the formula: N=[file length/(MTU-IP header length-MAC header length)]+1;

(4) It should judge the layer 4 is empty or not, and judge the layer 3 is empty or not, either. Only the layer is empty, the fragment packet can transmit, if it is not available, the packet has to wait for the leisure；

(5) Layer 3 receives the packet, and then the responsibility of layer 3 is to append the two IP headers that are from the sender and receiver. In this layer, it will analyze the IP header information and calculate the IP header checksum and add it to the header;

(6) After the packet reaches layer 2, and then the packet is appended to the two MAC headers that are from the sender and receiver.

(7) Then the packet reaches the medium, the first thing is to judge the shared medium is busy or not. Only the medium is not busy, the packet can be transmitted through the medium;

(8) And then the other two nodes will be judged whether the node can receive the packet from the medium. In the diagram above, we assumed that the node 2 can receive the packet, and the node 3 cannot. The node which cannot receive the packet will drop it;

(9) The process of receiving from node 2: packet reaches layer 2, and then it removes the two MAC headers. After reaching the layer 3, the packet will be removed the two IP headers;

(10)Until the packet reaches the layer 4, layer 4 will merge the fragments from layer 3 into a whole frame, in this process, it will check the checksum does or doesn’t correct, if the checksum is correct, then send the frame to layer 5. And the last step is sending the frame to node 2; .If the checksum is error, then drop the message;

*The processes about receiving and sending of node 2 and 3 are totally same with node 1.

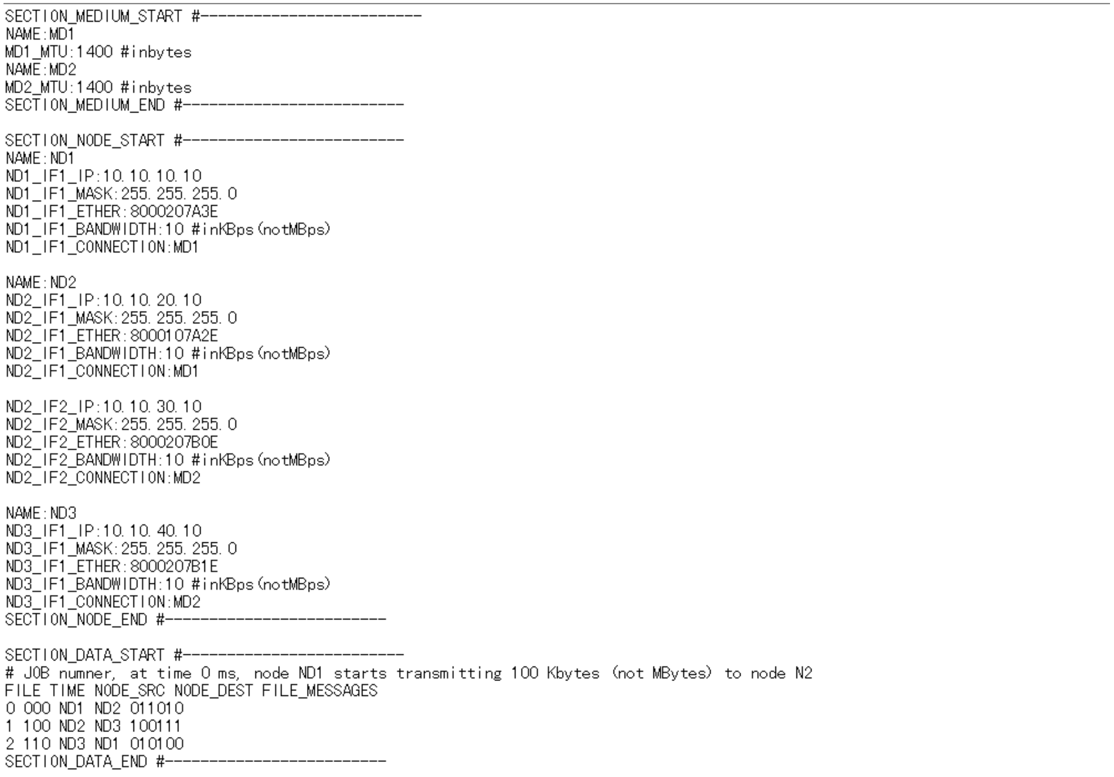
2. Instructions

Programming Environment: Microsoft Visual Studio 2010

TAG: classproject

We design an “inputfile.txt” and put it to the sub file (“class_project/inputfile”) of the project. When running the program, the file will be read into the program. The file includes the information of the mediums and the information about IP address, mask, ether, bandwidth and the medium of the three nodes. Additionally, the file includes the transmission process information between every two nodes. Then when we debugging the project, the project will run based on the file we designed. And the process of the project will display in the “cmd” window.

