# CS2031 Telecommunication II

# Assignment #1: Command & Control

Kai Suzuki, 18308704
3rd November, 2019

Content

1. Introduction
2. Theory
   2.1. Stop-and-Wait Protocol
   2.2. Go-Back-N ARQ protocol
   2.3. Go-Back-N ARQ protocol
3. Implementation
   3.1. Worker
   3.2. C&C
   3.3. Broker
   3.4. Packet
   3.5. Demo
4. Conclusion

## 1.Introduction

The task for this assignment is to create protocols bringing messages from Command & Control (C&C) to several workers through a broker by the use of sockets, datagram packets and threads. In other words, the aim of this assignment is to design a packet layout and handle packet for the communication between a number of nodes.
To achieve this task, this report will discuss the necessary knowledge and my understanding related to this area in the Theory section and my approach for this task in the Implement section.

## 2.Theory

First of all, it is necessary to introduce packet, flow control and the types of flow control protocols. The packet is an important concept in terms of the interaction with multiple nodes. Packetizing is compressing data in frame or cell as adding header and trailer. Next, flow controls are the process of dealing with the data transmission between two nodes, a sender and a receiver. Also, this is defined as “the control of the amount of data that a sender can transmit without overflowing the receiver” in class. Finally, the flow control protocol is divided into the noiseless channel and the noisy channel. Then the former has the simplest protocol and stop-and-wait protocol, on the other hand the later has Stop-and-Wait ARQ protocol, Go-Back-N ARQ protocol and Selective Repeat ARQ Protocol. From here, stop-and-wait protocol, Go-Back-N ARQ protocol and Selective Repeat ARQ will be explained much more.

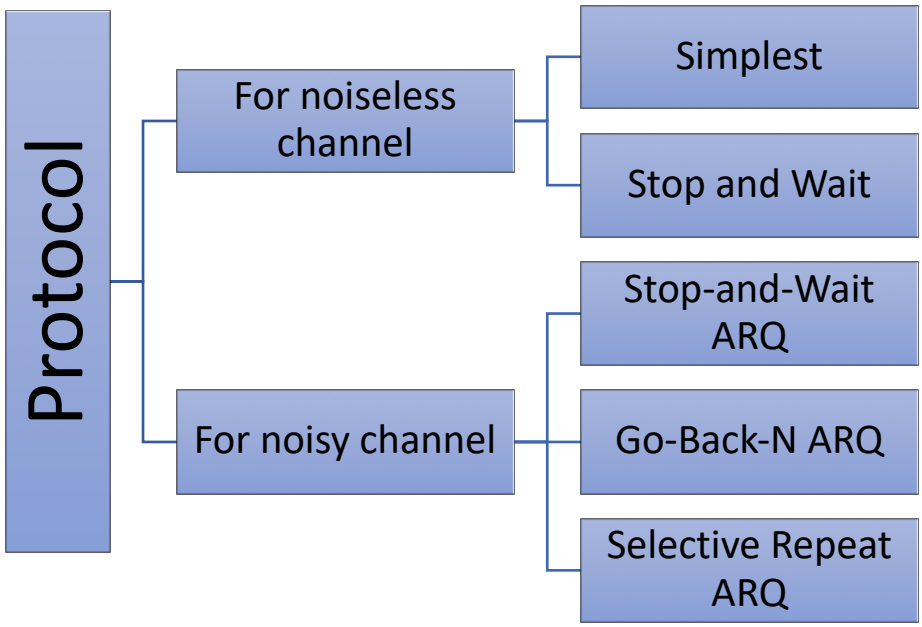


Figure 1: Types of protocols

## 2.1 Stop-and-Wait Protocol

This is a process of managing data between two connected nodes and guarantees the data from each other is not lost and received in the correct order. The sender sends the frame and waits for an acknowledgement (ACK) while the receiver replies to the received frame with ACK. Also, this type of protocol is implemented in my program.

