Installation and use of the “cgr-jni-Merge” packet

The cgr-jni-Merge packet has been primarily developed to insert CGR and OCGR support in The ONE simulator. However, it also adds a few auxiliary features that could be used independently of CGR/OCGR. These extensions are: the support of priorities (message generators with priorities, Epidemic routing with priorities); the possibility of converting a ONE log into an ION contact plan, and last but foremost, the possibility of enforcing deterministic contacts provided in an external contact plan.

This guide is intended for both developers and users.

Summary

1 Integration of the “cgr-jni-Merge” packet into The ONE

1.1 Preliminary steps

1.1.1 Operating system compatibility of cgr-jni

A Unix/Linux O.S. is required to run the native C code called by CGR and OCGR classes contained in cgr-jni-Merge. A 64 bit version should be preferred, as OCGR classes may show an inconsistent behaviour on 32 bit platforms. In Windows the compatibility is limited to the classes that do not interface with native C code, such as: the "PriorityEpidemicRouter", the priority message generators, the class to convert a log into an ION contact plan and the class that reads and enforces an ION contact plan.

1.1.2 Downloading The ONE and cgr-jni-Merge packages

The ONE (Opportunistic Network Environment Simulator) can be downloaded from **https://akeranen.github.io/the-one/** .

This page also contains links to several “Community Resources”, including a good, although not recently updated, tutorial: http://one-simuator-for-beginners.blogspot.it/2013/08/one-simulator-introduction.html.

Unzip the .zip file containing The ONE; a directory called “the-one master” will be created. The ONE folder contains all original Java classes, setting files and two scripts for compilation and running (one.sh and compile.sh respectively). This guide and the code refers to the latest version available at the time of writing (v1.6.0).

The cgr-jni-Merge (branch “Merge” of cgr-jni) extension can be downloaded from **https://github.com/alessandroberlati/cgr-jni/tree/Merge**

This package is described in Simone Pozza’s thesis (in English)

http://amslaurea.unibo.it/12150/1/pozza_simone_tesi.pdf

Once downloaded, create a new directory, e.g. cgr-jni, and unzip the cgr-jni-Merge.zip in it (4 subdirectories will be created). Note that while the ONE contains only Java code, cgr-jni-Merge contains both Java and native C code, to be compiled separately.

1.1.3 Modifications to The ONE

We tried hard to keep cgr-jni-Merge code completely separate from the ONE code, in order to distribute our package as an independent external module, thus only very few changes in the ONE

code are necessary. At present we need just two modifications in the code and one in the one.sh script file, used to start ONE manually.

1.1.3.1 Host addresses from 1

As ION cannot handle a node whose ipn number is 0, we need to modify ONE to start assigning the host addresses from 1. To this end, the ONE code needs to be changed as follows:

class core.DTNHost

line 22: initialize nextAddress with 1.

line 107: assign 1 to nextAddress.

Ctrl+S to save.

Note that in practice ONE will continue to use (and require in settings) numbers starting from 0. However, in logs and on the GUI they will be shifted by one (e.g. [0,8] will become in logs and GUI [1,9])

1.1.3.2 getMoreInfo() method

The second change consists in adding the method getMoreInfo() to the class report.MessageStatsReport.

Starting from line 178 of MessageStatsReport.java, add the lines in bold.

class report.MessageStatsReport

```
178   write(statsText);
179   write(getMoreInfo());
180   super.done();
181   }
182   protected String getMoreInfo() {
183         return "";

184   }
```

Ctrl+S to save.

This method is overridden in the report.OCGRMessageStatsReport class, so it is necessary to add it here, although it returns an empty string.

1.1.4 Adding paths in the file one.sh

To allow the JVM to find the cgr-jni-Merge Java classes, we need to modify a command in one.sh.

In the file one.sh, add $CGR_JNI_CLASSPATH to the -cp parameter of the java command invocation, and also “bin” in case the user wants to compile the ONE classes with an IDE, like Eclipse. In brief, add to the existing command one or both the parts in bold in the right order (note that the character “:” works as a separator).

```
java -Xmx512M -cp
bin:target:lib/ECLA.jar:lib/DTNConsoleConnection.jar:$CGR_JNI_C
LASSPATH core.DTNSim $*
```

1.2 Compiling and launching The ONE and cgr-jni from command line interface

Note that while the ONE contains only Java code, cgr-jni-Merge contains also native C code, to be compiled separately. The compilation can be done either from a command line interface or in Eclipse. Here we will refer to the former (easier for normal users). Developers may prefer the compilation in Eclipse, explained later.

