Installation and use of the “CGR-JNI” packet

CGR-JNI [1], [2] has been developed at University of Bologna in cooperation with Scott Burleigh, NASA-JPL. Its primary aim is to insert CGR [3], [4] [5] and OCGR [5] native C code, taken from the ION DTN implementation [7], in the Java based ONE simulator [8], [9], by means of the Java Native Interface (JNI). It also adds quite a few auxiliary features that can be used independently of (O-)CGR and that are written in Java. These extensions are:

1) support of priorities (message generators with priorities, Epidemic routing [10] with priorities);
2) the possibility of converting a ONE log into a contact plan (in ION format)
3) a new transmission interface with support to unidirectional links;
4) the possibility of enforcing deterministic contacts provided in an external contact plan (in ION format)
5) Support of deterministic message generators; they follow a syntax similar to DTNperf_3 [12]; and are provided in an external message plan file.
6) Possibility of enforcing both deterministic (provided in an external contact plan) and random contacts (due to node motion) at the same time.

This guide is intended for both developers and users. Further information in English can be found in Michele Rodolfi’s master thesis [13] (especially on the C code and the JNI interface), and in Simone Pozza’s thesis [14] (new classes). Giuseppe Tempesta refined the work done by Simone Pozza and latest updates (improved deterministic contacts, deterministic message generators) have been developed by Mattia Mengozzi.

For further information (or bug reporting) contact

Prof. Carlo Caini carlo.caini@unibo.it

Mattia Mengozzi: mattia.mengozzi5@studio.unibo.it

Giuseppe Tempesta: giuseppe.tempesta2@studio.unibo.it

Date: 5 Nov. 2017

Summary

1 Preliminary steps

1.1 Operating system compatibility of the CGR-JNI package

A Unix/Linux O.S. is required to run the native C code called by CGR and OCGR classes contained in CGR-JNI. A 64 bit version should be preferred, as OCGR classes might 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.

1.2 Downloading ONE and CGR-JNI packages

Download the ONE (Opportunistic Network Environment) simulator 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 ONE (e.g. in ~/mySources/one). The parent directory (/one) will contain all original Java classes, setting files and two scripts for compilation and running (one.sh and compile.sh respectively). This guide and the additional code refer to the latest version available at the time of writing (v1.6.0).

Download the CGR-JNI package. Pay attention that there are several branches. Should you opt for the "Merge" one (branch "Merge" of cgr-jni)

https://github.com/alessandroberlati/cgr-jni/tree/Merge

Unzip the file cgr-jni-Merge.zip (e.g. in ~/mySources/cgr-jni). As said, while ONE contains only Java code, CGR-JNI contains both Java and native C code (in /cgr-jni/src and /cgr-jni/ion-cgr-jni, respectively), to be compiled separately.

1.3 Modifications to ONE

Although for the sake of modularity we tried hard to keep the CGR-JNI code as separated as possible from the ONE code, a few changes to the ONE package resulted unavoidable. These changes are described in full details in chapter 6, but, for user convenience, all modifications can be done at once by applying the patch file "the-one_v1.6.0-cgr-jni.patch, in /cgr-jni. The script apply_patch.sh in the same directory helps the user in the patch application. It only requires the path of the one parent directory (e.g. ~/mySources/one)

2 Compiling and launching ONE and CGR-JNI from the command line interface

Java classes (belonging to both ONE and CGR-JNI) and C code (in /cgr-jni/ion-cgr.jni) must be compiled separately. The compilation can be done either from a command line interface or in Eclipse (with support to both C and Java). Here we will refer to the former method, which is easier for plain users. The latter, which may be more convenient to developers, will be explained later.

2.1 Compiling Java classes

2.1.1 Java compiler and JAVA_HOME

To compile both ONE and cgr-jni Java classes a Java compiler ver.8 or above is required; to check your version enter:

```
java -version
```

If you have not any Java compiler or your compiler is too old, download a new version. Otherviwse, you can go on by checking if the environment variable $JAVA_HOME is set:

```
echo $JAVA_HOME
```

This variable is used by the compiler to find the header files of the cgr-jni Java Native Interface. It is usually set to /usr/lib/jvm/java-8-oracle/ but it depends on which Java version is actually installed on your machine., As the compilation fails unless this variable is set, if necessary you can temporarily set the variable with this command after having found your version:

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

As there are many possibilities, to find your path go in /usr/lib/jvm and list all directories inside, and look for something like “java-8-oracle” or “java-1.8.0-openjdk”.

Note that because the $JAVA_HOME setting is temporarily, you must procced with the compilation on the same terminal otherwise you have to set JAVA_HOME again.

