2020

SOFTENG 364: Assignment 2

Craig Sutherland

Contents

Important Dates

Submission due date: Wednesday, 10 June, 2020 – 5pm on Canvas

Note: *this is week 12 of the semester, so we cannot give any extensions.*

Learning Objectives

At the end of this assignment you should be able to:

- Write a program in Java to communicate via pings and acknowledgements,
- Implement the Distance Vector algorithm in Java,
- Discuss the security issues in an unsecured network.

Welcome to Drone Swarm Rescue System

You have been hired as a new developer on the Drone Swarm Rescue System (DSRS) project. This project aims to build a system that allows multiple drones to explore a site as part of an on-going rescue operation. The concept is very simple: use multiple, semi-autonomous drones to map an area and detect any items of interest (e.g. people to rescue, hazards, etc.)

As there are no guarantees about the environment the drones will be deployed in, the project team has decided to implement their own network on the drones. The drones will use internal Wi-Fi routers to communicate with each other. In order to ensure communications with the operators, certain drones will be designated as relays. Messages from drones on the edges of the swarm will be relayed to the nearest relay drone, which will in turn relay the message onto other relay drones until the messages get to the operators, and vice-versa.

Your role on the team is to handle the message routing between the drones. The system architect has proposed building a protocol based on the Distance Vector algorithm, as there are no guarantees about the connections between the drones. The system architect has given you the following spec for implementing the DSRS network:

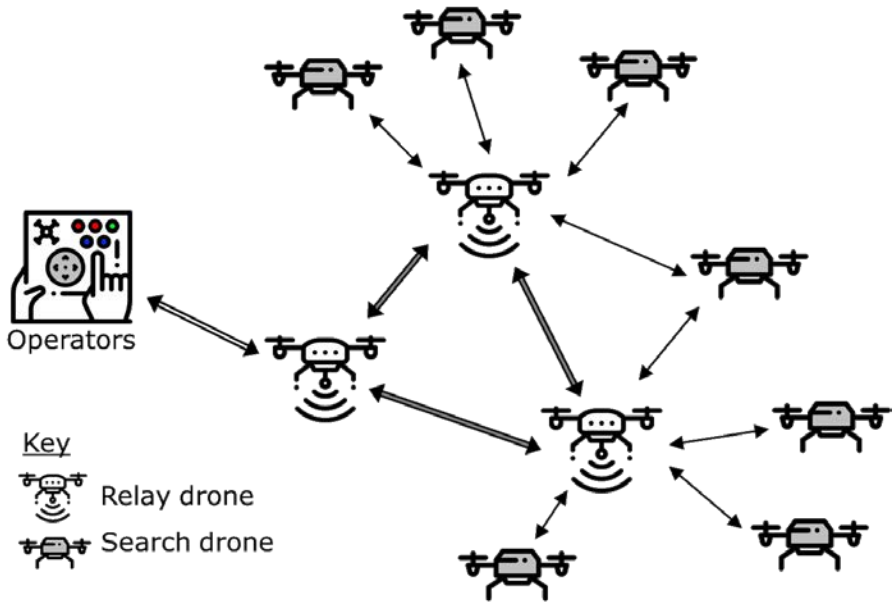


Figure 1: Diagram of drone communications.

As indicated in Figure 1, the relay drones form the heart of the DSRS network. Communications are initiated by the client, with each drone and operator in the system has a list of relay drones. When a drone (or operator) starts, it will attempt to connect to each relay drone in the system. To facilitate the initial connection, each drones has the IP addresses of all the possible relay drones[1]. On start-up, the drone will iterate through the list and send a connection request to each relay drone. If the connection is successful, the relay drone will send an acknowledgement, otherwise the request will timeout. On a regular basis, the drone will resend the connection request to all relay drones to confirm the connection is still active. This connection process is performed by all drones, both relay and search.

[1] Drones are only possible connections as there is no guarantee that a relay drone will be online and within range.

The relay drones will keep a list of all connected drones (and operators, but we will just assume they are a type of drone from now on.) When a drone connects, the relay drone will send an acknowledgement back to the drone and then store the IP address of the drone in a list. The relay drone will then send a "ping" request to the drone and measure the time it takes to receive a response from the drone. The relay will also send out pings to all connected drones on a regular basis. If a drone fails to respond after a set time, the drone is removed from the list of connected drones.

