

Developer Documentation On ONE Simulator

Sahil Gupta
sahilgupta221@gmail.com

Computer Engineering Division
Netaji Subhas Institute of Technology, New Delhi
India

Current Designation:
Software Developer
at
<http://www.shopclues.com/>

INDEX

1. CASE I: Normal Analysis in ONE

1.1 Introduction

1.2 Initialisation Phase working

1.3 Running Phase working

2. CASE II : External File Analysis in ONE

2.1 Introduction

2.2 Initialisation Phase working

2.3 Running Phase working

3. Appendix

3.1 Default_settings.txt file for CASE I Analysis

3.2 Extra default file setting for CASE II analysis

3.3 Sample file of CASE II Analysis

3.4 Sample section of file containing MessageEvent List

CASE I: Normal Analysis in ONE

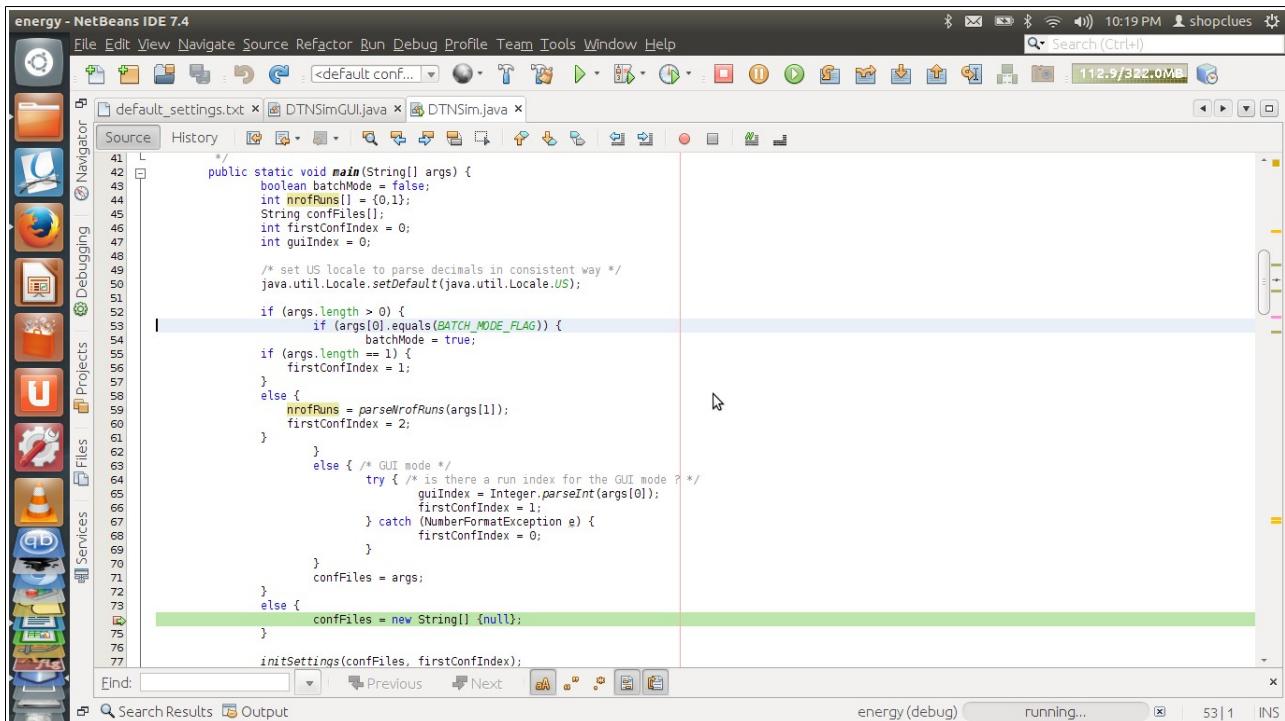
Introduction

Normal Analysis can be understood by seeing the Appendix default_file setting section. We have taken Prophet as router and Random Way Point as Mobility Model. This documentation aims to provide the explanation of ONE Architecture with the specific case studies of the default_setting.txt file as shown in the Appendix. Readers are advised to first read the author's written documentation of ONE from following link:

http://www.netlab.tkk.fi/tutkimus/dtn/theone/pub/the_one.pdf

This documentation also explains various components along with their functionality and responsibility code by code. This Study is divided into two phases

1. Initialisation of Simulation enviroment.
2. Running during simulation till end.



```
energy - NetBeans IDE 7.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Source History Navigator Projects Files Services
default_settings.txt DTNSimGUI.java DTNSim.java
41 L public static void main(String[] args) {
42     boolean batchMode = false;
43     int nrOfRuns[] = {0,1};
44     String confFiles[];
45     int firstConfIndex = 0;
46     int guiIndex = 0;
47
48     /* set US locale to parse decimals in consistent way */
49     java.util.Locale.setDefault(java.util.Locale.US);
50
51     if (args.length > 0) {
52         if (args[0].equals(BATCH_MODE_FLAG)) {
53             batchMode = true;
54         if (args.length == 1) {
55             firstConfIndex = 1;
56         }
57         else {
58             nrOfRuns = parseNrOfRuns(args[1]);
59             firstConfIndex = 2;
60         }
61         else /* GUI mode */
62             try { /* is there a run index for the GUI mode ? */
63                 guiIndex = Integer.parseInt(args[0]);
64                 firstConfIndex = 1;
65             } catch (NumberFormatException e) {
66                 firstConfIndex = 0;
67             }
68         }
69         confFiles = args;
70     }
71     else {
72         confFiles = new String[] {null};
73     }
74     initSettings(confFiles, firstConfIndex);
75
76
77 }
```

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+F) 3:52 PM shopclues

Projects Files Services

DTNSim.java x Settings.java x default_settings.txt x

Source History

```

else {
    confFiles = new String[] {null};
}
initSettings(confFiles, firstConfIndex);

```

Variables x Output

Name	Type	Value
confFiles	String[]	#89(length=1)
<Enter new watch>		...
Static		...
args	String[]	#85(length=0)
batchMode	boolean	false
nrofRuns	int[]	#86(length=2)
firstConfIndex	int	0
guiIndex	int	0
confFiles	String[]	#89(length=1)
[0]		null

Search Results Breakpoints

energy (debug) vikasgarg has gone offline

NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+F) 4:31 PM shopclues

Projects Files Services

DTNSim.java x default_settings.txt x

Source History

```

        double duration = (System.currentTimeMillis() - startTime)/1000.0;
        print("---\nAll done in " + String.format("%.2f", duration) + "s");
    } else {
        Settings.setRunIndex(guiIndex);
        new DTNSimGUI().start();
    }
}

```

Variables x Output

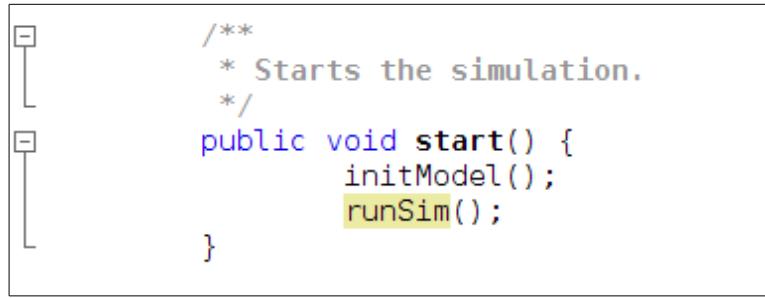
Name	Type	Value
<Enter new watch>		...
Static		...
args	String[]	#85(length=0)
batchMode	boolean	false
nrofRuns	int[]	#86(length=2)
[0]	int	0
[1]	int	1
firstConfIndex	int	0
guiIndex	int	0
confFiles	String[]	#87(length=1)
[0]		null

Search Results Breakpoints

energy (debug) running... 92 | 1 INS

Initialisation Phase working

I am discussing Simulator in GUI mode operation. It starts from DTNSim class. Here you will find main class of simulator. It reaches else part *if(arg.length>0)* condition as seen in green area figure. Here *initSettings(confFiles, firstConfIndex)* is doing noting. Then it moves to else part of *if(batchMode)* condition. Here *Settings.setRunIndex(guiIndex)* sets the index in setting file. Since there is no additional config file to read, this function plays no role here. Now *new DTNSimGUI().start()* is the point of interest. It invokes two methods *initModel()* and *runSim()*. First one is the topic of discussion in this section. Running Phase will be covered later on.



energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Source History Navigator

```

87     /**
88      * Starts the simulation.
89      */
90     public void start() {
91         initModel();
92         runSim();
93     }
94
95     /**
96      * Initializes the model.
97      */
98     private void initModel() {
99         Settings settings = null;
100
101        try {
102            settings = new Settings();
103            this.scen = SimScenario.getInstance();
104
105            // add reports
106            for (int i=1, n = settings.getInt(NROF_REPORT_S); i<=n; i++) {
107                String reportClass = settings.getSetting(REPORT_S + i);
108                addReport((Report)settings.createObject(REPORT_PAC + reportClass));
109            }
110
111            double warmupTime = 0;
112            if (settings.contains(MM_WARMUP_S)) {
113                warmupTime = settings.getDouble(MM_WARMUP_S);
114                if (warmupTime > 0) {
115                    SimClock c = SimClock.getInstance();
116                    c.setTime(-warmupTime);
117                }
118
119                this.world = this.scen.getWorld();
120                world.warmupMovementModel(warmupTime);
121            }
122
123            catch (SettingsError se) {
124                System.err.println("Can't start: error in configuration file(s)");
125                System.err.println(se.getMessage());
126                System.exit(-1);
127            }
128            catch (SimError er) {
129                System.err.println("Can't start: " + er.getMessage());
130                System.err.println("Caught at " + er.getStackTrace()[0]);
131                System.exit(-1);
132            }
133        }
134    }

```

Find: Previous Next

Search Results Output energy(debug) running... 87 | 12 | INS

Code is divided into three parts for discussion of their functionality.

First :

```

settings = new Settings();
this.scen = SimScenario.getInstance();

```

Second :

```

// add reports
for (int i=1, n = settings.getInt(NROF_REPORT_S); i<=n; i++) {
    String reportClass = settings.getSetting(REPORT_S + i);
    addReport((Report)settings.createObject(REPORT_PAC + reportClass));
}

```

Third:

```

double warmupTime = 0;
if (settings.contains(MM_WARMUP_S)) {
    warmupTime = settings.getDouble(MM_WARMUP_S);
    if (warmupTime > 0) {
        SimClock c = SimClock.getInstance();
    }
}

```

```
    c.setTime(-warmupTime);
}
this.world = this.scen.getWorld();
world.warmupMovementModel(warmupTime);
```

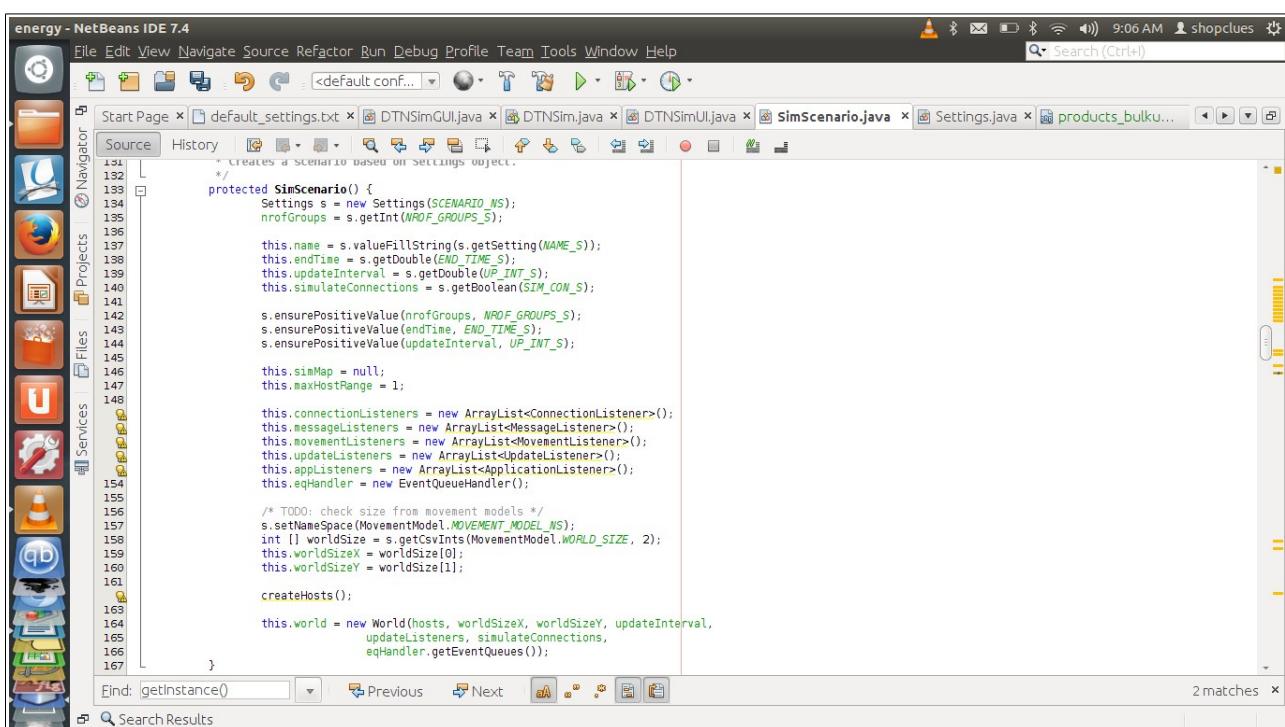
Lets us see the internal working of each task one by one in detail.

`settings = new Settings()` creates setting object. Setting object is used to read the value of parameters from Config files or generating object of class dynamically. The `Setting.java` method and attributes can be studied from Docs. This class can be specifically used when you want to obtain value from config file. This can be interger, decimal,string and even can be used to load class object dynamically.

After this, `this.scen = SimScenario.getInstance()` does the major task. Let us see what happens inside this method(see below snapshot). This will call the constructor of Simscenario class.

```
public static SimScenario getInstance() {  
    if (myinstance == null) {  
        myinstance = new SimScenario();  
    }  
    return myinstance;  
}
```

Now flow moves to construct where scenario elements are created and scenario variables are initialized



Bunch of stuff is happening here. Let us see one by one.

```
Settings s = new Settings(SCENARIO_NS);
nrofGroups = s.getInt(NROF_GROUPS_S);

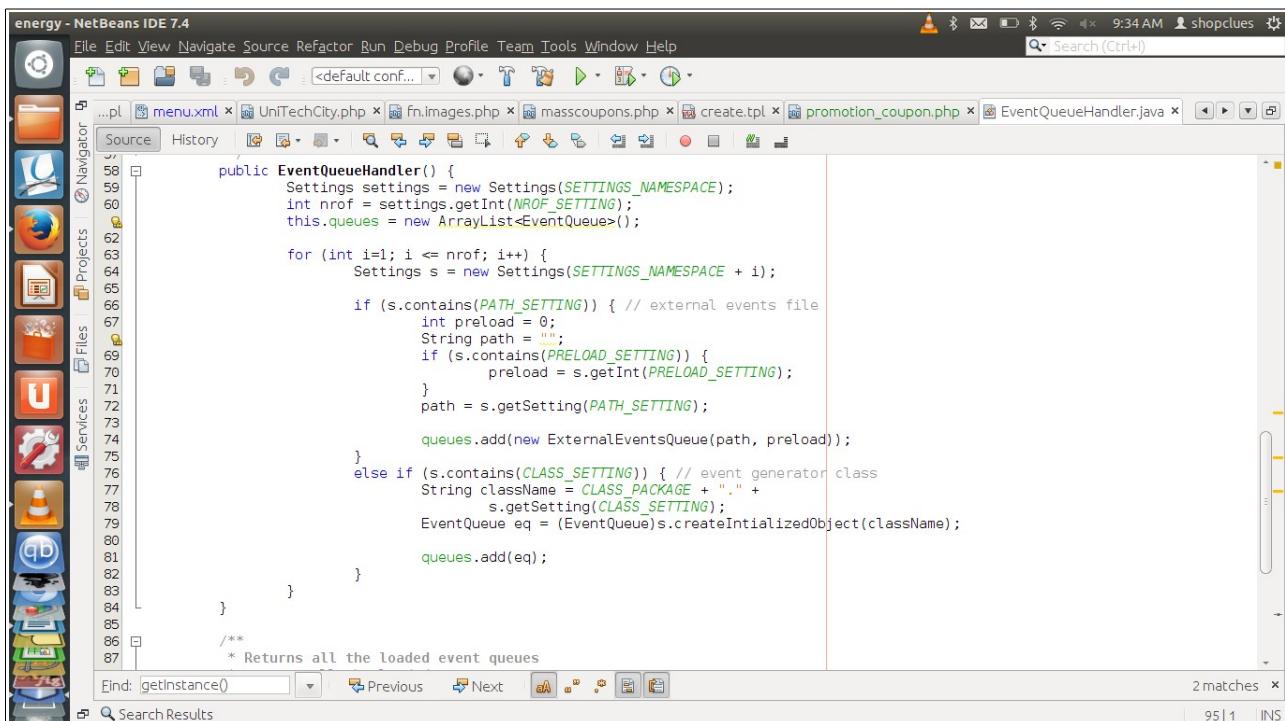
this.name = s.valueFillString(s.getSetting(NAME_S));
this.endTime = s.getDouble(END_TIME_S);
this.updateInterval = s.getDouble(UP_INT_S);
this.simulateConnections = s.getBoolean(SIM_CON_S);

s.ensurePositiveValue(nrofGroups, NROF_GROUPS_S);
s.ensurePositiveValue(endTime, END_TIME_S);
s.ensurePositiveValue(updateInterval, UP_INT_S);

this.simMap = null;
this.maxHostRange = 1;

this.connectionListeners = new ArrayList<ConnectionListener>();
this.messageListeners = new ArrayList<MessageListener>();
this.movementListeners = new ArrayList<MovementListener>();
this.updateListeners = new ArrayList<UpdateListener>();
this.appListeners = new ArrayList<ApplicationListener>();
this.eqHandler = new EventQueueHandler();
```

Much of the stuff is self explanatory. Various parameters are getting their values from config file. Various listeners are being initialised. But `this.eqHandler = new EventQueueHandler()` is of special interest to us. Let us see what happens in there.



The screenshot shows the NetBeans IDE 7.4 interface with the title bar "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run. The left sidebar has icons for Projects, Files, Services, and Navigator. The Navigator panel shows files like menu.xml, UniTechCity.php, fn.images.php, masscoupons.php, create.tpl, promotion_coupon.php, and EventQueueHandler.java. The main editor area displays the code for EventQueueHandler.java:

```
public EventQueueHandler() {
    Settings settings = new Settings(SETTINGS_NAMESPACE);
    int nrof = settings.getInt(NROF_SETTING);
    this.queues = new ArrayList<EventQueue>();

    for (int i=1; i <= nrof; i++) {
        Settings s = new Settings(SETTINGS_NAMESPACE + i);

        if (s.contains(PATH_SETTING)) { // external events file
            int preload = 0;
            String path = "";
            if (s.contains(PRELOAD_SETTING)) {
                preload = s.getInt(PRELOAD_SETTING);
            }
            path = s.getSetting(PATH_SETTING);

            queues.add(new ExternalEventsQueue(path, preload));
        } else if (s.contains(CLASS_SETTING)) { // event generator class
            String className = CLASS_PACKAGE + "." +
                s.getSetting(CLASS_SETTING);
            EventQueue eq = (EventQueue)s.createInitializedObject(className);
            queues.add(eq);
        }
    }
}

/**
 * Returns all the loaded event queues
 */

```

The code initializes an array of EventQueue objects. It iterates through a range of indices (1 to nrof) and for each index, it creates a new Settings object. It then checks if the setting contains a path. If it does, it creates an ExternalEventsQueue with the specified path and preload value. If it contains a class setting, it creates an EventQueue of that class type. Finally, it adds both types of queues to a list.

Clearly it performs two tasks. One External Event Path and Other is event generator initialisation. In

our case there is no external event file and Event generator class is MessageEventGenerator. This will be explained later in running phase the detail function of event generators. Here it is sufficient to know that this class will help in creating message for random source and destination. It also sets message in source node message buffer.

