COSC 364

Internet Technologies and Engineering

First Assignment

Kyran Stagg   78943881

Minfang Yu   75219495

The percentage contribution:

Kyran Stagg 50%

Minfang Yu 50%

Which aspects of your overall program (design or implementation) do you consider particularly well done?

In our program we think that the multithreading of our program was implemented in such a way that makes the readability of the program very good and it ensures that inputs and output events don’t cross over. The multi-threading also allows for better scaling with more sockets. The program also makes great use of our own Object classes to allow for quicker, cleaner entry table processing.

Which aspects of your overall program (design or implementation) could be improved?

Since java is an object-based programming language, it can be hard to use low level data structures like byte arrays and java developers are meant to avoid them and create their own objects. When creating and sending response packets, we had to make our objects into byte arrays to send them but could have done it with built in object sending functions. Because of this, our code can be harder to understand when we are translating between byte arrays and our entry table object.

How have you ensured atomicity of event processing?

As mentioned before our program makes great use of multi-threading. This ensures that one thread is meant to handle the sending of routing tables and another for receiving them. On top of this we have more threads for each socket to ensure socket data doesn’t get crossed over, and we make use of built in timer tasks to handle time related events.

Our testing plans

For testing in the early stages of development we made use of three separate configurations for three different routers. This was streamlined even more by the two main IDE's we used. We used VScode and Intellij which have very similar testing and debugging frameworks which allowed us to inspect any variable at almost anytime to ensure our code was producing correct input and outputs. When we needed to test a packet being sent was correct, we would write down the packets expected contents and use the debugging suites to check the program produced the correct input output.

Later when our program grew and could handle more routers, the program moved on to using the given seven router examples. We continued using VScode and intellij's debug suites but also output log files of resulting entry tables after a minute of all routers running for a minute. We would then use Diffchecker.com to ensure that the outputs matched our expected outputs.

If we were to write this program again, or had more time to work, we would have implemented Junit testing. This is a testing library for java that runs code with given input and checks for expected output. Since this was a smaller assignment we decided to stick with manual testing.

Source code

Filenames are given above each class definition, please ensure names are correct before compiling and building a jar file.

Runner.java

```java
import java.io.IOException;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

/**
 * Main entry class for program
 */
public class Runner {
    final static int INFINITY = 30;
    static EntryTable entryTable;
    static ConfigIO routerConfig = null;

    /**
     * method to call and organise router configuration before starting server
     * @param args String[] the arguments provided by the user
     */
    private void processConfig(String[] args) {
        String file = null;
        if (args.length > 0 && (args[0].equals("--file") || args[0].equals("-f"))) {
            file = args[1];
        } else {
            System.out.println("File is needed to configure RIP");
            System.exit(0);
        }

        // get the router config from file
        try {
            routerConfig = new ConfigIO(file);
        } catch (IOException e) {
            e.printStackTrace();
        }
        entryTable = new EntryTable();
        Entry self = new Entry(routerConfig.routerId, routerConfig.routerId, 0,
LocalTime.now());
        entryTable.update(self);
        for (List<String> output : routerConfig.outputs) {
            Entry entry = new Entry(Integer.parseInt(output.get(2)),
Integer.parseInt(output.get(2)), Integer.parseInt(output.get(1)), LocalTime.now());
            entryTable.update(entry);
        }
        System.out.println(entryTable.toString());
    }

    /**
     * main method to start program creates multiple threads for each input socket
     * @param args String[] user provided options for execution
     */
    public static void main(String[] args) {
        Runner runner = new Runner();

        // get the file from the launch arguments
        runner.processConfig(args);

        //set variables for later use
        ArrayList<SocketRunner> sockets = new ArrayList<>();
        ArrayList<Integer> inputPorts = routerConfig.inputPorts;
        SocketRunner socket;

        //create sockets and new threads for each socket
        for (int port : inputPorts) {
            if (port == inputPorts.get(0)) {
                socket = new InputSocketRunner(port);
                System.out.println("Port used for data transmission: " + port);
            } else {
                socket = new SocketRunner(port);
            }
            new Thread(socket).start();
            sockets.add(socket);
        }


    }
}
```

