Credit Hours System

CMPN405

Cairo University

Faculty of Engineering

Computer Networks-1

Project

Student names:

Ahmed Ibrahim Abdellatif - 1170024

Mahmoud Ezzat ELMaghraby - 1170532

Ahmed Mohamed ELKhatib - 1170100

Ahmed Hany Mubarak - 1170428

Team Member	Workload percentage (phase1&2 + reports)	Functionalities
Ahmed Ibrahim Abdellatif	25 %	Sliding-window protocol (Selective repeat)
Mahmoud ELMaghraby	25 %	Transmission channel noise modeling + Network Architecture + Statistics
Ahmed El-Khatib	25 %	Transmission channel noise modeling + Network Architecture + Statistics
Ahmed Hany	25 %	Hamming + Character count

Integration of selective repeat and topology was conducted by Ahmed Ibrahim and Mahmoud El-Maghraby

Integrating statistics and noise was conducted by Ahmed El-Khatib and Ahmed Hany

---

Selective Repeat

> The implemented code gives the node an ability to send data to more than one node (actually it has the capacity send to all nodes) at the same time which increases the randomness of choosing the pairs
>
> - This was an extra feature added to the original requirements of the project

```
void Node::handleMessage(cMessage *msg)
{
    /*This function handles the received message whether it is:
    - a self-message such as the timers (ack timeout and timeout), the signal to
enable the network layer
    and the signal that simulates receiving data from the network layer
    - a message received from another node such as data, ack, nack, handshake*/
}

//important code segments in this function:
// 1- simulating reciving data from network and buffering them in a queue to be
sent in order according to the window size
int file = (getIndex() % NUMBER_OF_FILES) + 1;
std::ifstream MyReadFile("node"+ std::to_string(file) +".txt");
std::string myText;
destination_index = (rand > node_number)? rand-1 : rand;
while (getline (MyReadFile, myText)) {
    mmsg = new MyMessage_Base("");
    mmsg->setM_Payload(myText.c_str());
    mmsg->setType(data);
    mmsg->setDestination_node(rand);
    buffered_from_network[destination_index].insert(mmsg);
}
// 2- sending the buffered data if network layer is enabled
```

```
while (network_layer_enabled[destination_index] == true &&
buffered_from_network[destination_index].front())
{
    statistics(1);
    statistics(4);
    frame_to_send = check_and_cast<MyMessage_Base *>
(buffered_from_network[destination_index].pop());
    number_of_sent_but_not_acked[destination_index]++;
    outbound_window[destination_index]
[sequence_to_send_next[destination_index]%WINDOW_WIDTH] = frame_to_send->dup();
send_frame(data,sequence_to_send_next[destination_index],seq_where_inbound_windo
w_begins[destination_index],frame_to_send->getDestination_node());
    circular_increment(sequence_to_send_next[destination_index]);
    if(number_of_sent_but_not_acked[destination_index] >= WINDOW_WIDTH)
disable_network_layer(destination_index);
    delete frame_to_send;
}
// 3- starting to send data when a handshake is received, also simulating
receiving data from network layer, then sending a signal to start sending.
// Here data is not sent immediately to be able to recognize the received data
right after the handshake and send a piggy-backed ack with it
else if(type_received == handshake){

    int file = (getIndex() % NUMBER_OF_FILES) + 1;
    std::ifstream MyReadFile("node"+ std::to_string(file) +".txt");
    std::string myText;
    int destination_index = (frame_received->getSource_node() > node_number)?
frame_received->getSource_node()-1 : frame_received->getSource_node();
    while (getline (MyReadFile, myText)) {
        MyMessage_Base *mmsg = new MyMessage_Base("");
        mmsg->setM_Payload(myText.c_str());
        mmsg->setType(data);
        mmsg->setDestination_node(frame_received->getSource_node());
        buffered_from_network[destination_index].insert(mmsg);
    }
    EV<<"handshake received from "<<frame_received->getSource_node()<<" at "<<
node_number <<"\n";
    signal_to_enable_network->setDestination_node(frame_received-
>getSource_node());
    scheduleAt(simTime()+0.1, signal_to_enable_network);
    signal_to_enable_network = signal_to_enable_network->dup();
}
// 4- dealing with piggy-backed acks sent with data frames
while(is_sequence_between(frame_received->getAck(), ack_expected[source_index],
sequence_to_send_next[source_index]) && ack_received_or_not[source_index]
[frame_received->getAck() % WINDOW_WIDTH] == false)
{
    EV <<"ACK received at "<< getIndex() <<" from "<<frame_received-
>getSource_node()<<" for sequence "<< (ack_expected[source_index] + 1) %
(MAX_SEQUENCE + 1) <<"\n";
    ack_received_or_not[source_index][ack_expected[source_index] % WINDOW_WIDTH]
= true;
    number_of_sent_but_not_acked[source_index]--;
    stop_timer(ack_expected[source_index],frame_received->getSource_node());
    circular_increment(ack_expected[source_index]);
}
delete frame_received;
```

```
// all other code segments are highly important but these are the ones I want to highlight for their importance in the report
```

```
void Node::send_frame(frame_type type, sequence_number frame_no, sequence_number ack, unsigned int destination)
{
    /*This function sends the messages according to the passed values as clarified by the names of its parameters*/
}
```

```
void Node::start_timer(sequence_number s_no, unsigned int destination)
{
    /*sending a self-message to simulate timer and setting is_timer_set flag*/
}
```

```
void Node::stop_timer(sequence_number s_no, unsigned int destination)
{
    /*unsetting the flag*/
}
```

```
void Node::start_ack_timer(unsigned int destination)
{
    /*sending a self-message to simulate timer and putting ack_sent_or_not flag as false*/
}
```

```
void Node::stop_ack_timer(unsigned int destination)
{
    /*putting the flag as true*/
}
```

```
void Node::enable_network_layer(int destination_index)
{
    /*setting a flag*/
}
```

```
void Node::disable_network_layer(int destination_index)
{
    /*unsetting the flag*/
}
```