Back to Simscenario constructure.

```
s.setNameSpace(MovementModel.MOVEMENT_MODEL_NS);
int [] worldSize = s.getCsvInts(MovementModel.WORLD_SIZE, 2);
this.worldSizeX = worldSize[0];
this.worldSizeY = worldSize[1];
```

Here World size is getting set.

```
createHosts();

this.world = new World(hosts, worldSizeX, worldSizeY, updateInterval,
                      updateListeners, simulateConnections,
                      eqHandler.getEventQueues());
```

Both portions are of special interest to us. Let us see first what createHosts() will do first.

```

protected void createHosts() {
    this.hosts = new ArrayList<OTNHost>();
    for (int i=1; i<profGroups; i++) {
        List<NetworkInterface> interfaces =
            new ArrayList<NetworkInterface>();
        Settings s = new Settings(GROUP_NS+i);
        s.setSecondaryNamespace(GROUP_NS);
        String gid = s.getSetting(GROUP_ID_S);
        int nrofHosts = s.getInt(NR_OF_HOSTS_S);
        int nrofInterfaces = s.getInt(NR_OF_INTERF_S);
        int appCount;
        // creates prototypes of MessageRouter and MovementModel
        MovementModel mmProto =
            (MovementModel)s.createInitializedObject(MY_PACKAGE +
                s.getSetting(MOVEMENT_MODEL_S));
        MessageRouter mRouterProto =
            (MessageRouter)s.createInitializedObject(ROUTING_PACKAGE +
                s.getSetting(ROUTER_S));
        /* checks that these values are positive (throws Error if not) */
        s.ensurePositiveValue(nrofHosts, NR_OF_HOSTS_S);
        s.ensurePositiveValue(nrofInterfaces, NR_OF_INTERF_S);
        // setup interfaces
        for (int j=1;j<nrofInterfaces;j++) {
            String intName = s.getSetting(INTERFACE_NAME_S + j);
            Settings intSettings = new Settings(intName);
            NetworkInterface iface =
                (NetworkInterface)intSettings.createInitializedObject(
                    INTTYPE_PACKAGE + intSettings.getSetting(INTTYPE_S));
            iface.setListeners(connectionListeners);
            iface.setGroupSettings(s);
            interfaces.add(iface);
        }
        // setup applications
        if (s.contains(APPCOUNT_S)) {
            appCount = s.getInt(APPCOUNT_S);
        } else {
            appCount = 0;
        }
    }
}

```

The screenshot shows the NetBeans IDE interface with the code editor open. The code is annotated with line numbers from 316 to 356. A red vertical line highlights the 'createHosts()' method. The code initializes hosts, MovementModel, and NetworkInterface objects based on settings from a configuration file. The NetBeans toolbar and menu bar are visible at the top, and the project navigation pane is on the left.

```

energy - NetBeans IDE 7.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Source History
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
for (int j=1; j<=appCount; j++) {
    String appname = null;
    Application protoApp = null;
    try {
        // Get name of the application for this group
        appname = s.getSetting(APPPNAME_S);
        // Get settings for the given application
        Settings t = new Settings(appname);
        // Load the instance of the application
        protoApp = (Application)t.createInitializedObject(
            APP_PACKAGE, t.getSetting(APFTYPE_S));
        // Set application
        protoApp.setAppListeners(this.applisteners);
        // Set the proto application in proto router
        //mRouterProto.setApplication(protoApp);
        mRouterProto.addApplication(protoApp);
    } catch (SettingsError se) {
        // Failed to create an application for this group
        System.err.println("Failed to setup an application: " + se);
        System.err.println("Caught at " + se.getStackTrace()[0]);
        System.exit(-1);
    }
}
if (mmProto instanceof MapBasedMovement) {
    this.simMap = ((MapBasedMovement)mmProto).getMap();
}
// creates hosts of its group
for (int j=0; j<nrofHosts; j++) {
    ModuleCommunicationBus comBus = new ModuleCommunicationBus();

    // prototypes are given to new DTNHost which replicates
    // new instances of movement model and message router
    DTNHost host = new DTNHost(this.messageListeners,
        this.movementListeners, gid, interfaces, comBus,
        mmProto, mRouterProto);
    hosts.add(host);
}

```

Find: getInstance() Previous Next Search Results 2 matches 368 | 69 | INS

Now here something happens that might interest the fellow reasearchers.

Story itself contained in for loop.

```
for (int i=1; i<=nrofGroups; i++) {
```

....

....

}

Here nodes are intialised as DTNHost class and all setting associated with it. Here Movement model, Router, Network Interfaces for communication, Application layer(if any) and module communication bus are initialised. The following code sets our nodes of scenario in the enviroment.

```

DTNHost host = new DTNHost(this.messageListeners,
    this.movementListeners,gid, interfaces, comBus,
    mmProto, mRouterProto);
hosts.add(host);

```

The details of this constructure calling and network interface position updatation working in the form of cells will be disscussed in running phase at its appropriate position.

Coming back to

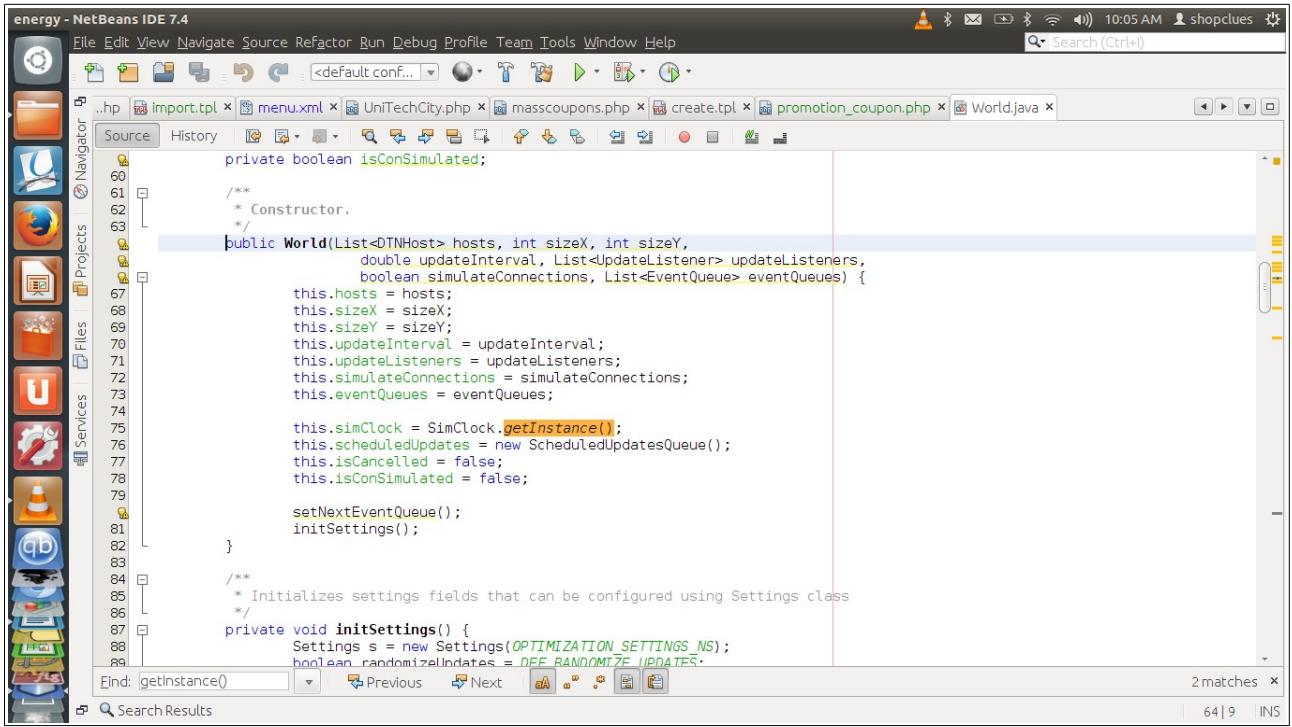
```
createHosts();
```

```

this.world = new World(hosts, worldSizeX, worldSizeY, updateInterval,
    updateListeners, simulateConnections,
    eqHandler.getEventQueues());

```

Now the world is intialised which is used in running phase to update nodes and carrying out the simulation. Lets have a look inside its constructure.



The screenshot shows the NetBeans IDE 7.4 interface with the title bar "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has icons for file operations like Open, Save, Find, and Run. The Navigator panel on the left shows the project structure with files like import.tpl, menu.xml, UniTechCity.php, masscoupons.php, create.tpl, promotion_coupon.php, and World.java. The main code editor displays Java code for the World class:

```
private boolean isConSimulated;

    /**
     * Constructor.
     */
    public World(List<DTNHost> hosts, int sizeX, int sizeY,
                double updateInterval, List<UpdateListener> updateListeners,
                boolean simulateConnections, List<EventQueue> eventQueues) {
        this.hosts = hosts;
        this.sizeX = sizeX;
        this.sizeY = sizeY;
        this.updateInterval = updateInterval;
        this.updateListeners = updateListeners;
        this.simulateConnections = simulateConnections;
        this.eventQueues = eventQueues;

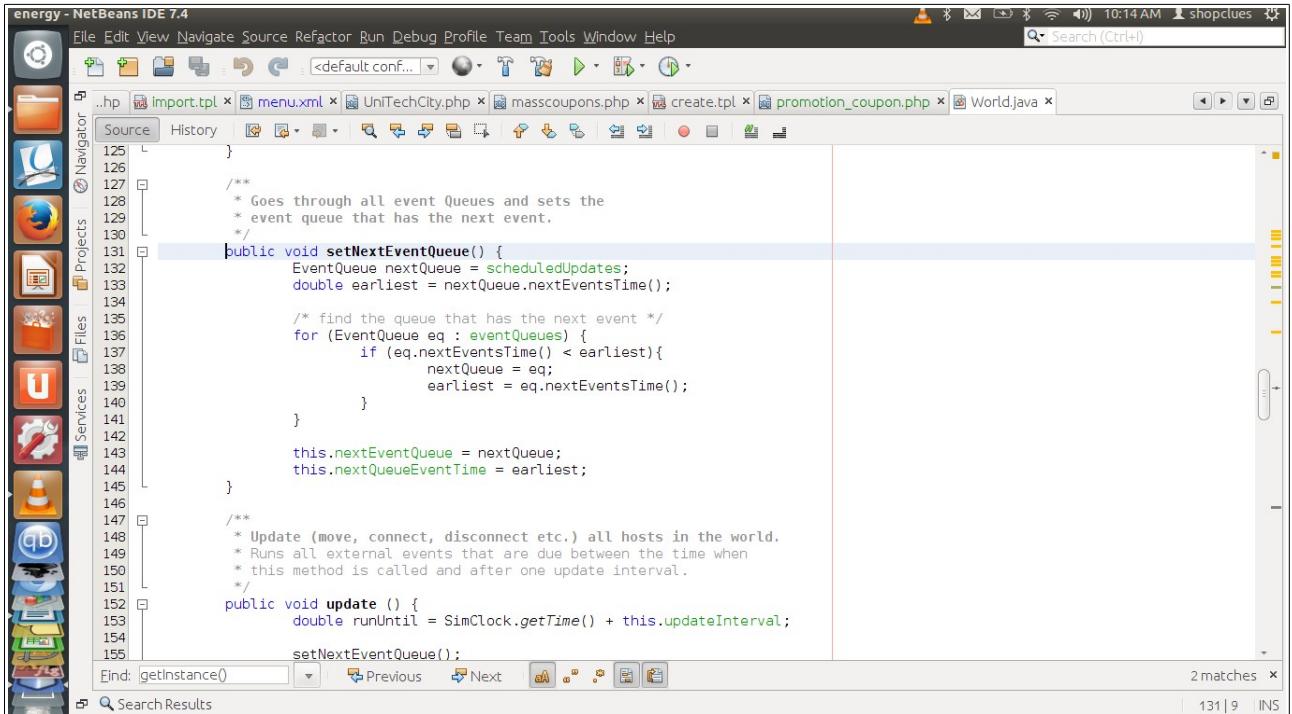
        this.simClock = SimClock.getInstance();
        this.scheduledUpdates = new ScheduledUpdatesQueue();
        this.isCancelled = false;
        this.isConSimulated = false;

        setNextEventQueue();
        initSettings();
    }

    /**
     * Initializes settings fields that can be configured using Settings class
     */
    private void initSettings() {
        Settings s = new Settings(OPTIMIZATION_SETTINGS_NS);
        boolean randomizeUpdates = DEF_RANDOMIZE_UPDATES;
    }
}
```

The search results panel at the bottom shows 2 matches found in 64 lines of code.

Everything is self explanatory. The `setNextEventQueue()` sets the `eventQueues` which is `MessageEventGenerator` class in our case. Let us see how it looks.



The screenshot shows the NetBeans IDE 7.4 interface with the title bar "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has icons for file operations like Open, Save, Find, and Run. The Navigator panel on the left shows the project structure with files like import.tpl, menu.xml, UniTechCity.php, masscoupons.php, create.tpl, promotion_coupon.php, and World.java. The main code editor displays Java code for the World class, specifically focusing on the `setNextEventQueue()` method:

```
    /**
     * Goes through all event Queues and sets the
     * event queue that has the next event.
     */
    public void setNextEventQueue() {
        EventQueue nextQueue = scheduledUpdates;
        double earliest = nextQueue.nextEventsTime();

        /* find the queue that has the next event */
        for (EventQueue eq : eventQueues) {
            if (eq.nextEventsTime() < earliest){
                nextQueue = eq;
                earliest = eq.nextEventsTime();
            }
        }

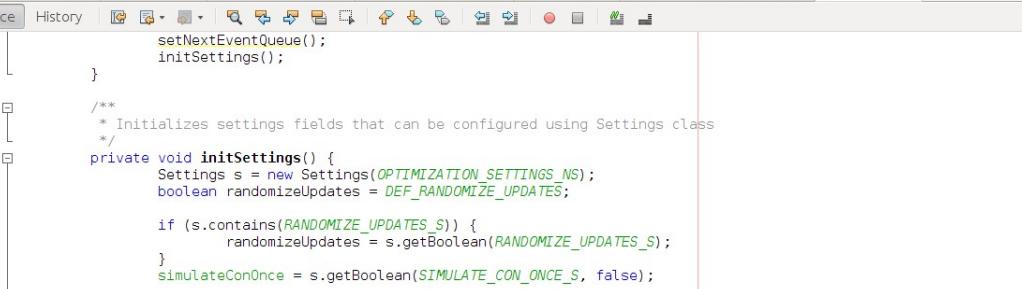
        this.nextEventQueue = nextQueue;
        this.nextQueueEventTime = earliest;
    }

    /**
     * Update (move, connect, disconnect etc.) all hosts in the world.
     * Runs all external events that are due between the time when
     * this method is called and after one update interval.
     */
    public void update () {
        double runUntil = SimClock.getTime() + this.updateInterval;

        setNextEventQueue();
    }
}
```

The search results panel at the bottom shows 2 matches found in 131 lines of code.

The last line `initSettings()` sets the order in which nodes will be updated. It can be Random or in accordance to network address of nodes according to the config file setting. Let us see how this looks like.



The screenshot shows the NetBeans IDE interface with the title bar "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has icons for file operations like Open, Save, Import, Export, and a search bar with "Search (Ctrl+F)" and a magnifying glass icon. The Navigator panel on the left lists "Source", "History", and other project components like "Import.tpl", "menu.xml", "UniTechCity.php", "masscoupons.php", "create.tpl", "promotion_coupon.php", and "World.java". The main code editor window displays the following Java code:

```
setNextEventQueue();
initSettings();

/**
 * Initializes settings fields that can be configured using Settings class
 */
private void initSettings() {
    Settings s = new Settings(OPTIMIZATION_SETTINGS_NS);
    boolean randomizeUpdates = DEF_RANDOMIZE_UPDATES;

    if (s.containsKey(RANDOMIZE_UPDATES_S)) {
        randomizeUpdates = s.getBoolean(RANDOMIZE_UPDATES_S);
    }
    simulateConOnce = s.getBoolean(SIMULATE_CON_ONCE_S, false);

    if (randomizeUpdates) {
        // creates the update order array that can be shuffled
        this.updateOrder = new ArrayList<DTNHost>(this.hosts);
    }
    else { // null pointer means "don't randomize"
        this.updateOrder = null;
    }
}

/**
 * Moves hosts in the world for the time given time initialize host
 * positions properly. SimClock must be set to <CODE>-time</CODE> before
 * calling this method.
 * @param time The total time (seconds) to move
 */
```

Comming back to DTNSimUI *initModel()* method.

Lets us see what task second will do.

```
// add reports
for (int i=1, n = settings.getInt(NROF_REPORT_S); i<=n; i++){
    String reportClass = settings.getSetting(REPORT_S + i);
    addReport((Report)settings.createObject(REPORT_PAC + reportClass));
}
```

Here three things will happen. First dynamic object loading through `createObject` method. Then typecasting into Report Class and then passing it to `addReport` method. Let us see them one by one.

```

742     }
743
744     /**
745      * Creates (and dynamically loads the class of) an object that
746      * initializes itself using the settings in this Settings object
747      * (given as the only parameter to the constructor).
748      * @param className Name of the class of the object
749      * @return Initialized object
750      * @throws SettingsError if object couldn't be created
751     */
752     public Object createInitializedObject(String className) {
753         Class<?>[] argsClass = {Settings.class};
754         Object[] args = {this};
755
756         return loadObject(className, argsClass, args);
757     }
758
759     /**
760      * Creates (and dynamically loads the class of) an object using the
761      * constructor without any parameters.
762      * @param className Name of the class of the object
763      * @return Initialized object
764      * @throws SettingsError if object couldn't be created
765     */
766     public Object createObject(String className) {
767         return loadObject(className, null, null);
768     }
769
770     /**
771      * Dynamically loads and creates an object.
772      * @param className Name of the class of the object
773      * @param argsClass Class(es) of the argument(s) or null if no-argument
774      * constructor should be called
775     */

```

```

779     private Object loadObject(String className, Class<?>[] argsClass,
780         Object[] args) {
781         Object o = null;
782         Class<?> objClass = getClass(className);
783         Constructor<?> constructor;
784
785         try {
786             if (argsClass != null) { // use a specific constructor
787                 constructor = objClass.getConstructor((Class[])argsClass);
788                 o = constructor.newInstance(args);
789             }
790             else { // call empty constructor
791                 o = objClass.newInstance();
792             }
793         } catch (SecurityException e) {
794             e.printStackTrace();
795             throw new SettingsError("Fatal exception " + e, e);
796         } catch (IllegalArgumentException e) {
797             e.printStackTrace();
798             throw new SettingsError("Fatal exception " + e, e);
799         } catch (NoSuchMethodException e) {
800             e.printStackTrace();
801             throw new SettingsError("Class '" + className +
802                 "' doesn't have a suitable constructor", e);
803         } catch (InstantiationException e) {
804             e.printStackTrace();
805             throw new SettingsError("Can't create an instance of '" +
806                 className + "'", e);
807         } catch (IllegalAccessException e) {
808             e.printStackTrace();
809             throw new SettingsError("Fatal exception " + e, e);
810         } catch (InvocationTargetException e) {
811             // this exception occurs if initialization of the object fails
812             if (e.getCause() instanceof SettingsError) {
813                 throw ((SettingsError)e.getCause());
814             }
815             else {
816                 e.printStackTrace();
817                 throw new SimError("Couldn't create settings-accepting object" +
818                     " for '" + className + "'\n" + "cause: " + e.getCause(), e);
819             }
820         }
821         return o;
822     }

```

Here `createInitializedObject` and `createObject` methods are used to load the object dynamically. This further use `loadObject` Method. `Class<?>` is the key here for loading object dynamically. Readers are advised to see Oracle Java Docs for further information on this topic.

Now see `addReport` Method in `DTNSimUI` class.

```
/**
```

```

* Adds a new report for simulator
* @param r Report to add
*/
protected void addReport(Report r) {
    if (r instanceof MessageListener) {
        scen.addMessageListener((MessageListener)r);
    }
    if (r instanceof ConnectionListener) {
        scen.addConnectionListener((ConnectionListener)r);
    }
    if (r instanceof MovementListener) {
        scen.addMovementListener((MovementListener)r);
    }
    if (r instanceof UpdateListener) {
        scen.addUpdateListener((UpdateListener)r);
    }
    if (r instanceof ApplicationListener) {
        scen.addApplicationListener((ApplicationListener)r);
    }
    this.reports.add(r);
}
}

```

Here you get your first clue how various reports in report Pakage generate report. Funda is pretty simple. Reports implement various kinds of interfaces depending upon which method they want to invoke or define. Later in running phase we will see that various classes get iterated in their interface loop like MessageListeners UpdateListener,MovementListener etc. This is again basic java stuff and to get indepth knowledge of such schemes, One can read Design Patterns in Java Head First Edition or any other book on Design Patterns can be reffered.