2.1.2 Compiling Java classes of ONE

Use the script “compile.sh”, in /one to compile all ONE classes. This script will put the .class files (i.e. the exe in Java) into /one/target, by default.

2.1.3 Compiling Java classes of CGR-JNI

After compiling ONE classes, set these two paths:

```
export ONE_BIN=<yourPathTo>/one/target
```

```
export CGR_JNI_CLASSPATH=<yourPathTo>/cgr-jni/bin
```

Pay attention as before that variables defined by means of export commands are temporary.

Then, run the “compile_cgr-jni.sh” script in /cgr-jni.

2.2 Compiling the native C code

To compile C code the use of command line interface is always recommended. To this end enter the following command from the directory /cgr-jni/ion-cgr-jni, which contains the C code:

```
make ONE_CLASSPATH=<yourPathTo>/one/target
```

You can optionally append DEBUG=1 to enable CGR debug prints.

2.3 Running simulations from a terminal

The ONE provides two ways to perform simulations, the graphic mode and the batch mode. The former makes use of a powerful Graphic User Interface (GUI); the latter is much faster and it 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 script /one/one.sh. Before invoking the script, the LD_LIBRARY_PATH and the CGR_JNI_CLASSPATH environment variables must be set. The first refers to the location of the libcgr_jni.so library, the second to the class files of CGR-JNI

```
export LD_LIBRARY_PATH=<yourPathTo>/cgr-jni/ion_cgr_jni/
export CGR_JNI_CLASSPATH=<yourPathTo>/cgr-jni/bin
```

Pay attention that paths defined by means of export commands are temporary, thus must be re-entered after a reboot. The directory “bin” will be automatically created later by the make, if not already present.

2.3.1 GUI mode

To run ONE in the GUI mode the syntax is.

```
one.sh [settings.txt]
```

For example the command

```
one.sh
```

starts ONE in the GUI mode, with the default settings (in /one/default_settings.txt);

```
one.sh <yourPathTo>/one/example_settings/epidemic_settings.txt
```

does the same, but after reading the default_settings.txt file, the epidemic_settings.txt is read too, (overriding parameters with the same name).

2.3.2 Batch mode

To run ONE in the batch mode the syntax is.

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

For example, the command

```
one.sh –b 1
```

starts ONE in the batch mode, with the default settings (given in /one/default_settings.txt); the advantage is that without the GUI the execution time is much shorter.

```
one.sh –b 7
```

starts ONE in the batch mode, with the default settings for seven times; this can be useful only if in the default setting file we have random.seed=1, in order to have different results at each run. As runs are independent, 7 sets of log files will be created.

```
one.sh –b 1 <yourPathTo>/one/example_settings/epidemic_settings.txt
```

starts ONE in the batch mode once, reading first the default settings and then epidemic_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 mysettings.txt we have

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

the command

```
one.sh –b 7 mysettings.txt
```

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

Compiling and launching ONE and CGR-JNI in Eclipse

For a developer integrating both ONE and CGR-JNI in an IDE, to modify, compile, run and debug it may result more convenient. We will refer here specifically to Eclipse (it is required Eclipse for Java Enterprise Edition with support to C).

Importing a project in Eclipse usually implies that the original sources files are copied inside the Eclipse workspace, as will be shown here. If this is the case, every change made in Eclipse will affect only files inside Eclipse, not the original ones.

From now on, to keep path notations simple, we will assume that the original ONE and CGR-JNI directories are in ~/mySources and the copies in ~/eclipse-workspace. If this is not the case, we let the reader change the paths accordingly.

2.4 Importing ONE in Eclipse

Patch the ONE code as described before in 1.3, unless already done. Then open Eclipse and create a new Java project (new-> project->Java project), called “one”.

Then import the original files

- Click with the right button of the mouse on the just created project directory icon, then import->general->file system, click on browser and choose the directory created before by unzipping the original file (~/mySources/one).

2.4.1 Adding libraries

Add DTNConcsoleConnection.jar and ECLA.jar jar libraries to “one” project:

- click with the right button on the “one” project, Build Path->add External Archives, select the “one”, lib, and select both files

Add Junit4 to the “one” project too:

- click with the right button on the “one” project, Build Path->add Libraries then ->Add library, select Junit, Next select Junit4, Finish 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.

2.5 Importing CGR-JNI in Eclipse and linking

In Eclipse create a new Java project (new-> project->Java project), called for example “cgr-jni”.

