THE ONE - Case1: Capturing Average Connecting Time among Two Nodes

Xu Liu *and Yuanzhu Chen

Department of Computer Science
Memorial University of Newfoundland

May 11, 2013

1 Introduction

In the previous report, I have gave a detailed introduction of THE ONE (related to how to install, how to configure and how to debug). In this report, I plan to show a small case which is to calculate the average connecting time between two nodes in a given scenario by THE ONE.

1.1 Outline

this report is consisted of three sections

- the first section is about how to set up a simple scenario in THE ONE
- the second one is related to improve THE ONE code for our personal experiment purpose
- the third part shows how to get some basic value for analysis from our experiment data

2 Scenario Setting in THE ONE

2.1 Definition of Scenario

In this report, I plan to create a scenario for DTN test. In this scenario, I suppose that there are 10 persons in same region and they have some abilities to communicate with each other by Bluetooth. Moreover, I assume that the transmission rate of Bluetooth is 90 Kb/S (90 Kb is the maximum transmission rate of Bluetooth on iOS device [iOS 6.0]) , and the transmission range of the Bluetooth is 50 meters in non-barrier space (although I test that Bluetooth of iOS device can communicate with each other in 100 meters distance). As for the map of this scenario, I plan to use the default map of THE ONE on the street. Also, in terms of the moving model of person, I will use the default moving model from THE ONE in this scenario.

2.2 Convert Scenario to Configuration File

After we finish defining the scenario that we want to use for our test. Then, we need to create a script (configuration file) to let THE ONE know what the scenario is:

*liuxu.mun@gmail.com

- Firstly, we need to create a new file. This file is a pure text file. It will record all the information that used to create our scenario. In this report, let's call the file "simple_scene.one" (select our project "the_one_1.4.1", then right click the project -> click "New"->"File", Typing "simple_scene.one" in the given text box, Then click "Finish", "simple_cene.one" file will automatically open) as Figure 1.

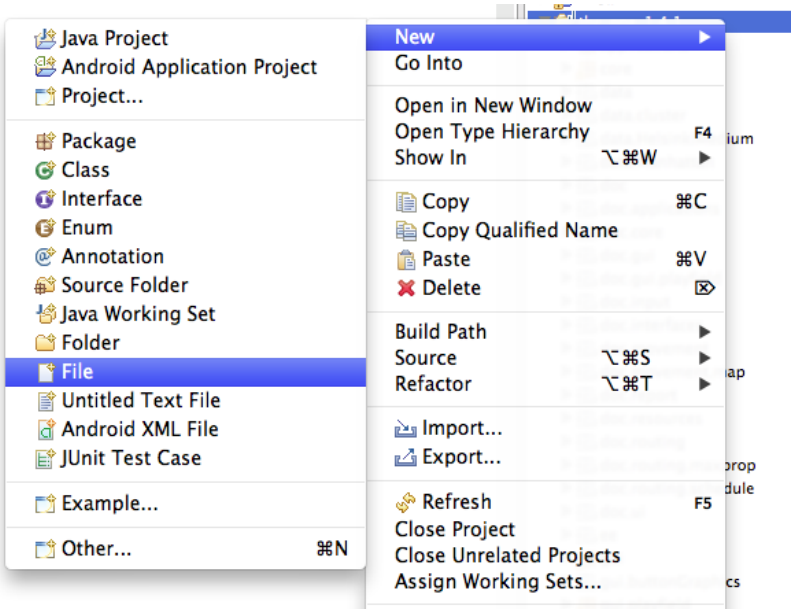

Figure 1: Create configuration file for our scenario

- Secondly, we need to write our scenario information to this configuration file by using THE ONE's configuration command. From officially setting files, we know that , the basic information we have to write is for scenario setting. The configuration command is as show in Figure 2. In Figure 2, we have to pay more attention to the syntax of THE ONE's command. "#" is the sign to start a comment. After "#", it is better to leave a space, then type our comment. Sometimes, I try to comment some lines without space behind "#". THE ONE simulator will give some warning or errors during our simulation. Moreover, we can see that the basic syntax for assign a value to THE ONE is that the left side of "=" is the key words for THE ONE to decide which functions or features, it should enable during the running time (There are many commands, in Figure 2, I just list the commands that useful for our simple_scene scenario). the right side of "=" is the value that we want to give to THE ONE. Based on the right side value, THE ONE can know what kind of scenario we want to get.

```
########### Scenario settings
# A name to identify the present scenario
Scenario.name = simple_scene
# Whether to create connections among nodes
Scenario.simulateConnections = true
# How frequently to sample the simulation
Scenario.updateInterval = 0.1
# 43200s == 12h
# How long the simulation will run (in seconds)
Scenario.endTime = 43200
```