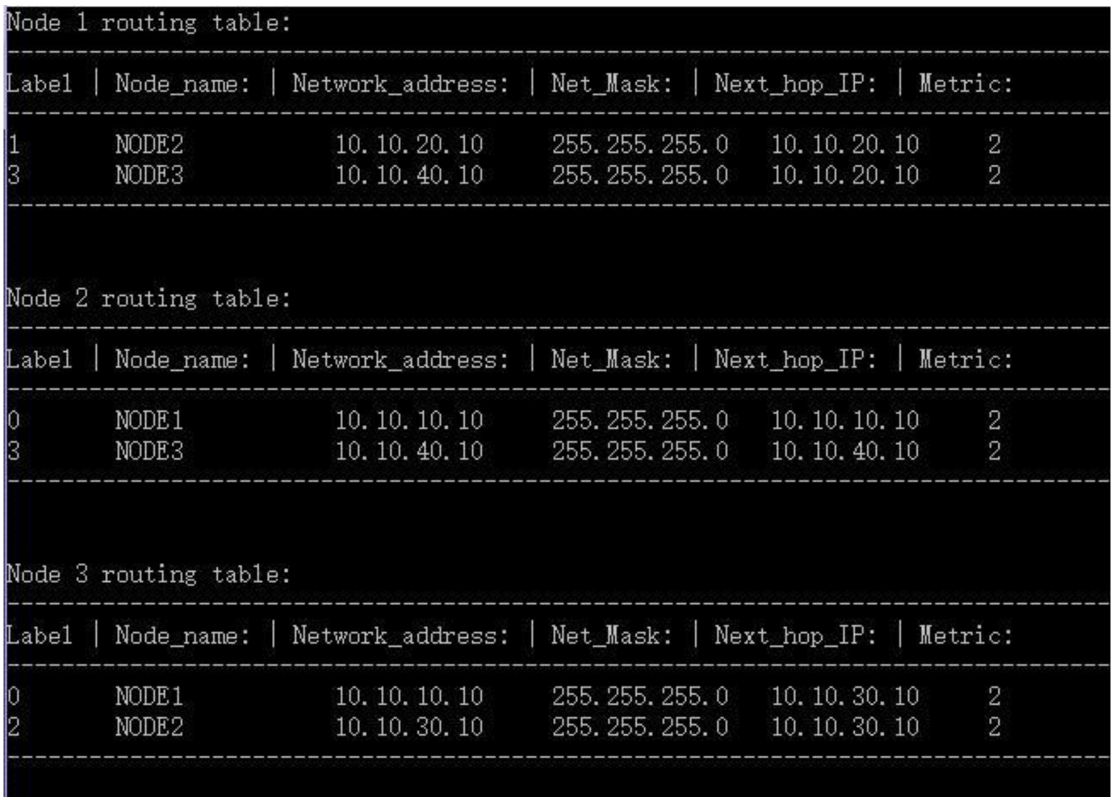
The “inputfile.txt” file:



```
SECTION_MEDIUM_START #-------------------------
NAME:MD1
MD1_MTU:1400 #inbytes
NAME:MD2
MD2_MTU:1400 #inbytes
SECTION_MEDIUM_END #-------------------------

SECTION_NODE_START #-------------------------
NAME:ND1
ND1_IF1_IP:10.10.10.10
ND1_IF1_MASK:255.255.255.0
ND1_IF1_ETHER:8000207A3E
ND1_IF1_BANDWIDTH:10 #inKBps(notMBps)
ND1_IF1_CONNECTION:MD1

NAME:ND2
ND2_IF1_IP:10.10.20.10
ND2_IF1_MASK:255.255.255.0
ND2_IF1_ETHER:8000107A2E
ND2_IF1_BANDWIDTH:10 #inKBps(notMBps)
ND2_IF1_CONNECTION:MD1

ND2_IF2_IP:10.10.30.10
ND2_IF2_MASK:255.255.255.0
ND2_IF2_ETHER:8000207B0E
ND2_IF2_BANDWIDTH:10 #inKBps(notMBps)
ND2_IF2_CONNECTION:MD2

NAME:ND3
ND3_IF1_IP:10.10.40.10
ND3_IF1_MASK:255.255.255.0
ND3_IF1_ETHER:8000207B1E
ND3_IF1_BANDWIDTH:10 #inKBps(notMBps)
ND3_IF1_CONNECTION:MD2
SECTION_NODE_END #-------------------------

SECTION_DATA_START #-------------------------
# JOB numner, at time 0 ms, node ND1 starts transmitting 100 Kbytes (not MBytes) to node N2
FILE TIME NODE_SRC NODE_DEST FILE_MESSAGES
0 000 ND1 ND2 011010
1 100 ND2 ND3 100111
2 110 ND3 ND1 010100
SECTION_DATA_END #-------------------------
```