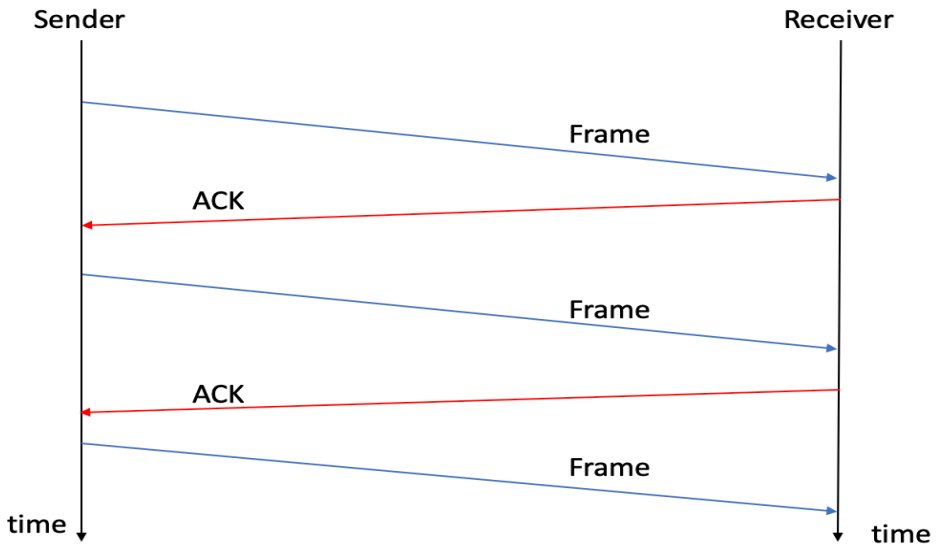


Figure 2: Stop-and-Wait Protocol

## 2.2 Go-Back-N ARQ protocol

This protocol is a method sending and receiving data specified by window size. The start of the sender's window size and the frame number are 0. After sending frame number 0, the next frame number is added by 1. When the receiver received the proper data from the sender, it sends ACK which contains the next required frame number. Then the sender receives ACK from the receiver and the start of the sender's window size becomes the number contained in the ACK.

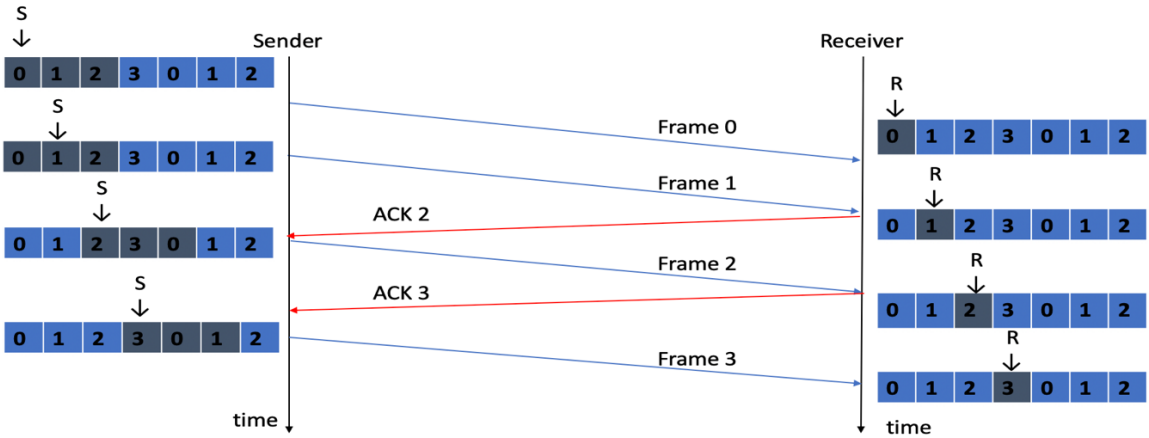

Figure 3: Go-Back-N ARQ protocol

### 2.3 Selective Repeat ARQ Protocol

This is a protocol that sends a message specified window size from the sender. This protocol looks similar to the previous one, Go-Back-N. However, when a frame is lost, this method needs to resend only the lost frame. For example, if the frame 1 from the sender is lost as below diagram and the sender sends the frame 2, the receiver received the frame 2 which is not required and then sends NAK to the sender. After getting the NAK from the receiver, the sender sends frame 1 again.