1.2.1 Compiling Java classes (of both The ONE and cgr-jni-Merge)

First, as the JAVA 8 compiler (or above) is required, check your version:

java -version

Then check if the environment variable $JAVA_HOME is properly set:

echo $JAVA_HOME

This variable is necessary to the compiler to find the cgr-jni-Merge JNI (Java Native Interface) header files. It is usually set to /usr/lib/jvm/java-8-oracle/ depending on which Java version is actually installed on your machine. If this variable is not set, the compilation fails. If needed you can (temporarily) set the variable with this command:

export JAVA_HOME=/usr/lib/jvm/YOUR-VERSION/

You can find a lot of possibility, go in /usr/lib/jvm and list directories inside, you have to find something like "java-1.8.0-openjdk".

It is important to use the same terminal because if you open a new terminal you have to set JAVA_HOME again. To facilitate the resolution of dependencies and the compilation we use two scripts. The first, “compile.sh”, is provided in the ONE package and is used to compile all the ONE classes together. This script put the .class files (i.e. the exe in Java) into the “target” folder, by default.

After running “compile.sh” it is necessary to enter the following commands:

export ONE_BIN=/Path/To/ONE/target

where "/Path/To/ONE/target" denotes the folder containing the ONE .class files and then

export CGR_JNI_CLASSPATH=/Path/To/cgr-jni-Merge/bin

also used in the script one.sh, used to launch ONE (see below). Note that paths defined by means of export commands are temporary.

Finally, run the "compile_cgr-jni.sh" script to compile the cgr-jni-Merge Java classes.

1.2.2 Compiling the cgr-jni-Merge native C code

To compile the native C code the use of command line interface is recommended. To this end the user should enter in the cgr-jni-Merge/ion_cgr_jni directory, and enter the following command:

make ONE_CLASSPATH=/Path/To/ONE/target

where "/Path/To/ONE" is the path to the root directory of the packages containing the .class files obtained as a result of the previous ONE compilation. You can optionally append DEBUG=1 to enable CGR debug prints.

It is important to note that the Makefile assumes that the Java classes of the cgr_jni packet are put by the Java compiler into the bin directory, as Eclipse does. If this is not the case, the actual classpath of these classes needs to be added to the make command, as follows (the character colon represents a list separator in Java, while ONE_classpath and cgr_jni_classpath denotes your paths):

make ONE_CLASSPATH=/Path/To/ONE:/Path/To/cgr-jni-Merge

1.2.3 Running simulations

The ONE provides two ways to perform simulations, the graphic mode and the batch mode. The former makes use of a Graphic User Interface (GUI) that shows the nodes moving in the map (default Helsinki) and allows the user to interact with the simulation with pause, step and fast forward buttons. The latter is much faster and it also allows the user to perform a series of simulations automatically, by changing one parameter (such as the bundle size or the expiration time) each run.

The ONE can be manually started by means of the one.sh script in TheONE folder. Before invoking the script, we need to set the LD_LIBRARY_PATH and the CGR_JNI_CLASSPATH environment variables. The first refers to the location of the libcgr_jni.so library, the second to the class files of the cgr-jni packet:

export LD_LIBRARY_PATH=/Path/To/libcgr_jni.so

export CGR_JNI_CLASSPATH=/Path/To/cgr-jni-Merge/bin

Note that paths defined by means of export commands are temporary, thus must be re-entered after a reboot.

1.2.3.1 GUI mode

To run ONE in the GUI mode the syntax is.

one.sh [/Path/To/settings.txt]

For example:

one.sh

will launch ONE in the GUI mode, with the settings specified in default_settings.txt

one.sh /Path/To/settings.txt

will do the same, but after reading the default_settings.txt the file settings.txt will be read too (by overriding parameters with the same name).

1.2.3.2 Batch mode

To run ONE in the batch mode the syntax is.

one.sh –b <number of times to run> [/Path/To/settings.txt]

For example

one.sh –b 1

will launch ONE in the batch mode, with the settings specified in default_settings.txt; the advantage is that without the GUI the execution time will be much shorter.

one.sh –b 7

will launch ONE in the batch mode, with the settings specified in default_settings.txt for seven times; this can be useful only if in the default setting file the random.seed=1, in order to not have exactly the same results at each run. Note that as runs are independent, 7 sets of log files will be created.

one.sh –b 1 /Path/To/settings.txt

will launch ONE in the batch mode once, reading first the settings specified in default_settings.txt and then those given in settings.txt.