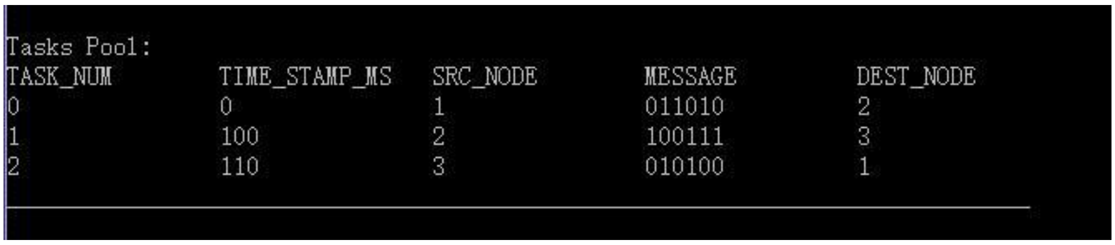
3. Output

(1) The routing table information on node 1, 2, 3, the table includes the “Node_name”, ”Network_address”, “Net_mask”, “Next_hop_IP”, “Metric”:



```
Node 1 routing table:
--------------------------------------------------------------------------------
Label | Node_name: | Network_address: | Net_Mask: | Next_hop_IP: | Metric:
--------------------------------------------------------------------------------
1       NODE2           10.10.20.10      255.255.255.0   10.10.20.10     2
3       NODE3           10.10.40.10      255.255.255.0   10.10.20.10     2
--------------------------------------------------------------------------------


Node 2 routing table:
--------------------------------------------------------------------------------
Label | Node_name: | Network_address: | Net_Mask: | Next_hop_IP: | Metric:
--------------------------------------------------------------------------------
0       NODE1           10.10.10.10      255.255.255.0   10.10.10.10     2
3       NODE3           10.10.40.10      255.255.255.0   10.10.40.10     2
--------------------------------------------------------------------------------


Node 3 routing table:
--------------------------------------------------------------------------------
Label | Node_name: | Network_address: | Net_Mask: | Next_hop_IP: | Metric:
--------------------------------------------------------------------------------
0       NODE1           10.10.10.10      255.255.255.0   10.10.30.10     2
2       NODE2           10.10.30.10      255.255.255.0   10.10.30.10     2
--------------------------------------------------------------------------------
```

(2)The program contains 3 tasks which are transmitting message between nodes:



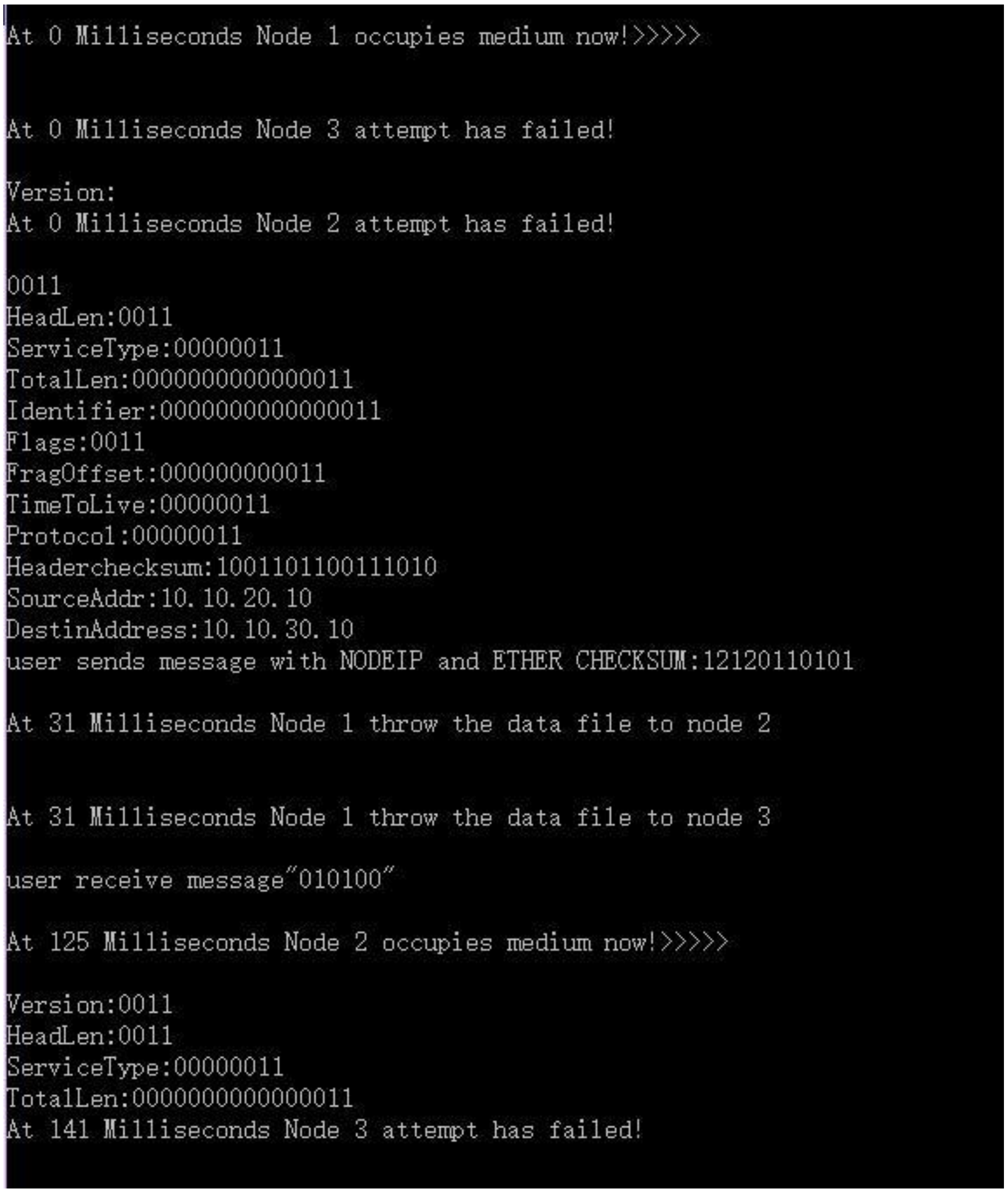
```
Tasks Pool:
TASK_NUM        TIME_STAMP_MS   SRC_NODE        MESSAGE         DEST_NODE
0               0               1               011010          2
1               100             2               100111          3
2               110             3               010100          1

______________________________________________________________________________
```

(3)The process of transmitting message between two nodes begins. If one node occupies t he medium, the other two nodes will fail to transmit messages. And when user receives t he message from node, user will gain the ether checksum and IP header information inclu ding the header checksum.

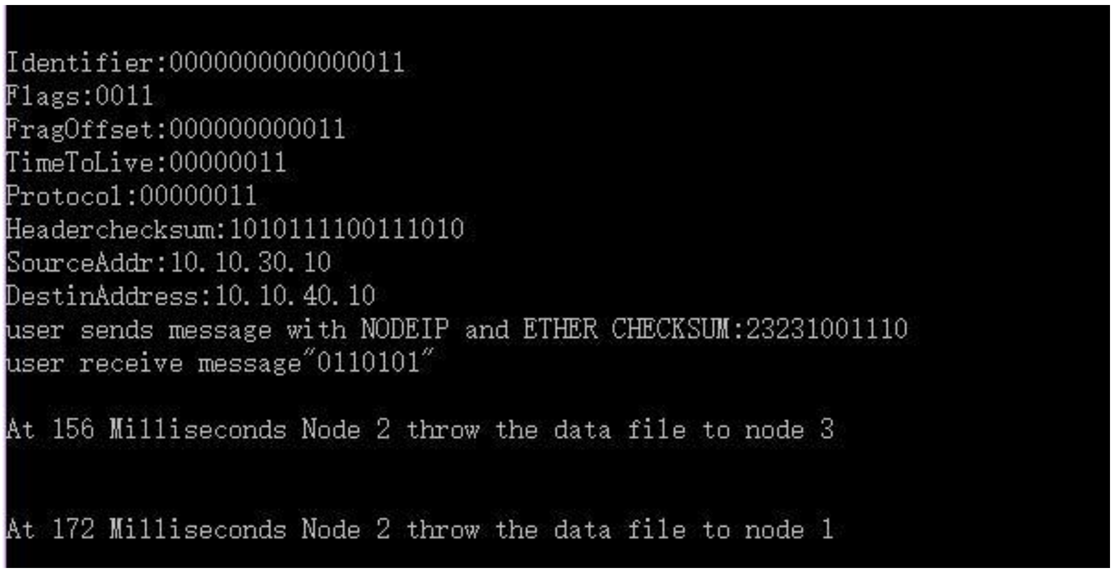
Completed function:

- Show the header information of the message and ether checksum;
- Show the Time_STAMP；
- At particular setting time, one node occupies the medium and other two nodes fail. And user will receive the message from the node which occupies the medium and send message successfully.