Moving on to the third task.

```

double warmupTime = 0;
if (settings.contains(MM_WARMUP_S)) {
    warmupTime = settings.getDouble(MM_WARMUP_S);
    if (warmupTime > 0) {
        SimClock c = SimClock.getInstance();
        c.setTime(-warmupTime);
    }
}

this.world = this.scen.getWorld();
world.warmupMovementModel(warmupTime);

```

Here, First Warmup time is obtained. Then set in the simulator clock. Then world warmupMovementModel Method is called.

```

/**
 * Moves hosts in the world for the time given time initialize host
 * positions properly. SimClock must be set to <CODE>-time</CODE> before
 * calling this method.
 * @param time The total time (seconds) to move
 */
public void warmupMovementModel(double time) {
    if (time <= 0) {
        return;
    }

    while(SimClock.getTime() < -updateInterval) {
        moveHosts(updateInterval);
        simClock.advance(updateInterval);
    }

    double finalStep = -SimClock.getTime();

    moveHosts(finalStep);
    simClock.setTime(0);
}

```

moveHost method will be discussed in the running phase section.

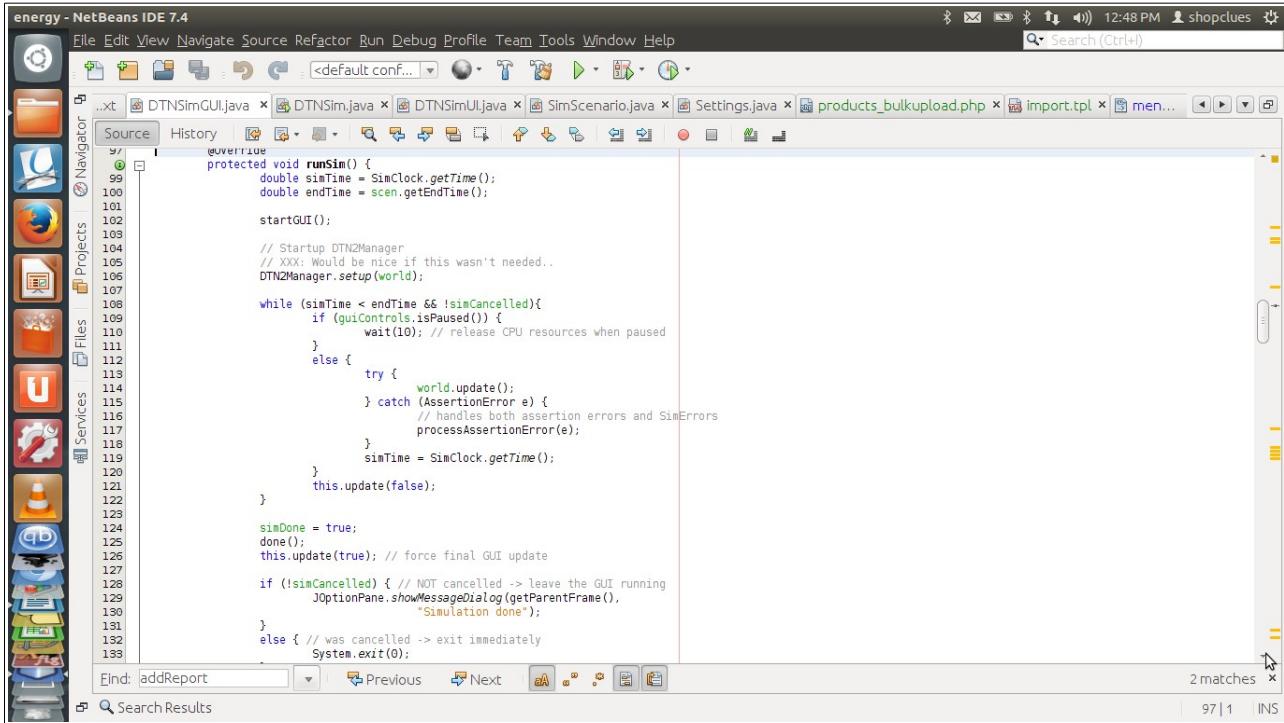
Now again back to start() Method of DTNSimUI class. Till now all required objects are loaded and are interlinked as seen above. Now as the time passes simulator call methods on UpdateInterval value(as defined in config file). Let us see Running Phase.

```


    /**
     * Starts the simulation.
     */
    public void start() {
        initModel();
        runSim();
    }


```

Running Phase working



The screenshot shows the NetBeans IDE 7.4 interface. The title bar reads "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run. The code editor window displays Java code for the `runSim()` method. The code starts with an overridden protected void declaration, initializes `simTime` and `endTime`, calls `startGUI()`, and then enters a loop. Inside the loop, it checks if the simulation is paused. If paused, it waits for 10 seconds. If not, it tries to update the world, catches assertion errors, processes them, and then updates the GUI. After each iteration, it checks if the simulation is done. If not done, it continues the loop. If done, it shows a message dialog and then updates the GUI again. Finally, it exits the loop if not cancelled and exits the application if cancelled. The code editor has a search bar at the bottom with "Find: addReport" and "Search Results" buttons.

```
@Override
protected void runSim() {
    double simTime = SimClock.getTime();
    double endTime = scen.getEndTime();

    startGUI();

    // Startup DTN2Manager
    // XXX: Would be nice if this wasn't needed..
    DTN2Manager.setup(world);

    while (simTime < endTime && !simCancelled) {
        if (guiControls.isPaused()) {
            wait(10); // release CPU resources when paused
        } else {
            try {
                world.update();
            } catch (AssertionError e) {
                // handles both assertion errors and SimErrors
                processAssertionError(e);
            }
            simTime = SimClock.getTime();
        }
        this.update(false);
    }

    simDone = true;
    done();
    this.update(true); // force final GUI update

    if (!simCancelled) { // NOT cancelled -> leave the GUI running
        JOptionPane.showMessageDialog getParentFrame(),
            "Simulation done");
    }
    else { // was cancelled -> exit immediately
        System.exit(0);
    }
}
```

The run sim code has been divided into following section for discussion:

First:

```
double simTime = SimClock.getTime();
double endTime = scen.getEndTime();

startGUI();

// Startup DTN2Manager
// XXX: Would be nice if this wasn't needed..
DTN2Manager.setup(world);
```

Second:

```
while (simTime < endTime && !simCancelled){
    if (guiControls.isPaused()) {
        wait(10); // release CPU resources when paused
    }
    else {
        try {
            world.update();
        } catch (AssertionError e) {
            // handles both assertion errors and SimErrors
            processAssertionError(e);
        }
    }
}
```

```

        simTime = SimClock.getTime());
    }
    this.update(false);
}

```

Third:

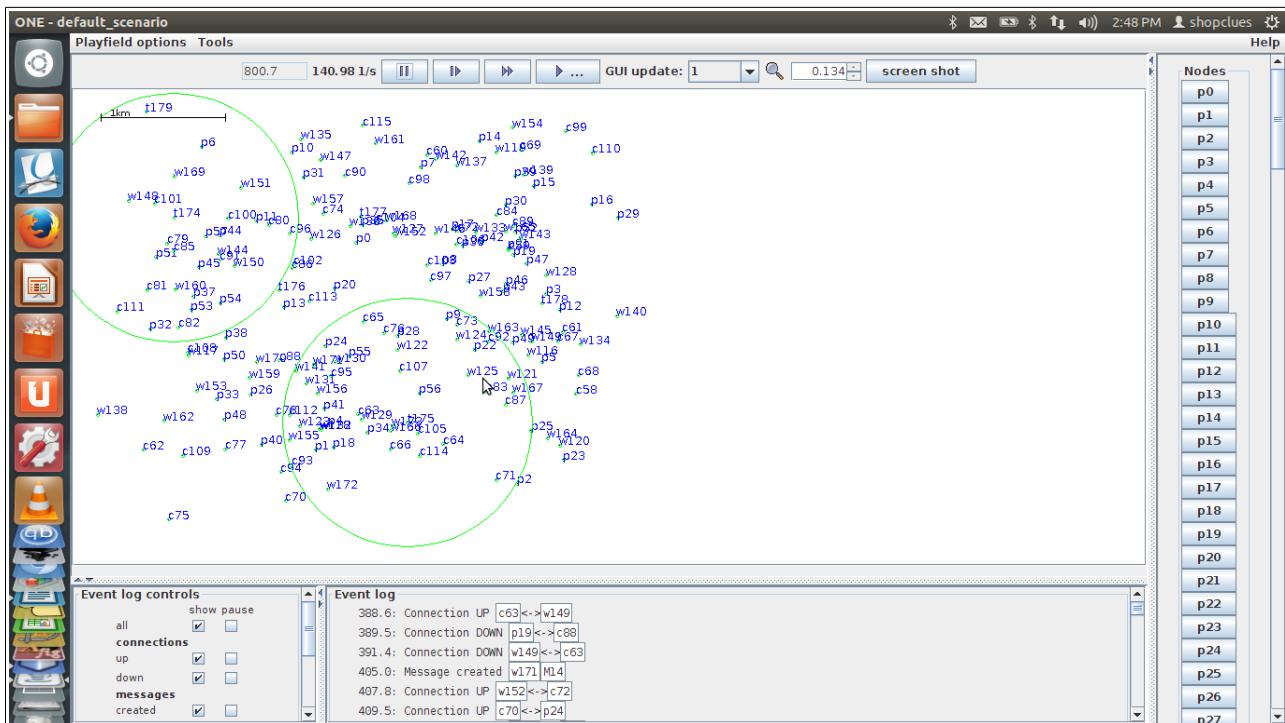
```

simDone = true;
done();
this.update(true); // force final GUI update

if (!simCancelled) { // NOT cancelled -> leave the GUI running
    JOptionPane.showMessageDialog(getParentFrame(),
        "Simulation done");
}
else { // was cancelled -> exit immediately
    System.exit(0);
}

```

In first task, simulator time and end time is obtained. *StartGUI()* method will initialise the GUI interface. See the figure. Various Pannels in the main frame can be seen. Further snapshot shows flow of code. Basics of Java frames and Jpanel etc is sufficient to explore futher GUI component.



energy - NetBeans IDE 7.4

```

28 * Graphical User Interface for simulator
29 */
30 public class DTNSimGUI extends DTNSimUI {
31     private MainWindow main;
32     private PlayField field;
33     private GUIControls guiControls;
34     private EventLogPanel eventLogPanel;
35     private InfoPanel infoPanel;
36
37     private void startGUI() {
38         try {
39             SwingUtilities.invokeAndWait(new Runnable() {
40                 public void run() {
41                     try {
42                         initGUI();
43                     } catch (AssertionError e) {
44                         processAssertionError(e);
45                     }
46                 }
47             });
48         } catch (InterruptedException e) {
49             e.printStackTrace();
50             System.exit(-1);
51         } catch (InvocationTargetException e) {
52             e.printStackTrace();
53             System.exit(-1);
54         }
55     }
56
57     /**
58      * Initializes the GUI
59     */
60     private void initGUI() {
61         this.field = new PlayField(world, this);
62
63         this.field.addMouseListener(new PlayfieldMouseListener());
64         this.field.addMouseWheelListener(new PlayfieldMouseHandler());
65
66         this.guiControls = new GUIControls(this, this.field);
67         this.eventLogPanel = new EventLogPanel(this);
68         this.infoPanel = new InfoPanel(this);
69         this.main = new MainWindow(this.scen.getName(), world, field,
70             guiControls, infoPanel, eventLogPanel, this);
71
72         scen.addMessageListener(eventLogPanel);
73         scen.addConnectionListener(eventLogPanel);
74
75         if (scen.getMap() != null) {
76             field.setMap(scen.getMap());
77         }
78
79         // if user closes the main window, call closeSim()
80         this.main.addWindowListener(new WindowAdapter() {
81             private boolean closeAgain = false;
82             public void windowClosing(WindowEvent e) {
83                 closeSim();
84                 if (closeAgain) {
85                     // if method is called again, force closing
86                     System.err.println("Forced close. " +
87                         "Some reports may have not been finalized.");
88                     System.exit(-1);
89                 }
90                 closeAgain = true;
91             }
92         });
93
94         this.main.setVisible(true);
95     }

```

Find: simCancelled Previous Next Output 3 matches 37 | 9 INS

energy - NetBeans IDE 7.4

```

58     /**
59      * Initializes the GUI
60     */
61     private void initGUI() {
62         this.field = new PlayField(world, this);
63
64         this.field.addMouseListener(new PlayfieldMouseListener());
65         this.field.addMouseWheelListener(new PlayfieldMouseHandler());
66
67         this.guiControls = new GUIControls(this, this.field);
68         this.eventLogPanel = new EventLogPanel(this);
69         this.infoPanel = new InfoPanel(this);
70         this.main = new MainWindow(this.scen.getName(), world, field,
71             guiControls, infoPanel, eventLogPanel, this);
72
73         scen.addMessageListener(eventLogPanel);
74         scen.addConnectionListener(eventLogPanel);
75
76         if (scen.getMap() != null) {
77             field.setMap(scen.getMap());
78         }
79
80         // if user closes the main window, call closeSim()
81         this.main.addWindowListener(new WindowAdapter() {
82             private boolean closeAgain = false;
83             public void windowClosing(WindowEvent e) {
84                 closeSim();
85                 if (closeAgain) {
86                     // if method is called again, force closing
87                     System.err.println("Forced close. " +
88                         "Some reports may have not been finalized.");
89                     System.exit(-1);
90                 }
91                 closeAgain = true;
92             }
93         });
94
95         this.main.setVisible(true);

```

Find: simCancelled Previous Next Output 3 matches 37 | 10 INS

In our case *DTN2Manager.setup(world)* is not required. Now see second task.

In second task GUI components are having the same meaning as their names. While loop will run till endtime time is reached and update functions will call every updateInterval value from config file. Now if pause button is not pressed it enters the else part of *if(guiControls.isPaused())* then *world.update()* method is called. This *world.update* method is of pretty much interest to Our DTN research Community.

Real Game begins from now. In this method Events are called, Hosts are updated, Routers are

updated and Hosts are moved to new incremental location as per mobility model chosen from config. Here connections are also made and broke. Get a cup of coffee now. You need to start this section from fresh look. Here the implementations of most of the research problems lie like one way connections or connection manipulation, Queuing policy, routing design, mobility design etc. Most of the people are interested in such kind of implementations as per the experience. Here we go.

Here we start with the update method of world.

```

energy - NetBeans IDE 7.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
...va masscoupons.php x create.tpl x promotion_coupon.php x DTNSimUI.java x SimScenario.java x DTNSim.java x World.java x
Source History Navigator Projects Files Services
147 /**
148 * Update (move, connect, disconnect etc.) all hosts in the world.
149 * Runs all external events that are due between the time when
150 * this method is called and after one update interval.
151 */
152 public void update () {
153     double runUntil = SimClock.getTime() + this.updateInterval;
154
155     setNextEventQueue();
156
157     /* process all events that are due until next interval update */
158     while (this.nextQueueEventTime <= runUntil) {
159         simClock.setTime(this.nextQueueEventTime);
160         ExternalEvent ee = this.nextEventQueue.nextEvent();
161         ee.processEvent(this);
162         updateHosts(); // update all hosts after every event
163         setNextEventQueue();
164     }
165
166     moveHosts(this.updateInterval);
167     simClock.setTime(runUntil);
168
169     updateHosts();
170
171     /* inform all update listeners */
172     for (UpdateListener ul : this.updateListeners) {
173         ul.updated(this.hosts);
174     }
175 }
176 /**
177 */

```

Here we again divide the code in three as follows:

First:

```

double runUntil = SimClock.getTime() + this.updateInterval;
setNextEventQueue();

```

Second:

```

while (this.nextQueueEventTime <= runUntil) {
    simClock.setTime(this.nextQueueEventTime);
    ExternalEvent ee = this.nextEventQueue.nextEvent();
    ee.processEvent(this);
    updateHosts(); // update all hosts after every event
    setNextEventQueue();
}

```

Third:

```

moveHosts(this.updateInterval);
simClock.setTime(runUntil);
updateHosts();
/* inform all update listeners */
for (UpdateListener ul : this.updateListeners) {
    ul.updated(this.hosts);
}

```

```
}
```

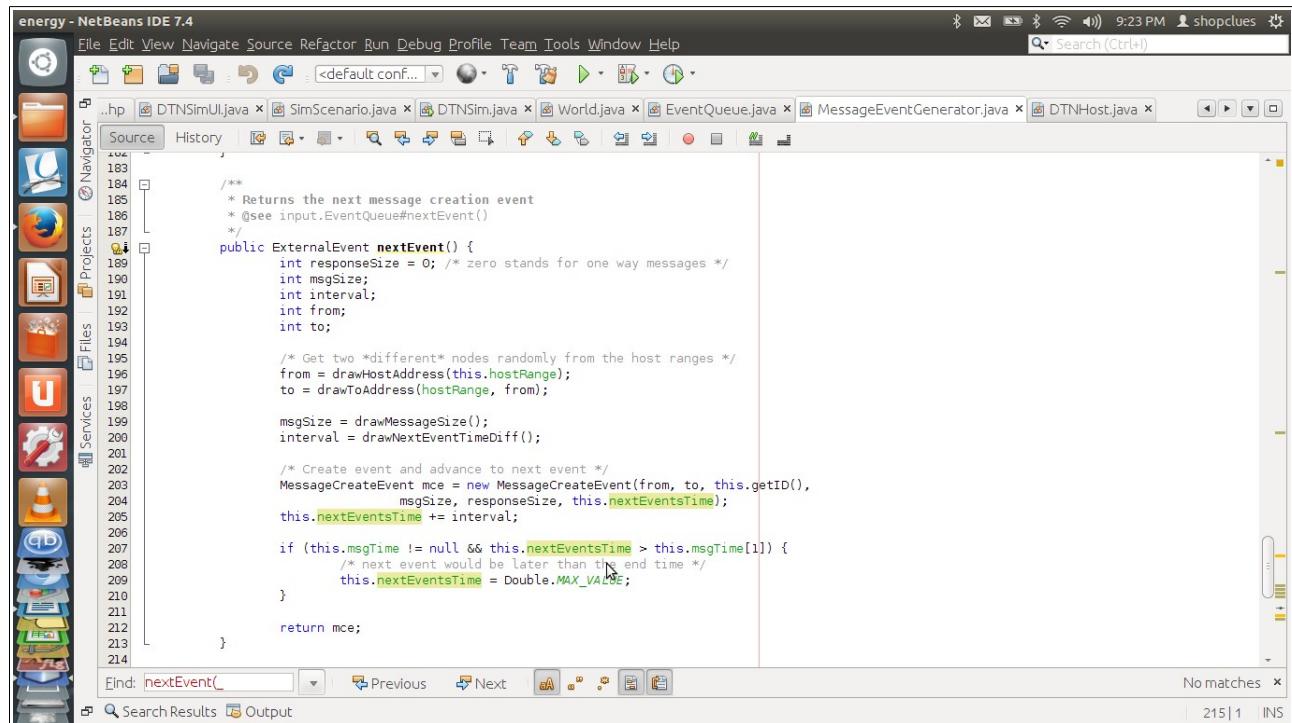
Lets us discuss it one by one.

```
]
    /**
     * Goes through all event Queues and sets the
     * event queue that has the next event.
     */
    public void setNextEventQueue() {
        EventQueue nextQueue = scheduledUpdates;
        double earliest = nextQueue.nextEventsTime();

        /* find the queue that has the next event */
        for (EventQueue eq : eventQueues) {
            if (eq.nextEventsTime() < earliest){
                nextQueue = eq;
                earliest = eq.nextEventsTime();
            }
        }

        this.nextEventQueue = nextQueue;
        this.nextQueueEventTime = earliest;
    }
```

Here it set next queue that has next event to be executed. In our case it will be only MessageEventGenerator class. Here nextEventsTime method will return random time between Events1.interval = 25,35 value in config file.



In second task, Simulator clock sets time upto *nextQueueEventTime* value.
Here, *this.nextEventQueue* is MessageEventGenerator. The nextEvent Method of

MessageEventGenerator is shown above.