If in the configuration files we have n values for the same parameter, it is possible to run one n times with the different values, by exploiting the batch mode. For example, if in settings.txt we have

Group.buffrsize=[5M; 10M; 15M; 25M; 30M; 35M]

the command

one.sh –b 7 /Path/To/settings.txt

will result into 7 consecutive and independent execution of ONE, each with a different buffer value. For user convenience, three example files are provided in the folder cgr-jni-Merge/simulations/test_batch to change BufferSize, messageSize and TTL, respectively.

1.3 Compiling and launching The ONE and cgr-jni in Eclipse

A developer could be interested in integrating both TheONE and cgr-jni-Merge, in an IDE, to modify, compile, run and debug. We will refer here to a specific IDE, Eclipse. It is important to note that it is required Eclipse for Java Enterprise Edition with support to C.

1.3.1 Importing The ONE in Eclipse

Open Eclipse; create a new Java project, called for example TheONE. Click with the right button of the mouse on the just created project folder, then import->general->file system, click on browser and choose the-one-master folder created before by unzipping the original file. Importing a project in Eclipse usually implies that the original sources files are copied inside the Eclipse workspace. If this is the case, every change made in Eclipse will affect only files inside Eclipse, not the original ones.

1.3.1.1 Adding .jar libraries

We need to add three .jar libraries.

1. click with the right button on the TheONE project, Build Path->add External Archives, select the “the-one-master”, lib, and select both files (DTNConcsoleConnection.jar and ECLA.jar).
2. click with the right button on the TheONE project, then properties-> java Build Path, click on libraries->Add library, select Junit, Next select Junit4, Finsh and click apply and then OK.

These steps corresponds to steps from 10 to 12 in http://one-simuator-for-beginners.blogspot.it/2013/08/how-to-integrate-one-with-eclipse.html.

1.3.1.2 Modifications to The ONE

Add the modifications for cgr-jni-Merge compatibility as described before, unless already done.

1.3.2 Importing cgr-jni-Merge in Eclipse and linking

Create a new Java project, called for example cgr-jni-Merge. Click with the right button of the mouse on the just created project folder, then import->general->file system, click on browser and choose cgr-jni-Merge folder created before, by unzipping the original file.

Then the two different projects must be linked:

1. Right Click on The ONE project > Build Path > Configure Build Path > Left Click on the Source tab > Link Source >browser select the path to the just imported cgr-jni-Merge/src and enter as "folder name" a name at your choice, e.g. srcCGR.
2. Right Click on the new source, e.g. srcCGR, Properties > Native Library > Location /Path/To/cgr-jni-Merge/ion_cgr_jni
3. Right Click on the project cgr-jni-Merge->properties->Java Build Path->Project, add and select the ONE project.

1.3.3 Compiling the cgr-jni-Merge native C code

Native library compilation is carried out as described before but with a different path. From the cgr-jni-Merge/ion_cgr_jni directory\, open the terminal and enter the following command:

make ONE_CLASSPATH=/Path/To/TheOne/bin

where "/Path/To/" is the path to the root directory of the packages containing the .class files obtained as a result of ONE compilation. You can optionally append DEBUG=1 to enable CGR debug prints. Note that Eclipse puts class files in the bin directory and not in the target directory as the ONE script "compile.sh" does.

1.3.4 Running simulations

After compiling the native library, the simulation can be launched either by invoking the one.sh script as before (see 1.2.3), or directly from Eclipse. In the latter case, the same syntax applies, but optional parameters must be entered in a different way: Right Click on TheONE project, Run As -> Run Configurations and select Arguments; now in the box "Program arguments", add the command line arguments, such as batch simulation (-b), and/or additional setting files. As said, you have to follow the same syntax of one.sh (see 1.2.3.2 for details)

For example, to run ONE in GUI mode with the settings specified in defaults_settings.txt file it is enough to launch ONE without specifying any arguments. Vice versa, to run a simulation in batch mode once, just to be faster: Right Click on TheONE project, Run As -> Run Configurations and

select Arguments; now in the box "Program arguments", add command "-b 1". See 1.2.3.2 for further information and other examples.

2 Setting files

The ONE is a very powerful and flexible simulator; the user can select different movement models, different routing algorithms, different classes of nodes, etc. All these settings are specified in configuration files that must be passed as arguments when the simulation is launched. This section aims to help the user familiarize with the ONE configuration files and in particular with settings specific to CGR and OCGR routing classes.