Then, import the original files:

- Click with the right button of the mouse on the just created project directory, then import->general->file system, click on browser and choose the original directory created by unzipping the original CGR-JNI file (~/mySources/cgr-jni)

Link the Java source files of the “cgr-jni” project to the “one” project:

- Right Click on the “one” project > Build Path > Configure Build Path > Left Click on the Source tab > Link Source >browser select the path to cgr-jni/src (/eclipse-workspace/cgr-jni/src) and enter as a “directory name” a name at your choice, e.g. srcCGR.

Link the C native source files of the cgr-jni project to TheONE project:

- Right Click on the new source directory, e.g. srcCGR, Properties > Native Library > /eclipse-workspace/cgr-jni/ion_cgr_jni
- Right Click on the project cgr-jni ->properties->Java Build Path->Project, add and select the “one” project.

Add libraries to the “cgr-jni” project:

- click with the right button on the cgr-jni project, Build Path->add Libraries then ->Add library, select Junit, Next select Junit4, Finsh and click apply and then OK.

2.6 Compiling Java classes and native C code

Compile Java classes (unless you have the Project->Build Automatically option checked)

- Right Click on the cgr-jni project and select “build compile”.

Then compile the C code of CGR-JNI. This can be carried out as described before but with a different path, because Eclipse puts the .class files in the “bin” directory and not in the “target” directory as the script “compile.sh” does.
In brief, open a terminal and from the /eclipse-workspace/cgr-jni/ion_cgr_jni directory enter the following command:

```
make ONE_CLASSPATH=<yourPathTo>/eclipse-workspace/one/bin
```

You can optionally append DEBUG=1 to enable CGR debug prints.

2.7 Running simulations in Eclipse

Simulations can be launched either by invoking the “one.sh” script as before (see 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 the “one” 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.

For example, to run ONE in GUI mode with the settings specified in defaults_settings.txt file it is enough to start ONE without specifying any arguments. Vice versa, to run a simulation in batch mode once, just to be faster:

- Right Click on the “one” project, Run As -> Run Configurations and select Arguments; now in the box "Program arguments", add command "-b 1".

See 2.3.2 for further information and other examples.

Note that Eclipse offer the user the possibility of saving the configuration created, so that they can be used later without entering additional settings, which is very convenient.

3 ONE 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 "default_settings.txt" in the /one directory, 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.

3.1 Default_settings.txt

Original default settings are contained in default_settings.txt and are organized into a few sections. As other settings (e.g. for specific routers) are usually set in a differential, it is paramount to have a good knowledge of this file, which can also be used as a template. Here we will present our comments on the most important settings, following the organization of the original default_settings.txt file. The user is referred to The ONE documentation for a comprehensive information. The default_setting.txt file we normally use in our tests is marginally different from the original one. Our values are highlighted (in yellow).

3.1.1 Scenario settings

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

The first line contains the scenario name that will be used in the GUI interface and in logs to identify the scenario in use. For this reason, it is opportune to override it in differential files. *Scenario.simulateConnections* enables/disables 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). *Scenario.updateInterval* the time granularity of the simulation i.e. the interval between consecutive updates; Scenario.endTime the end of simulation (in s). Both times are in seconds and refer to simulated time, not to the actual

duration of the simulation run, which is usually much lower, depending on the complexity of the scenario and the computational power of the CPU in use.

3.1.2 Interface-specific settings:

```
## 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 50
```

This section defines the transmit "interfaces", i.e. the characteristics of radio system supposed in use on nodes. Two "interfaces" are defined here, the *bt* (for Bluetooth) and the *highspeed*. First it must be set the type (i.e. one of the java classes defined in /theONE/src/interfaces), which is *SimpleBroadcastInterface* for both. Then other characteristics, such as transmission rates, (speed, in B/s, thus the settings 250k and 10M correspond to 2 Mbit/s and 80Mbps, respectively) and ranges (in m). Different group of nodes can have different interfaces (e.g. either *bt* or *highspeed*, or both). Communication is possible only between nodes sharing the same interfaces, when their distance is in the range. Note that for interfaces of *SimpleBroadcastInterface* type the transmission rates are independent of the distance when the two nodes are in the range. Other types, such as the *DistanceCapacityInterface*, have a different behaviour.