Figure 2: Scenario Basic Settings

- After we finish giving the basic setting to our scenario, the next step is setting up the network interface for each node (host). In our scenario, we use Bluetooth with 90kbits/s transmission rate in 50 meters transmission range. Therefore, in our "simple_scene", we should add the following commands in Figure 3. The variable "btInterface"

is defined by user themselves. Thus, we can define "btInterface" to any names like "bt","BluetoothInterface" and so on. Compared to the basic scenario setting, the variable name "Scenario" in the previous settings can not be modified by user. If we changed "Scenario" variable name, THE ONE will load the default configuration file.

```
# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 90kByte = 90 x 1024 kbit (it is the maximum speed in iOS device)
btInterface.transmitSpeed = 92160k
# Range of transmission (in meter)
btInterface.transmitRange = 50
```

Figure 3: Network Interface Settings

- When we set up our scenario and the network interface. Then, we need to add our nodes in our scenario. As for the nodes, THE ONE uses the concept - "Group" to distinguish them. In THE ONE simulator, at least, there must be one group in the scenario. Many nodes can be in the same group or different group. It is decided by the setting of our configuration file. As for our test scenario, there is only one group (pedestrian). Therefore, we just need to define one group in our configuration file (Figure 4). If we need cars, bicycle and other kind of nodes, we can create a new group for different type of transportation later. As we can see in the Figure 4, if we want to set up a group, the variable name should start with "Group" plus a digital number. and the digital number should start from value 1. If we start from value 2, but we just have one group, THE ONE will use the default configuration file. Moreover, if we have several groups, most of them have the same properties. Giving the value to them would cost us too much time in set up the configuration file. Therefore, if we only use the variable "Group" without adding a digital number behind. It can save us time. Here, the meaning of "Group" is setting up all the groups as the same properties. such as "Group.router = EpidemicRouter". In this part, there is one thing we need to be careful. Once we set up the number of nodes in this group ("Group1.nrofHosts = 10"). We must guarantee that in the events configuration setting, We need to control the range of the host in events from 0 to less or equal than "Group1.nrofHost". Here the value should be "Events1.hosts = 0,10". Otherwise, THE ONE will throw an error during the running time.

```
# Define the number of group for our scenario, at least one group
Scenario.nrofHostGroups = 1
# group1 (pedestrians) specific settings
Group1.groupID = G
# Mobility model for all the nodes
Group1.movementModel = ShortestPathMapBasedMovement
# Routing protocol to be used by a node in the group
Group1.router = EpidemicRouter
# Buffer size of any node
Group1.bufferSize = 5M
# minimum and maximum wait times (seconds) after reaching destination
Group1.waitTime = 0, 120
# All nodes have the bluetooth interface
Group1.nrofInterfaces = 1
Group1.interface1 = btInterface
# Walking speeds
Group1.speed = 0.5, 1.5
# Message TTL of 300 minutes (5 hours)
Group1.msgTtl = 300
# nrofHosts: number of hosts in the group
Group1.nrofHosts = 10
```

Figure 4: Set up Group and Nodes

- The next step of setting is related to the events generation (mainly about new message creation) in our scenario. As we do the same thing in the previous mention, firstly , we have to set up the number of our event generator (the variable naming rule is the same as that of "Group" definition). Then, we give some properties for our event

generator in order to let him to frequently generate the message that useful for our test as Figure 5. Because in this test, we just need to calculate the connecting time between two nodes. There is no need to guarantee that we successfully deliver each message. So it is not necessary to change the message size to fit one maximum message transmission. Therefore, we randomly pick up a message size range. As for the event host, the range of it must be fit the value we set in the previous part. Otherwise, THE ONE will give us error during the running time.

```
## Message creation parameters
# How many event generators
Events.nrof = 1
# Class of the first event generator
Events1.class = MessageEventGenerator
# (following settings are specific for the MessageEventGenerator class)
# Creation interval in seconds (one new message every 25 to 35 seconds)
Events1.interval = 25,35
# Message sizes (20kBytes - 50kB)
Events1.size = 20k,50k
# range of message source/destination addresses
Events1.hosts = 0,10
# Message ID prefix
Events1.prefix = M
```