It is worth noting that settings can be passed to The ONE in a differential way, by means of multiple files. General settings should be written in the default settings file, called “default_settings.txt”, while specific settings should be put in specific files. When a file is added, new settings are added, while old settings are overridden by new settings with the same name.

2.1 The ONE default settings

The default settings file, default_settings.txt, contains general settings. Here we will present a fw comments on the most important settings, referring the user to The ONE documentation for a comprehensive information.

The following four lines contain the base settings for the simulation scenario; simulateConnections enable the opening and closing of contacts as a result of node position (two nodes can communicate if they have in common one radio interface and their distance is lower than the coverage radius of the common interface). UpdateInterval indicates, in seconds, the interval between consecutive updates, i.e. the time granularity of the simulation. The end time the simulation length in seconds. The actual simulation time is usually much lower, depending on the complexity of the scenario and the computational power of the machine running The ONE.

```
Scenario.name = default_scenario
Scenario.simulateConnections = true
Scenario.updateInterval = 0.1
Scenario.endTime = 43200
```

Below two transmit interfaces are created, with different types, transmission rates (speed) and ranges. The type of an interface is a java class (ONE package connections), and different types have different setting fields; here the type is *SimpleBroadcastInterface* for both. The transmit speed is in

kBps, so the 250kBps of the btInterface (Bluetooth) correspond to 2Mbps, 10M to 80Mbps; finally, the range is in meters.

```
btInterface.type = SimpleBroadcastInterface
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 50
```

The following lines define the groups of different nodes (hosts) used in the simulation and their settings, three in the example. First, the settings common to all groups unless specifically overwritten are set (“group” settings). Then specific settings (“group$” settings), or settings that override default values, are added for each group. The movement model is a Java class from The ONE package Movement, while the router is the Java class implementing the desired routing algorithm. Note that speeds are in m/s; when two values are given they identify a range; TTL is in minutes. Although in The one each node is uniquely identified by its node number, for convenience this identifier has a prefix to identifies the gruoup it belongs to. In the example we have “p” for pedestrians, “c” for car (note the redefinition of speed for cars).

```
Scenario.nrofHostGroups = 3
Group.movementModel = ShortestPathMapBasedMovement
Group.router = EpidemicRouter
Group.bufferSize = 5M
Group.waitTime = 0, 120
Group.nrofInterfaces = 1
Group.interface1 = btInterface
Group.speed = 0.5, 1.5
Group.msgTtl = 300
Group.nrofHosts = 5
Group1.groupID = p
Group2.groupID = c
Group2.okMaps = 1
Group2.speed = 2.7, 13.9
Group3.groupID = w
```

Below a message generator is defined: the range of the interval between two consecutive messages is in seconds. The range “hosts” contains the subset of nodes that can act as sources and destinations, all others nodes being just relays. In the example, as each group as a cardinality 5, and there are 3 groups, the total

number of nodes is 15 and thus coincides with the source/destination subset. In simple terms, all nodes will generate/receive messages. In The ONE every message has an identifier, every identifier starts with a prefix given by the generator.

Note that the instruction "Events1.hosts = 0, 126" will generate nodes in the range [0,126[i.e. 126 nodes numbered from 0 to 125. This is the numbering internal to ONE, shifted by 1 with the ONE modifications reported in 1.1.3.1

```
Events.nrof = 1
Events1.class = MessageEventGenerator
Events1.interval = 40,60
Events1.size = 50k
Events1.hosts = 0, 126
Events1.prefix = M
```

In The ONE it is possible to set the random generator seed, rngSeed. Different simulation runs will produce exactly the same results starting from the same seed, which is fundamental to assure the reproducibility of results. Then is set the world size and the warmup time (how many seconds nodes will move through the map before starting exchanging messages). The map files used are provided in the ONE folder and are automatically read by the movement model.

```
MovementModel.rngSeed = 1
MovementModel.worldSize = 4500, 3400
MovementModel.warmup = 1000
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
```

Here we define which reports must be created. Report names correspond to class names, as each class generates one report. It is possible to set a "warmup" period (in s) before the actual start of data collection and the default report directory.

```
Report.nrofReports = 3
Report.warmup = 0
Report.reportDir = reports/
Report.report1 = MessageStatsReport
Report.report2 = EventLogReport
Report.report3 = DeliveredMessagesReport
```

The following lines contain some optimization settings included in the default settings file provided by ONE.

```
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
GUI.UnderlayImage.offset = 64, 20
GUI.UnderlayImage.scale = 4.75
GUI.UnderlayImage.rotate = -0.015
GUI.EventLogPanel.nrofEvents = 100
```

2.2 cgr-jni-Merge specific settings

2.2.1 Settings for generating messages with priorities

In cgr-jni-Merge it is possible to generate massages with priorities, which is not possible in standard ONE. Priority messages are necessary to run "priority-aware" routers, such the priority version of Epidemic, CGR and OCGR. They are also compatible with all standard routers, but in this case priority is not enforced. The settings necessary to add three message generators, one for each priority class, i.e. bulk, normal and expedited, are given below. The classes are denoted as 0, 1 and 2 and labelled as B, N and E. Note that the host range must be set in accordance with the general settings.