3.1.3 Group-specific settings:

```
# Define 6 different node groups
Scenario.nrofHostGroups = 6 3

## 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 MapRouteMovement
# routeType: route's type - for MapRouteMovement


# Common settings for all groups
Group.movementModel = ShortestPathMapBasedMovement
Group.router = EpidemicRouter
Group.bufferSize = 5M
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 = 40 5

# 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 = MapRouteMovement
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 = MapRouteMovement
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 = MapRouteMovement
Group6.routeFile = data/tram10.wkt
Group6.routeType = 2
Group6.waitTime = 10, 30
Group6.speed = 7, 10
Group6.nrofHosts = 2
```

This section defines the groups of different nodes (hosts) used in the simulation and their settings. First, the number of groups is set with the *Scenario.nrofHostGroups*, which clearly formally belongs to scenario settings and thus has been likely inserted here for practical convenience by the file authors. Then “the common settings for all groups” sub-section defines the settings that are valid for all groups unless specifically overwritten later (“group” settings). Finally, specific settings (“group$” settings), or settings that override default values, are added for each group. The movement model is a Java class in /theONE/src/movement, while the router is a Java class in /theONE/src/routing . When two values are given, such as in *`Group.speed = 0.5, 1.5`*, (speeds are in m/s) they define a range (the actual value is chosen randomly in this range). Although in The ONE each node has as a unique node number (i.e. two nodes cannot have the same number even if belonging to different groups), for use convenience in reading logs, each group as a prefix label Note that these prefixes are just labels and not unique identifiers as suggested by the name of

the setting *.GroupID*. For example, in the file we have the label "p" (pedestrians) for groups 1 and 3, "c" (car) for group 2 and "t" (trams) for groups 4,5, and 6).

Note that for cars the common (*Group.speed = 0.5, 1.5*) speed range is overridden by *Group2.speed = 2.7, 13.9*, ; while it is not in all other groups as they lack a specific setting.

Each group can have a different movement model, and different constraints.

The total number of nodes is set in an indirect way, being given by the sum of all *Group$.nrofHosts* settings.

Last, let us point out that while speeds are defined in m/s, following SI rules although impractical, the time to live, TTL, is in minutes, which is practical but also inconsistent with other time settings in The ONE, given in seconds. The user must pay attention to these inconsistencies, as unit of measures are never entered in setting lines, thus there is not any control enforced by ONE. Of course, results can be hugely affected by a user misunderstanding on the right unit of measure implicitly required.

3.1.4 Message creation parameters

```
## 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 40,60
# Message sizes (500kB - 1MB)
Events1.size = 500k,1M 50k
# range of message source/destination addresses
Events1.hosts = 0,126 0,126
# Message ID prefix
Events1.prefix = M
```

First of all, it must be stressed that messages are a kind of events, but not all events are messages. Here only one event generator is set (*Events.nrof = 1*). By contrast to group this setting does not belong to the scenario, as "Events" are treated a part. Events$ refer to the different event generators. *Events1.class = MessageEventGeneratorgenerators* specifies that the

first (and sole) event generator is a *MessageEventGeneratorgenerators* class (to be find in TheONE/src/input). Each event generator class has its own parameters. We have *Events1.interval* which here is to be randomly chosen in the [25,35[interval (in s), and analogously for message sizes, and "hosts". The range "hosts" contains the subset of nodes that can act as sources and destinations, all others nodes being just relays. Node that *Events1.hosts = 0,126* means that hosts are in the range [0,126[, i.e. that we have 126 hosts, numbered form 0, 125. Frankly speaking, a peculiar choice. This numbering is preserved in settings, but is shifted by1 in internal parts of the ONE code and in logs when our modifications are applied, to make ONE numbering compatible with the DTN "ipn" scheme, where numbers start from 1. This means that the same identical setting *Events1.hosts = 0,126* will produce node numbers in the range [1,126] in our logs.

3.1.5 Movement model settings

```
## 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
```

In The ONE it is possible to set the random generator seed, e.g. to 1 as in `MovementModel.rngSeed=1`. In this way, different simulation runs will produce exactly the same results as they start from the same seed, which is fundamental to assure the reproducibility of results. Then is set the "world size" (in m) and the "warmup time" (in s), i.e. how many seconds nodes will move through the map before starting any message exchange. The name of map files used must be set starting from The ONE directory (i.e. /data/roads.wkt is in /theONE/data) and are automatically read by the movement model.

3.1.6 Reports

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

```
# how many reports to load
Report.nrofReports = 2 3
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report with
output setting)
Report.reportDir = reports/
# Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = ContactTimesReport
Report.report3 = DeliveredMessagesReport
```

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.

3.1.7 Default settings for some routers settings

```
## Default settings for some routers settings
ProphetRouter.secondsInTimeUnit = 30
SprayAndWaitRouter.nrofCopies = 6
SprayAndWaitRouter.binaryMode = true
```

3.1.8 Optimization settings

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

3.1.9 GUI settings

```
## 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]$
```

4 CGR-JNI settings

In CGR-JNI several examples files referring to new classes are included in the directory /cgr-jni/example_settings. In a few of them, paths to contact plans and/or to message plans are set. In order to avoid absolute path, which would depend on the particular installation, we decided to give these paths only from the root of the example_settings directory. This way it is enough to add your path to /cgr-jni/example_settings to have all the examples working. Alternatively, and likely easier, the user can copy all the content of this directory into "/one/example_settings". This latter solution is simpler because it does not require any change to example files.