Figure 5: Set up Event for Message Generation

- After we finish all the settings for our test scenario, the last things for the configuration file is that we just need to keep the same configuration commands from default_settings.txt to set up the movement, map and GUI information. In this report, I will not describe them. Because they are not extremely useful for our scenario. The values of the setting values in these three parts are list in Figure 6

```
## Movement model settings
# seed for movement models' pseudo random number generator (default = 0)
MovementModel.rngSeed = 1
# World's size for Movement Models without implicit size (width, height; meters)
MovementModel.worldSize = 4500, 3400
# How long time to move hosts in the world before real simulation
MovementModel.warmup = 1000

## Map based movement -movement model specific settings
MapBasedMovement.nrofMapFiles = 4
MapBasedMovement.mapFile1 = data/roads.wkt
MapBasedMovement.mapFile2 = data/main_roads.wkt
MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
MapBasedMovement.mapFile4 = data/shops.wkt

## GUI settings
# GUI underlay image settings
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
# Image offset in pixels (x, y)
GUI.UnderlayImage.offset = 64, 20
# Scaling factor for the image
GUI.UnderlayImage.scale = 4.75
# Image rotation (radians)
GUI.UnderlayImage.rotate = -0.015
# how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents = 100
# Regular Expression log filter (see Pattern-class from the Java API for RE-matching details)
#GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$
```

Figure 6: Movement, Map, GUI Settings

- When we have our configuration file for our scenario. We can run THE ONE and test our scenario(configuration). But before that, we need to do something for Eclipse environment. Otherwise, THE ONE will load the default_settings.txt

file instead of loading our configuration. Therefore, Firstly, we need to select our project "the_one_1.4.1" then right click what we select as Figure 7. Secondly, we select "run as" and click "Run Configurations".

After we click "Run Configurations", we will go to the configuration panel of THE ONE as Figure 8. In this

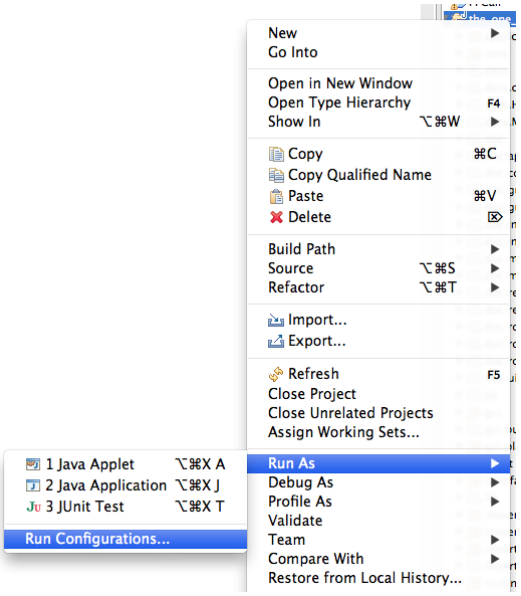

Figure 7: New Settings for Eclipse

panel, we select the second tab "Arguments". In this tab, we type "${string_prompt}" in program arguments area. Then, we run our THE ONE and get a pop up window as Figure 9. In this pop up window, we need to type our "simple_scene.one" file in order to let THE ONE know, we want to run our customized configuration instead of the "default_setting.txt" file.

Finally, we can find our scenario (I zoom out the graph by 0.15 proportion) as shown in the Figure 10. There are ten nodes with 50 meters range of Bluetooth.

3 Modify Code of THE ONE to fit our own purpose

When we have a scenario.Now, we can test some attributes of this scenario for our personal test purpose. In this report, I mainly focus on testing the average connecting time between two nodes. Therefore, we should capture connecting information in each node. Then record the time when THE ONE created a connection from one node to another node. Also, record the time when THE ONE drop a connection between two nodes. After that, we select all nodes connecting information and analyzing them.

In order to reach our goal, there are two approaches to implement our idea. The first one requires us to write some code for our output. The second one is the original one that THE ONE offers to us. As for the first one, we have a very flexibility to output what we want to get. In terms of the second one, it is very easy for us to get the result with less code writing. I will explain these two approaches.