ConfigIO.java

```java
import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Class to parse the config files
 */
public class ConfigIO {

    Integer routerId = null;
    ArrayList<Integer> inputPorts = new ArrayList<>();
    public ArrayList<List<String>> outputs = new ArrayList<>();

    /**
     * method to get and check all the config files attributes. saves them to lists in
the runner class
     * @param file String the file to access
     * @throws IOException exception to handle file opening errors
     */
    ConfigIO(String file) throws IOException {
        // Initialize variables
        ArrayList<String> listOutputs = new ArrayList<>();
        // Open the file
        File f = new File(file);
        BufferedReader buffer = new BufferedReader(new FileReader(f));
        String readLine;
        try {
            // Iterate over every line
            while ((readLine = buffer.readLine()) != null) {
                // When router-id is found, save it
                if (readLine.equals("router-id:")) {
                    if ((readLine = buffer.readLine()) != null) {
                        routerId = Integer.parseInt(readLine);
                        if (!(routerId >= 1 && routerId <= 64000)){
                            throw new RuntimeException("Router id must be between 1 and
64000 inclusive id: "+routerId);
                        }
                    }
                }

                // When input-ports are found, save them
                assert readLine != null;
                if (readLine.equals("input-ports:")) {
                    while ((readLine = buffer.readLine()) != null &&
!readLine.equals("")) {
                        if (!(Integer.parseInt(readLine) >= 1024 &&
Integer.parseInt(readLine) <= 64000)){
                            throw new RuntimeException("Input ports must be between 1024
and 64000 inclusive port: " + Integer.parseInt(readLine));
                        }
                        inputPorts.add(Integer.parseInt(readLine));
                    }
                }

                // When output information is found, save it
                assert readLine != null;
                if (readLine.equals("outputs:")) {
                    while ((readLine = buffer.readLine()) != null &&
!readLine.equals("")) {
                        listOutputs.add(readLine);
                    }
                }
            }

            //split the outputs lines up into a list
            for (String str : listOutputs) {
                outputs.add(Arrays.asList(str.split("-")));
            }

            //boundary checking
            for (List<String> output: outputs){
                for (Integer i: inputPorts) {
                    if (Integer.parseInt(output.get(0)) == i){
                        throw new RuntimeException("Input and Output ports must be
different port: " + i);
                    }
                }
            }


        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                buffer.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Entry.java

```
import java.time.LocalTime;

import static java.time.temporal.ChronoUnit.SECONDS;

import java.io.Serializable;

/**
 * class to hold a single entry about a router in the network
 */
public class Entry implements Serializable {
    static final long serialVersionUID = 42L;
    private int dest;
    private int next_hop;
    private int metric;
    private LocalTime time;

    /**
     * constructor to create a new entry for a router
     * @param dest int the destination router
     * @param next_hop int the next hop to get to the destination
     * @param metric int the cost to get to the destination
     * @param t LocalTime the time the link was last reported
     */
    Entry(int dest, int next_hop, int metric, LocalTime t) {
        super();
        this.dest = dest;
        this.next_hop = next_hop;
        this.metric = metric;
        this.time = t;
    }

    /**
     * method to stringify the entry
     * @return String a string representing the entry
     */
```

```java
    public String toString() {

        String result = "";
        result += "dest: " + String.valueOf(this.dest) + " ";
        result += "next hop: " + String.valueOf(this.next_hop) + " ";
        result += "metric: " + String.valueOf(this.metric) + " ";
        result += "time: " + this.time.toString() + " ";

        return result;
    }


    /**
     * method to check how long it has been since the last update of the entry
     * @return long the amount of seconds since the entry was reported
     */
    long timer() {
        LocalTime now = LocalTime.now();
        return SECONDS.between(this.time, now);
    }

    void setTime(LocalTime time) {
        this.time = time;
    }

    int getDest() {
        return dest;
    }

    int getNextHop() {
        return next_hop;
    }

    int getMetric() {
        return metric;
    }

    private LocalTime getTime() {
        return time;
    }

    void setNextHop(int next_hop) {
        this.next_hop = next_hop;
    }

    void setMetric(int metric) {
        this.metric = metric;
    }

    /**
     * method to duplicate the entry
     * @return Entry a new clone of the entry
     */
    Entry duplicateEntry() {
        return new Entry(getDest(), getNextHop(), getMetric(), getTime());
    }
}
```

EntryTable.java

```java
import java.io.Serializable;
import java.util.*;