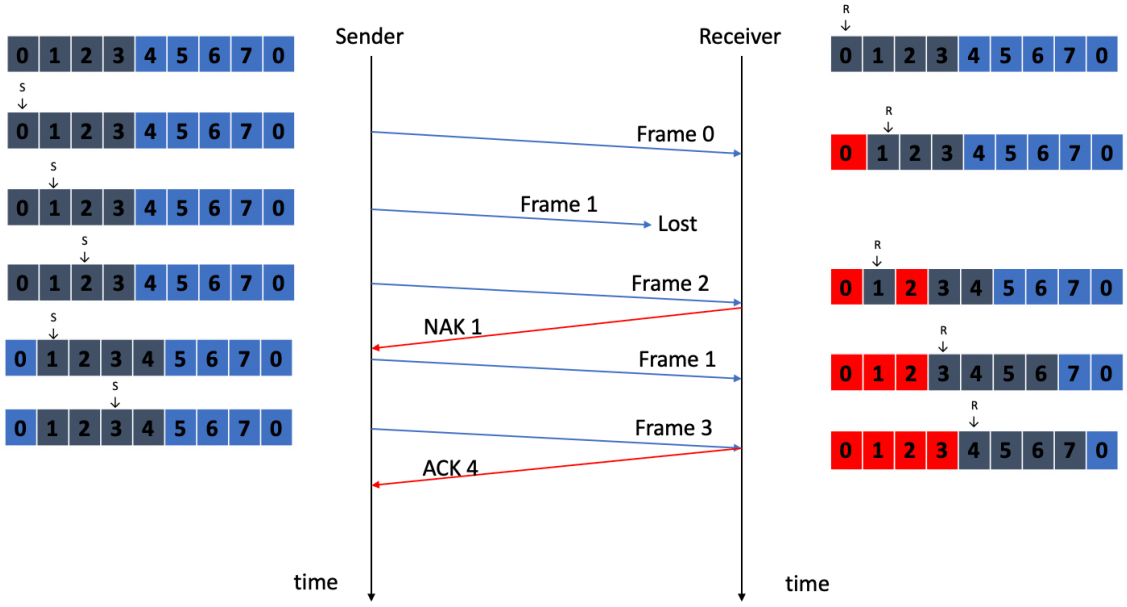


Figure 4: Selective Repeat ARQ Protocol

### 3.Implementation

This section consists of my approaches to solve the problem given in this assignment, following worker, C&C, Broker and packet.

In my code, there are two workers, two C&C applications and one broker and the Stop-and-Wait protocol is implemented.

Also, the job my program can do is only to print a specific word certain times, my program cannot calculate and send any file.

## 3.1 Worker

This class has main two functions which are "`start()`" and "`onReceipt()`."
In the first function, a protocol is created with the inputted worker's name for sending a job request to the broker.
The worker can handle any received messages within the second function. The first if statement is to deal with the reply from the broker, to the previous worker's job request. The next else if statement can print the certain sent word. The last else statement can restart "`start()`" function again if this worker does not receive the message containing reply to the job request from the broker. These three statements can solve the problem which is caused by the message lost.

```
public synchronized void start() throws Exception {
    if(!getBack) {
        DatagramPacket packet= null;

        byte[] payload= null;
        byte[] header= null;
        byte[] buffer= null;

            payload= (terminal.readString("If you are volunteering for work, enter your name: \n")).getBytes();
            header= new byte[PacketContent.HEADERLENGTH];

            buffer= new byte[header.length + payload.length];
            System.arraycopy(header, 0, buffer, 0, header.length);
            System.arraycopy(payload, 0, buffer, header.length, payload.length);

            terminal.println("Sending packet...");
            packet= new DatagramPacket(buffer, buffer.length, dstAddress);
            socket.send(packet);
            terminal.println("Packet sent");
            this.wait();
    }

}
```

Figure 5: "`start()`" function in worker1 and worker2

```
/**
 * Assume that incoming packets contain a String and print the string.
 */
public synchronized void onReceipt(DatagramPacket packet) {

    try {
        StringContent content= new StringContent(packet);
        this.notify();
        if(content.toString().equals("accept your request")) {        // worker's request is accepted by the broker
            getBack=true;
            terminal.println("get reply from broker");
        }
        else if(getBack) {                                            // get the word to print
            terminal.println("\n");
            terminal.println(content.toString());
        }
        else {                                                        // get other types of messages, restart again
            start();
        }
    }
    catch(Exception e) {e.printStackTrace();}

}
```