Discussion on nextEvent Method of MessageEventGenerator:

Parameters like *responseSize*, *msgSize*, *interval*, *from* and *to* obtained their value. Now this line
*MessageCreateEvent mce = new MessageCreateEvent(from, to, this.getID(),
msgSize, responseSize, this.nextEventsTime);*

creates the MessageCreateEvent object which is required in world.update method for process events.

```
public MessageCreateEvent(int from, int to, String id, int size,  
                           int responseSize, double time) {  
    super(from,to, id, time);  
    this.size = size;  
    this.responseSize = responseSize;  
}
```

Now back to update method. Line *ee.processEvent(this)* calls the processEvent Method of MessageCreateEvent which creates Message and put it in the source message buffer.

```
/**  
 * Creates the message this event represents.  
 */  
@Override  
public void processEvent(World world) {  
    DTNHost to = world.getNodeByAddress(this.toAddr);  
    DTNHost from = world.getNodeByAddress(this.fromAddr);  
  
    Message m = new Message(from, to, this.id, this.size);  
    m.setResponseSize(this.responseSize);  
    from.createNewMessage(m);  
}
```

Back to world.update Method. It is time to look at updateHosts Method of world class.

```

177     /**
178      * Updates all hosts (calls update for every one of them). If update
179      * order randomizing is on (updateOrder array is defined), the calls
180      * are made in random order.
181      */
182     private void updateHosts() {
183         if (this.updateOrder == null) { // randomizing is off
184             for (int i=0, n = hosts.size();i < n; i++) {
185                 if (this.isCancelled) {
186                     break;
187                 }
188                 hosts.get(i).update(simulateConnections);
189             }
190         } else { // update order randomizing is on
191             assert this.updateOrder.size() == this.hosts.size() :
192                 "Nrof hosts has changed unexpectedly";
193             Random rng = new Random(SimClock.getIntTime());
194             Collections.shuffle(this.updateOrder, rng);
195             for (int i=0, n = hosts.size();i < n; i++) {
196                 if (this.isCancelled) {
197                     break;
198                 }
199                 this.updateOrder.get(i).update(simulateConnections);
200             }
201         }
202         if (simulateConOnce && simulateConnections) {
203             simulateConnections = false;
204         }
205     }
206 }
207

```

In our case control reaches to red line. So its obvious it is randomise update in nature. There update method of each node(DTNHost) in scenario is called. Let us see now update method of DTNHost.

```

328     /**
329      * Updates node's network layer and router.
330      * @param simulateConnections Should network layer be updated too
331      */
332     public void update(boolean simulateConnections) {
333         if (!isRadioActive()) {
334             // Make sure inactive nodes don't have connections
335             tearDownAllConnections();
336             return;
337         }
338
339         if (simulateConnections) {
340             for (NetworkInterface i : net) {
341                 i.update();
342             }
343         }
344         this.router.update();
345     }
346

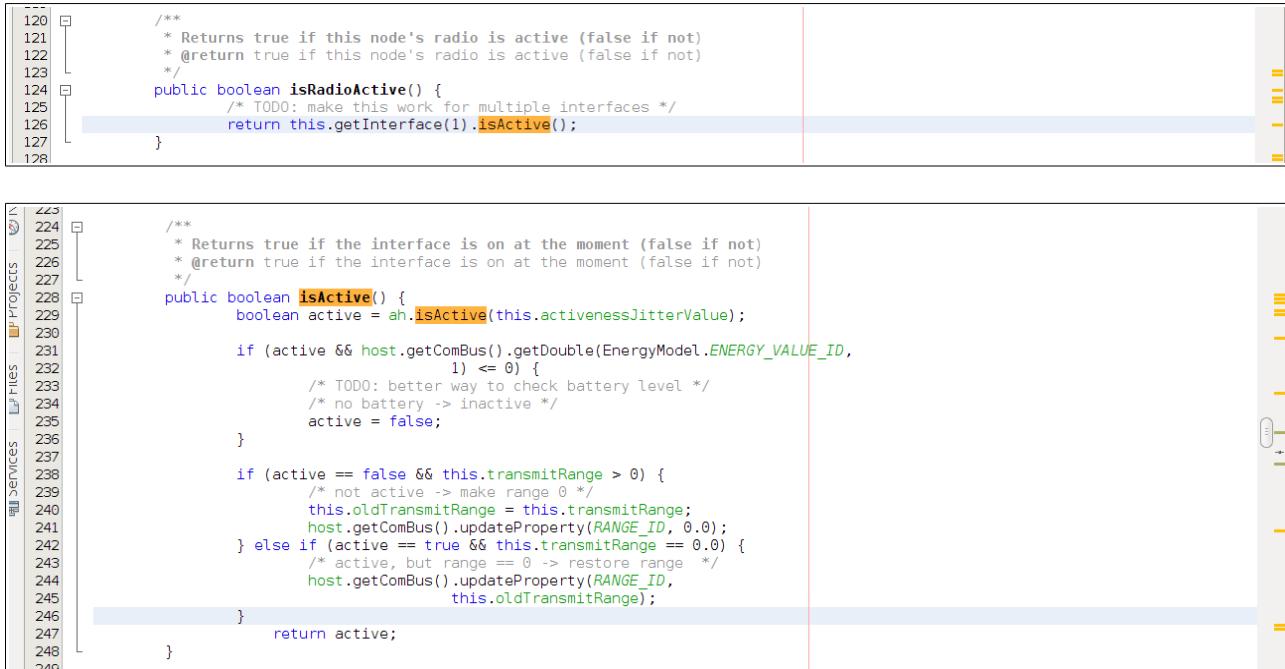
```

Meanwhile in DTNHost class.

In its construction at initialisation phase values of MessageListeners are MessageStatsReport and EventLogPannel. Router is prophet and Network Interface is SimpleBroadcastInterface.

comBus	ModuleCommunicationBus	#2116
destination		null
energy_threshold_counter	int	0
location	Coord	#2110
movListeners	ArrayList	"size = 0"
movement	RandomWaypoint	#2112
msgListeners	ArrayList	"size = 2"
[0]	MessageStatsReport	#2139
[1]	EventLogPanel	#2140
name	String	"D55"
net	ArrayList	"size = 1"
[0]	SimpleBroadcastInterface	#2137
nextTimeToMove	double	38.72030292546405
path		null
router	ProphetRouter	#2111
speed	double	0.0
total energy charged	double	0.0

Back to DTNHost.update Method. *IsRadioActive* method first checks interface at first position of interface list. This returns true or false depending upon whether node is active or not. It is also decide based on reason like config value, Current Energy value and transmission range is 0 or not. See Pics for clarity.



```

120 /**
121 * Returns true if this node's radio is active (false if not)
122 * @return true if this node's radio is active (false if not)
123 */
124 public boolean isRadioActive() {
125     /* TODO: make this work for multiple interfaces */
126     return this.getInterface(1).isActive();
127 }

```

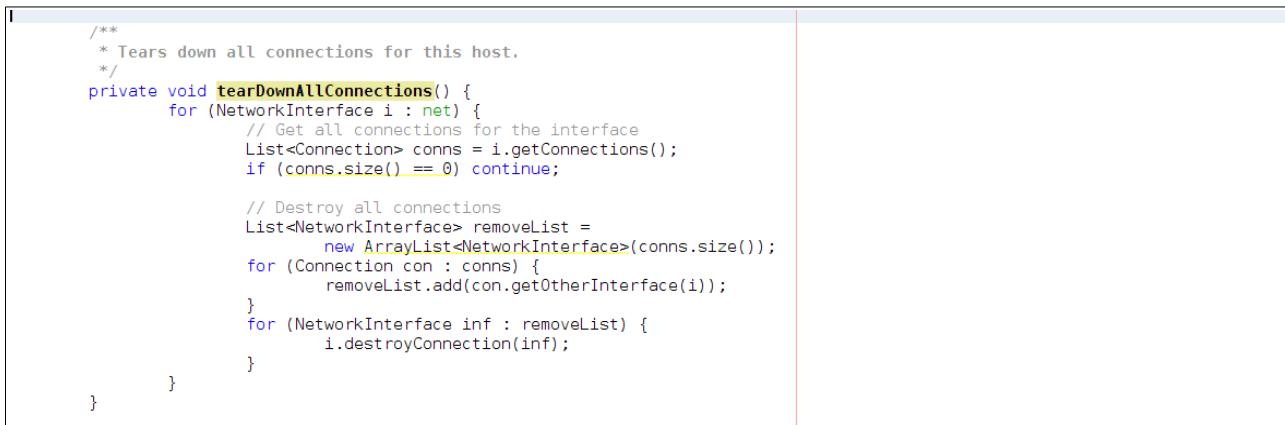


```

223 /**
224 * Returns true if the interface is on at the moment (false if not)
225 * @return true if the interface is on at the moment (false if not)
226 */
227 public boolean isActive() {
228     boolean active = ah.isActive(this.activenessJitterValue);
229
230     if (active && host.getComBus().getDouble(EnergyModel.ENERGY_VALUE_ID,
231         1) <= 0) {
232         /* TODO: better way to check battery level */
233         /* no battery -> inactive */
234         active = false;
235     }
236
237     if (active == false && this.transmitRange > 0) {
238         /* not active -> make range 0 */
239         this.oldTransmitRange = this.transmitRange;
240         host.getComBus().updateProperty(RANGE_ID, 0.0);
241     } else if (active == true && this.transmitRange == 0.0) {
242         /* active, but range == 0 -> restore range */
243         host.getComBus().updateProperty(RANGE_ID,
244             this.oldTransmitRange);
245     }
246 }
247
248 return active;
249

```

In our case it is true. So it will not enter here. The case in which it enters in this section of code *tearDownAllConnections* method will remove all connection associated with it. See snapshot of it.



```

/**
 * Tears down all connections for this host.
 */
private void tearDownAllConnections() {
    for (NetworkInterface i : net) {
        // Get all connections for the interface
        List<Connection> conns = i.getConnections();
        if (conns.size() == 0) continue;

        // Destroy all connections
        List<NetworkInterface> removeList =
            new ArrayList<NetworkInterface>(conns.size());
        for (Connection con : conns) {
            removeList.add(con.getOtherInterface(i));
        }
        for (NetworkInterface inf : removeList) {
            i.destroyConnection(inf);
        }
    }
}

```

According to our case study. We reach at the for loop of *NetworkInterface* iteration. Here its update method is called. This will call the update method of *SimpleBroadcastInterface*.

Before proceeding further it is the right time to discuss the working of DTNHost construction from *createHost* method of *Simsenario* that we left at initialisation phase and some files associated in its working.

```

public DTNHost(List<MessageListener> msgLs,
    List<MovementListener> movLs,
    String groupId, List<NetworkInterface> interf,
    ModuleCommunicationBus comBus,
    MovementModel mmProto, MessageRouter mRouterProto) {
    this.comBus = comBus;
    this.location = new Coord(0,0);
    this.address = getNextAddress();
    this.name = groupId+address;
    this.net = new ArrayList<NetworkInterface>();

    for (NetworkInterface i : interf) {
        NetworkInterface ni = i.replicate();
        ni.setHost(this);
        net.add(ni);
    }

    // TODO - think about the names of the interfaces and the nodes

    this.msgListeners = msgLs;
    this.movListeners = movLs;

    // create instances by replicating the prototypes
    this.movement = mmProto.replicate();
    this.movement.setComBus(comBus);
    this.movement.setHost(this);
    setRouter(mRouterProto.replicate());

    this.location = movement.getInitialLocation();

    this.nextTimeToMove = movement.nextPathAvailable();
    this.path = null;

    if (movLs != null) { // inform movement listeners about the location
        for (MovementListener l : movLs) {
            l.initialLocation(this, this.location);
        }
    }
}

```

Every part of code is self explanatory but there is something more interesting going on in the highlighted parts.

```

for (NetworkInterface i : interf) {
    NetworkInterface ni = i.replicate();
    ni.setHost(this);
    net.add(ni);
}

```

Here ni.setHost method assosiates interface of DTNHost with cell of grid which is based on it transmission range. So if it has n interfaces, then it has ten different topologies of grids in which it is associated. Now let us step by step see what is going on.

```

135 /**
136 * For setting the host - needed when a prototype is copied for several
137 * hosts
138 * @param host The host where the network interface is
139 */
140 public void setHost(DTNHost host) {
141     this.host = host;
142     ModuleCommunicationBus comBus = host.getComBus();
143
144     if (!comBus.containsProperty(SCAN_INTERVAL_ID)) {
145         /* add properties and subscriptions only for the 1st interface */
146         /* TODO: support for multiple interfaces */
147         comBus.addProperty(SCAN_INTERVAL_ID, this.scanInterval);
148         comBus.addProperty(RANGE_ID, this.transmitRange);
149         comBus.addProperty(SPEED_ID, this.transmitSpeed);
150         comBus.subscribe(SCAN_INTERVAL_ID, this);
151         comBus.subscribe(RANGE_ID, this);
152         comBus.subscribe(SPEED_ID, this);
153     }
154
155     if (transmitRange > 0) {
156         optimizer = ConnectivityGrid.ConnectivityGridFactory(
157             this.interfacetype.hashCode(), transmitRange);
158         optimizer.addInterface(this);
159     } else {
160         optimizer = null;
161     }
162 }
163
164 /**
165 */

```

Here First it registers basic properties to communication bus of its DTNHost class. Then optimizer

makes grids based on transmission range and cellSizeMultiplier value.

```

116     * Returns a connectivity grid object based on a hash value
117     * @param key A hash value that separates different interfaces from each other
118     * @param maxRange Maximum range used by the radio technology using this
119     * connectivity grid.
120     * @return The connectivity grid object for a specific interface
121     */
122    public static ConnectivityGrid ConnectivityGridFactory(int key,
123        double maxRange) {
124        if (gridobjects.containsKey((Integer)key)) {
125            return (ConnectivityGrid)gridobjects.get((Integer)key);
126        } else {
127            ConnectivityGrid newgrid =
128                new ConnectivityGrid((int)Math.ceil(maxRange *
129                    cellSizeMultiplier));
130            gridobjects.put((Integer)key,newgrid);
131        }
132    }
133
134    /**
135     * Adds a network interface to the overlay grid
136

```

Variables table:

Name	Type	Value
<code><Enter new watch></code>		
<code>\$_db_field</code>		<code>>"\$_db_field" is not a known variable in the current context.</code>
<code>Static</code>		
<code>cellSizeMultiplier</code>	int	5
<code>guiUpdateInterval</code>		<code>>"guiUpdateInterval" is not a known variable in the current context.</code>
<code>i</code>		<code>>"i" is not a known variable in the current context.</code>
<code>key</code>	int	1178287367

Lets see what is really happening behind the scence.

Here it makes ConnectivityGrid new object which futher makes rows and column and so the cells array is filled with the new GridCell inner class object. GridCell is the inner class to ConnectivityGrid. *ginterfaces* is intialised.

```

95
96     /**
97      * Creates a new overlay connectivity grid
98      * @param cellSize Cells edge's length (must be larger than the largest
99      * radio coverage's diameter)
100
101     private ConnectivityGrid(int cellSize) {
102         this.rows = worldsizeY/cellSize + 1;
103         this.cols = worldsizeX/cellSize + 1;
104         // leave empty cells on both sides to make neighbor search easier
105         this.cells = new GridCell[rows+2][cols+2];
106         this.cellSize = cellSize;
107
108         for (int i=0; i<rows+2; i++) {
109             for (int j=0; j<cols+2; j++) {
110                 this.cells[i][j] = new GridCell();
111             }
112         }
113         ginterfaces = new HashMap<NetworkInterface,GridCell>();
114     }
115

```

Variables table:

Name	Type	Value
<code><Enter new watch></code>		
<code>cellSize</code>	int	50
<code>cellSize</code>	int	50
<code>cells</code>	ConnectivityGrid\$GridCell[]	<code>#225(length=71)</code>
<code>ginterfaces</code>	HashMap	<code>size = 0</code>
<code>this</code>	ConnectivityGrid	<code>#223</code>

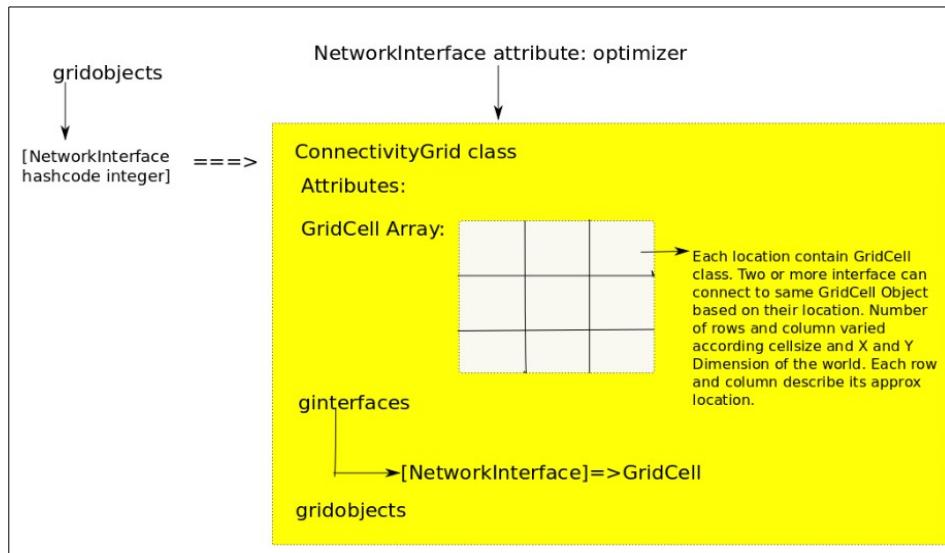
`gridobjects.put((Integer)key,newgrid)` registers this grid with above mention topology with hashcode of interface in `gridobjects`. This attribute is static and so the method in which all things are happening. So it is a global variable by nature and available to all.

Comming back to NetworkInterface.setHost method next line:
`optimizer.addInterface(this);`

addinterface associates cell with interface and interface with cell.

```
134
135     /**
136      * Adds a network interface to the overlay grid
137      * @param ni The new network interface
138      */
139
140     public void addInterface(NetworkInterface ni) {
141         GridCell c = cellFromCoord(ni.getLocation());
142         c.addInterface(ni);
143         ginterfaces.put(ni,c);
144 }
```

Discussion on ConnectivityGrid class and its purpose:



This is the approximate model that describes the functionality of ConnectivityGrid Class. Here static attribute gridobject associates each ConnectivityGrid to NetworkInterface of Host. This is checked during initialization phase for getting ConnectivityGrid class for particular interface. If not present then one is created with cell array initialised and put into gridobject. Now according to the location of the node and so the interface, the proper cell from cells array is taken.

The interface is added in the inner class object of GridCell and GridCell object joins the interface in ginterface object. This ConnectivityGrid is also taken in attribute of NetworkInterface Class in the name of optimizer. This also explains why two different interfaces can't communicate even if they are provided the same connection parameters(transmission speed, range etc). Since getNeighborCellsByCoord method will only return interfaces of nodes having same interface class like SimpleBroadcastInterface.

This information will be helpful when we come back to the update method of SimpleBroadcastInterface update method. Understanding of making and breaking connection will become much clearer.

Moving back to update method of DTNhost from where this all intermediate story started.

```

328
329     /**
330      * Updates node's network layer and router.
331      * @param simulateConnections Should network layer be updated too
332      */
333     public void update(boolean simulateConnections) {
334         if (!isRadioActive()) {
335             // Make sure inactive nodes don't have connections
336             tearDownAllConnections();
337             return;
338         }
339
340         if (simulateConnections) {
341             for (NetworkInterface i : net) {
342                 i.update();
343             }
344         }
345     }
346 
```

Now its easy to analyse what this update method is doing.

The screenshot shows the NetBeans IDE interface with the title bar "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The left sidebar has icons for Projects, Navigator, Files, and Services. The main editor window displays the Java code for the update() method:

```

64     * Updates the state of current connections (i.e. tears down connections
65     * that are out of range and creates new ones).
66     */
67     public void update() {
68         if (optimizer == null) {
69             return; /* nothing to do */
70         }
71
72         // First break the old ones
73         optimizer.updateLocation(this);
74         for (int i=0; i<this.connections.size(); ) {
75             Connection con = this.connections.get(i);
76             NetworkInterface anotherInterface = con.getOtherInterface(this);
77
78             // all connections should be up at this stage
79             assert con.isUp() : "Connection " + con + " was down!";
80
81             if (!isWithinRange(anotherInterface)) {
82                 disconnect(con,anotherInterface);
83                 connections.remove(i);
84             }
85             else {
86                 i++;
87             }
88
89         } // Then find new possible connections
90         collection<NetworkInterface> interfaces =
91         optimizer.getNearInterfaces(this);
92         for (NetworkInterface i : interfaces) {
93             connect(i);
94         }
95     }

```

The code is annotated with comments explaining its purpose. The search bar at the top right shows "Search (Ctrl+F)" and the status bar indicates "2:21PM shopclues".

Let analyse code in three parts.