/**
 * class to hold all entries of routers in the network
 */
public class EntryTable implements Serializable {
    static final long serialVersionUID = 42L;
    private Map<Integer, Entry> entries;
    private Integer sourceRouter;


    /**
     * constructor to create a new entry table to hold all the link entries
     */
    EntryTable() {
        super();
        this.entries = new HashMap<>();
        this.sourceRouter = Runner.routerConfig.routerId;
    }

    Integer getSourceRouter() {
        return this.sourceRouter;
    }

    /**
     * method to stringify the entry table
     * @return String a string representing the entry table
     */
    private String tostr() {
        StringBuilder result = new StringBuilder();
        for (Entry entry : this.entries.values()) {
            result.append(entry.toString()).append("\n");
        }

        return result.toString();
    }

    public String toString() {
        return this.tostr();
    }

    /**
     * method to return all destinations in the entry table
     * @return List list of destinations
     */
    private List<Integer> destinations() {
        List<Integer> dest = new ArrayList<>(this.entries.keySet());
        Collections.sort(dest);
        return dest;
    }

    private Entry getEntry(int dest) {
        return this.entries.get(dest);
    }

    /**
     * method to get a list of all entries
     * @return List list of all entries
     */
    List<Entry> getEntries() {
        List<Entry> result = new ArrayList<Entry>();
        for (int dest : this.destinations()) {
            result.add(this.entries.get(dest));
        }
        return result;
    }

    /**
```

```java
     * method to add a new entry to the entry table
     * @param entry the new entry to add to the entry table
     */
    void update(Entry entry) {
        Entry current = this.getEntry(entry.getDest());

        if (current == null) {
            if (entry.getMetric() < Runner.INFINITY) {
                this.entries.put(entry.getDest(), entry);
            }
        } else if (current.getNextHop() == entry.getNextHop()) {
            if (entry.getMetric() > Runner.INFINITY) {
                entry.setMetric(Runner.INFINITY);
            }
            this.entries.put(entry.getDest(), entry);
        } else if (entry.getMetric() < current.getMetric()) {
            this.entries.put(entry.getDest(), entry);
        }

    }

    /**
     * method to remove an entry by destination
     * @param dest int the destination to remove from the table
     */
    void removeEntry(int dest) {
        if (!(this.entries.get(dest) == null)) {
            this.entries.remove(dest);
        }
    }

    /**
     * method to duplicate the entry table
     * @return EntryTable a new clone of the entry table
     */
    EntryTable duplicateTable() {
        EntryTable clone = new EntryTable();
        for (Entry entry : getEntries()) {
            clone.update(entry.duplicateEntry());
        }
        return clone;
    }
}
```

SendEntryTable.java

```java
import java.util.TimerTask;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.time.LocalTime;
import java.util.List;

/**
 * class to send the entry table to other directly connected routers
 */
public class SendEntryTable extends TimerTask {

    /**
     * method to create a packet containing the current entry table
     * @param output the current output socket details
     * @return DatagramPacket a packet to be sent to a specific output socket
     */
    private DatagramPacket createPacket(List<String> output) {
        try {
            //remove all garbage entries
            for (Entry entry : Runner.entryTable.getEntries()) {
                if ((entry.timer() > 60) && (entry.getDest() !=
Runner.routerConfig.routerId)) {
                    Runner.entryTable.removeEntry(entry.getDest());
                }
            }
            //duplicate the table so infinity can be inserted without effecting the real
table
            EntryTable tableToSend = Runner.entryTable.duplicateTable();
            for (Entry entry : tableToSend.getEntries()) {
                entry.setMetric(entry.getMetric() + Integer.parseInt(output.get(1)));
                if (entry.getNextHop() == Integer.parseInt(output.get(2))) {
                    entry.setMetric(Runner.INFINITY);
                }
            }

            //create the packet from the entry table object
            ByteArrayOutputStream arrayOutputStream = new ByteArrayOutputStream();
            ObjectOutputStream objectOutputStream = new
ObjectOutputStream(arrayOutputStream);
            objectOutputStream.writeObject(tableToSend);
            objectOutputStream.flush();
            byte[] buffer = arrayOutputStream.toByteArray();

            return new DatagramPacket(buffer, buffer.length, InetAddress.getLocalHost(),
                    Integer.parseInt(output.get(0)));
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * method to make this class runnable in its own thread.
     * when the thread is started the class will send packets to all output ports
     */
    public void run() {
        for (List<String> output : Runner.routerConfig.outputs) {
            DatagramPacket packet = createPacket(output);
            try {
                System.out.println("RESPONSE\tRouter id:" + output.get(2) + "\t" +
LocalTime.now());
                DatagramSocket outputSocket = new DatagramSocket();
                assert packet != null;
                outputSocket.send(packet);
                outputSocket.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