Random messages with priorities

In CGR-JNI 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 random message generators, one for each priority class, i.e. bulk, normal and expedited, are given below, assuming that there are not any other events generators of any kind. 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. As said, "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 in logs with The ONE modification reported in 6.2.

```
#Begin settings for priority message random generators and
priority reports
# one message generator for each priority level: Bulk, Normal and
Expedited
Events.nrof = 3
# generator of "Bulk" messages
Events1.class = PriorityMessageEventGenerator
Events1.interval = 1,160
Events1.size = 100k
Events1.hosts = 0,126
Events1.prefix = B
Events1.priority = 0
# generator of "Normal" messages
Events2.class = PriorityMessageEventGenerator
Events2.interval = 1,255
Events2.size = 100k
Events2.hosts = 0,126
Events2.prefix = N
```

```
Events2.priority = 1
# generator of "Expedited" messages
Events3.class = PriorityMessageEventGenerator
Events3.interval = 1,300
Events3.size = 100k
Events3.hosts = 0,126
Events3.prefix = E
Events3.priority = 2
Report.report1 = PriorityMessageStatsReport
#End settings for priority message random generators and priority
reports
```

4.1 Priority Epidemic: priority_epidemic.txt

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
Group.router = PriorityEpidemicRouter

#Begin settings for priority message random gebnerators
….
….
#End settings for priority message random generators

#Reports
Report.report1 = PriorityMessageStatsReport
```

4.2 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. As said, CGR and OCGR classes need native C code, i.e. CGR or OCGR algorithms are not simulated but executed. At present they can run only on (preferentially 64 bit) Linux/Unix O.S.

Contacts must be necessarily passed to CGR by means of a contact.plan file, as usual with CGR. However, it is important to stress that these contacts are not per se enforced by ONE. To have the necessary correspondence there are two possibilities: either to enforce scheduled contacts in ONE as additional events, or to pass a contact plan that corresponds exactly to the pseudo-random contact autonomously enforced by ONE. Both techniques will be shown later.

The contact plan file is optional for OCGR as the opportunistic version adds discovered and predicted contacts to the contact plan autonomously, thus the external file is necessary only if some

scheduled contacts are to be provided in the simulation (e.g. the user wants to simulate an hybrid scenario with both scheduled and opportunistic contacts, as shown later).

4.2.1 ContactGraphRouter

Let us consider CGR without priorities first (/example_settings/cgr/cgr.txt). 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 (but not automatically by ONE as said).

```
Scenario.name = CGR
Group.router = ContactGraphRouter
ContactGraphRouter.debug = false
ContactGraphRouter.ContactPlanPath =
     example_settings/cgr/contact_plan.txt
```

4.2.2 PriorityContactGraphRouter

For priority CGR (/example_settings/cgr/pcgr.txt), 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 random generators. As these settings are the same as for Priority Epidemic router, it is enough to copy the corresponding block of settings.

```
Scenario.name = PriorityCGR
Group.router = PriorityContactGraphRouter
ContactGraphRouter.debug = false
ContactGraphRouter.ContactPlanPath =
     example_settings/cgr/contact_plan.txt

#Begin settings for priority message random generators
…
#End settings for priority message random generators
#Reports
Report.report1 = PriorityMessageStatsReport
```

4.2.3 OpportunisticContactGraphRouter:

The following settings (in /example_settings/ocgr/ocgr.txt) 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. As said, for OCGR it is not strictly necessary to provide an external contact plan. 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 when *true* disables CGR forwarding, which may be convenient only to developers (e.g. to verify contact information exchanges between nodes).

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

```
Scenario.name = OpportunisticCGR
Group.router = OpportunisticContactGraphRouter
OpportunisticContactGraphRouter.epidemicDropBack = true
OpportunisticContactGraphRouter.preventCGRForward = false
OpportunisticContactGraphRouter.debug = false
#Set your path to the contact plan (optional in OCGR; decomment if
you want to provide it)
#OpportunisticContactGraphRouter.ContactPlanPath =
/yourPathTo/contact_plan.txt
#Reports
Report.report2 = OCGRMessageStatsReport
```

4.2.4 PriorityOpportunisticContactGraphRouter

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