First:

`optimizer.updateLocation(this);`

Second:

```

for (int i=0; i<this.connections.size(); ) {
    Connection con = this.connections.get(i);
    NetworkInterface anotherInterface = con.getOtherInterface(this);

    // all connections should be up at this stage
    assert con.isUp() : "Connection " + con + " was down!";

    if (!isWithinRange(anotherInterface)) {
        disconnect(con,anotherInterface);
        connections.remove(i);
    }
}

```

```

        }
    else {
        i++;
    }
}

```

Third:

```

// Then find new possible connections
Collection<NetworkInterface> interfaces =
    optimizer.getNearInterfaces(this);
for (NetworkInterface i : interfaces) {
    connect(i);
}

```

First section just update the cell it must belong to based on its current location.

```

166
167     /**
168      * Checks and updates (if necessary) interface's position in the grid
169      * @param ni The interface to update
170     */
171     public void updateLocation(NetworkInterface ni) {
172         GridCell oldCell = (GridCell)ginterfaces.get(ni);
173         GridCell newCell = cellFromCoord(ni.getLocation());
174
175         if (newCell != oldCell) {
176             oldCell.moveInterface(ni, newCell);
177             ginterfaces.put(ni,newCell);
178         }
179     }
180

```

```

207
208     /**
209      * Get the cell having the specific coordinates
210      * @param c Coordinates
211      * @return The cell
212     */
213     private GridCell cellFromCoord(Coord c) {
214         // +1 due empty cells on both sides of the matrix
215         int row = (int)(c.getY()/cellSize) + 1;
216         int col = (int)(c.getX()/cellSize) + 1;
217
218         assert row > 0 && row <= rows && col > 0 && col <= cols : "Location " +
219         c + " is out of world's bounds";
220
221         return this.cells[row][col];
222     }
223

```

Now second section is easy. It removes those connections which are not in range. Here some interesting thing is going in disconnect method and same in connect of third section.

```

340
341     /**
342      * Returns true if another interface is within radio range of this interface
343      * and this interface is also within radio range of the another interface.
344      * @param anotherInterface The another interface
345      * @return True if the interface is within range, false if not
346     */
347     protected boolean isWithinRange(NetworkInterface anotherInterface) {
348         double smallerRange = anotherInterface.getTransmitRange();
349         double myRange = getTransmitRange();
350         if (myRange < smallerRange) {
351             smallerRange = myRange;
352         }
353
354         return this.host.getLocation().distance(
355             anotherInterface.getHost().getLocation()) <= smallerRange;
356     }
357

```

```

320
321     /**
322      * Disconnects this host from another host. The derived class should
323      * make the decision whether to disconnect or not
324      * @param con The connection to tear down
325      */
326     protected void disconnect(Connection con,
327         NetworkInterface anotherInterface) {
328         con.setUpState(false);
329         notifyConnectionListeners(CON_DOWN, anotherInterface.getHost());
330
331         // tear down bidirectional connection
332         if (!anotherInterface.getConnections().remove(con)) {
333             throw new SimError("No connection " + con + " found in " +
334             anotherInterface);
335
336         }
337
338         this.host.connectionDown(con);
339         anotherInterface.getHost().connectionDown(con);
340     }

```

Here connectionDown method of DTNHost is called. Let us see what happened there.

```

    /**
     * Informs the router of this host about state change in a connection
     * object.
     * @param con The connection object whose state changed
     */
    public void connectionUp(Connection con) {
        this.router.changedConnection(con);
    }

    public void connectionDown(Connection con) {
        this.router.changedConnection(con);
    }

```

Important point is `this.router.changedConnection(con)` is called from here. ProphetRouter uses this function to update its delivery probability value. Also `con.setUpState(false)` is a function to make `this.isUp = false` which is again used in Prophet Router.

Now Third section First it gets near interface. Near interface are those who lie in +- 1 row and column in cells array.

```

    /**
     * Returns all interfaces that are "near" (i.e., in neighboring grid cells)
     * and use the same technology and channel as the given interface
     * @param ni The interface whose neighboring interfaces are returned
     * @return List of near interfaces
     */
    public Collection<NetworkInterface> getNearInterfaces(
        NetworkInterface ni) {
        ArrayList<NetworkInterface> niList = new ArrayList<NetworkInterface>();
        GridCell loc = (GridCell)ginterfaces.get(ni);

        if (loc != null) {
            GridCell[] neighbors =
                getNeighborCellsByCoord(ni.getLocation());
            for (int i=0; i < neighbors.length; i++) {
                niList.addAll(neighbors[i].getInterfaces());
            }
        }
        return niList;
    }

```

```

    /**
     * Finds all neighboring cells and the cell itself based on the coordinates
     * @param c The coordinates
     * @return Array of neighboring cells
     */
    private GridCell[] getNeighborCellsByCoord(Coord c) {
        // +1 due empty cells on both sides of the matrix
        int row = (int)(c.getY()/cellSize) + 1;
        int col = (int)(c.getX()/cellSize) + 1;
        return getNeighborCells(row,col);
    }

```

```

92
93     /**
94      * Returns an array of Cells that contains the neighbors of a certain
95      * cell and the cell itself.
96      * @param row Row index of the cell
97      * @param col Column index of the cell
98      * @return Array of neighboring Cells
99     */
00     private GridCell[] getNeighborCells(int row, int col) {
01         return new GridCell[] {
02             cells[row-1][col-1],cells[row-1][col],cells[row-1][col+1],//1st row
03             cells[row][col-1],cells[row][col],cells[row][col+1],//2nd row
04             cells[row+1][col-1],cells[row+1][col],cells[row+1][col+1]]//3rd row
05         };
06     }
07

```

Now back to update Method of SimpleBroadcastInterface. Here connect Method checks the returned nearby interfaces which are in range or not. Then make new Connection.

Connection con = new CBRConnection(this.host, this.anotherInterface.getHost(), anotherInterface, conSpeed);

Then connect Method is called which is again *this.host.connectionUp(con)*.

```

301
302     public void connect(NetworkInterface anotherInterface) {
303         if (isScanning())
304             && anotherInterface.getHost().isRadioActive()
305             && isWithinRange(anotherInterface)
306             && !isConnected(anotherInterface)
307             && (this != anotherInterface)) {
308             // new contact within range
309             // connection speed is the lower one of the two speeds
310             int conSpeed = anotherInterface.getTransmitSpeed();
311             if (conSpeed > this.transmitSpeed) {
312                 conSpeed = this.transmitSpeed;
313             }
314
315             Connection con = new CBRConnection(this.host, this,
316                     anotherInterface.getHost(), anotherInterface, conSpeed);
317             connect(con,anotherInterface);
318         }
319     }
320

```

```

321
322     /**
323      * Connects this host to another host. The derived class should check
324      * that all pre-requisites for making a connection are satisfied before
325      * actually connecting.
326      * @param con The new connection object
327      * @param anotherInterface The interface to connect to
328     */
329     protected void connect(Connection con, NetworkInterface anotherInterface) {
330         this.connections.add(con);
331         notifyConnectionListeners(CON_UP, anotherInterface.getHost());
332
333         // set up bidirectional connection
334         anotherInterface.getConnections().add(con);
335
336         // inform routers about the connection
337         this.host.connectionUp(con);
338         anotherInterface.getHost().connectionUp(con);
339     }
340

```

Here Note that Bidirectional Connections are made. So if any body wants to have one way connection, they can dissable or comment any *add* method based on their conditions.

```

328
329     /**
330      * Updates node's network layer and router.
331      * @param simulateConnections Should network layer be updated too
332      */
333     @Override
334     public void update(boolean simulateConnections) {
335         if (!isRadioActive()) {
336             // Make sure inactive nodes don't have connections
337             tearDownAllConnections();
338             return;
339         }
340         if (simulateConnections) {
341             for (NetworkInterface i : net) {
342                 i.update();
343             }
344         }
345     }
346 }
```

Now back to `this.router.update()` method. It reaches the to ProphetRouter update Method in our case study scenario.

```

@Override
public void update() {
    super.update();
    if (!canStartTransfer() || isTransferring()) {
        return; // nothing to transfer or is currently transferring
    }
    // try messages that could be delivered to final recipient
    if (exchangeDeliverableMessages() != null) {
        return;
    }
    tryOtherMessages();
}
```

Here we observe Five calls:

1. `super.update()`
2. `canStartTransfer()`
3. `isTransferring()`
4. `exchangeDeliverableMessages()`
5. `tryOtherMessages()`

Lets us discuss all activities of routing one by one.

```

@Override
public void update() {
    super.update();

    /* in theory we can have multiple sending connections even though
     * currently all routers allow only one concurrent sending connection */
    for (int i=0; i<this.sendingConnections.size(); ) {
        boolean removeCurrent = false;
        Connection con = sendingConnections.get(i);

        /* finalize ready transfers */
        if (con.isMessageTransferred()) {
            if (con.getMessage() != null) {
                transferDone(con);
                con.finalizeTransfer();
            } /* else: some other entity aborted transfer */
            removeCurrent = true;
        }

        /* remove connections that have gone down */
        else if (!con.isOpen()) {
            if (con.getMessage() != null) {
                transferAborted(con);
                con.abortTransfer();
            }
            removeCurrent = true;
        }

        if (removeCurrent) {
            // if the message being sent was holding excess buffer, free it
            if (this.getFreeBufferSize() < 0) {
                this.makeRoomForMessage(0);
            }
            sendingConnections.remove(i);
        }
        else {
            /* index increase needed only if nothing was removed */
            i++;
        }
    }

    /* time to do a TTL check and drop old messages? Only if not sending */
    if (SimClock.getTime() - lastTtlCheck >= ttlCheckInterval &&
        sendingConnections.size() == 0) {

```

Find: tryOtherMessages() Previous Next

2 matches | 586 | 9 | INS

Search Results Output

```

    /**
     * Updates router.
     * This method should be called (at least once) on every simulation
     * interval to update the status of transfer(s).
     */
    public void update(){
        for (Collection<Application> apps : this.applications.values()) {
            for (Application app : apps) {
                app.update(this.host);
            }
        }
    }
}

```

First it enters the update of ActiveRouter class. `super.update()` send control to MessageRouter update Method which updates application layer. In our case study it is not used.

Comming back to ActiveRouter.update Method. Let us see for loop iteration.

SendingConnection contains the list of connection which has started transferring the Message.

```

    /**
     * Returns true if the current message transfer is done.
     * @return True if the transfer is done, false if not
     */
    public boolean isMessageTransferred() {
        return getRemainingByteCount() == 0;
    }
}

```

```

    /**
     * Returns the amount of bytes to be transferred before ongoing transfer
     * is ready or 0 if there's no ongoing transfer or it has finished
     * already
     * @return the amount of bytes to be transferred
     */
    public int getRemainingByteCount() {
        int remaining;

        if (msgOnFly == null) {
            return 0;
        }

        remaining = (int)((this.transferDoneTime - SimClock.getTime())
                         * this.speed);

        return (remaining > 0 ? remaining : 0);
    }

```

This method checks whether message has been transferred or not. `this.transferDoneTime` is calculated as per the message size which has been send through this connection. Next `con.getMessage()` returns `msgOnFly` which tells that message is finalised or not. All these methods will going to be discussed in the Routing part discussion.

```

    /**
     * Gets the message that this connection is currently transferring.
     * @return The message or null if no message is being transferred
     */
    public Message getMessage() {
        return this.msgOnFly;
    }

```

Then `transferDone(con)` method is doing nothing here.

```

    public void finalizeTransfer() {
        assert this.msgOnFly != null : "Nothing to finalize in " + this;
        assert msgFromNode != null : "msgFromNode is not set";

        this.bytesTransferred += msgOnFly.getSize();

        getOtherNode(msgFromNode).messageTransferred(this.msgOnFly.getId(),
                                                      msgFromNode);
        clearMsgOnFly();
    }

```

The `con.abortTransfer()`

```

    /**
     * Clears the message that is currently being transferred.
     * Calls to {@link #getMessage()} will return null after this.
     */
    protected void clearMsgOnFly() {
        this.msgOnFly = null;
        this.msgFromNode = null;
    }

```

The `con.isUp()` is used which is controlled in `NetworkInterface update` method.

```

    public void abortTransfer() {
        assert msgOnFly != null : "No message to abort at " + msgFromNode;
        int bytesRemaining = getRemainingByteCount();

        this.bytesTransferred += msgOnFly.getSize() - bytesRemaining;

        getOtherNode(msgFromNode).messageAborted(this.msgOnFly.getId(),
                                                msgFromNode, bytesRemaining);
        clearMsgOnFly();
    }
}

```

```

482
483     /**
484      * Informs the host that a message transfer was aborted.
485      * @param id Identifier of the message
486      * @param from From who the message was from
487      * @param bytesRemaining Not bytes that were left before the transfer
488      * would have been ready; or -1 if the number of bytes is not known
489      */
490     @Override
491     public void messageAborted(String id, DTNHost from, int bytesRemaining) {
492         this.router.messageAborted(id, from, bytesRemaining);
493     }
494

```

```

458
459     /**
460      * This method should be called (on the receiving host) when a message
461      * transfer was aborted.
462      * @param id Id of the message that was being transferred
463      * @param from Host the message was from (previous hop)
464      * @param bytesRemaining Not bytes that were left before the transfer
465      * would have been ready; or -1 if the number of bytes is not known
466      */
467     public void messageAborted(String id, DTNHost from, int bytesRemaining) {
468         Message incoming = removeFromIncomingBuffer(id, from);
469         if (incoming == null) {
470             throw new SimError("No incoming message for id " + id +
471                               " to abort in " + this.host);
472         }
473
474         for (MessageListener ml : this.mListeners) {
475             ml.messageTransferAborted(incoming, from, this.host);
476         }
477     }
478

```

In rest part, if removeCurrent is true then it makes space in message buffer if required and removes sendingconnections. Otherwise increment i which means move to next connection to check. If TTL is expired and it is not transferring the message then messages are dropped whose ttl is expired.

```

295
296     /**
297      * Drops messages whose TTL is less than zero.
298      */
299     protected void dropExpiredMessages() {
300         Message[] messages = getMessageCollection().toArray(new Message[0]);
301         for (int i=0; i<messages.length; i++) {
302             int ttl = messages[i].getTtl();
303             if (ttl <= 0) {
304                 deleteMessage(messages[i].getId(), true);
305             }
306         }
307     }
308

```

```

490
491     /**
492      * Deletes a message from the buffer and informs message listeners
493      * about the event
494      * @param id Identifier of the message to delete
495      * @param drop If the message is dropped (e.g. because of full buffer) this
496      * should be set to true. False value indicates e.g. remove of message
497      * because it was delivered to final destination.
498      */
499     public void deleteMessage(String id, boolean drop) {
500         Message removed = removeFromMessages(id);
501         if (removed == null) throw new SimError("no message for id " +
502                                         id + " to remove at " + this.host);
503
504         for (MessageListener ml : this.mListeners) {
505             ml.messageDeleted(removed, this.host, drop);
506         }
507     }
508

```

Also at last if energy is defined it updates its value also.

```
632
633     if (energy != null) {
634         /* TODO: add support for other interfaces */
635         NetworkInterface iface = getHost().getInterface(1);
636         energy.update(iface, getHost().getComBus());
637     }
```

Now coming back to *update* Method of Prophet.

```
207
208     /**
209      * Makes rudimentary checks (that we have at least one message and one
210      * connection) about can this router start transfer.
211      * @return True if router can start transfer, false if not
212      */
213     protected boolean canStartTransfer() {
214         if (this.getNrofMessages() == 0) {
215             return false;
216         }
217         if (this.getConnections().size() == 0) {
218             return false;
219         }
220
221         return true;
222     }
```

It basically checks either number of messages in message buffer is not zero or connection list must not be empty.

```
525
526     /**
527      * Returns true if this router is transferring something at the moment or
528      * some transfer has not been finalized.
529      * @return true if this router is transferring something
530      */
531     public boolean isTransferring() {
532         if (this.sendingConnections.size() > 0) {
533             return true; // sending something
534         }
535
536         List<Connection> connections = getConnections();
537
538         if (connections.size() == 0) {
539             return false; // not connected
540         }
541
542         for (int i=0, n=connections.size(); i<n; i++) {
543             Connection con = connections.get(i);
544             if (!con.isReadyForTransfer()) {
545                 return true; // a connection isn't ready for new transfer
546             }
547         }
548
549         return false;
550     }
```

```
    /**
     * Returns true if the connection is ready to transfer a message (connection
     * is up and there is no message being transferred).
     * @return true if the connection is ready to transfer a message
     */
    public boolean isReadyForTransfer() {
        return this.isUp && this.msgOnFly == null;
    }
```

IsTransferring() method checks whether connection are made or not.

```

468
469     /**
470      * Exchanges deliverable (to final recipient) messages between this host
471      * and all hosts this host is currently connected to. First all messages
472      * from this host are checked and then all other hosts are asked for
473      * messages to this host. If a transfer is started, the search ends.
474      * @return A connection that started a transfer or null if no transfer
475      * was started
476      */
477     protected Connection exchangeDeliverableMessages() {
478         List<Connection> connections = getConnections();
479
480         if (connections.size() == 0) {
481             return null;
482         }
483
484         @SuppressWarnings(value = "unchecked")
485         Tuple<Message, Connection> t =
486             tryMessagesForConnected(sortByQueueMode(getMessagesForConnected()));
487
488         if (t != null) {
489             return t.getValue(); // started transfer
490         }
491
492         // didn't start transfer to any node -> ask messages from connected
493         for (Connection con : connections) {
494             if (con.getOtherNode(getHost()).requestDeliverableMessages(con)) {
495                 return con;
496             }
497         }
498
499         return null;
500     }
501

```

```

351     /**
352      * Returns a list of message-connections tuples of the messages whose
353      * recipient is some host that we're connected to at the moment.
354      * @return a list of message-connections tuples
355      */
356     protected List<Tuple<Message, Connection>> getMessagesForConnected() {
357         if (getNrofMessages() == 0 || getConnections().size() == 0) {
358             /* no messages -> empty list */
359             return new ArrayList<Tuple<Message, Connection>>(0);
360         }
361
362         List<Tuple<Message, Connection>> forTuples =
363             new ArrayList<Tuple<Message, Connection>>();
364         for (Message m : getMessageCollection()) {
365             for (Connection con : getConnections()) {
366                 DTNHost to = con.getOtherNode(getHost());
367                 if (m.getTo() == to) {
368                     forTuples.add(new Tuple<Message, Connection>(m, con));
369                 }
370             }
371         }
372     }
373
374     return forTuples;
375

```

redundant type arguments in new expression (use diamond operator)
 ...
 (Alt-Enter shows hints)

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+F)

Source History

```
516     @SuppressWarnings("unchecked") /* ugly way to make this generic */
517     protected List sortByQueueMode(List list) {
518         switch (sendQueueMode) {
519             case Q_MODE_PDR:
520                 break;
521             case Q_MODE_FIFO:
522                 Collections.shuffle(list, new Random(SimClock.getIntTime()));
523                 Collections.sort(list,
524                     new Comparator() {
525                         /** Compares tuples by their messages, receiving time */
526                         public int compare(Object o1, Object o2) {
527                             double diff;
528                             Message m1, m2;
529
530                             if (o1 instanceof Tuple) {
531                                 m1 = ((Tuple<Message, Connection>)o1).getKey();
532                                 m2 = ((Tuple<Message, Connection>)o2).getKey();
533                             }
534                             else if (o1 instanceof Message) {
535                                 m1 = (Message)o1;
536                                 m2 = (Message)o2;
537                             }
538                             else {
539                                 throw new SimError("Invalid type of objects in " +
540                                     "the list");
541                             }
542
543                             diff = m1.getReceiveTime() - m2.getReceiveTime();
544                             if (diff == 0) {
545                                 return 0;
546                             }
547                             return (diff < 0 ? -1 : 1);
548                         }
549                     });
550                 break;
551             /* add more queue modes here */
552             default:
553                 throw new SimError("Unknown queue mode " + sendQueueMode);
554             }
555         }
556         return list;
557     }
558 }
```

Find: DeliverableMessages Previous Next

Search Results

2 matches X

516 | 9 | INS

```
376     /**
377      * Tries to send messages for the connections that are mentioned
378      * in the Tuples in the order they are in the list until one of
379      * the connections starts transferring or all tuples have been tried.
380      * @param tuples The tuples to try
381      * @return The tuple whose connection accepted the message or null if
382      * none of the connections accepted the message that was meant for them.
383      */
384     protected Tuple<Message, Connection> tryMessagesForConnected(
385         List<Tuple<Message, Connection>> tuples) {
386         if (tuples.size() == 0) {
387             return null;
388         }
389
390         for (Tuple<Message, Connection> t : tuples) {
391             Message m = t.getKey();
392             Connection con = t.getValue();
393             if (startTransfer(m, con) == RCV_OK) {
394                 return t;
395             }
396         }
397
398         return null;
399     }
```

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+F)

Source History Navigator

```
175 /**
176 * Tries to start a transfer of message using a connection. Is starting
177 * succeeds, the connection is added to the watch list of active connections
178 * @param m The message to transfer
179 * @param con The connection to use
180 * @return the value returned by
181 * {@link Connection#startTransfer(DTNHost, Message)}
182 */
183 protected int startTransfer(Message m, Connection con) {
184     int retVal;
185
186     if (!con.isReadyForTransfer()) {
187         return TRY_LATER_BUSY;
188     }
189
190     if (!policy.acceptSending(getHost(),
191         con.getOtherNode(getHost()), con, m)) {
192         return MessageRouter.DENIED_POLICY;
193     }
194
195     retVal = con.startTransfer(getHost(), m);
196     if (retVal == RCV_OK) { // started transfer
197         addToSendingConnections(con);
198     }
199     else if (deleteDelivered &amp; retVal == DENIED_OLD &&
200             m.getTo() == con.getOtherNode(this.getHost())) {
201         /* final recipient has already received the msg -> delete it */
202         this.deleteMessage(m.getId(), false);
203     }
204
205     return retVal;
206 }
```