Here is the node 1 tries to send data to node 2 and node 3 at 31 milliseconds. And node2 occupies medium at 125 milliseconds:

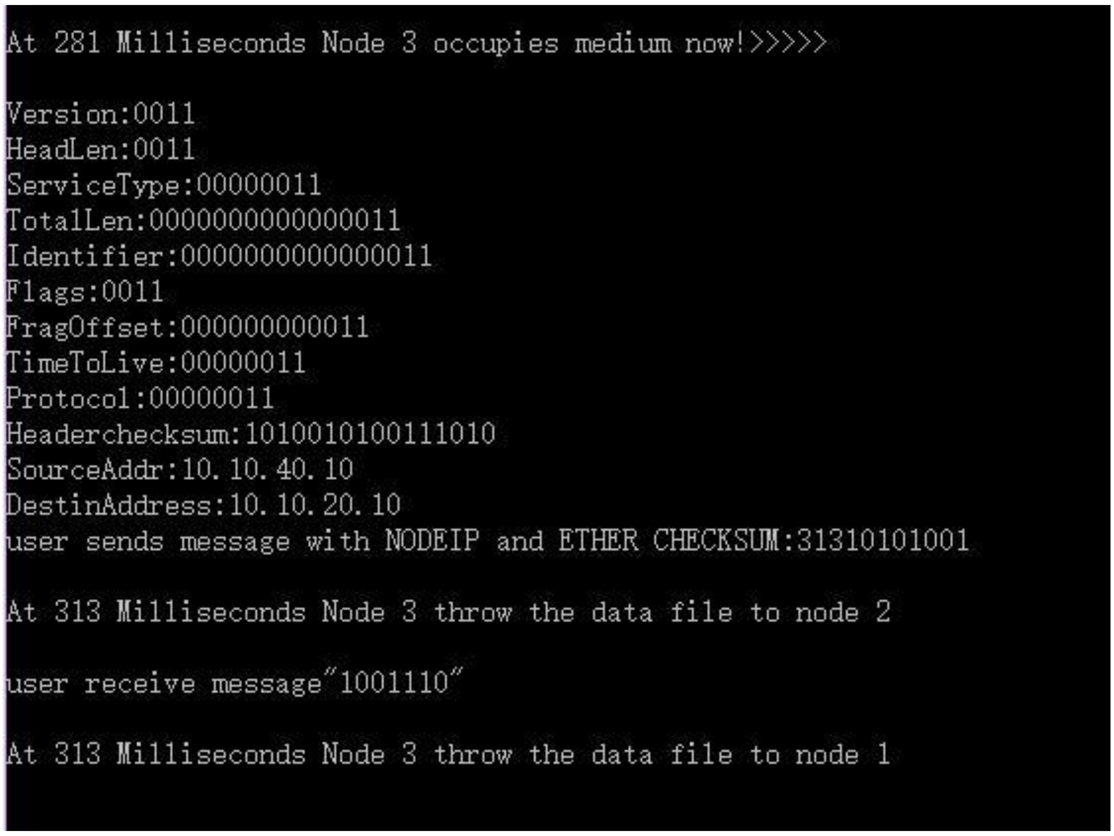


```
At 0 Milliseconds Node 1 occupies medium now!>>>>>


At 0 Milliseconds Node 3 attempt has failed!

Version:
At 0 Milliseconds Node 2 attempt has failed!

0011
HeadLen:0011
ServiceType:00000011
TotalLen:0000000000000011
Identifier:0000000000000011
Flags:0011
FragOffset:000000000011
TimeToLive:00000011
Protocol:00000011
Headerchecksum:1001101100111010
SourceAddr:10.10.20.10
DestinAddress:10.10.30.10
user sends message with NODEIP and ETHER CHECKSUM:12120110101

At 31 Milliseconds Node 1 throw the data file to node 2


At 31 Milliseconds Node 1 throw the data file to node 3

user receive message"010100"

At 125 Milliseconds Node 2 occupies medium now!>>>>>

Version:0011
HeadLen:0011
ServiceType:00000011
TotalLen:0000000000000011
At 141 Milliseconds Node 3 attempt has failed!
```



```
Identifier:0000000000000011
Flags:0011
FragOffset:000000000011
TimeToLive:00000011
Protocol:00000011
Headerchecksum:1010111100111010
SourceAddr:10.10.30.10
DestinAddress:10.10.40.10
user sends message with NODEIP and ETHER CHECKSUM:23231001110
user receive message"0110101"

At 156 Milliseconds Node 2 throw the data file to node 3


At 172 Milliseconds Node 2 throw the data file to node 1
```