```
Scenario.name = PriorityOpportunisticCGR
Group.router = PriorityOpportunisticContactGraphRouter
OpportunisticContactGraphRouter.epidemicDropBack = true
OpportunisticContactGraphRouter.preventCGRForward = false
OpportunisticContactGraphRouter.debug = false
ContactGraphRouter.debug = false

#Set your path to the contact plan (optional with OCGR; decomment
if you want to provide it)
#ContactGraphRouter.ContactPlanPath = /yourPathTo/contact_plan.txt

#Begin settings for priority message random generators and
priority reports
….
#End settings for priority message random generators

#Reports
Report.report1 = PriorityMessageStatsReport
```

4.3 Conversion of a ONE log into an ION contact plan

CGR routers needs a contact plan (an external file in ION format). Here, we will explain how to use a class of CGR-JNI to convert a LogFile generated by ONE into an ION contact plan, which can have multiple uses, but in particular to let CGR know in advance the "pseudo" random contacts that will be generated by ONE. This class does not use native C code.

More precisely, the contact plan converter consists of a stand - alone Java application including two classes, *ContactPlanConverter.java* and *ContactPlanLine.java*. The former class creates an ION contact plan file starting from a ONE report file, namely the *CPEventLogReport* created by ONE after a simulation.

In order to let CGR know in advance the "pseudo" random contacts generated by ONE, it is necessary to run the same simulation twice, with the same seed. The former run just to collect the pseudo random contact data; the latter to carry out the wanted CGR simulation, with CGR provided with a contact plan that corresponds exactly to pseudo random contacts enforced by ONE. Of course, it is crucial that the two simulations have the same configuration and seed. This process is shown below.

First, it is necessary to generate a suitable ONE log file; to this end, open "default_settings.txt" and change the Report.report2 from `Report.report2 = ContactTimesReport` to `Report.report2 = CPEventLogReport`

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

The report files generated are saved in the /one/reports directory. The CPEventLogReport has the name of the scenario as a prefix, e.g. “default_scenario_CPEventLogReport.txt”. Now, we need to convert this file, by running /cgr-jni/CPConverter.sh and following the instructions on the screen. A few comments:

- *Insert the directory (full path) of java classes "ContactPlanConverter.java" and "ContactPlanLine.java"*; these are in the directory /cgr-jni/src, these classes are used to generate the contact plan.
- *Insert full path of the input file for the Contact Plan Converter*;

e.g. <yourPathTo>/one/reports/default_scenario_CPEventLogReport.txt.

- *Insert the output contact plan path*;

  e.g.: <yourPathTo>/cgr-jni/example_settings/cgr/Converted_CP.txt"

In Eclipse, you can call the converter by right clicking on ContactPlanConverter.java, Run As->Java Application.

4.4 Deterministic contacts

ONE has been designed for opportunistic environments only. 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 scheduled contacts specified in a contact plan file in ION format, with minimal limitations.

4.4.1 Basic settings

To enforce deterministic contacts it is necessary to insert an additional event generator and pass the contact plan file to ONE. A few optional settings may come first, as in the example below (/example_settings/deterministic/deterministic_contacts.txt).

First a specific scenario name is set; then we disable the establishment of contacts as a result of node position (if two nodes are close enough a contract is established), to prevent ONE from creating any contacts that are not included in the contact plan. Then we inhibit the movement of nodes, by selecting the "stationaryMovement". This also implies that al nodes will be located on the same point on the map (e.g. 0,0).

After these optional settings, we have added one event generator (the second) to default settings, specified its class and finally given the contact plan file path.

```
## Start of Ancillary settings
Scenario.name = Deterministic_Contacts
# This disable contacts as a result of distance between nodes
Scenario.simulateConnections = false
# This disable node movemnt for all nodes
Group.movementModel = StationaryMovement
#All nodes will be colocated on the map at the origin (no
possibility to distibuish them on the map)
Group.nodeLocation = 0,0
## End of Ancillary settings
## Start of Core settings
Events.nrof = 2
Events2.class = ExternalEventsQueue
Events2.filePath =
example_settings/deterministic/contact_plan_det.txt
## End of Core settings
```

More complex examples are included in the directory /example_settings/cgr/ASMS2014 as the normal situation is that the same contact plan file is passed both to CGR and to ONE (by means of

independent settings). Note, however, that the use of deterministic contacts is not limited to CGR experiments. In fact, they can be used jointly with whatever router class.

4.5 Deterministic message generators

In ONE not only are contacts random, but also messages. To extend the scope of ONE to deterministic environments (e.g. to evaluate CGR), but also to have the possibility of verifying the correct implementation of other routers, it is necessary to add the possibility of generating deterministic messages.

4.5.1 The message plan (syntax of message generators)

As for deterministic contacts, deterministic message generators must be set in an external file (the message plan). Each line of the message plan defines a deterministic message generator, following a syntax that is deliberately like that of the client of the DTNperrf_3 tool, included in ION (/contrib).

The syntax is