The relay drones will use this list, with the ping response times, to build a forwarding table to any other drone in the network. Whenever an updated ping response time is received, the relay drone will check if the response time has changed. If it has, the relay drone will perform an update using the Bellman-Ford algorithm and update the Distance Vector to drone. It will then send out the updated Distance Vectors to all the connected relay drones. The same process also happens when a Distance Vector update is received from any other drone.

Sound simple? Good – your job is to implement this system.

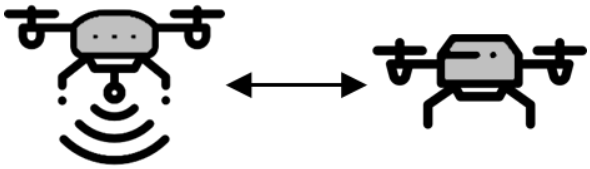
Assignment Tasks

This assignment has two parts to it. You need to implement the following parts of the spec:

- Pinging a client and recording the response time
- Calculating the forwarding table

Each part is worth half of the total mark for the assignment.

Task 1: Pinging Clients



Using your knowledge of socket programming (see Lab 7), you will need to implement the relay drone to client ping.

The process has the following steps:

1. Read in a list of clients
2. Iterate through the list and ping each client, for each client record the time to receive a response (in seconds)
3. Write out the list of clients who responded with the ping response time

The process must be implemented in Java. The entry point for your program must be a class called DSRSNetwork.java.

File Format

The list of client is a plain text file, containing the entries in a CSV format. Each line contains the name and type of the client, the client's IP address and the last response time. If the client has never been pinged or the last ping did not connect, the last response time will be -1. The following is an example of the file content[2]:

```
Search1,Search,127.0.0.1:10124,-1
Search2,Search,127.0.0.1:10125,-1
Operator1,Operator,127.0.0.1:10126,0
Relay2,Relay,127.0.0.1:10128,6
```

The data uses the following rules:

- Name: alphanumeric characters only (A-Z, a-z and 0-9), case sensitive,
- Type: one of "Operator", "Relay" and "Search",
- IP address: a valid IP address and port,
- Last response time: a non-negative integer value or -1.

The input and output formats are the same. The filename is `clients-`*`dronename`*`.csv` where *`dronename`* is the name of the drone. Assuming Relay1 is the drone opening the file, it will be called `clients-Relay1.csv`.

Note: if the drone loads an entry with the same name as itself, it should ignore that entry.

Ping Message Format

As the main focus of this assignment is sending and receiving messages, the format is very simple. To ping a neighbour, the drone sends the text "PING" and a newline character (\n). When the neighbour receives a ping, it should reply with the text "ACK" and a newline character (\n).

The ACK response is a common pattern in network programming to confirm a message has been received[3]. As you can guess, it is short for ACKnowledgement. In this assignment it is also needed to time the time to ping the neighbour and receive the response.

Application Output

To help with testing (and marking), when the relay drone pings all the clients, it should display a message to the console saying when each step starts and finishes, as well as when each client is being pinged. Finally, the app should also display a message saying the process has started and finished. For example:

---

[2] For this assignment, all IP addresses will use the local loopback address (127.0.0.1) and a port.

[3] There is also a NAK response, which is a negative acknowledgement. I'll leave it to you to figure out where a NAK might be useful…

```
Starting ping process #1
Reading client list: starting
Reading client list: finished – 3 clients read
Pinging all clients: starting
- Pinging Search1...ping received after 1s
- Pinging Search2...could not ping
- Pinging Operator1...ping received after 0s
Pinging all clients: finished – 3 clients pinged
Writing client list: started
Writing client list: finished – 3 clients written
Ping process #1 finished
```

Assumptions

For this assignment you can assume all input data is valid, i.e. you do not need to validate any data from the command line or input CSV file[4].

The relay drone application does not need to reply to pings from other drones.

Assume Relay1 is the name of the drone.

Testing

To assist with this assignment, we have written a small testing application called testbed. This application will listen on a port and reply to any messages with an acknowledge ("ACK\n").

The application is distributed as a JAR file. To run the application, use the following command:

```
java –jar testbed.jar -port=port
```

Where *`port`* is the port to listen on (e.g. 10124.)