InputSocketRunner.java

```java
import java.io.ByteArrayInputStream;
import java.io.IOException;

import java.io.ObjectInputStream;
import java.net.DatagramPacket;
import java.util.Random;
import java.util.Timer;

/**
 * class to handle the single socket that sends out response packets
 */
public class InputSocketRunner extends SocketRunner {

    /**
     * constructor to create a socketRunner
     * @param port int the port to bind a socket to
     */
    InputSocketRunner(int port) {
        super(port);
    }


    /**
     * runnable method to use its own thread.
     * this method is used to block its socket to listen for incoming data
     * and also to send data when a timer runs out
     */
    public void run() {
        synchronized (this) {
            this.runningThread = Thread.currentThread();
        }
        super.openServerSocket();

        //create the timed task (with some randomness to avoid missing a transmission)
        Random rand = new Random();
        Timer UpdateTimer = new Timer();
        SendEntryTable sendEntryTable = new SendEntryTable();
        UpdateTimer.schedule(sendEntryTable, 10000 + rand.nextInt(20000), 10000 +
rand.nextInt(20000));

        while (!isStopped()) {
            byte[] buffer = new byte[1000000];
            DatagramPacket packetReceived = new DatagramPacket(buffer, buffer.length);
            try {
                //blocking the socket to get data
                this.serverSocket.receive(packetReceived);
                try {
                    //convert recived data to entry table object and pass onto worker
thread
                    ByteArrayInputStream byteArray = new ByteArrayInputStream(buffer);
                    ObjectInputStream inputStream = new ObjectInputStream(byteArray);
                    try {
                        EntryTable table = (EntryTable) inputStream.readObject();
                        new Thread(new SocketWorker(table, serverSocket)).start();
                    } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            } catch (IOException e) {
                if (isStopped()) {
                    System.out.println("Server Stopped.");
                    return;
                }
                throw new RuntimeException("Error accepting client connection", e);
            }
        }
        System.out.println("Server Stopped.");
    }
}
```

SocketRunner.java

```java
import java.io.ByteArrayInputStream;
import java.io.IOException;

import java.io.ObjectInputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

/**
 * class to handle sockets that receive data only
 */
public class SocketRunner implements Runnable {

    private int serverPort;
    DatagramSocket serverSocket = null;
    private boolean isStopped = false;
    Thread runningThread = null;

    /**
     * constructor to create a socketRunner
     * @param port int the port to bind a socket to
     */
    SocketRunner(int port) {
        this.serverPort = port;
    }

    /**
     * runnable method to use its own thread.
     * this method is used to block its socket to listen for incoming data
     */
    public void run() {
        synchronized (this) {
            this.runningThread = Thread.currentThread();
        }
        openServerSocket();
        while (!isStopped()) {
            byte[] buffer = new byte[1000000];
            DatagramPacket packetReceived = new DatagramPacket(buffer, buffer.length);

            try {
                //blocking the socket to get data
                this.serverSocket.receive(packetReceived);
                try {
                    //convert recived data to entry table object and pass onto worker
thread
                    ByteArrayInputStream byteArray = new ByteArrayInputStream(buffer);

                    ObjectInputStream inputStream = new ObjectInputStream(byteArray);
                    try {
                        EntryTable table = (EntryTable) inputStream.readObject();
                        new Thread(new SocketWorker(table, serverSocket)).start();
                    } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            } catch (IOException e) {
                if (isStopped()) {
                    System.out.println("Server Stopped.");
                    return;
                }
                throw new RuntimeException("Error accepting client connection", e);
            }
        }
        System.out.println("Server Stopped.");
    }

    synchronized boolean isStopped() {
        return this.isStopped;
    }

    /**
     * method to open the socket for this thread to handle
     */
    void openServerSocket() {
        try {
            this.serverSocket = new DatagramSocket(this.serverPort);
        } catch (java.net.BindException e) {
            throw new RuntimeException("Input ports must be unique and not already bound
port: " + serverPort, e);
        } catch (IOException e) {
            throw new RuntimeException("Cannot open port " + serverPort, e);
        }
    }

}
```