```
start_time <s> -s <source> -d <destination> <[-T <s>|-D <num>]> -P <num>-
R <rate> [options]
```

Compulsory parameters are:

- Start_time: the offset (in s) from the simulation start,
- source and destination: the node numbers corresponding to these;
- -T (--time): defines the duration of message generation (in s);
- in alternative to –T, -D defines the total amount of data to be generated before termination. The value must be followed by either the postfix B (byte), k (kilobyte) o M (megabyte).
- -R (--rate): is the message generation rate; the value must be followed by either the postfix b (to (bundles/s), or k (kbit/s) or M (Mbit/s).

Optional parameters are:

- -P (--payload): the bundle payload dimension; The value must be followed by the same postfixes as Data (B,k,M). The default is 50 kB
- -p (or --priority): bundle priority [0|1|2]; the default is 0.

E.g.

```
10 -s 1 –d 4 –D 200k – R 1.5b –P 100k –p2
```

Starts a deterministic generator at 10s from the simulation beginning; bundles (messages in ONE trminology) are generated by node 1 and are to be delivered to node 4; message generation will terminate after sending 200kB of data; bundles are sent at a rate of 1.5 bundle per second, have a payload of 100k and have priority 2 (expedited).

4.5.2 Basic settings

Let us consider a very simple example here , where default random generators are replaced by deterministic ones (/example_settings/deterministic/deterministic_messages.txt). First the default_settings.txt Events1.class must be overridden, so that the DMExternalEventsQueue (Deterministic Message External Event Queue) is set. Then we have to insert the path to the message plan and finally to set to D (Deterministic) the prefix.

```
Scenario.name = Deterministic_Messages
## Start of Core settings
Events1.class = DMExternalEventsQueue
Events1.filePath =
example_settings/deterministic/message_plan_det.txt
Events1.prefix = D
## End of Core settings
```

5 Advanced CGR-JNI settings: example scenarios

5.1 ASMS2014: CGR with deterministic contacts and messages

This scenario aims at showing the use of CGR-JNI for the evaluation of CGR. The scenario is deliberately fully deterministic (fixed nodes, external contacts, deterministic message generators) and is a reply of a couple of tests presented in [5]. We have four nodes, 1-4; 1 is the source, 4 the destination, 2 and 3 two intermediate nodes. 1 is connected to 2 and 3 by means of space intermittent links; 2 and 3 to node 4 by means of terrestrial continuous links. We will consider two tests.

5.1.1 Contact plan

In both tests the contacts are the same. They are given in contactPlanASMS2014.txt:

```
a range   +1   +3600     1    2    1
a contact      +60  +80  1    2    64000
a contact      +60  +80  2    1    64000
a range   +1   +3600     1    3    1
a contact      +30  +90  1    3    16000
a contact      +30  +90  3    1    16000
a range   +1   +3600     1    3    1
a contact      +105 +135 1    3    16000
a contact      +105 +135 3    1    16000
a range   +1   +3600     2    4    1
a contact      +1   +3600     2    4    1250000
a contact      +1   +3600     4    2    1250000
a range   +1   +3600     3    4    1
a contact      +1   +3600     3    4    1250000
a contact      +1   +3600     4    3    1250000
```

Note the nesting of contact 1-2 into the first 1-3 contact, a situation particularly challenging for CGR. (see [5]).

5.1.2 First test: cgr_settings_ASMS2014.txt

In the first text 14 bundles of 100kB are generated, all with the same priority (e.g. 0). The messagePlanASMS2014.txt contains only the following line:

```
+1 -s 1 -d 4 -D 1500k -p 0 -R 1b -P 100k
```

The ONE settings, in the file cgr_settings_ASMS2014.txt, are reported below. Note that in order to have the four fixed nodes in different positions on the map, we defined 4 group of hosts, each consisting of one host only. This is very convenient to check graphically the correct opening and closing of deterministic contacts given in the contact plan.