Let us look at the first approach - writing code by ourselves. In order to monitor a connection between two nodes, we have to put our attention to each node. In THE ONE, class "core.DTNHost" is used to generate a node and maintain some attributes of each node. Therefore, if we want to implement our monitor function. Going to and improving "core.DTNHost" is the correct direction. In class "core.DTNHost" we can find two functions. One is "connectionUp", the other is "connectionDown". As their name shows, one will be triggered when there is a connection created between two nodes by Bluetooth. The other will be called when the connection is broken. Thus, we can guess that in order to reach our goal, we need to do some in these two functions. Because "core.DTNHost" is a class for each node. THE ONE will create instances of each node based on this class. Therefore, we change the code in this two functions. It potentially controls and affects all nodes in the scenarios.

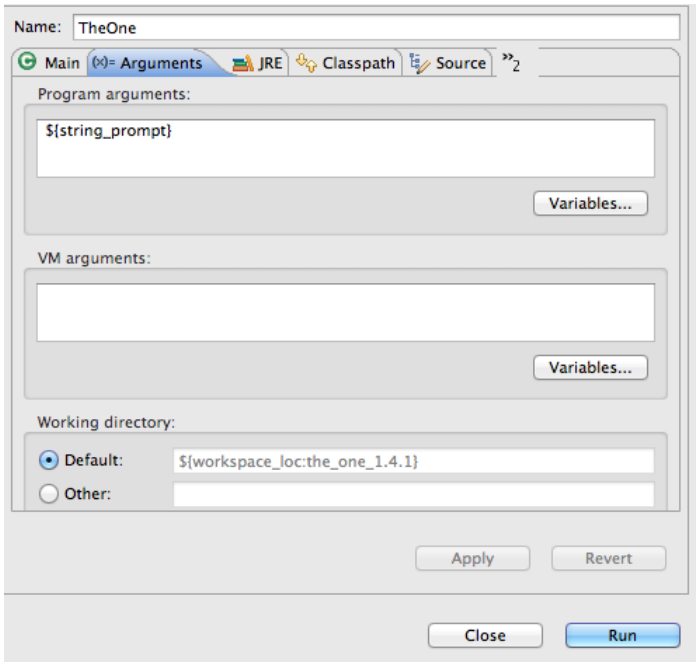

Figure 8: Run Configuration

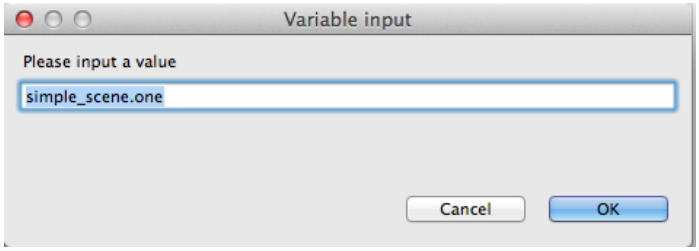

Figure 9: Change Running Configuration File for THE ONE

As we can easily think about that, we can print our connection creating time in "connectionUP" and connection dropping time in "connectionDown" to console. Then compare these two time value based on two nodes. Afterward, it is subconscious for the people to use Java system clock to capture the connection up and down. However, it is wrong. Because we use the simulator, therefore, the time in the simulator is different from the system clock of Java (simulator time will start when we begin the simulation). Thus, we should use the simulator start time instead of system clock. In THE ONE, there is a class called "core.SimClock". When we call the static function "getTime()" of this class, we will get a double value of time. This value is the duration when we start the simulation until now. So, compare this value in connection up status and connection down status, it will be more precise that directly use the system clock of Java. Likewise, we have to consider that if we only print out the time of connection up and down. It is not enough. Because every node will print out the same information to the console. At that time, it is difficult for us to calculate and analyze. Therefore, when we print out our time, we should print out the node id. In THE ONE, we can directly get current node id (in THE ONE, it is called "name") by write code "this.name" in class "core.DTNHost". Then, based on the parameters of two functions ("connectionUp","connectionDown"). we can call function "con.getOtherNode(this)' to get the node which connected to current node. Finally, we are able to calculate the average connecting time by using these kinds of information. Figure 11 shows the modified code in class "core.DTNHost" and the result of running the simulator.