Find: startTransfer Previous Next

Search Results

7 matches

183 | 10 | INS

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+F)

Source History Navigator

```
34 /**
35 * Sets a message that this connection is currently transferring. If message
36 * passing is controlled by external events, this method is not needed
37 * (but then e.g. {@link #finalizeTransfer()} and
38 * {@link #isMessageTransferred()} will not work either). Only a one message
39 * at a time can be transferred using one connection.
40 * @param from The host sending the message
41 * @param m The message
42 * @return The value returned by
43 * {@link MessageRouter#receiveMessage(Message, DTNHost)}
44 */
45 public int startTransfer(DTNHost from, Message m) {
46     assert this.msgOnFly == null : "Already transferring " +
47         this.msgOnFly + " from " + this.msgFromNode + " to " +
48         this.getOtherNode(this.msgFromNode) + ". Can't " +
49         "start transfer of " + m + " from " + from;
50
51     this.msgFromNode = from;
52     Message newMessage = m.replicate();
53     int retVal = getOtherNode(from).receiveMessage(newMessage, from);
54
55     if (retVal == MessageRouter.RCV_OK) {
56         this.msgOnFly = newMessage;
57         this.transferDoneTime = SimClock.getTime() +
58             (1.0*m.getSize()) / this.speed;
59     }
60
61     return retVal;
62 }
63
64 /**
```

Find: startTransfer Previous Next

Search Results

7 matches

45 | 33 | INS

```
446
447     /**
448      * Start receiving a message from another host
449      * @param m The message
450      * @param from Who the message is from
451      * @return The value returned by
452      * {@link MessageRouter#receiveMessage(Message, DTNHost)}
453      */
454     @Override
455     public int receiveMessage(Message m, DTNHost from) {
456         int retVal = this.router.receiveMessage(m, from);
457
458         if (retVal == MessageRouter.RCV_OK) {
459             m.addNodeOnPath(this); // add this node on the messages path
460         }
461
462         return retVal;
463     }

```

```

@Override
public int receiveMessage(Message m, DTNHost from) {
    int recvCheck = checkReceiving(m, from);
    if (recvCheck != RCV_OK) {
        return recvCheck;
    }

    // seems OK, start receiving the message
    return super.receiveMessage(m, from);
}
```

```

/**
 * Try to start receiving a message from another host.
 * @param m Message to put in the receiving buffer
 * @param from Who the message is from
 * @return Value zero if the node accepted the message (RCV_OK), value less
 * than zero if node rejected the message (e.g. DENIED_OLD), value bigger
 * than zero if the other node should try later (e.g. TRY_LATER_BUSY).
 */
public int receiveMessage(Message m, DTNHost from) {
    Message newMessage = m.replicate();

    this.putToIncomingBuffer(newMessage, from);
    newMessage.addNodeOnPath(this.host);

    for (MessageListener ml : this.mListeners) {
        ml.messageTransferStarted(newMessage, from, getHost());
    }

    return RCV_OK; // superclass always accepts messages
}
```

```

104     public void changedConnection(Connection con) {
105         if (this.energy != null && con.isUp() && !con.isInitiator(getHost())) {
106             this.energy.reduceDiscoveryEnergy();
107         }
108     }
109
110     @Override
111     public boolean requestDeliverableMessages(Connection con) {
112         if (isTransferString()) {
113             return false;
114         }
115         DTNHost other = con.getOtherNode(getHost());
116         /* do a copy to avoid concurrent modification exceptions
117          * (startTransfer may remove messages) */
118         ArrayList<Message> temp =
119             new ArrayList<Message>(this.getMessageCollection());
120         for (Message m : temp) {
121             if (other == m.getTo()) {
122                 if (startTransfer(m, con) == RCV_OK) {
123                     return true;
124                 }
125             }
126         }
127         return false;
128     }
129
130     @Override
131     public boolean createNewMessage(Message m) {
132         makeRoomForNewMessage(m.getSize());
133         return super.createNewMessage(m);
134     }

```

exchangeDeliverableMessages() Method starts transfer of message between two nodes. Message is exchanged only if other node is destination. Let us see what happens one by one.
RequestDeliverableMessages. We observe four major method here.

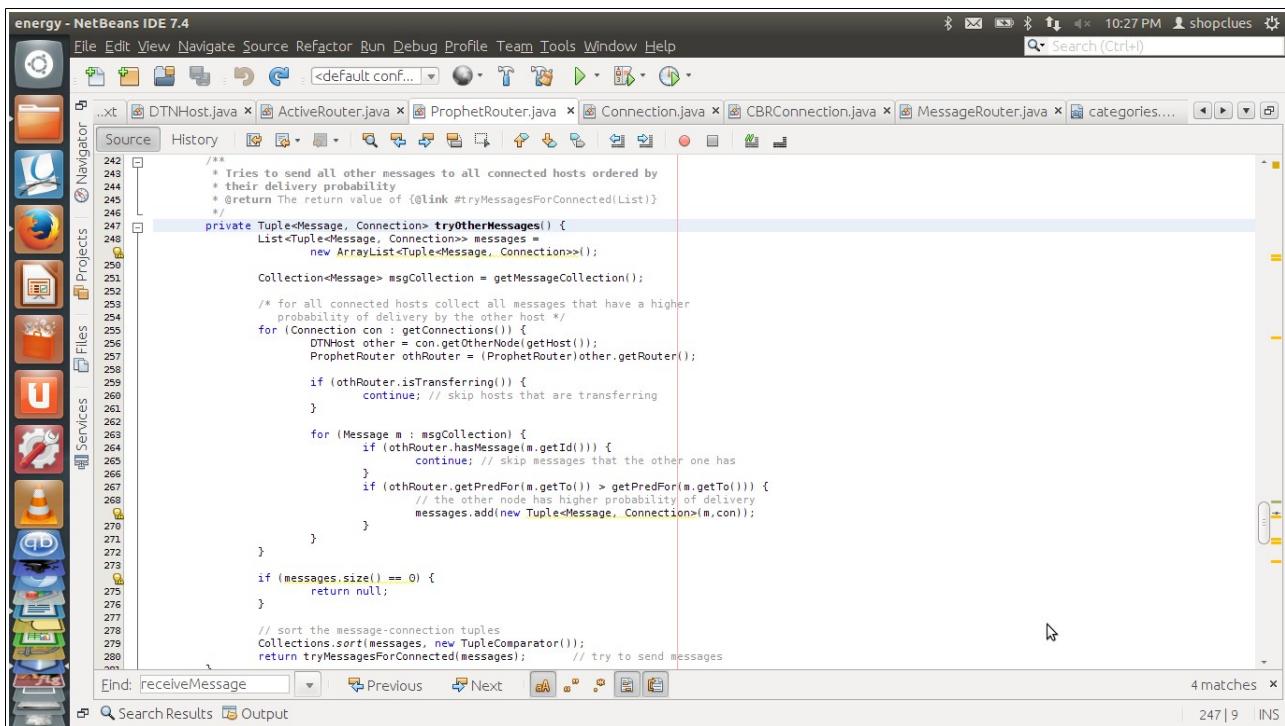
1. `getMessagesForConnected`
2. `sortByQueueMode`
3. `tryMessagesForConnected`
4. `requestDeliverableMessages`

First Method returns tuple object with message and connection list. The host associated with connection must be destination to message. In second method, messages are shuffle from list on bases of two modes taken from default config file. Two modes are FIFO and RANDOM modes. Third starts transfer between two nodes. Fourth will also do same as third.

Here `startTransfer` of `ActiveRouter` call `CBRConnection startTransfer` Method. Here we observe the method calling of `getOtherNode(from).receiveMessage(newMessage, from)` and intialisation of `this.msgOnFly = newMessage, this.transferDoneTime = SimClock.getTime() + (1.0*m.getSize()) / this.speed;`

Message is send to incoming buffer of other host. `MsgOnFly` is set and `this.transferDoneTime` gets value when transfer has to be finished based upon the size of message.

Now again comming out of `exchangeDeliverableMessage` Method. Last call is done to `tryOtherMessages()` method of `ProphetRouter`.



```

242     /**
243      * Tries to send all other messages to all connected hosts ordered by
244      * their delivery probability
245      * @return The return value of {@link #tryMessagesForConnected(List)}
246     */
247     private Tuple<Message, Connection> tryOtherMessages() {
248         List<Tuple<Message, Connection>> messages =
249             new ArrayList<Tuple<Message, Connection>>();
250
251         Collection<Message> msgCollection = getMessageCollection();
252
253         /* for all connected hosts collect all messages that have a higher
254          probability of delivery by the other host */
255         for (Connection con : getConnections()) {
256             DTNHost other = con.getOtherNode().getHost();
257             ProphetRouter othRouter = (ProphetRouter)other.getRouter();
258
259             if (othRouter.isTransferring())
260                 continue; // skip hosts that are transferring
261
262             for (Message m : msgCollection) {
263                 if (othRouter.hasMessage(m.getId())) {
264                     continue; // skip messages that the other one has
265                 }
266                 if (othRouter.getPredFor(m.getId()) > getPredFor(m.getId())) {
267                     // the other node has higher probability of delivery
268                     messages.add(new Tuple<Message, Connection>(m,con));
269                 }
270             }
271
272             if (messages.size() == 0)
273                 return null;
274
275             // sort the message-connection tuples
276             Collections.sort(messages, new TupleComparator());
277             return tryMessagesForConnected(messages); // try to send messages
278         }
279     }
280

```

Find: receiveMessage Previous Next

Search Results Output

4 matches X

247 | 9 INS

All the code is pretty clear. After all filtering criterias messages list of tuple is created. This is passed to `tryMessagesForConnected(messages)`. This method has been already discussed previously.

CBRConnection holds responsibility of the Message transfer information. Connection object made only when interfaces are in range of each other. That define its purpose.

Now back to Original world update Method the `moveHosts(this.updateInterval)` which is yet to discuss.

This method moves the hosts to new location that must be with updateInterval value.

```

]     /**
* Moves all hosts in the world for a given amount of time
* @param timeIncrement The time how long all nodes should move
*/
]     private void moveHosts(double timeIncrement) {
]         for (int i=0,n = hosts.size(); i<n; i++) {
]             DTNHost host = hosts.get(i);
]             host.move(timeIncrement);
]
}

```

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

default_settings.txt DTNHost.java World.java

Source History

```
369 /**
370  * Moves the node towards the next waypoint or waits if it is
371  * not time to move yet
372  * @param timeIncrement How long time the node moves
373 */
374 public void move(double timeIncrement) {
375     double possibleMovement;
376     double distance;
377     double dx, dy;
378
379     if (!isMovementActive() || SimClock.getTime() < this.nextTimeToMove) {
380         return;
381     }
382     if (this.destination == null) {
383         if (!setNextWaypoint()) {
384             return;
385         }
386     }
387
388     possibleMovement = timeIncrement * speed;
389     distance = this.location.distance(this.destination);
390
391     while (possibleMovement >= distance) {
392         // node can move past its next destination
393         this.location.setLocation(this.destination); // snap to destination
394         possibleMovement -= distance;
395         if (!setNextWaypoint()) { // get a new waypoint
396             return; // no more waypoints left
397         }
398         distance = this.location.distance(this.destination);
399     }
400
401     // move towards the point for possibleMovement amount
402     dx = (possibleMovement/distance) * (this.destination.getX() -
403         this.location.getX());
404     dy = (possibleMovement/distance) * (this.destination.getY() -
405         this.location.getY());
406     this.location.translate(dx, dy);
407 }
```

Find: update() Previous Next

Search Results Output

2 matches X

374 | 49 | INS

```
/***
 * Sets the next destination and speed to correspond the next waypoint
 * on the path.
 * @return True if there was a next waypoint to set, false if node still
 * should wait
 */
private boolean setNextWaypoint() {
    if (path == null) {
        path = movement.getPath();
    }

    if (path == null || !path.hasNext()) {
        this.nextTimeToMove = movement.nextPathAvailable();
        this.path = null;
        return false;
    }

    this.destination = path.getNextWaypoint();
    this.speed = path.getSpeed();

    if (this.movListeners != null) {
        for (MovementListener l : this.movListeners) {
            l.newDestination(this, this.destination, this.speed);
        }
    }
}

return true;
}
```

```

    * Returns a possible (random) placement for a host
    * @return Random position on the map
    */
@Override
public Coord getInitialLocation() {
    assert rng != null : "MovementModel not initialized!";
    Coord c = randomCoord();

    this.lastWaypoint = c;
    return c;
}

@Override
public Path getPath() {
    Path p;
    p = new Path(generateSpeed());
    p.addWaypoint(lastWaypoint.clone());
    Coord c = lastWaypoint;

    for (int i=0; i<PATH_LENGTH; i++) {
        c = randomCoord();
        p.addWaypoint(c);
    }

    this.lastWaypoint = c;
    return p;
}

```

```

    /**
     * Returns true if the path has more waypoints, false if not
     * @return true if the path has more waypoints, false if not
     */
public boolean hasNext() {
    return nextWpIndex < this.coords.size();
}

```

```

    /**
     * Returns a sim time when the next path is available. This implementation
     * returns a random time in future that is {@link #WAIT_TIME} from now.
     * @return The sim time when node should ask the next time for a path
     */
public double nextPathAvailable() {
    return SimClock.getTime() + generateWaitTime();
}

```

```

    /**
     * Returns the next waypoint on this path
     * @return the next waypoint
     */
public Coord getNextWaypoint() {
    assert hasNext() : "Path didn't have " + (nextWpIndex+1) + ". waypoint";
    return coords.get(nextWpIndex++);
}

```

```

    /**
     * Returns the speed towards the next waypoint (asked with
     * {@link #getNextWaypoint()}).
     * @return the speed towards the next waypoint.
     */
    public double getSpeed() {
        assert speeds.size() != 0 : "No speed set";
        assert nextWpIndex != 0 : "No waypoint asked";

        if (speeds.size() == 1) {
            return speeds.get(0);
        }
        else {
            return speeds.get(nextWpIndex-1);
        }
    }
}

```

Process starts with basic validation checks. If destination is not set then it obtains its new destination.

This is done through `setNextWaypoint()`. `possibleMovement` and `distance` values are compared against each other in the while loop. This while loop makes sure that chosen destination position is not crossed or when node moves to destination position then set host position is same as destination. Then through `setNextWaypoint()` new destination is set. Then distance is calculated in each dimension and position is updated. Lets have a look at `setNextWaypoint()` method. Here first Path is obtained and then destination through `this.destination = path.getNextWaypoint()` and speed is updated and Listeners are informed. In this case scenario it goes to RandomWaypoint `getPath` Method.

Important discussion on Mobility Model:

During initialisation phase each host gets its location based on Mobility Model. Like in case of MapBasedMobiliy Model each group host will get dispersed on the points in the their WKT file. Reading these file itself clears the concept. Also `getPath` Method is very important part of Mobilty Model. Since what will be your strategy to choose next destination for node lies here. People interested in modifying mobility according to certain parameters must look this method implementation in various mobility classes. Path class is very simple as it simply carries the coordinates of all places that become its destination point from time to time. It also records the list of speed that it has taken from source to destination. The `nextWpIndex` helps in getting destination point for host as it has lastest index value of List attributes.

At last in DTNSimUI class `done()` method is called which calls all reports `done` method included from config.

```

141     "\nmsg_time: " + format(getSimTime()));
142     double deliveryProb = 0; // delivery probability
143     double responseProb = 0; // request-response success probability
144     double overhead = Double.NaN; // overhead ratio
145
146     if (this.nrofCreated > 0) {
147         deliveryProb = (1.0 * this.nrofDelivered) / this.nrofCreated;
148     }
149     if (this.nrofDelivered > 0) {
150         overhead = (1.0 * (this.nrofRelayed - this.nrofDelivered)) /
151                         this.nrofDelivered;
152     }
153     if (this.nrofResponseReqCreated > 0) {
154         responseProb = (1.0* this.nrofResponseDelivered) /
155                         this.nrofResponseReqCreated;
156     }
157
158     String statsText = "created: " + this.nrofCreated +
159                         "\nstarted: " + this.nrofStarted +
160                         "\nrelayed: " + this.nrofRelayed +
161                         "\naborted: " + this.nrofAborted +
162                         "\ndropped: " + this.nrofDropped +
163                         "\nremoved: " + this.nrofRemoved +
164                         "\ndelivered: " + this.nrofDelivered +
165                         "\ndelivery_prob: " + format(deliveryProb) +
166                         "\nresponse_prob: " + format(responseProb) +
167                         "\noverhead_ratio: " + format(overhead) +
168                         "\nlatency_avg: " + getAverage(this.latencies) +
169                         "\nlatency_med: " + getMedian(this.latencies) +
170                         "\nhopcount_avg: " + getNtAverage(this.hopCounts) +
171                         "\nhopcount_med: " + getNtMedian(this.hopCounts) +
172                         "\nbuffertime_avg: " + getAverage(this.msgBufferTime) +
173                         "\nbuffertime_med: " + getMedian(this.msgBufferTime) +
174                         "\nrtt_avg: " + getAverage(this.rtt) +
175                         "\nrtt_med: " + getMedian(this.rtt)
176
177     write(statsText);
178 }
179
180 }
181
182 }
183
184 }

```

Find: update (1 match)
 Search Results | Output
 1|1 INS

All the variables of such class is updated through various Method calling of various Listener. In this case it is MessageListener.

Possible Network Characteristics in this Research area:

1. Presently only one connection transfer message and bidirectional connection are made.
We can control this bidirectional property and can make it unidirectional or can control under the condition where A makes connection to B and B to A. It is possible that A is connected to B but
B is not connected to A.
2. Mobility Pattern on the basis of Message Traffic.
3. Routing Pattern on basis of Mobility Pattern.
4. Queing or Scheduling Policy changes to lighten/reduce the load on Network.
5. Encryption/Decryption of Message.
6. Identity Based Cryptography etc.

Once the architecture of ONE is understood ,it is easy to do anything in it. Innovation is the limit.

CASE II : External File Analysis in ONE

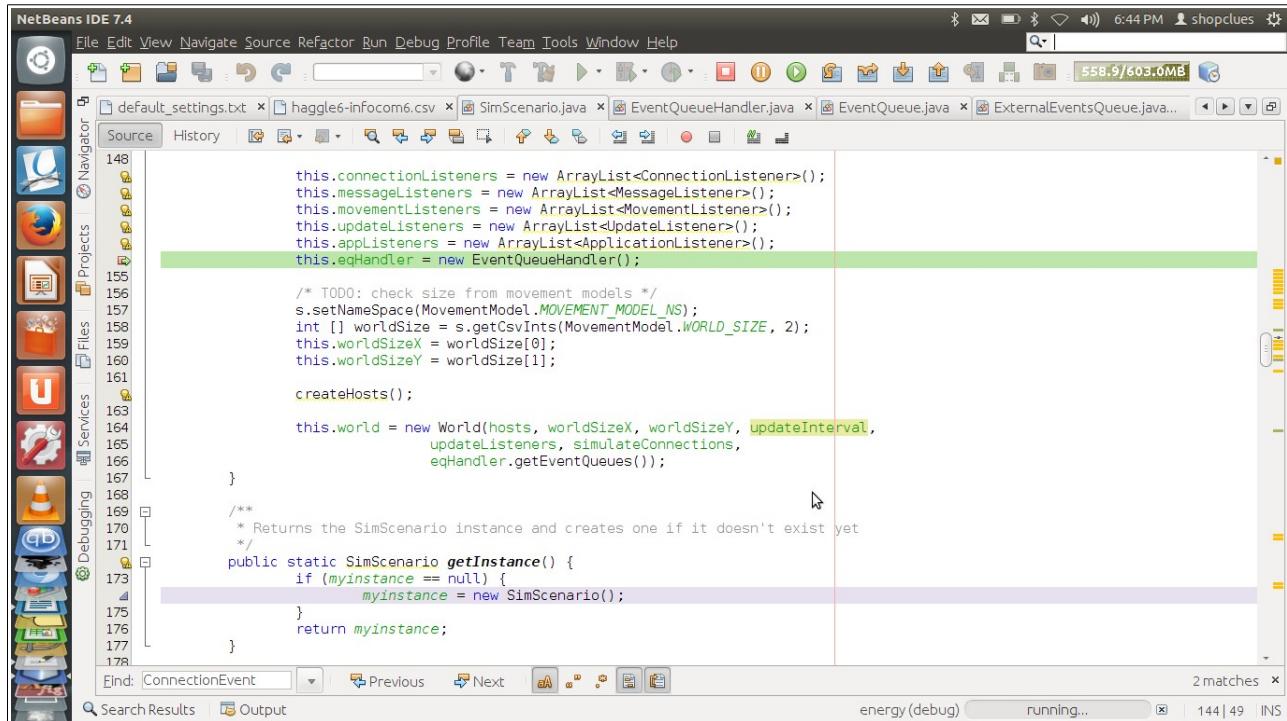
Introduction

There is also a provision for real data analysis in this simulator. One can define the connection between two nodes and message creation events in specific file format as shown in Appendix section. This section explains the working of code in such scenario.