Figure 6: "`onReceipt()`" function in worker1 and worker2

## 3.2 C&C

This class has the same main two functions as well as Worker.
Within the "`start()`" function, the protocol for a job description is crafted and sent to the broker. To make workers print a certain term, there is a fixed format: "write ( 'word' ) 'number' times." The user can substitute any favourite word for 'word' and favourite number for 'number'; for example, "write (hello) 15 times" and "write (world) 7 times."
The "`onReceipt()`" function can cope with received messages. The first if statement can initiate again the "`start()`" function if the condition is matched. The next else if statement can report the job described in "`start()`" function is finished. The end else statement can continue the condition in which this C&C is waiting for any messages.

```
public synchronized void start() throws Exception {

    DatagramPacket packet= null;

    byte[] payload= null;
    byte[] header= null;
    byte[] buffer= null;

        payload= (terminal.readString("Enter work discription\n"+"(e.g) write ( 'word' ) 'number' times  //change within ' '\n")).getBytes();


        header= new byte[PacketContent.HEADERLENGTH];

        buffer= new byte[header.length + payload.length];
        System.arraycopy(header, 0, buffer, 0, header.length);
        System.arraycopy(payload, 0, buffer, header.length, payload.length);

        terminal.println("Sending packet...");
        packet= new DatagramPacket(buffer, buffer.length, dstAddress);
        socket.send(packet);
        terminal.println("Packet sent");
        this.wait();

}
```

Figure 7: "`start()`" function in CandC and CandC2

```
public void onReceipt(DatagramPacket packet) {
    try {
        StringContent content= new StringContent(packet);

        if(content.toString().equals("Incomplete")) {
            terminal.println("Fail to send. Start again");
            this.start();
        }
        else if(content.toString().equals("Job done")) {
            terminal.println("accept job done");
            //for ACK
            DatagramPacket response = null;
            String replyMessage="accept job done";
            content.swap(replyMessage);
            response=content.toDatagramPacket();
            response.setSocketAddress(packet.getSocketAddress());
            socket.send(response);
        }
        else {
            this.wait();
        }


    }
    catch(Exception e) {e.printStackTrace();}
}
```

Figure 8: "`onReceipt()`" function in CandC and CandC2

### 3.3 Broker

This broker has to receive messages from workers and C&C applications and send messages to both of them. In my program, the core function is "`onReceipt()`" function and almost all of the processes are executed in this function.

There are three statements including if, else if and else statements.

The first if statement and the second else if statement are for receiving a message from worker1 and worker2, sending replies to them and distributing a job if necessary.

```
@Override
public synchronized void onReceipt(DatagramPacket packet) {
    // TODO Auto-generated method stub
    try {

        if(packet.getPort()==DST_PORT_WORKER1) {
            //From worker1
            StringContent content= new StringContent(packet);

            if(content.toString().equals("task done")) {
                if(((fromCandC&&!fromCandC2)||(!fromCandC&&fromCandC2)))distributeWork(content,jobMessage);
            }
            else {
                terminal.println(content.toString());
                packetFromWorker1 = packet;
                availableWorkers.add(packetFromWorker1);

                //for ACK
                DatagramPacket response = null;
                String replyMessage="accept your request";
                content.swap(replyMessage);
                response=content.toDatagramPacket();
                response.setSocketAddress(packetFromWorker1.getSocketAddress());
                socket.send(response);
            }
        }
        else if(packet.getPort()==DST_PORT_WORKER2) {
            //From worker2
            StringContent content= new StringContent(packet);

            if(content.toString().equals("task done")) {
                if(((fromCandC&&!fromCandC2)||(!fromCandC&&fromCandC2)))distributeWork(content,jobMessage);
            }
            else {
                terminal.println(content.toString());
                packetFromWorker2 = packet;
                availableWorkers.add(packetFromWorker2);

                //for ACK
                DatagramPacket response = null;
                String replyMessage="accept your request";
                content.swap(replyMessage);
                response=content.toDatagramPacket();
                response.setSocketAddress(packetFromWorker2.getSocketAddress());
                socket.send(response);
            }
        }
```

Figure 9: the first if and the next else if statement in "`onReceipt()`" function in broker