Here is node 3 occupying medium and transmitting message at 281 millisecond s:



```
At 281 Milliseconds Node 3 occupies medium now!>>>>>

Version:0011
HeadLen:0011
ServiceType:00000011
TotalLen:0000000000000011
Identifier:0000000000000011
Flags:0011
FragOffset:000000000011
TimeToLive:00000011
Protocol:00000011
Headerchecksum:1010010100111010
SourceAddr:10.10.40.10
DestinAddress:10.10.20.10
user sends message with NODEIP and ETHER CHECKSUM:31310101001

At 313 Milliseconds Node 3 throw the data file to node 2

user receive message"1001110"

At 313 Milliseconds Node 3 throw the data file to node 1
```

4. Code Explanation:

(1)The first step is to read file, after opening file, based on the “config.txt”, and complete the process of counting lines from the documentary, set the timestamp, and set the source node and destination node. It reads data into arrays and builds the array of structures. In the end, output the array of config in struction.

```
int LINES;
void readfile(){
    ifstream file;
    file.open("Config.txt",ios::in);  //open file
    if(file.fail())
    {
        cout<<"File doesn't exist!"<<endl; // if file doesn't exist, output error
        file.close();
        cin.get();
        cin.get();
    }
    else//
    {
        LINES=CountLines("Config.txt");  // count the lines from config documentary
        int *ts=new int[LINES];  // timestamp
        string *sn=new string[LINES]; //source node
        string *msg=new string[LINES];  //message
        string *dn=new string[LINES];  //destination node
        int label;
        int i=0;
        while(!file.eof()) //reading data into arrays
        {
            file>>ts[i];
            file>>sn[i];
            file>>msg[i];
            file>>dn[i];
            label = i+1;

            //build the array of structures
            data_in[i].Time_stamp_ms=ts[i];
            data_in[i].Src_node=sn[i];
            data_in[i].message=msg[i];
            data_in[i].Dest_node=dn[i];
            data_in[i].label=label;

            i++;
        }
        file.close(); //close file

    cout<<"LABEL"<<"\t\t"<<"TIME_STAMP_MS"<<"\t"<<"SRC_NODE"<<"\t"<<"MSG"<<"\t\t"<<"DE
ST_NODE"<<endl;
        for(i=0;i<LINES;i++)    //output the array
        {
            cout<<data_in[i].label
                <<"\t\t"
                <<data_in[i].Time_stamp_ms
                <<"\t\t"
                <<data_in[i].Src_node
                <<"\t\t"
                <<data_in[i].message
                <<"\t\t"
                <<data_in[i].Dest_node
                <<endl;
        }
    }
```

(2)Create the thread for medium, the thread is a sequential control process for the multiple programs running simultaneously. If the medium is not busy, the message between nodes can transmit through the medium. And once one node occupied the medium, the process of other nodes cannot be called comparing to the medium. At this time, the node will call the "sleep" function to wait for the medium to transmit. After the task over, the medium relieves the space and available to other nodes, the process of node 2 and node 3 are the same with the node 1:

```
DWORD WINAPI Node1(LPVOID lpParamter)
{
    char* message_r1; // Message from medium
    string msg2to1;
    while(allset==false) {

        if (WaitForSingleObject(Medium.isbusy,0)==WAIT_OBJECT_0)  //Medium is not busy
Node 1 take it!
        {
            cout<<"<<<<<Node 1 occupies medium now!>>>>>\n"<<endl;

            //*************Task begin at here!***************//
            for (int j=0;j<LINES;j++){

    msg2to1=send5(data_in[j].message,data_in[j].Src_node,data_in[j].Dest_node); // user
calls layer5 and send the message layer by layer
            }
            cout<<msg2to1<<endl;
            //send2(data_in[0].message,"2","5");
            string str;
            char *cstr = new char[msg2to1.length() + 1];
            stringstream stream;
            stream << msg2to1[1];
            str = stream.str(); // change char into string
            if ("1" == str)
            {
                strcpy(message_t1, msg2to1.c_str());
            };
            Medium.Open_connection(1);  // Node 1 begin to transmit
            Medium.Transmit(message_t1);
            Medium.Close_connection(1);  // Task over


            ReleaseMutex(Medium.isbusy);
            Sleep(3000);
        }
        else
        {
            cout<<"Node 1 attempt has failed!\n"<<endl;
            Sleep(500);  // Time interval to check the medium

        }
        Receive("1",public_message);
    }
    return 0;
```

(3) Sending Process: we create a class about the media which is the common medium to transmit messages between nodes, and if it needs transmit the message, it will call the "Open_connection" to connect, in the process of transmit, it calls the "Transmit" function, and after the accomplish of transmission, it will call the"Close_connetion".

```
class Media
{
public:
    HANDLE isbusy;
    void Open_connection(int);
    void Transmit(char*);
    void Close_connection(int);
private:
    int ID;
    char Message[MTU];
} Medium;
```