After we finish going through the first approach of getting the connecting time of two nodes. There is another easy way for us to reach the goal. It is easy but not flexible. We can use the class "Report" of THE ONE to output our

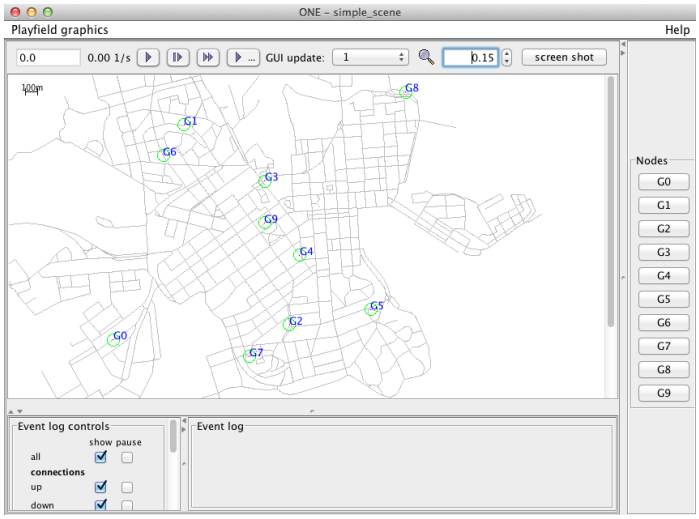

Figure 10: Final Result of Our Test Scenario

```
public void connectionUp(Connection con) {
    System.out.println("UP:"+ this.name + "->"
+ con.getOtherNode(this).name + ":" + SimClock.getTime());
    this.router.changedConnection(con);
}

public void connectionDown(Connection con) {
    System.out.println("DO:"+ this.name + "->"
+ con.getOtherNode(this).name + ":" + SimClock.getTime());
    this.router.changedConnection(con);
}
```

```
Console  Problems  Javadoc  LogCat
<terminated> TheOne [Java Application] /Library/Java/JavaVirtualMachines/1.6.0_32-b05-420.jdk/Cor
UP:G7->G4:542.0000000000547
UP:G4->G7:542.0000000000547
DO:G7->G4:568.6000000000607
DO:G4->G7:568.6000000000607
UP:G9->G7:891.6000000001342
UP:G7->G9:891.6000000001342
DO:G9->G7:911.5000000001387
DO:G7->G9:911.5000000001387
```

Figure 11: Final Result of Our Test Scenario

results. To be detailed, if we want to use the class "Report" of THE ONE to output our results. We have to configure our "simple_scene.one" file in order to let THE ONE know that we want to export the results. The configuration of report exporting is similar to the setting of Group. To begin with, we need to set the number of report by writing command "Report.nrofReports = 1". After that, we need to tell THE ONE where and which class used to save and generate the report file. The settings and results of report are show in Figure 12 and Figure 13. Once we set up these commands in our configuration file. After we close our simulator, in our "reports" folder, we will find a txt file. That file is the file contain the time when two nodes connected to each other, when two nodes disconnect and the index number of two nodes.

In fact, when we try to use both the first approach and second approach to export our result. we can find that the result value is totally the same. But compare to the second approach, the result value only keep the integer part and two decimal number, for the first approach it is more precise. Also, for the first approach we can assign our own format to output. To some extend, the first approach is more flexible for us to customize and trace our result.

```
# how many reports to load
Report.nrofReports = 1
# default directory of reports (can be overridden per Report with output setting)
Report.reportDir = reports/
# Report classes to load
Report.report1 = ConnectivityONEReport
```

Figure 12: Report Settings of THE ONE

```
542.00 CONN 4 7 up
568.60 CONN 4 7 down
891.60 CONN 7 9 up
911.50 CONN 7 9 down
1655.00 CONN 0 5 up
1729.70 CONN 0 5 down
1827.10 CONN 3 7 up
1882.80 CONN 3 7 down
2273.50 CONN 2 4 up
2531.60 CONN 2 4 down
```

Figure 13: Report Export of THE ONE

4 Analysis of THE ONE Experiment Data

As for analysis, THE ONE offers us some basic mathematic functions to get some useful value from a large data set. In class "report.Report", we can find severals functions like "getAverage(List<double>values)","getMedian(List<double> values)","getVariance(List<double> values)" and so on. From these functions, we can pass the value that we get from connectionUp and connectionDown functions. Then generate the value that we want to get to analyze the network. Also, due to the export result, we also can use some statistic application to analyze the result. Such as R and Excel. In the later report, I will conduct another experiment and describe how to use R to analyze a large number of data set.

5 Conclusion

In this report, I mainly describe how to configure THE ONE to load our own setting file in Eclipse. Moreover, I talk about how to use THE ONE to conduct a basic experiment for capturing the connecting time of two nodes. Then I explain how to call class "Report" of THE ONE to generate report and export the report. From these experiments, I can see that THE ONE is a good organized simulator for DTN research.