Note that the instructions "Events*.hosts = 0, 126" will generate nodes in the range [0,126[i.e. 126 nodes numbered from 0 to 125. This is the numbering internal to ONE, shifted by 1 with the ONE modifications reported in 1.1.3.1

```
# Begin settings for priority message generators
Report.report1 = PriorityMessageStatsReport
Events.nrof = 3
Events1.class = PriorityMessageEventGenerator
Events1.interval = 1,160
Events1.size = 100k
Events1.hosts = 0,126
Events1.prefix = B
Events1.priority = 0
Events2.class = PriorityMessageEventGenerator
Events2.interval = 1,255
Events2.size = 100k
```

```
Events2.hosts = 0,126
Events2.prefix = N
Events2.priority = 1
Events3.class = PriorityMessageEventGenerator
Events3.interval = 1,300
Events3.size = 100k
Events3.hosts = 0,126
Events3.prefix = E
Events3.priority = 2
#End settings for priority message generators
```

2.2.2 Settings for Priority Epidemic

The Epidemic router class has been extended to support priorities. Note that related classes do not require native C code. The following settings can be added in a differential way to ONE default settings.

```
Scenario.name = PriorityEpidemic_settings
Group.router = PriorityEpidemicRouter
Report.report1 = PriorityMessageStatsReport
```

2.2.3 Settings for CGR and OCGR

The following settings can be added in a differential way to the ONE default settings to carry out a simulation with CGR or OCGR, without or with priorities. CGR and OCGR classes need native C code, i.e. CGR or OCGR is not simulated but executed. At present they can run only on 64 bit Linux/Unix O.S., as said.

2.2.3.1 ContactGraphRouter settings

Let consider CGR without priorities first. It is necessary to override the scenario name (now CGR_settings) and the router. Third, we have to insert the path to the contact plan used by CGR in route computations.

```
Scenario.name = CGR_settings
Group.router = ContactGraphRouter
ContactGraphRouter.ContactPlanPath = /path/to/contact_plan.txt
```

2.2.3.2 PriorityContactGraphRouter settings

For priority CGR, below, the first three lines are necessary to set the scenario name, select the priority CGR and read the contact plan as before. Because of priority, however, we need to specify

the message report with priority information (fourth line) and the priority message generators. As these settings are the same as for Priority Epidemic router, it is enough to copy the corresponding block of settings.

```
Scenario.name = PCGR_settings
Group.router = PriorityContactGraphRouter
ContactGraphRouter.ContactPlanPath = /path/to/contact_plan.txt
Report.report1 = PriorityMessageStatsReport
#Begin settings for priority message generators
…
#End settings for priority message generators
```

2.2.3.3 OpportunisticContactGraphRouter settings

The following settings can be added in a differential way to carry out simulations with OCGR without or with priorities (OpportunisticContactGraphRouter and PriorityOpportunisticContact GraphRouter respectivly). The first three line are in common to CGR settings. Note, however, that for OCGR it is not strictly necessary to provide an external contact plan. In fact, OCGR adds discovered and predicted contacts to the contact plan autonomously; the external file is necessary only if scheduled contacts are to be provided in the simulation. Then we have to add a specific report. Finally, as this routing algorithm is still in a testing phase, there are some particular features. By setting true the epidemic dropback, if CGR does not manage to find a route for a bundle, as it usually happens at the beginning of a simulation, the bundle is forwarded with epidemic routing. If debug is true, debug information about the current simulation (found routes, bundles sent, etc.) is shown on the console. The last, if true, disable CGR forwarding, which may be convenient only to developers (e.g. to test contact information exchanges between nodes only).