It is also possible to call the testbed application with a `-delay` argument, which specifies the delay in seconds. For example:

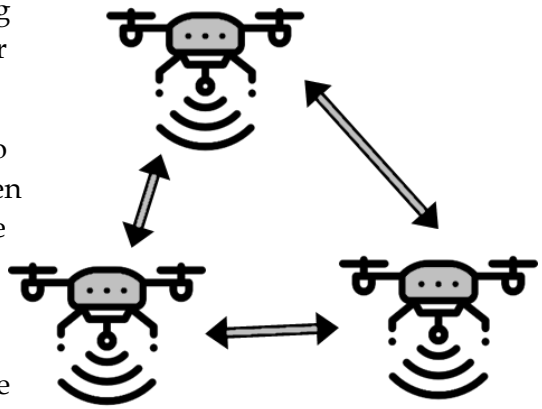
```
java –jar testbed.jar -port=10124 -delay=3
```

This example will start the testbed application listening on port 10124 and will delay for three seconds before responding.

An example `clients-relay1.csv` file has also been provided which includes several drones.

[4] Do **NOT** assume the data is valid in a real world application. Always, always, always! validate any input data.

Task 2: Calculating Routes



In Lab 6, you implemented the Link-State algorithm for calculating routes. In this assignment, you will implement the Distance Vector algorithm for calculating routes.

As Distance Vector is an asynchronous algorithm, you will need to first implement the communications between relay drones and then implement the actual routing calculations. Remember the Distance Vector algorithm waits for incoming Distance Vector updates before updating its own Distance Vector table. You will need to modify the client code from the previous task to receive incoming Distance Vector updates as well as ping requests. If you need more details on the Distance Vector algorithm, it was covered in video 02 of the Network Layer: Control Plane module.

You can use the same input file format as Task 1, along with the same command line arguments.

The relay application should function as follows:

1. On start-up, run the ping client process to retrieve the initial costs for all connected clients
2. Open a socket to listen for incoming messages on port 10120 and wait
3. When the application receives a distance vector update, it should calculate the new costs to all nodes and store them in a forwarding table.

Once the program is started, it should handle multiple distance vectors updates. Each update should update the forwarding table and send out Distance Vector Updates as required.

Application Output

The application should display the same console output as task 1 for the ping process. It should also add the following output for calculating new costs:

```
New DVs received
Starting DV update calculation
- Calculating cost for Search1...cost updated to 1 via Relay1
- Calculating cost for Search2...cost updated to 1 via Relay1
- Calculating cost for Operator1...no change
Sending updated DVs
- Sending to Relay2...done
DV update calculation finished
```

Alternatively, if there are no updated Distance Vectors, it should display the following output:

```
New DVs received
Starting DV update calculation
- Calculating cost for Search1...no change
- Calculating cost for Search2...no change
- Calculating cost for Operator1...no change
Skipping DV update send
DV update calculation finished
```

Distance Vector Update Message Format

With the first task, the message format was not important (it just mattered that the client responded.) For this task, the message format is very important! Both the sending and the receiving relay drones need the same message format. To simplify the task, the system architect has given you the following message format.

The message format consists of three parts:

- Header
- Data
- Footer

The header consist of a single word, "UPDATE", followed by a colon (:) then the name of the originating drone and a final colon.

The data is a set of comma separated values. Each value consists of the name of the destination drone, followed by an equal sign (=) then the cost to the drone. Only include the DVs that have been changed.

The footer consists of a colon, followed by a number stating the number of updates transmitted and finally a newline character (\n).

Here is an example:

```
UPDATE:Relay1:Search1=1,Search2=1:2\n
```

Thus, when your application receives a message, it must first check for the word "UPDATE". You can then split the message into four parts using a colon. The first part can be ignored, the second part is the source name, the third part is the data and the fourth part is the number of data values received[5].

When a drone receives a Distance Vector Update message, it should respond with an acknowledgement ("ACK\n") in the same way as a ping.

Forwarding Table

After every Distance Vector update operation, the application must write out a forwarding table to the current directory. The table will have one row per destination that can be reached, and only rows for the destinations that can be reached. If there is a client in the clients table that cannot be reached, then the row for this client must be omitted.

Each row should have two values, separated by commas. The first value is the name of the destination node (e.g. Search1, Operator1, etc.) The second value is the name of the node to forward the messages onto. If the destination node is a neighbour of the current node, then the destination node will be the second value.

The following is an example of a forwarding table:

```
Search1,Search1
Search2,Relay2
Operator1,Operator1
Relay2,Relay2
```

The filename is `forwarding-`*`dronename`*`.csv` where *`dronename`* is the name of the drone. Assuming Relay1 is the drone opening the file, it will be called `forwarding-Relay1.csv`.

