THE ONE: How The Scenario Is Created by THE ONE

Xu Liu *and Yuanzhu Chen

Department of Computer Science
Memorial University of Newfoundland

May 11, 2013

1 Introduction

As we know that, compare to the famous network simulator-NS2, THE ONE is a relatively young kid. However, it is still a complex system for us to understand in a short time. If we want to fully utilize its features, modify its components to adapt our experiment environment, the best thing is to understand its basic structure and work process. Then, based on our basic understanding, we pick up the important and essential parts of THE ONE to study further and do research deeply. This report gives a relatively detailed description of how THE ONE create a scenario in the java class for a DTN simulation.

1.1 Outline

This report is consisted of two sections, each sections follow the order that THE ONE call them.

- Initialization of running a configuration file in THE ONE
- Creating a scenario in THE ONE

2 Initialization

As everyone knows, any application should have an entry to start. THE ONE has a "main" function in package "Core" of "DTNSim" class. This function will be called at the beginning when the simulator start. In this function. firstly, THE ONE load the "input start arguments". The arguments can be the name of "configuration files" (we mentioned in Case 1 report. By typing "./one.sh simple_scene.one", we can start our THE ONE and pass configuration file "simple_scene.one" as the configuration file value) or the mode of running THE ONE (by typing "one.sh -b 5 simple_scene.one", we can going to the batch mode. In short word, batch mode is console mode [only white letters with black background], if we do not give "-b" parameter, THE ONE will run in GUI mode (show graph)).

When we go into "main" function (core.DTNSim.main()). We can find a very confused concept(variable) "nrofRuns" (it records the number of runs). At the begin of studying THE ONE, we can not fully understand what's its meaning. But after we looking for THE ONE official documents [1]. We know that "run" here means one kind of combination of experiments (simulation) parameters. For example, I want to set two random seeds (random seeds are used to assign different patterns (ways) to generate random numbers, we can simply think about it as trend to generate value. If I have two random seeds, maybe my random value from this two random seeds will be like (1,2,3,4,5,...)[each value is randomly generated by random seeds A] and (1,3,5,7,9...)[each value is randomly generated by random seeds B], every time they are different, but the degree of difference is controlled by random seeds). Ok, Let's go back to our example. When we have

*liuxu.mun@gmail.com

two random seeds (means we have two trends to generate random value), then we have another factor, say three kinds of time intervals for nodes to move. Here, we have 6 combinations of these two factors. if we define the value of "run". we can run our 6 combination in one experiments of THE ONE instead of starting and stop THE ONE six times. In other word, we can easily use one turn to finish conducting the six experiments and get results. Therefore, we can summarize that "run" here means the combination of the parameters. one more thing need to be mentioned here, if we assign 5 run to this situation, even there are 6 combination, it only triggers 5 combinations. The order of these combinations can be found in document [1]. In GUI Mode (window with visualized nodes), we can not use "run" to assign the number of combination. "run" settings are only available in batch mode (white letter with black background).

When we finish selecting our running mode of THE ONE. The next step for THE ONE to do is that it loads the basic configuration file and saves these values into the running memory for later use. As we can see that, in THE ONE, "core.DTNSim.initSettings" function is used to do this work in "main" function. In "initSettings" function, it call the static function of class "Setting.init()". This function loads the basic settings from "default_setting.txt" file. Once it finish loading the default settings, it will call function "Properties.load()" to load the customized settings from other configuration files (here it will load "simple_scene.one" file). Then, use the new value to override the default value in order to create users personal experimental environment. Afterwards, we can get a "Properties" instance "props" in "Setting" class. This instance will save all the information of our configuration file ("default_setting" and "simple_scene.one"). Later, every class will call "Setting" class to obtain their interested attribute value. The main process of initialized THE ONE is shown in Figure 1.

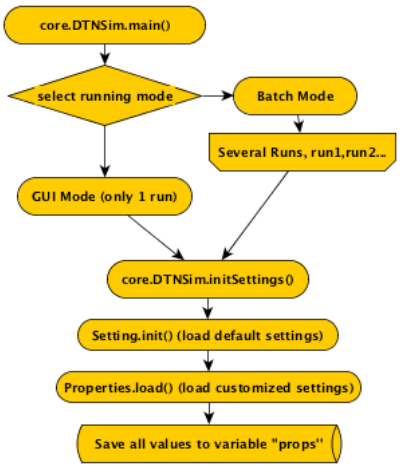

Figure 1: Initialization Process of THE ONE

3 Establish Scenario

When we finished loading the initialization information from our configuration file. Based on our mode, THE ONE will call "DTNSimTextUI" class (batch mode) or "DTNSimGUI" class (GUI mode) to run our simulation. Because GUI mode is straightforward and easy to trace, Also (batch mode) is the same as the GUI mode (just different combination). Therefore, in this report we select the GUI mode.