In layer 3 and layer 2, we need to add IP address and MAC address as header to the packets.

```
string send2(string msg,string src,string des){ //ad mac address to message
    msg=src+des+msg;
    return msg;
}

string send3(string msg,string src, string des){ //ad ip adress to message
    msg=src+des+msg;
    return send2(msg,src,des);
}
```

(4) In layer 4, it needs to split the message split the message into equal size packet in receiving process. We set a vector to store the temporary result and then the divided packets are put into the result and push back to the string. Because we need to save size for fill into the IP and MAC addresses. So we define the packet size is the MTU minus the sum of IP header and MAC header. So the next position of the next packet is the original position plus (MTU-4+1):

```
std::vector<std::string> split(std::string str) //split the message into equal size packet
{
    std::vector<std::string> result;  // set up vector to store the result
    int size=str.size();
    int i;
    for(i=0; i<size; i++)
    {
        int MTU=100;
        std::string s=str.substr(i,MTU-4); // The new packet size is set to (MTU minus
the length of two IP headers and two MAC headers), and we set the length of two headers
are 2 respectively
        result.push_back(s);    //The divided packets are put into the result and push
back to the string
        i=i+95; //The position of next i is set to the packet body length minus 1
    }
    return result;
}
```

After split process, layer 4 sends the divided packets, it creates a vector named “result” and assigned the “split”. And return to the send3 to output the result.

(5) Receiving Process: the process of transmitting message from layer 5

to layer 2, layer5 to layer 2 have to call “receive” function to receive messages between nodes, and the lower layer received message from the higher layer have to call the higher layer function. For example, the “receive” function on layer 4:

```
void rec4(string recmsg,string node){ //send received message to layer5
    rec5(recmsg, node);
}
```

As the same, after the medium transmitting message and send it to the user, layer2 to layer 5 have to call the “send” function. Additionally, the layer 2 needs to erase MAC address to message, and layer 3 needs to erase IP address to message. At layer 2, node will decide whether this file is sent to it, then decides whether sends this to layer 3. The sum is 4. The code:

```
void rec2(string node,string msg ,string i){  // send received and processed message to
layer3
    string str;
    stringstream stream;
    stream << msg[1];
    str = stream.str(); // change char into string
    if (node == str)
    {
        msg=msg.erase (0,4); // erase the mac ip address
        msg=msg.erase(msg.length(),1); //erase checksum
        rec3(node,msg);    //keep the messge only
    }else
        cout<<"\nAt "<<GetTickCount()-start<<" Milliseconds Node "<<i<<" throw the
data file to node "<<node<<"\n"<<endl;

}
```

(6)Output the information of message’s IP header and calculate the header checksum:
After indentifying the data structure of IP header, and identifying the data of every part of IP header, the IP source address and destination address are based on the transmission information.
Then we use “bitset” function to transform decimal system to binary system. When reading IP addresses, we identify the address as four digits and convert it respectively by using the function.
Example of the IP source address:

```
    string pp1 = ip.SourceAddr; //convert ip source address to binary coding
    string ppp1[4];
    for (int j=0;j<4;j++){
        ppp1[j]=pp1.substr(j*3,2);
    }
    int ips[4]={0,0,0,0};
    for(int i=0;i<4;i++){
        ips[i]=atoi(ppp1[i].c_str());
    }
    bitset<8> bss1=ips[0];
    bitset<8> bss2=ips[1];
    bitset<8> bss3=ips[2];
    bitset<8> bss4=ips[3];
```

Before calculating the header checksum, we combine the two 8-digit binary number into a 16-digit binary number by shifting the number to the left. For example: `bitset<16> a=((ip.Version<<12)|(ip.HeadLen<<8)|ip.ServiceType);`

At last, we add the all 16-digit binary number to obtain the result.

```
    bitset<16> a=((ip.Version<<12)|(ip.HeadLen<<8)|ip.ServiceType);  //connect header
coding in binary to 16-digits
    bitset<16> b=TotalLen2;
    bitset<16> c=Identifier2;
    bitset<16> d=((ip.Flags<<12)|ip.FragOffset);
    bitset<16> e=((ip.TimeToLive<<8)|ip.Protocol);
    bitset<16> f=((ips[0]<<8)|ips[1]);
    bitset<16> g=((ips[2]<<8)|ips[3]);
    bitset<16> h=((ipd[0]<<8)|ipd[1]);
    bitset<16> i=((ipd[2]<<8)|ipd[3]);

    hdchecksumt=a.to_ulong()+b.to_ulong()+c.to_ulong()+d.to_ulong()+e.to_ulong()+f.to_
ulong()+g.to_ulong()+h.to_ulong()+i.to_ulong();
    bitset<16> hdchecksum(hdchecksumt);
```