As OCGR at present is much slower than other routers, a significant reduction of nodes (e.g. 8) is suggested (corresponding settings are not reported here).

```
Scenario.name = OCGR
Group.router = OpportunisticContactGraphRouter
ContactGraphRouter.ContactPlanPath = /path/to/contact_plan.txt
Report.report1 = OCGRMessageStatsReport
OpportunisticContactGraphRouter.epidemicDropBack = false
OpportunisticContactGraphRouter.debug = false
OpportunisticContactGraphRouter.preventCGRForward = false
```

2.2.3.4 PriorityOpportunisticContactGraphRouter settings

If PriorityOpportunisticContactGraphRouter is used, the first four lines must be slightly modified to consider the priority version. Then the settings for defining priority message generators must follow. They can be the same already used in Priority Epidemic Routing and Priority CGR .

```
Scenario.name = POCGR
Group.router = POpportunisticContactGraphRouter
ContactGraphRouter.ContactPlanPath = /path/to/contact_plan.txt
Report.report1 = PriorityMessageStatsReport
#Begin settings for priority message generators
…
#End settings for priority message generators
```

2.3 How to generate an ION Contact Plan

CGR routers needs a contact plan. Here, we will explain how to use a class of cgr-jni-Merge to convert a LogFile generated by TheOne into an ION Contact Plan. This class does not use native C code. First , you need to generate the log file; to this end, open "default_settings.txt" and change the Report.report2 as follow:

```
#Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = CPEventLogReport
```

Then save the file and start simulation using only "default_settings.txt".

The report files generated are in the folder "reports", inside theOne folder. The CPEventLogReport has the name of the scenario as a prefix , e.g. “default_scenario_CPEventLogReport.txt”. Now, you have to convert the file. You can run "CPCreator.sh" (in "cgr-jni-Merge" folder) and follow the instructions on the screen. A few comments:

- *Insert full path of java class "ContactPlanCreator.java" and "ContactPlanLine.java"*; these are in the folder /Path/To/cgr-jni-Merge/src, these classes are used to generate the contact plan.
- *Insert full path of the input file for the Contact Plan Creator*; this file, just generated, is in folder "reports", so you have to insert the complete path /Path/To/reports/default_scenario_CPEventLogReport.txt.
- *Insert the output contact plan path*; you have to give the NAME_FILE and the PATH of the ION contact plan: "/Path/To/Just_Generated_CP.txt"

(Remember: /Path/To/ refer to your path, e.g. /home/USER/workspace/...)
Alternatively, if you have cgr-jni-Merge on Eclipse, you can convert the file following this procedure: /src/(default package), then right click on ContactPlanCreator.java, Run As->Java Application.

2.4 How to open and close contacts following an external ION Contact Plan (and not as a result of node movements).

ONE has been designed for opportunistic environments. As that, contacts are normally the results of node movement. To extend the scope of ONE to deterministic environments (e.g. to evaluate CGR), it is however paramount to enforce deterministic contacts. Although this is possible by writing test classes, this is time consuming and quite inconvenient. As an alternative, we have added the possibility to enforce contacts specified in an ION Contact Plan, with minimal limitations.
To make use of this feature, the following additional changes are required to TheONE code:

- *class input.ExternalEventsQueue.java*

  at line 83 replace the StandardEvantsReader with the CPEventsReader. To this end add this line

  ```
  this.reader = new CPEventsReader(eventsFile);
  ```

  and comment with '//' the old one

  ```
  //this.reader = new StandardEventsReader(eventsFile);
  ```

  Ctrl+S to save.

- *class core.NetworkInterface.java*

  The new method transmissionSpeed must be added to this class; to this end, add the lines below in the code of the class:

  ```
  public void setTransmissionSpeed(int transmitSpeed) {
       this.transmitSpeed = transmitSpeed;
  }
  ```

  Ctrl+S to save.

- *class* core.DTNHost.*java*

  This modification consists in replacing the original DTNHost.java class with our version; to this end, delete the original DTNHost.java in TheONE, then **copy** our DTNHost.java version, which is in folder cgr-jni-Merge/simulations/external_cp, into TheONE (package core).

It is of course necessary to recompile ONE and then to pass the list of the wanted contacts to ONE, i.e. the “external contact plan”. An example, external_cp.txt, is provided in the folder cgr-jni-Merge/simulations/external_cp fro the user convenience. It can be added to default settings as usual. Note that the external contact plan is used only to enforce deterministic contacts in TheONE and that being this feature completely independent of CGR, it can be used with whatever router class. Should CGR be selected, these contacts would NOT be automatically passed to CGR, as CGR requires its own contact plan file (see 2.2.3).

See the README.txt file in the folder cgr-jni-Merge/simulations/external_cp for further instructions if necessary.