SocketWorker.java

```
import java.net.DatagramSocket;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

/**
 * class to handle deconstructing an entry table to be merged with this routers one
 */
public class SocketWorker implements Runnable {

    private EntryTable newTable;
    private DatagramSocket socket;

    /**
     * constructor method to get the information needed to work
     * @param newTable EntryTable the table that we received
     * @param socket DatagramSocket the socket that received the data
     */
    SocketWorker(EntryTable newTable, DatagramSocket socket) {
        this.newTable = newTable;
        this.socket = socket;
    }

    /**
     * runnable method to interperate the incoming entry table
     */
    public void run() {
        LocalTime now = LocalTime.now();
        System.out.println("RECIEVED \tPort: " + socket.getLocalPort() + "\t" +
now.toString());

        Integer nextHop = newTable.getSourceRouter();

        ArrayList<Entry> entriesToAdd = new ArrayList<>();
        for (Entry newEntry : newTable.getEntries()) {
            boolean flag = false;
            for (Entry currentEntry : Runner.entryTable.getEntries()) {
                if (newEntry.getDest() == currentEntry.getDest()) { // we have an
existing record to update!
                    flag = true;
                    if (newEntry.getMetric() < currentEntry.getMetric() &&
currentEntry.getMetric() != Runner.INFINITY) { // the new entry has a smaller metric then
our one
                        Runner.entryTable.removeEntry(currentEntry.getDest());
                        newEntry.setNextHop(nextHop);
                        newEntry.setTime(LocalTime.now());
                        entriesToAdd.add(newEntry);
                    } else if (currentEntry.getNextHop() == nextHop) { //the next hops
and destination are the same
                        currentEntry.setMetric(newEntry.getMetric());
                    }
                }
                if (nextHop == currentEntry.getDest() && currentEntry.getMetric() ==
Runner.INFINITY) { //router has come back from the dead, re add its data
                    currentEntry.setTime(LocalTime.now());
                    for (List<String> output : Runner.routerConfig.outputs) {
                        if (socket.getLocalPort() == Integer.parseInt(output.get(0))) {
                            currentEntry.setMetric(Integer.parseInt(output.get(1)));
                        }
                    }
                }
            }
            if (!flag && newEntry.getMetric() < Runner.INFINITY) { // this is a new entry
to add to our table
                newEntry.setNextHop(nextHop);
                newEntry.setTime(LocalTime.now());
                entriesToAdd.add(newEntry);
            }
        }
        for (Entry entryToAdd : entriesToAdd) { // add the new entries
            Runner.entryTable.update(entryToAdd);
        }

        System.out.println(Runner.entryTable.toString()); //output for user to see

    }
}
```

Configuration Files

Given filenames are recommend to be used, but must be of .cfg file format.

router1.cfg

```
router-id:
1

input-ports:
1111
1112
1113

outputs:
2221-1-2
6661-5-6
7771-8-7
```

router2.cfg

```
router-id:
2

input-ports:
2221
2222

outputs:
1111-1-1
3331-3-3
```

router3.cfg

```
router-id:
3

input-ports:
3331
3332

outputs:
2222-3-2
4441-4-4
```

router4.cfg

```
router-id:
4

input-ports:
4441
4442
4443

outputs:
3332-4-3
5551-2-5
7772-6-7
```

router5.cfg

```
router-id:
5

input-ports:
5551
5552

outputs:
4442-2-4
6662-1-6
```

router6.cfg

```
router-id:
6

input-ports:
6661
6662

outputs:
1112-5-1
5552-1-5
```

router7.cfg

```
router-id:
7

input-ports:
7771
7772

outputs:
1113-8-1
4443-6-4
```

Running the program

Java requires a jar to execute a program. To obtain this execute the follow terminal commands: Ensure all .java source files are in the same directory and are the only files in the directory, these commands assume that the directory only has the source code in it. Also create a folder in said directory called ‘build’. This will be used to store the compiled class files.

```
javac -d ./build *.java
```

This command is to compile the .java files into .class files and place them in the build folder

```
cd ./build
```