(7)Output routing table: based on the condition of transmission between nodes, we set the different cases of routing table. For example, viewing the routing table on node 1, it contains the node 2 and node 3 transmitting messages to node 1, and the next_hop IP is the same one “10.10.20.10”:

```
string Next_hop_IP;
    switch ( Node.label )
    {
    case 1:
        Next_hop_IP="10.10.20.10";
        RT_input(1,Next_hop_IP);
        RT_input(3,Next_hop_IP);

    cout<<"---------------------------------------------------------------------------
-------\n\n"<<endl;
        break;
    case 2:
        Next_hop_IP="10.10.10.10";
        RT_input(0,Next_hop_IP);
        Next_hop_IP="10.10.40.10";
        RT_input(3,Next_hop_IP);

    cout<<"---------------------------------------------------------------------------
-------\n\n"<<endl;
        break;
    case 3:
        Next_hop_IP="10.10.30.10";
        RT_input(0,Next_hop_IP);
        RT_input(2,Next_hop_IP);


    cout<<"---------------------------------------------------------------------------
-------\n\n"<<endl;
        break;

    }
}
```

(8) Accept a formatted general input file defining the topology and traffic.
Claim the function to handle the confliction between nodes. If the node layer 2 is busy, the node layer 4 detects the confliction, and the transmitting process will stop.

```
 void De_confliction(NODE Node)
{
     if (Node.layer2.isbusy==true)
     {
          Node.layer4.confliction=true;
          //Node.layer4.Stop;
          cout<<"Confliction detected! Node "<<Node.label<<" stopped proccessing"<<endl;

     }
}
```

(11)Main function:
Reading file("inputfile.txt")to stream, if the file exists, then reads data from file and reads medias MTU from the file. And read information about the nodes, for example, fetch the IP address of nodes:

```
    int pos_ND1_IF1_IP = DataBuffer.find("ND1_IF1_IP", 0);// find the position of IP
    int pos_ND2_IF1_IP = DataBuffer.find("ND2_IF1_IP", 0);
    int pos_ND2_IF2_IP = DataBuffer.find("ND2_IF2_IP", 0);
    int pos_ND3_IF1_IP = DataBuffer.find("ND3_IF1_IP", 0);
    ND1_IF1_IP = DataBuffer.substr(pos_ND1_IF1_IP+sizeof("ND1_IF1_IP"), 11); //fetch IP 
Address
    ND2_IF1_IP = DataBuffer.substr(pos_ND2_IF1_IP+sizeof("ND2_IF1_IP"), 11);
    ND2_IF2_IP = DataBuffer.substr(pos_ND2_IF2_IP+sizeof("ND2_IF2_IP"), 11);
    ND3_IF1_IP = DataBuffer.substr(pos_ND3_IF1_IP+sizeof("ND3_IF1_IP"), 11);
```

We use array to structure node and initial node according to “inputfile”. After finding the position of file_messages from the “inputfile”, we process theread tasks from file, and handle the process of transmitting by create threads:

```
int pos_FILE_MESSAGES = DataBuffer.find("FILE_MESSAGES", 0);// find the position 
of FILE_SIZE (KB)
    for (int i=0;i<3;i++){
    data_in[i].label=i;
data_in[i].Time_stamp_ms=atoi(DataBuffer.substr(pos_FILE_MESSAGES+sizeof("FILE
_MESSAGES")+i*16,3).c_str());
data_in[i].Src_node=DataBuffer.substr(pos_FILE_MESSAGES+sizeof("FILE_MESSAGES"
)+5+i*16,1);
data_in[i].Dest_node=DataBuffer.substr(pos_FILE_MESSAGES+sizeof("FILE_MESSAGES
")+8+i*16,1);
data_in[i].message=DataBuffer.substr(pos_FILE_MESSAGES+sizeof("FILE_MESSAGES")
+9+i*16,6);
    }

HANDLE Node_1 = CreateThread(NULL, 0, Node1, NULL, 0, NULL);
    HANDLE Node_2 = CreateThread(NULL, 0, Node2, NULL, 0, NULL);
    HANDLE Node_3 = CreateThread(NULL, 0, Node3, NULL, 0, NULL);
    Medium.isbusy = CreateMutex(NULL, FALSE, Medium_busy_check);
    CloseHandle(Node_1);
    CloseHandle(Node_2);
    CloseHandle(Node_3);

    cin.get();
    return 0;
}
```