Transmission and Noise

```
//sending with random noise
    rand=uniform(0,1)*10;
    if(rand >= noise)
    {
        rand=uniform(0,1)*10;
        if(rand < noise)
        {
            double delay=uniform(0.1,1);
            EV<<"Frame delayed with time = "<<std::to_string(delay)<<endl;
            rand=uniform(0,1)*10;
            if(rand < noise)
            {
                EV<<"Frame Duplicated"<<endl;
                MyMessage_Base *dupMsg = frame_to_send->dup();
                sendDelayed(dupMsg, delay, "out");
            }
            sendDelayed(frame_to_send, delay, "out");
        }
        else
        {
            rand=uniform(0,1)*10;
            if(rand < noise)
            {
                EV<<"Frame Duplicated"<<endl;
                MyMessage_Base *dupMsg = frame_to_send->dup();
                send(dupMsg, "out");
            }
            send(frame_to_send,"out");
        }
    }
    else
    {
        statistics(2);
        EV<<"Frame Lost"<<endl;
        delete frame_to_send;
    }
```

```
//Modifying payload
        rand = uniform(0,1)*10;
        if(rand < noise) // prob to delay the message
        {
            rand=uniform(0,mypayload.length());
            mypayload[rand] = mypayload[rand] == '0'? '1':'0';
            EV<<"Frame modification at bit "<<std::to_string(rand)<<endl;
            EV<<"frame after modification: "<< mypayload<<"\n";
        }
        mypayload = charCount.to_string() + mypayload;
        frame_to_send->setM_Payload(mypayload.c_str());
        EV<<"frame: "<< mypayload<<"\n";
```

Statistics

```
void Node::statistics(int choice)
{   //assumption: duplicated frames is not considered a retransmitted frame
    /*checking the number of choice and incrementing the statistics variables
accordingly, then print the
    current results*/
}
```

Network Architecture

Using a ring topology (Distributed) and is suitable for any number of nodes

Hamming

```cpp
char Node::calculateParityAtPosition(int position, int newSize, std::string newCode) {
    /*calc hamming parity*/
}
```

```cpp
std::string Node::Hamming(int m, int r, std::string code) {
    /*loops over the payload and uses the above function to apply hamming code over it*/
}
```

```cpp
std::string Node::convertToChar(std::string message) {
    /*inverting the above process*/
}
```

```cpp
int Node::checkHamming(int m, int r, std::string newCode) {
    /*return the error bit position if any*/
}
```

```cpp
std::string Node::convertToCode(std::string message) {
    /*converting string to a string of bits*/
}
```

Character Count

```cpp
std::string Node::characterCount(std::string message, int size) {
    /*framing using character count*/
}
```