Testing

The testbed program can also be used for testing the second part of the assignment. In addition to the arguments for starting the ping server, you can change it to send mode by setting an action argument[6]:

[5] Normally you would use the fourth part to check you have received the correct number of entries. This validation check can be ignored for this assignment.

[6] There is also a sendping action, I added it to test the testbed. Feel free to use it for testing the testbed on your own machine

```
java –jar testbed.jar -port=port -action=senddvs
```

In send mode, the port is the destination port (i.e. the port on which a client is listening.)

This mode has the following optional arguments:

- -name: the name of the sending drone, defaults to "Relay1" if this argument is omitted.
- -dv: a Distance Vector to include in the message. This argument can be repeated multiple times.

The following is an example of calling the testbed:

```
java –jar testbed.jar -port=10121 -action=senddvs -dv=S1,1 -dv=S2,5
```

Which will generate the following message to a client on port 10121:

```
UPDATE:Relay1:S1=1,S2=5:2\n
```

Task 3: Reflection on Security

The system you have currently implemented is an open, unsecured system. As you will know from the module on security, this is very **BAD**[7]! However, implementing a secure system to lock a hacker out of the DSRS is beyond the scope of this course.

Instead, your task is to reflect on the current security of the system and write a short (two or three paragraphs, one page maximum) report on how you could secure the DSRS network. Think about where the open points in the network are, how a hacker might mess up the system and what could be done to secure each point.

[7] Notice the capitals – it really is very, very, very bad to have an open system. You don't want to know what a bad guy could do with our nice expensive Drone Swarm Rescue System!

Marking

We won't be spending long testing your system, so if you want the marks, make it easy for us.

Really, I mean it!

Our testing process will be really simple. We will do the following steps:

1. Extract your files to an empty folder
2. Compile your code
3. Copy a new `clients-Relay1.csv` file to folder
4. Start some testbed instances to respond to your pings
5. Run your program
6. Check the updated `clients-Relay1.csv` file
7. Send two or three DV updates to your program
8. Check there is `forwarding-Relay1.csv` file and it contains the correct output
9. Check the console output matches the expected output

And that's it (well, yes, I will read your report as well.)

Given we already have the testbed application, we will run the following commands:

```
javac *.java
java DSRSNetwork.class
```

Then check that everything is working fine. Make sure these commands work, then we will be happy and give you more marks.

Criteria (Total 8% of course):

Criteria	Ratings
Task 1: pings remote client **[3 marks]**	3 marks: sends ping, waits for reply and records correct duration 2 marks: sends ping and waits for reply 1 mark: sends ping 0 marks: multiple errors
Task 1: reads and writes client file **[2 marks]**	2 marks: correctly reads and writes client list 1 mark: either correctly reads or correctly writes client 0 marks: neither reads nor writes
Task 1: application output **[3 marks]**	3 marks: all messages are correctly displayed 2 marks: 1 or 2 messages missing or there are some errors in the display 1 mark: some output is displayed 0 marks: no output is displayed
Task 2: application output **[3 marks]**	3 marks: all messages are correctly displayed 2 marks: 1 or 2 messages missing or there are some errors in the display 1 mark: some output is displayed 0 marks: no output is displayed
Task 2: calculates DV routes **[4 marks]**	4 marks: correctly calculates routes after multiple DVs received 3 marks: correctly calculates routes after a single DV is received 2 marks: some routes have incorrect values 1 mark: correctly parses incoming DV update 0 marks: cannot correctly process DV updates at all

Criteria	Ratings
Task 2: writes forwarding table **[2 marks]**	2 marks: correctly writes entire forwarding table 1 mark: some errors in forwarding table 0 marks: forwarding table is not generated or empty
Task 3: security report **[3 marks]**	1 mark per valid security point identified 0 marks: no valid points or report omitted
Penalties:	-2 marks: your code does not compile -1 mark: your code crashes
Total:	**24 Marks**

Files to submit

For this assignment, you must submit a single zip file to Canvas containing all your code and the security report. Do not include any .class or .jar files. The report should be a PDF file called Security_Report.pdf, with your name and UPI at the top of the report, followed by the 2-3 paragraphs.

The zip file should be named `Assignment_2_`*`UPI`*`.zip`, where *`UPI`* is your own UPI. For example, `Assignment_2_JDOE123.zip`.

You do not need to include any data files, the marker will use their own set of list files for the drone.

Credits

Top image on page 2 is modified from an image by Alexander Lesnitsky from Pixabay.