Initialisation Phase working

Simscenario.java constructre

this.eqHandler = new EventQueueHandler() returns the object that contains the connection up and down timing along with source and destination network address.

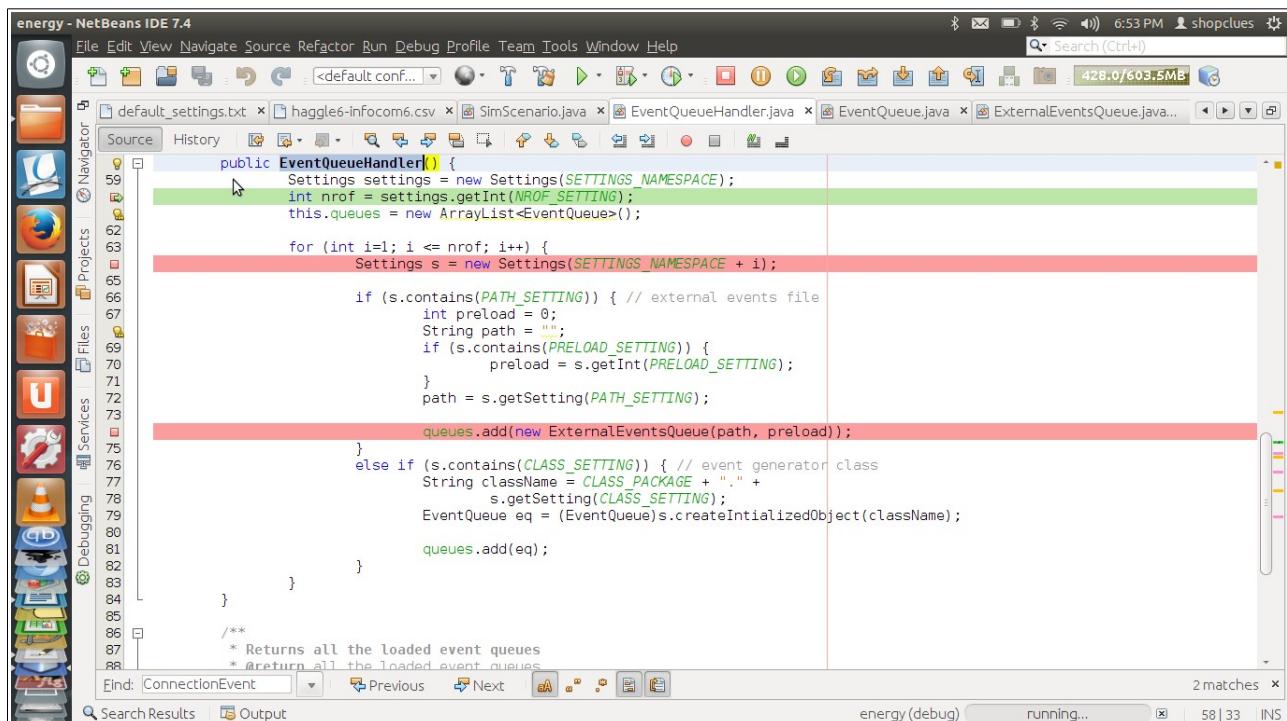


The screenshot shows the NetBeans IDE 7.4 interface with the SimScenario.java file open. The code is as follows:

```
148     this.connectionListeners = new ArrayList<ConnectionListener>();
149     this.messageListeners = new ArrayList<MessageListener>();
150     this.movementListeners = new ArrayList<MovementListener>();
151     this.updateListeners = new ArrayList<UpdateListener>();
152     this.appListeners = new ArrayList<ApplicationListener>();
153     this.eqHandler = new EventQueueHandler();
154
155     /* TODO: check size from movement models */
156     s.setNameSpace(MovementModel.MOVEMENT_MODEL_NS);
157     int [] worldsize = s.getCsvInts(MovementModel.WORLD_SIZE, 2);
158     this.worldSizeX = worldsize[0];
159     this.worldSizeY = worldsize[1];
160
161     createHosts();
162
163     this.world = new World(hosts, worldSizeX, worldSizeY, updateInterval,
164                           updateListeners, simulateConnections,
165                           eqHandler.getEventQueues());
166
167 }
168
169 /**
170  * Returns the SimScenario instance and creates one if it doesn't exist yet
171  */
172 public static SimScenario getInstance() {
173     if (myinstance == null) {
174         myinstance = new SimScenario();
175     }
176     return myinstance;
177 }
```

The code is annotated with several TODO comments and imports for ConnectionListener, MessageListener, MovementListener, UpdateListener, ApplicationListener, and EventQueueHandler.

EventQueueHandler looks like this:



The screenshot shows the NetBeans IDE 7.4 interface with the EventQueueHandler.java file open. The code is as follows:

```
59     public EventQueueHandler() {
60         Settings settings = new Settings(SETTINGS_NAMESPACE);
61         int nrof = settings.getInt(NROF_SETTING);
62         this.queues = new ArrayList<EventQueue>();
63
64         for (int i=1; i <= nrof; i++) {
65             Settings s = new Settings(SETTINGS_NAMESPACE + i);
66
67             if (s.contains(PATH_SETTING)) { // external events file
68                 int preload = 0;
69                 String path = "";
70                 if (s.contains(PRELOAD_SETTING)) {
71                     preload = s.getInt(PRELOAD_SETTING);
72                 }
73                 path = s.getSetting(PATH_SETTING);
74
75                 queues.add(new ExternalEventsQueue(path, preload));
76             }
77             else if (s.contains(CLASS_SETTING)) { // event generator class
78                 String className = CLASS_PACKAGE + "." +
79                             s.getSetting(CLASS_SETTING);
80                 EventQueue eq = (EventQueue)s.createInitializedObject(className);
81
82                 queues.add(eq);
83             }
84         }
85
86         /**
87          * Returns all the loaded event queues
88          * @return all the loaded event queues
89     }
```

The code initializes an ArrayList of EventQueue objects. It iterates through a list of settings, creating ExternalEventsQueue or EventQueue objects based on the setting type (PATH_SETTING or CLASS_SETTING).

In this case it goes to *if (s.contains(PATH_SETTING))* part. Here Path is

/home/shopclues/energy/ee/haggle6-infocom6.csv and preload is 0. Then `queues.add(new ExternalEventsQueue(path, preload))` line gets executed. In this it reaches to ExternalEventsQueue Constructore.

```
    /**
     * Creates a new Queue from a file
     * @param filePath Path to the file where the events are read from. If
     * file ends with extension defined in {@link BinaryEventsReader#BINARY_EXT}
     * the file is assumed to be a binary file.
     * @param nrofPreload How many events to preload
     * @see BinaryEventsReader#BINARY_EXT
     * @see BinaryEventsReader#storeToBinaryFile(String, List)
     */
    public ExternalEventsQueue(String filePath, int nrofPreload) {
        setNrofPreload(nrofPreload);
        init(filePath);
    }
```

```
    /**
     * Sets maximum number of events that are read when the next preload occurs
     * @param nrof Maximum number of events to read. If less than 1, default
     * value ( {@value DEFAULT_NROF_PRELOAD} ) is used.
     */
    public void setNrofPreload(int nrof) {
        if (nrof < 1) {
            nrof = DEFAULT_NROF_PRELOAD;
        }
        this.nrofPreload = nrof;
    }
```

```
private void init(String eeFilePath) {
    this.eventsFile = new File(eeFilePath);

    if (BinaryEventsReader.isBinaryEeFile(eventsFile)) {
        this.reader = new BinaryEventsReader(eventsFile);
    } else {
        this.reader = new StandardEventsReader(eventsFile);
    }

    this.queue = readEvents(nrofPreload);
    this.nextEventIndex = 0;
}
```

Here it reaches the `this.reader = new StandardEventsReader(eventsFile)` line.

```
public StandardEventsReader(File eventsFile){
    try {
        //this.scanner = new Scanner(eventsFile);
        this.reader = new BufferedReader(new FileReader(eventsFile));
    } catch (FileNotFoundException e) {
        throw new SimError(e.getMessage(), e);
    }
}
```

Then `this.queue = readEvents(nrofPreload)` will get executed.

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+F) 333.6 / 533.5MB

Source History Navigator Projects Files Services Debugging

```
139     }
140
141     /**
142      * Read some events from the external events reader
143      * @param nrof Maximum number of events to read
144      * @return A List of events that were read or an empty list if no events
145      * could be read
146     */
147     private List<ExternalEvent> readEvents(int nrof) {
148         if (allEventsRead) {
149             return new ArrayList<ExternalEvent>(0);
150         }
151
152         List<ExternalEvent> events = reader.readEvents(nrof);
153
154         if (nrof > 0 && events.size() == 0) {
155             reader.close();
156             allEventsRead = true;
157         }
158
159         return events;
160     }
161
162
163
164 }
```

Find: ConnectionEvent Previous Next Variables 2 matches

Search Results Output Variables energy (debug) running... 156 | 1 INS

This is a very important Method of StandardEventsReader in this case.

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+F) 368.8 / 533.0MB

Source History Navigator Projects Files Services Debugging

```
78     }
79
80     public List<ExternalEvent> readEvents(int nrof) {
81         ArrayList<ExternalEvent> events = new ArrayList<ExternalEvent>(nrof);
82         int eventsRead = 0;
83         // skip empty and comment lines
84         Pattern skipPattern = Pattern.compile("#.*|(^\\s*$)");
85
86         String line;
87         try {
88             line = this.reader.readLine();
89         } catch (IOException e1) {
90             throw new SimError("Reading from external event file failed.");
91         }
92         while (eventsRead < nrof && line != null) {
93             Scanner lineScan = new Scanner(line);
94             if (skipPattern.matcher(line).matches()) {
95                 // skip empty and comment lines
96                 try {
97                     line = this.reader.readLine();
98                 } catch (IOException e) {
99                     throw new SimError("Reading from external event file " +
100                                     "failed.");
101                 }
102                 continue;
103             }
104
105             double time;
106             String action;
107             String msgId;
108             int hostAddr;
109             int host2Addr;
110
111             try {
112                 time = lineScan.nextDouble();
113                 action = (lineScan.next());
114
115                 if (action.equals(DROP)) {
```

Find: ConnectionEvent Previous Next Variables 2 matches

Search Results Output Variables energy (debug) running... 78 | 9 INS

energy - NetBeans IDE 7.4

```

try {
    time = lineScan.nextDouble();
    action = lineScan.next();

    if (action.equals(ABORT)) {
        msgId = lineScan.nextInt();
        hostAddr = getHostAddress(lineScan.nextInt());
        events.add(new MessageDeleteEvent(hostAddr, msgId,
            time, true));
    }
    else if (action.equals(REMOVE)) {
        msgId = lineScan.nextInt();
        hostAddr = getHostAddress(lineScan.nextInt());
        events.add(new MessageDeleteEvent(hostAddr, msgId,
            time, false));
    }
    else if (action.equals(CONNECTION)) {
        String connEventType;
        boolean isUp;
        hostAddr = getHostAddress(lineScan.nextInt());
        host2Addr = getHostAddress(lineScan.nextInt());
        connEventType = lineScan.next();

        String interfaceId = null;
        if (lineScan.hasNext()) {
            interfaceId = lineScan.next();
        }

        if (connEventType.equalsIgnoreCase(CONNECTION_UP)) {
            isUp = true;
        }
        else if (connEventType.equalsIgnoreCase(CONNECTION_DOWN)) {
            isUp = false;
        }
        else {
            throw new SimError("Unknown up/down value '" +
                connEventType + "'");
        }
    }
}

```

Find: ConnectionEvent Previous Next

Search Results | Output | Variables

energy (debug) running... | 104 | 25 | INS

energy - NetBeans IDE 7.4

```

else {
    throw new SimError("Unknown up/down value '" +
        connEventType + "'");
}

ConnectionEvent ce = new ConnectionEvent(hostAddr,
    host2Addr, interfaceId, isUp, time);

events.add(ce);
}
else {
    msgId = lineScan.nextInt();
    hostAddr = getHostAddress(lineScan.nextInt());

    host2Addr = getHostAddress(lineScan.nextInt());
    if (action.equals(CREATE)){
        int size = lineScan.nextInt();
        int respSize = 0;
        if (lineScan.hasNextInt()) {
            respSize = lineScan.nextInt();
        }
        events.add(new MessageCreateEvent(hostAddr, host2Addr,
            msgId, size, respSize, time));
    }
    else {
        int stage = -1;
        if (action.equals(SEND)) {
            stage = MessageRelayEvent.SENDING;
        }
        else if (action.equals(DELIVERED)) {
            stage = MessageRelayEvent.TRANSFERRED;
        }
        else if (action.equals(ABORT)) {
            stage = MessageRelayEvent.ABORTED;
        }
        else {
            throw new SimError("Unknown action '" + action +
                "' in external events");
        }
    }
}

```

Find: ConnectionEvent Previous Next

Search Results | Output | Variables

energy (debug) running... | 134 | 67 | INS

Here heart of External file reading lies. In our Case it reaches to
`elseif(action.equals(CONNECTION)) {` and
`ConnectionEvent ce = new ConnectionEvent(hostAddr,host2Addr, interfaceId, isUp, time)`
is made.

```

21
22
23     protected String interfaceId;
24
25     /**
26      * Creates a new connection event
27      * @param from End point of connection
28      * @param to Another end of connection
29      * @param interf The number of interface for the connection
30      * @param up If true, this was a "connection up" event, if false, this
31      * was a "connection down" event
32      * @param time Time when the Connection event occurs
33     */
34     public ConnectionEvent(int from, int to, String interf, boolean up, double time) {
35         super(time);
36         assert to != from : "Can't self connect";
37         this.fromAddr = from;
38         this.toAddr = to;
39         this.isUp = up;
40         this.interfaceId = interf;
41     }
42
43     @Override
44     public void processEvent(World world) {
45         DTNHost from = world.getNodeByAddress(this.fromAddr);
46         DTNHost to = world.getNodeByAddress(this.toAddr);
47
48         from.forceConnection(to, interfaceId, this.isUp);
49     }
50
51     @Override
52     public String toString() {
53         return "CONN " + (isUp ? "up" : "down") + " @ " + this.time + " ";
54     }

```

Find: ConnectionEvent Previous Next ⌂ Variables energy(debug) running... 38 | 1 INS

This will return and create events of upto 500 lines of file. Again back to readEvents Method of ExternalEventsQueue Class. Then back to *EventQueueHandler()* constructre. There *queues.add(new ExternalEventsQueue(path, preload))* line execution gets finished. Then it comes back to Simscenario.java constructre where *this.eqHandler = new EventQueueHandler()* line execution gets finished. Now *List<ExternalEvent>* events attribute of ExternalEventsQueue object contains ConnectionEvent objects list.

Lets us see what happens next.

Then it reaches the world constructre.

```

58
59     private boolean simulateOnce;
60     private boolean isConSimulated;
61
62     /**
63      * Constructor.
64     */
65     public World(List<DTNHost> hosts, int sizeX, int sizeY,
66                 double updateInterval, List<UpdateListener> updateListeners,
67                 boolean simulateConnections, List<EventQueue> eventQueues) {
68         this.hosts = hosts;
69         this.sizeX = sizeX;
70         this.sizeY = sizeY;
71         this.updateInterval = updateInterval;
72         this.updateListeners = updateListeners;
73         this.simulateConnections = simulateConnections;
74         this.eventQueues = eventQueues;
75
76         this.simClock = SimClock.getInstance();
77         this.scheduledUpdates = new ScheduledUpdatesQueue();
78         this.isCancelled = false;
79         this.isConSimulated = false;
80
81         setNextEventQueue();
82         initSettings();
83     }
84
85     /**
86      * Initializes settings fields that can be configured using Settings class
87     */
88     private void initSettings() {
89         Settings s = new Settings(OPTIMIZATION_SETTINGS_NS);

```

Find: ConnectionEvent Previous Next ⌂ Variables energy(debug) running... 77 | 1 INS

Here eventQueues has ExternalEventsQueue which has queues of event list of ConnectionEvent. Then it comes to `setNextEventQueue()`.

```

    /**
     * Goes through all event Queues and sets the
     * event queue that has the next event.
     */
    public void setNextEventQueue() {
        EventQueue nextQueue = scheduledUpdates;
        double earliest = nextQueue.nextEventsTime();

        /* find the queue that has the next event */
        for (EventQueue eq : eventQueues) {
            if (eq.nextEventsTime() < earliest){
                nextQueue = eq;
                earliest = eq.nextEventsTime();
            }
        }

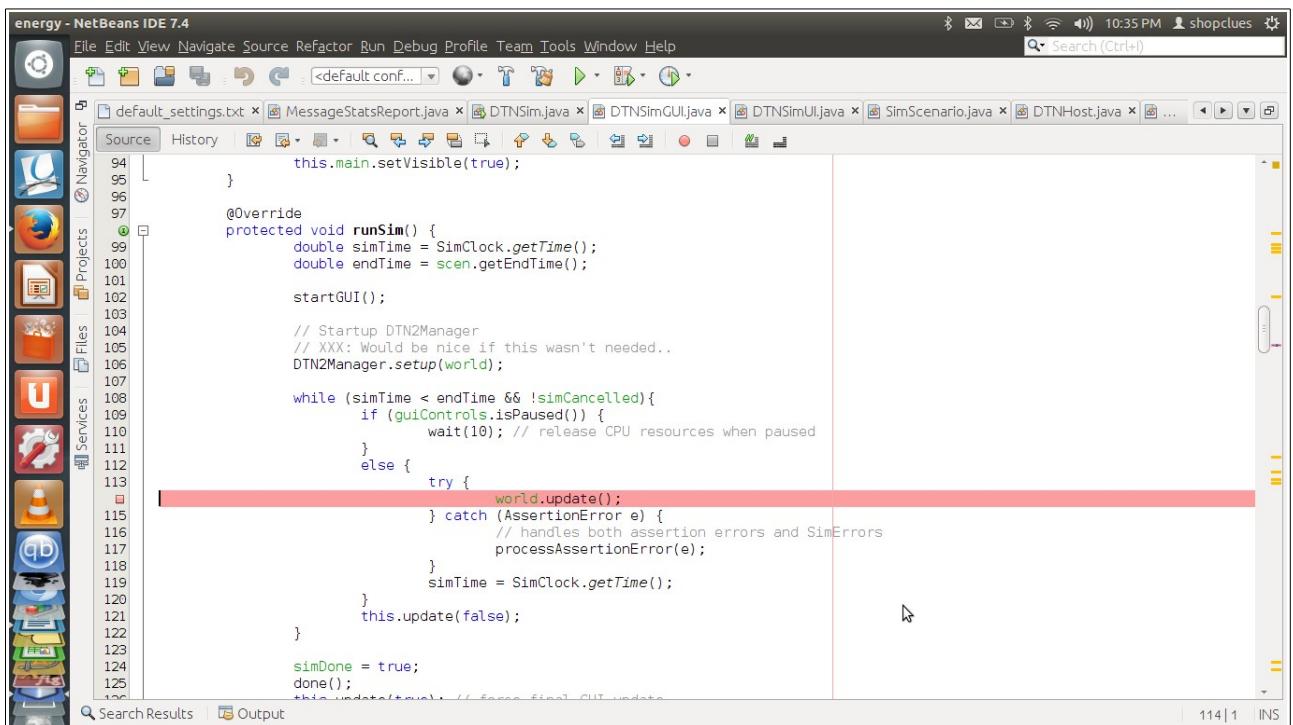
        this.nextEventQueue = nextQueue;
        this.nextQueueEventTime = earliest;
    }

```

Here earliest value is 4500 seconds as defined in first line of connection file. `this.nextEventQueue` has `ExternalEventsQueue` object in it.

Running Phase working

Directly moving to `world.update` method of in `DTNSimGUI runSim` Method.



```

149     * Runs all external events that are due between the time when
150     * this method is called and after one update interval.
151     */
152     public void update () {
153         double runUntil = SimClock.getTime() + this.updateInterval;
154
155         setNextEventQueue();
156
157         /* process all events that are due until next interval update */
158         while (this.nextQueueEventTime <= runUntil) {
159             simClock.setTime(this.nextQueueEventTime);
160             ExternalEvent ee = this.nextEventQueue.nextEvent();
161             ee.processEvent(this);
162             updateHosts(); // update all hosts after every event
163             setNextEventQueue();
164         }
165         moveHosts(this.updateInterval);
166         simClock.setTime(runUntil);
167
168         updateHosts();
169
170         /* inform all update listeners */
171         for (UpdateListener ul : this.updateListeners) {
172             ul.updated(this.hosts);
173         }
174     }
175
176     /**
177      * Updates all hosts (calls update for every one of them). If update
178      * order randomizing is on (updateOrder array is defined), the calls
179      * are made in random order.
180     */

```

We already see the activity in world update class for the case of the First case study.