To be detailed, both "DTNSimTextUI" and "DTNSimGUI" class inherit from "DTNUI" class. Therefore, when we run "DTNSimGUI" class, we need to check "DTNUI" class in advance. In "DTNUI" class there is a function called "start". This function will be called in "core.DTNSim.main()" function once the initialization work is done. "start" function mainly focuses on two things one is "initModel" and another is "runSim".

3.1 Function "initModel"

"initModel" is a function that create the basic scenario for us to conduct our simulation. As we can see in this function, It is consisted of three main parts. 1) create an instance of "SimScenario" class; 2) generate report class to trace our result; 3) warmup our nodes (before we start the real simulation, nodes can move in a period before it start to move in simulation, but when we conduct the experiments, it doesn't works well). In these three parts, part 2: report, part 3: warm up are easy for reader to understand. We mainly focus on part 1: create scenario.

We can deem the scenario as a director in a movie. It organizes everything and takes care of everything. Let's go into the constructor function of "SimScenario" class. The first operation in this constructor is that THE ONE loads the settings value from "Setting" class to initialize the number of groups (each group contains many hosts(nodes), each group can have different features. In "simple_scene.one" configuration file, we only have one group. Therefore, we just have one set of nodes which have the same features in this scenario.); Also, the constructor initializes the update interval of the simulation and some basic parameters. Then, the constructor instance a set of listeners, these listeners are not important for simulation, they are only useful for display some information on THE ONE user interface. In short word, once there are something or value changed in THE ONE. THE ONE will notify the GUI user interface based on these listener. Such as a connection created, message generated and so on. These information will show on the GUI window.

When the constructor finished setting up the listeners. THE ONE initializes a "EventQueueHandler" class. This class is used to handle the event (In THE ONE, we find that, it mainly focus on Message Generated Event). Let's check "EventQueueHandler class". In "EventQueueHandler" constructor, we can clear seen that it load "Events1.class" value and instance a object from "input.MessageEventGenerator" (based on our "simple_scene.one" configuration file) class and put this object as a "Event" to "EventQueue" of "EventQueueHandler" class for later use. Therefore, it is necessary for us to detect and go deeply into what's the function "MessageEventGenerator". When we check "MessageEventGenerator" class. We find that it implement EventQueue interface to return external event object and next event time. "ExternalEvent" is a class used to save the next event information and contain the operation that what this event should do while this event is called by THE ONE (To some extent, it is a Linked List Data Structure). Therefore, we can proof that "MessageEventGenerator" is a event and it contain the operation what it should do while it is called by THE ONE.

Until now, we know that in the "scenario constructor" we have a "EventQueueHandler" object. and this object has only one event (MessageEventGenerator) to generate the message. Then, in the constructor of "scenario", it starts to create our hosts based on our configuration file ("Setting" class has already obtained the information at the begin of THE ONE starting, THE ONE reads that information from "Setting" class and start to create hosts). As for each host, it has network interface, move pattern, routing approach as well as application layer. Therefore, when scenario creates hosts. It needs to helps each host finishes these kinds of initialization. From the codes we know that THE ONE first set up the network interface for its hosts (network interface is responsible to create the connection between two nodes and check the transmission range). Then, it creates the movement pattern and maps (the detailed information related to movement is show in my Case 2 report) for the host. After that, based on the configuration, THE ONE loads the routing information["simple_scene.one" configuration file] (Firstly->EpidemicRouter; Secondly->ActiveRouter(EpidemicRouter inherits from ActiveRouter);Thirdly->MessageRouter (ActiveRouter inherits from MessageRouter)). EpidemicRouter is used for delivering all message to all connections by calling "update" function. This function is implemented by "ActiveRouter". In ActiveRouter, this class is used to detect the state of connection between two nodes. Not only like this, but also, it focus on buffer management and message exchange approaches. MessageRouter does the similar work as ActiveRouter, but it is more detailed than ActiveRouter. Finally, Because we do not set up our application layer, Therefore, there is no application layer class will be called at this time. Then, we put all the information to a host and create each host.

Once we finish creating our hosts, we should give all the hosts, even queue information and so on to our "world". Only in "world", these nodes can get movement and message can be created. There is one thing we need to pay more attention. In the world. it has its own "schedule update queue". this queue used to update message generation, hosts movement and so on. Therefore, we should know that the updating is not based on the "EventQueue" information. It is a combination between "EventQueue" information and "schedule update queue" information (function "setNextEventQueue"). Finally. Our scenario is done. Then we can start to run our simulator.

4 Conclusion

THE ONE offers a mechanism to control the events and update information. As for us, there still are some confused things we can not fully explain. We plan to go deeply of the process of THE ONE in the later life.

References

[1] Ari Keranen. Opportunistic network environment simulator. May 2008.