The else statement is worked when receiving a message from C&C or C&C2. This statement is divided into the case in which there are available workers to do a job and the other case in which there are no available workers. In the first case, the distribution of a job to each worker is executed by the use of "`distributeWork()`" if the job is to print a specific word certain time. In the second case, the broker sends a message to C&C or C&C2 and it mentions the given job cannot be achieved because of the lack of available workers.

```
else {
    //From C&C or C&C2
    if(packetFromWorker1!=null && packetFromWorker2!=null) {    // there are workable workers in the lists


        if(packet.getPort()==DST_PORT_CANDC) {
            fromCandC = true;
            fromCandC2 = false;
        }
        else {
            fromCandC = false;
            fromCandC2 = true;
        }

        keptPacket = packet;
        StringContent content= new StringContent(packet);
        terminal.println(content.toString());

        if(typeOfOrder(content.toString())) {    // if the recieved work description is to print a specific word

            index =0;
            workerIndex =0;
            if(startIndex(content.toString())<0 || endIndex(content.toString())<0) jobMessage=null;
            else jobMessage = (content.toString()).substring(startIndex(content.toString())+1, endIndex(content.toString())-1);
                 // pick up a specific word that should be printed by workers

            times = Integer.parseInt(checkTimes(content.toString()));  //times to print the word
            DatagramPacket response = null;
            content.swap(jobMessage);
            if(index<times) {
                distributeWork(content, jobMessage); // distribute job to each worker
                System.out.println("after distibution index is "+index);
            }
            else {
                jobMessage = "Job done";
                content.swap(jobMessage);
                terminal.println(jobMessage);
                response=content.toDatagramPacket();
                response.setSocketAddress(packet.getSocketAddress());
                socket.send(response);
            }
        }
    }
    else {                                                  // there are no workable workers in the lists
        System.out.println("through incomplete");
        String reply = "Incomplete";
        StringContent content= new StringContent(packet);
        terminal.println(content.toString());
        DatagramPacket response;
        response= (new StringContent(reply)).toDatagramPacket();
        response.setSocketAddress(packet.getSocketAddress());
        socket.send(response);
    }
}
```

Figure 10: the last else statement in "`onReceipt()`" function in broker

There are two important functions apart from above "`onReceipt().`" The first one is "`distributeWork()`" used for giving a job to each worker at a given time. After distributing, this program progresses forward "`jobDone().`" From here, the broker sends a message to inform C&C or C&C2 of the done job.

```
public void distributeWork(StringContent content, String jobMessage) throws Exception{
    DatagramPacket response = null;
    content.swap(jobMessage);
    response=content.toDatagramPacket();
    response.setSocketAddress((availableWorkers.get(workerIndex)).getSocketAddress());
    socket.send(response);
    index++;
    workerIndex++;
    if(workerIndex==availableWorkers.size()) workerIndex =0;

    if(index>=times) {
        System.out.println("before go to jobDone");
        jobDone(content,jobMessage);
    }
}

public void jobDone(StringContent content, String replyMessage) throws Exception{
    DatagramPacket response = null;
    replyMessage = "Job done";
    content.swap(replyMessage);
    terminal.println(replyMessage);
    response=content.toDatagramPacket();
    response.setSocketAddress(keptPacket.getSocketAddress());
    index=0;
    workerIndex=0;
    fromCandC = false;
    fromCandC2 = false;
    socket.send(response);
}
```

Figure 11: `distributeWork()` and `jobDone()` in broker

## 3.4 Packet

In my program, every node such as worker1, worker2, C&C, C&C2 and broker creates each own packet when sending a new message to others. Furthermore, the process of creating a packet is almost the same. It consists of a buffer and the address of the destination. The payload is the core content of a message to be sent to others and the length of the header is 10 bytes in my program. The buffer includes both the payload and the header.

```
DatagramPacket packet= null;

byte[] payload= null;
byte[] header= null;
byte[] buffer= null;

    payload= (terminal.readString("If you are volunteering for work, enter your name: \n")).getBytes();
    header= new byte[PacketContent.HEADERLENGTH];

    buffer= new byte[header.length + payload.length];
    System.arraycopy(header, 0, buffer, 0, header.length);
    System.arraycopy(payload, 0, buffer, header.length, payload.length);

    terminal.println("Sending packet...");
    packet= new DatagramPacket(buffer, buffer.length, dstAddress);
    socket.send(packet);
    terminal.println("Packet sent");
    this.wait();
```

Figure 11: the process of creating a packet (derived from a part of "`start()`" function in worker1)

## 3.5 Demo

From here, the way of running this program is explained.

After running all classes: worker1, worker2, broker, C&C and C&C2, the first step is to input a worker's name at each terminal of workers such as John and Bob.