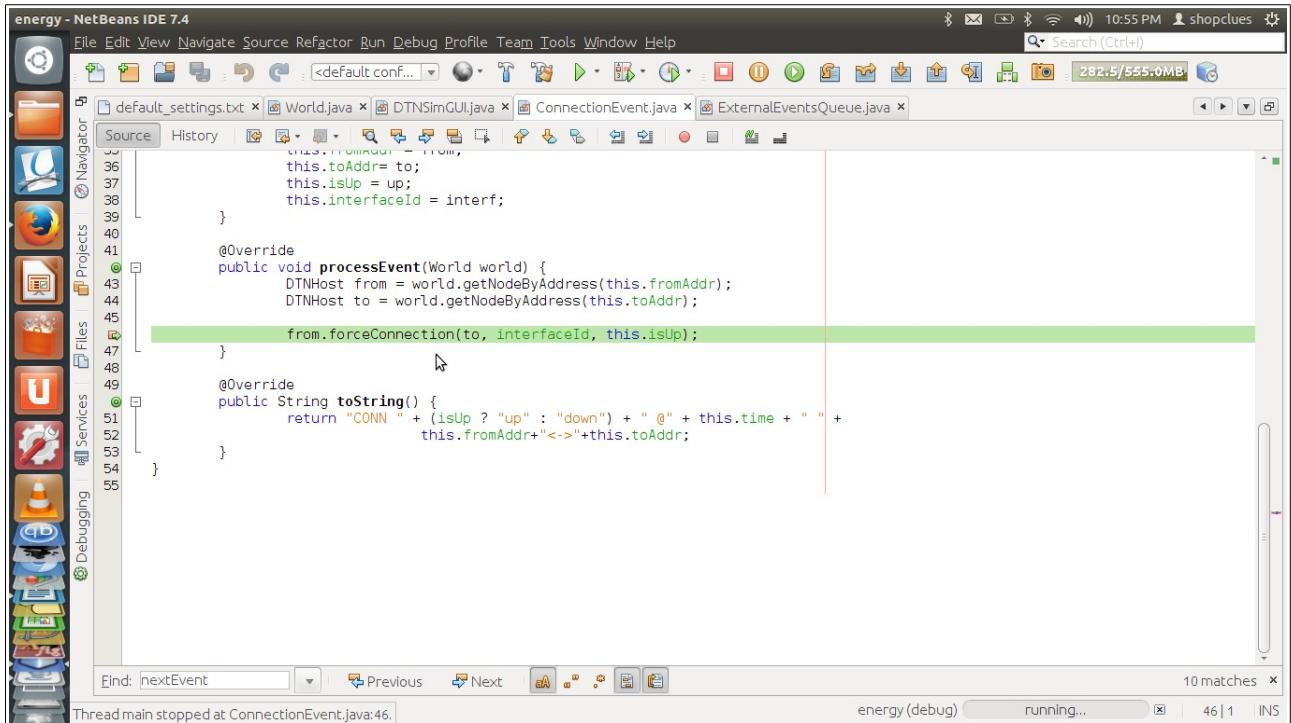
Let us again see for this case study. `setNextEventQueue()` will do same thing as in initialisation phase. But when `while (this.nextQueueEventTime <= runUntil)` will become true then inner code will execute. `this.nextQueueEventTime` will remain 4500 till `runUntil` will reach 4500.

```

100
101
102     else {
103         return queue.get(nextEventIndex).getTime();
104     }
105
106     /**
107      * Returns the next event in the queue or ExternalEvent with time of
108      * Double.MAX_VALUE if there are no events left
109      * @return The next event
110     */
111     public ExternalEvent nextEvent() {
112         if (queue.size() == 0) { // no more events
113             return new ExternalEvent(Double.MAX_VALUE);
114         }
115
116         ExternalEvent ee = queue.get(nextEventIndex);
117         nextEventIndex++;
118
119         if (nextEventIndex >= queue.size()) { // ran out of events
120             queue = readEvents(nrofPreload);
121             nextEventIndex = 0;
122         }
123
124         return ee;
125     }

```

Here ee is ConnectionEvent object that is being returned.



The screenshot shows the NetBeans IDE 7.4 interface with the 'energy' project open. The 'ConnectionEvent.java' file is the active editor. The code snippet highlights a call to the 'forceConnection' method:

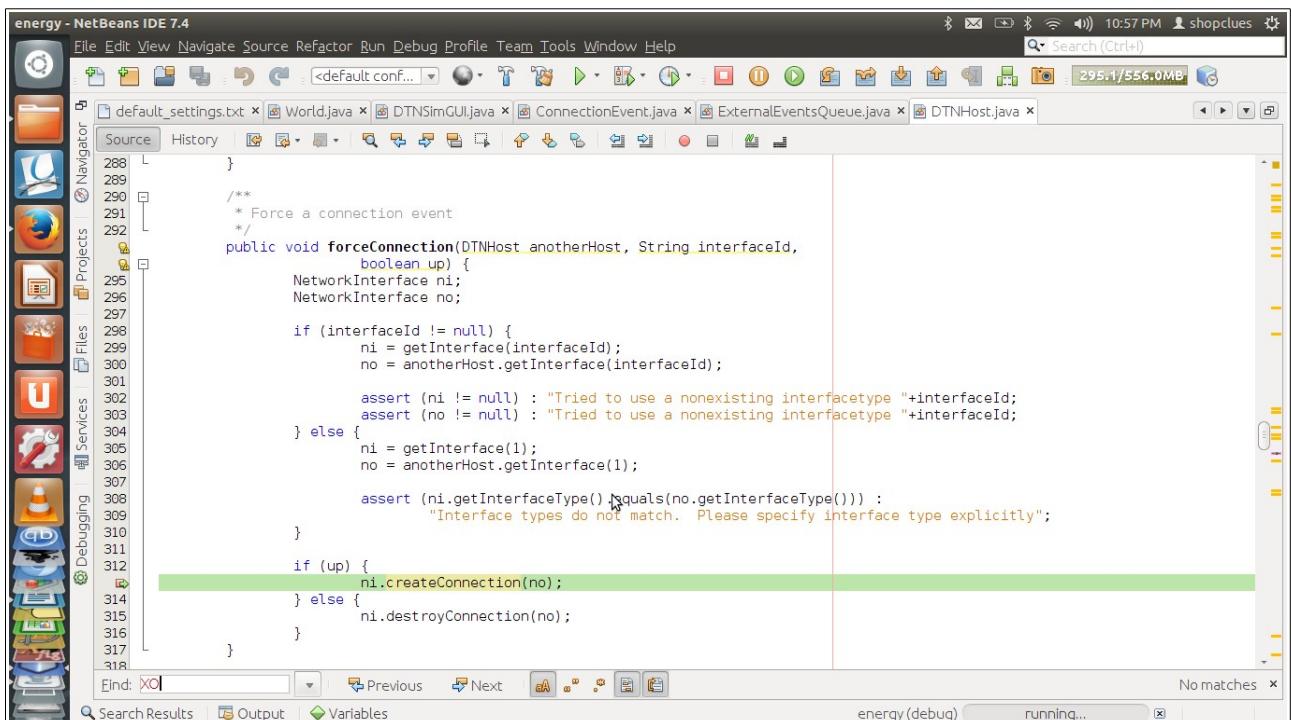
```
    from.forceConnection(to, interfaceld, this.isUp);
```

The 'from' variable is annotated with '@Override'. The 'forceConnection' method is defined as follows:

```
@Override  
public void forceConnection(DTNHost anotherHost, String interfaceld,  
    boolean up) {  
    NetworkInterface ni;  
    NetworkInterface no;  
  
    if (interfaceld != null) {  
        ni = getInterface(interfaceld);  
        no = anotherHost.getInterface(interfaceld);  
  
        assert (ni != null) : "Tried to use a nonexisting interface "+interfaceld;  
        assert (no != null) : "Tried to use a nonexisting interface "+interfaceld;  
    } else {  
        ni = getInterface(1);  
        no = anotherHost.getInterface(1);  
  
        assert (ni.getInterfaceType() .equals(no.getInterfaceType())) :  
            "Interface types do not match. Please specify interface type explicitly";  
    }  
  
    if (up) {  
        ni.createConnection(no);  
    } else {  
        ni.destroyConnection(no);  
    }  
}
```

The status bar at the bottom indicates 'Thread main stopped at ConnectionEvent.java:46.'

Here forced connections are made or Broken down as per file instruction.



The screenshot shows the NetBeans IDE 7.4 interface with the 'energy' project open. The 'DTNHost.java' file is the active editor. The code snippet highlights a call to the 'createConnection' method:

```
    ni.createConnection(no);
```

The 'createConnection' method is defined as follows:

```
public void createConnection(NetworkInterface no) {  
    if (no == null) {  
        throw new NullPointerException("no");  
    }  
  
    if (no.getInterfaceType() .equals(interfaceType)) {  
        if (no.getInterfaceId() == interfaceld) {  
            if (isUp) {  
                if (no.isUp()) {  
                    return;  
                } else {  
                    no.setUp(true);  
                    fireConnectionEvent(new ConnectionEvent(this, to, interfaceld, true));  
                }  
            } else {  
                if (!no.isUp()) {  
                    no.setUp(false);  
                    fireConnectionEvent(new ConnectionEvent(this, to, interfaceld, false));  
                }  
            }  
        } else {  
            throw new IllegalArgumentException("Interface id mismatch");  
        }  
    } else {  
        throw new IllegalArgumentException("Interface type mismatch");  
    }  
}
```

The status bar at the bottom indicates 'No matches'.

It calls SimpleBroadCastInterface createConnection Method.

```

91     optimizer.getNearInterfaces(this);
92     for (NetworkInterface i : interfaces) {
93         connect(i);
94     }
95
96     /**
97      * Creates a connection to another host. This method does not do any checks
98      * on whether the other node is in range or active
99      * @param anotherInterface The interface to create the connection to
100     */
101    public void createConnection(NetworkInterface anotherInterface) {
102        if (!isConnected(anotherInterface) && (this != anotherInterface)) {
103            // connection speed is the lower one of the two speeds
104            int conSpeed = anotherInterface.getTransmitSpeed();
105            if (conSpeed > this.transmitSpeed) {
106                conSpeed = this.transmitSpeed;
107            }
108
109            Connection con = new CBRConnection(this.host, this,
110                                              anotherInterface.getHost(), anotherInterface, conSpeed);
111            connect(con,anotherInterface);
112        }
113    }
114
115    /**
116     * Returns a string representation of the object.
117     * @return a string representation of the object.
118     */
119    public String toString() {
120        return "SimpleBroadcastInterface " + super.toString();
121    }

```

Thread main stopped at SimpleBroadcastInterface.java:112.

energy (debug) running... 112|1 INS

It calls the connect. Rest process is already discussed.

It also calls `ni.destroyConnection(no)` which destroys connections as can be seen in snapshots.

```

443
444     /**
445      * Disconnect a connection between this and another host.
446      * @param anotherInterface The other host's network interface to disconnect
447      * from this host
448      */
449    public void destroyConnection(NetworkInterface anotherInterface) {
450        DTNHost anotherHost = anotherInterface.getHost();
451        for (int i=0; i < this.connections.size(); i++) {
452            if (this.connections.get(i).getOtherNode(this.host) == anotherHost){
453                removeConnectionByIndex(i, anotherInterface);
454            }
455        }
456        // the connection didn't exist, do nothing
457    }
458
459    /**

```

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

The screenshot shows the NetBeans IDE 7.4 interface. The title bar says "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The left sidebar has icons for Projects, Files, Services, and Debugging. The main area shows the "NetworkInterface.java" file with the following code:

```

457 /**
458 * Removes a connection by its position (index) in the connections array
459 * of the interface
460 * @param index The array index of the connection to be removed
461 * @param anotherInterface The interface of the other host
462 */
463 private void removeConnectionByIndex(int index,
464                                     NetworkInterface anotherInterface) {
465     Connection con = this.connections.get(index);
466     DTNHost anotherNode = anotherInterface.getHost();
467     con.setAppState(false);
468     notifyConnectionListeners(CON_DOWN, anotherNode);
469
470     // tear down bidirectional connection
471     if (!anotherInterface.getConnections().remove(con)) {
472         throw new SimError("No connection " + con + " found in " +
473                           anotherNode);
474     }
475
476     this.host.connectionDown(con);
477     anotherNode.connectionDown(con);
478
479     connections.remove(index);
480 }
481
482
483

```

energy - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

The screenshot shows the NetBeans IDE 7.4 interface. The title bar says "energy - NetBeans IDE 7.4". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The left sidebar has icons for Projects, Files, Services, and Debugging. The main area shows the "DTNHost.java" file with the following code:

```

158     return this.comBus;
159 }
160
161 /**
162 * Informs the router of this host about state change in a
163 * object.
164 * @param con The connection object whose state changed
165 */
166 public void connectionUp(Connection con) {
167     this.router.changedConnection(con);
168 }
169
170 public void connectionDown(Connection con) {
171     this.router.changedConnection(con);
172 }
173

```

ConnectionUp and ConnectionDown can be used by router to get information of connections. In this setting Only One Event is used to simulate Connections. If Messages Generation Phenomean have to be added then MessageEventGent Generator can be used or file containing MessageEvent can be used. Sample shown in Appendix section.

Appendix

Default_settings.txt file for CASE I Analysis

```
## Scenario settings
Scenario.name = default_scenario
Scenario.simulateConnections = true
Scenario.updateInterval = 0.1
# 43200s == 12h
Scenario.endTime = 43200

## Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per second)
# transmitRange : range of the interface (meters)

# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 2 Mbps = 250kBps
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10

# High speed, long range, interface for group 4
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 1000

# Define 6 different node groups
Scenario.nrofHostGroups = 6

## Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name from movement package)
# waitTime: minimum and maximum wait times (seconds) after reaching destination
# speed: minimum and maximum speeds (m/s) when moving on a path
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from routing package)
# activeTimes: Time intervals when the nodes in the group are active (start1, end1, start2, end2, ... )
# msgTtl : TTL (minutes) of the messages created by this host group, default=infinite

## Group and movement model specific settings
# pois: Points Of Interest indexes and probabilities (poiIndex1, poiProb1, poiIndex2, poiProb2, ... )
#     for ShortestPathMapBasedMovement
```

```

# okMaps : which map nodes are OK for the group (map file indexes), default=all
# for all MapBasedMovent models
# routeFile: route's file path - for RandomWaypoint
# routeType: route's type - for RandomWaypoint

# Common settings for all groups
Group.movementModel = RandomWaypoint
Group.router = ProphetRouter
Group.bufferSize = 10M
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
Group.speed = 0.5, 1.5
# Message TTL of 300 minutes (5 hours)
Group.msgTtl = 300
Group.nrofHosts = 58

#####
Group.initialEnergy = 4800
Group.scanEnergy = 0.06
Group.scanResponseEnergy = 0.08
Group.transmitEnergy = 0.08
Group.baseEnergy = 0.07
Group.charging_coefficient = 20
Group.thrushold_energy = 3000
#####

# group1 (pedestrians) specific settings
Group1.groupID = p

# group2 specific settings
Group2.groupID = c
# cars can drive only on roads
Group2.okMaps = 1
# 10-50 km/h
Group2.speed = 2.7, 13.9

# another group of pedestrians
Group3.groupID = w

# The Tram groups
Group4.groupID = t
Group4.bufferSize = 50M
Group4.movementModel = RandomWaypoint
Group4.routeFile = data/tram3.wkt
Group4.routeType = 1
Group4.waitTime = 10, 30

```

```

Group4.speed = 7, 10
Group4.nrofHosts = 2
Group4.nrofInterfaces = 2
Group4.interface1 = btInterface
Group4.interface2 = highspeedInterface

Group5.groupID = t
Group5.bufferSize = 50M
Group5.movementModel = RandomWaypoint
Group5.routeFile = data/tram4.wkt
Group5.routeType = 2
Group5.waitTime = 10, 30
Group5.speed = 7, 10
Group5.nrofHosts = 2

Group6.groupID = t
Group6.bufferSize = 50M
Group6.movementModel = RandomWaypoint
Group6.routeFile = data/tram10.wkt
Group6.routeType = 2
Group6.waitTime = 10, 30
Group6.speed = 7, 10
Group6.nrofHosts = 2

```

```

## 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 (500kB - 1MB)
Events1.size = 500k,1M
# range of message source/destination addresses
Events1.hosts = 0,180
# Message ID prefix
Events1.prefix = M

## 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

## Reports - all report names have to be valid report classes

# how many reports to load
Report.nrofReports = 2
# length of the warm up period (simulated seconds)
Report.warmup = 0
Report.granularity = 43000
# default directory of reports (can be overridden per Report with output setting)
Report.reportDir = reports/
# Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = EnergyLevelReport

## Default settings for some routers settings
ProphetRouter.secondsInTimeUnit = 30
ProphetRouter.alpha = 0.5
ProphetRouter.beta = 0.5
ProphetRouter.energy_max = 4800

SprayAndWaitRouter.nrofCopies = 6
SprayAndWaitRouter.binaryMode = true

## Optimization settings -- these affect the speed of the simulation
## see World class for details.
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true

## 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]$

```

Extra default file setting for CASE II analysis

```
Events.nrof = 2
Events1.class = StandardEventsReader
Events1.filePath = <<directory containing the trace file>>/mytracefile.txt

Events2.class = MessageEventGenerator
# change interval to have more or less messages, see javadocs for more information.
Events2.interval = 30,40
Events2.size = 1k
# range of message source/destination addresses
Events2.hosts = <<min node ID>>,<<max node ID>>
Events2.prefix = M
```

In this case, *setNextEventQueue()* method of world will use MessageEventGenerator also as in first case study to set Message in the nodes. This is also set from *EventQueueHandler()* construction call from Simscenario initialisation.

Discussion on StandardEventsReaders.java

This class is specifically designed for events readings. Reading this file and special focus on *public List<ExternalEvent> readEvents(int nrof)* method will explain all.

Sample file of CASE II Analysis

Portion of **haggle6-infocom6.csv** used in our test case.

```
4500.00 CONN 16 12 up
5623.00 CONN 12 16 up
5872.00 CONN 2 13 up
5960.00 CONN 15 52 up
6010.00 CONN 13 2 up
6023.00 CONN 4 3 up
6207.00 CONN 21 24 up
6639.00 CONN 55 24 up
6802.00 CONN 84 90 up
6848.00 CONN 3 4 up
6850.00 CONN 4 0 up
6900.00 CONN 0 39 up
6915.00 CONN 84 95 up
6942.00 CONN 86 50 up
7025.00 CONN 15 16 up
7032.00 CONN 45 82 up
7033.00 CONN 84 67 up
7035.00 CONN 66 94 up
7035.00 CONN 69 40 up
7045.00 CONN 21 55 up
7128.00 CONN 0 51 up
```

7146.00 CONN 85 70 up
7152.00 CONN 92 63 up
7156.00 CONN 89 92 up
7159.00 CONN 64 94 up
7163.00 CONN 44 37 up
7163.00 CONN 44 77 up
7170.00 CONN 27 95 up
7173.00 CONN 86 41 up
7205.00 CONN 4 15 up
7217.00 CONN 70 86 up
7255.00 CONN 14 37 up
7259.00 CONN 85 20 up
7259.00 CONN 88 22 up
7259.00 CONN 88 41 up
7261.00 CONN 12 1 up
7262.00 CONN 15 12 up
7267.00 CONN 71 80 up
7268.00 CONN 68 53 up
7269.00 CONN 68 39 up
7269.00 CONN 94 53 up
7269.00 CONN 95 36 up
7270.00 CONN 61 35 up

Sample section of file containing MessageEvent List

3662.5 C	M1	4	2	1262799
3721.7 C	M2	90	76	1292645
3730.6 C	M3	33	99	1236983
3805.4 C	M4	62	12	1191810
3817.3 C	M5	67	19	606629
3824.3 C	M6	1	73	1221142
3829.0 C	M7	19	13	291138
3833.8 C	M8	59	71	1633127
3836.6 C	M9	38	26	1333674
3843.8 C	M10	45	99	446071
3911.7 C	M11	16	32	1302999
3931.6 C	M12	98	55	355706
937.4 C	M13	71	47	1461414
3971.9 C	M14	89	85	1740698
3979.8 C	M15	10	53	368358
3982.3 C	M16	4	65	1403045