This command is to move to the build folder

```
jar cvf rip.jar *
```

This command builds all the files in build into a jar file called rip.jar

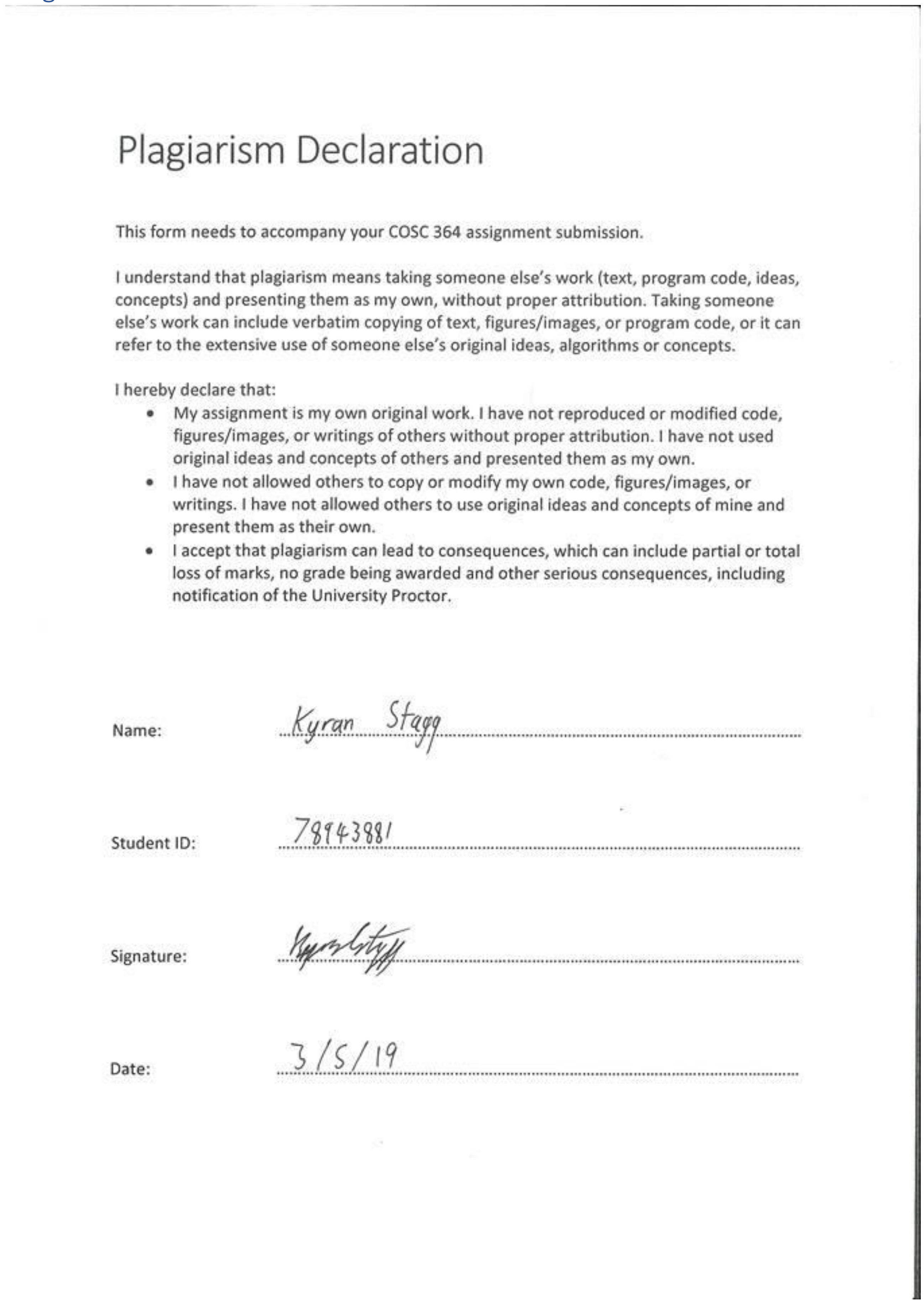
To start the program run the command:

```
Java -cp rip.jar Runner -f <config file>
```

Where <config file> is a given router.cfg file.

Creating a script to run all instances of the config files is recommend

Plagiarism declarations



Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name:	Kyran Stagg
Student ID:	78943881
Signature:	[illegible]
Date:	3/5/19

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name:	Mintang Yu
Student ID:	75219495
Signature:	Mintang Yu
Date:	3/05/2019