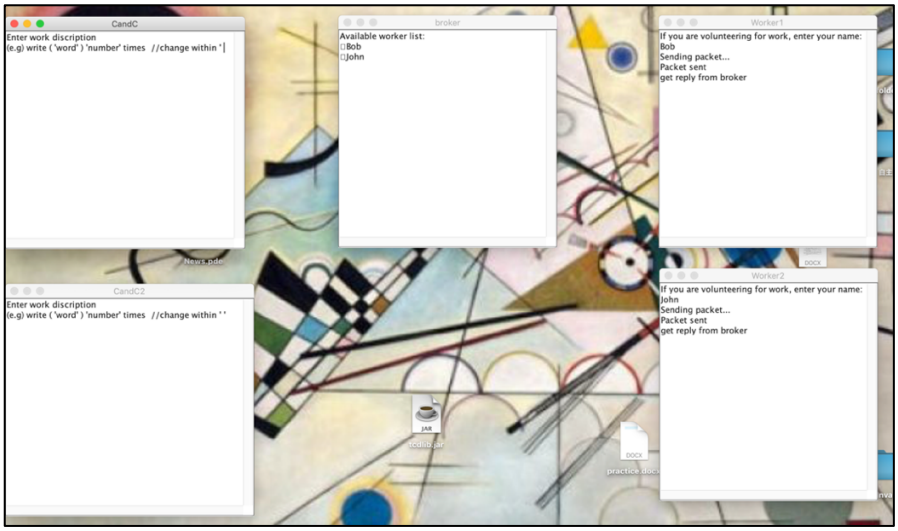


Figure12: putting the names of workers

The next step is just putting a job description at each C&C terminal. For instance, “write (hello) 15 times” is inputted at C&C terminal, the word “hello” is printed 8 times at worker1’s terminal and 7 times at worker2’s terminal. Likewise, when the user inputs “write (world) 7 times” at C&C2 terminal, the word “world” is printed 4 times at worker1’terminal and 3 times at worker2’s terminal. My program can only print a word and cannot print a number.

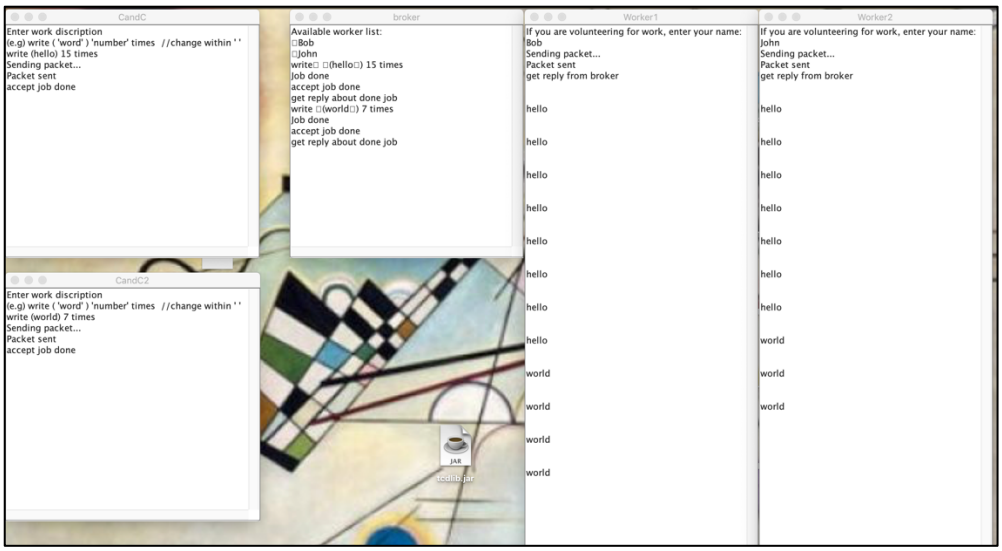


Figure13: the result of inputting work descriptions

## Conclusion

My program can work along with the description of this assignment and helped me to understand the mechanism of communication between nodes by the use of a protocol. For the avoidance of message lost, Stop-and-Wait is implemented this time; however, go-back-n and selective-repeat solutions are not. These two methods are so important as well that to be improved and implemented. Furthermore, docker and wireshark are not used for checking the communication between nodes, therefore, these should be used in assignment 2.