```
# Scenario settings
Scenario.name = CGR_ASMS_2014
Scenario.simulateConnections = false
Scenario.updateInterval = 0.1
Scenario.endTime = 140

#Intefaces
AsymmetricInterface.type = SimpleAsymmetricInterface
AsymmetricInterface.transmitSpeed = 500k
AsymmetricInterface.transmitRange = 200

#Group common
# Fixed nodes must belong to different groups to be
distinguishable on
# the map.
Scenario.nrofHostGroups = 4
#Group.router = PriorityContactGraphRouter
Group.router=ContactGraphRouter
ContactGraphRouter.ContactPlanPath =
example_settings/cgr/ASMS2014/contact_plan_ASMS2014.txt
ContactGraphRouter.debug = false

#Group specific
Group1.nrofHosts = 1
Group2.nrofHosts = 1
Group3.nrofHosts = 1
Group4.nrofHosts = 1
Group1.groupID = h
Group1.nrofInterfaces = 1
Group2.groupID = h
Group2.nrofInterfaces = 1
Group3.groupID = h
Group3.nrofInterfaces = 1
Group4.groupID = h
Group4.nrofInterfaces = 1
Group1.interface1 = AsymmetricInterface
Group2.interface1 = AsymmetricInterface
Group3.interface1 = AsymmetricInterface
Group4.interface1 = AsymmetricInterface
Group1.movementModel = StationaryMovement
Group1.nodeLocation = 2000,1000
Group2.movementModel = StationaryMovement
Group2.nodeLocation = 1000,2000
Group3.movementModel = StationaryMovement
Group3.nodeLocation = 3000,2000
Group4.movementModel = StationaryMovement
Group4.nodeLocation = 2000,3000

#Events
Events.nrof = 2
```

```
#Deterministic contacts
Events1.class = CPExternalEventsQueue
Events1.filePath =
example_settings/cgr/ASMS2014/contact_plan_ASMS2014.txt
#Deterministic messages
Events2.class = DMExternalEventsQueue
#Events2.filePath =
example_settings/cgr/ASMS2014/message_plan_ASMS2014_priority.txt
Events2.filePath =
example_settings/cgr/ASMS2014/message_plan_ASMS2014.txt
Events2.prefix = D

#Reports
#Report.report1 = PriorityMessageStatsReport
Report.report1 =MessageStatsReportatsReport
```

5.1.3 Second test: pcgr_settings_ASMS2014.txt

The latter test is the same as the former, but the last 4 bundles have a higher priority (e.g.2). To carry out this second test we simply need to slightly modify the message plan. New setting, in message_plan_ASMS2014_prioririty.txt are:

```
+1 -s 1 -d 4 -D 1100k -p 0 -R 1b -P 100k
+20 -s 1 -d 4 -D 400k -p 2 -R 1b -P 100k
```

The setting file pcgr_settings_ASMS2014.txt is the same as the previous one, but the call to priority versions of CGR, message plan and reports.

5.2 Hybrid: OCGR with both deterministic and random contacts

These last two examples (in /example_settings/ocgr/hybrid/) are the most complex one, as many additional classes of CGR-JNI are used together in an orchestrated way. The aim is to show how opportunistic and deterministic components can be used to evaluate the OCGR router [6] and to compare it with Epidemic [10] and ProPHET [11]. The deterministic component consists of 6 fixed host ("h"); the random component by a group of ten cars, identified by the letter "c". Each car moves randomly, and has its own random message generator. They emit one bundle (message) of 100kB every 40-60 s; the destination is always another car. Radio interfaces of all nodes have a Tx rate of 500kB/s and a radius of 200m.

5.2.1 Contact plan

Each host is connected (symmetrically) with the next host (e.g. 1-2-3-4-5-6) for all the test duration.

```
a range   +1   +21600    1    2    1
a contact      +1   +21600    1    2    1000000
a contact      +1   +21600    2    1    1000000
a range   +1   +21600    2    3    1
```

```
a contact      +1   +21600    2    3    1000000
a contact      +1   +21600    3    2    1000000
a range   +1   +21600    3    4    1
a contact      +1   +21600    3    4    1000000
a contact      +1   +21600    4    3    1000000
a range   +1   +21600    4    5    1
a contact      +1   +21600    4    5    1000000
a contact      +1   +21600    5    4    1000000
a range   +1   +21600    5    6    1
a contact      +1   +21600    5    6    1000000
a contact      +1   +21600    6    5    1000000
```

5.2.2 Test: ocgr_hybrid.txt

In the ocgr_hybrid test there are no priorities and all generators are random. Contacts are planned for hosts, which are fixed, (given by mans of an external contact plan) and random for cars, which are moving. We can also have opportunistic contacts between cars and hosts.

The differential instructions that must be set to override and complement the default settings are many, given the complexity of this test.

```
#Scenario
Scenario.name = Simulation_OCGR_hybrid
Scenario.endTime = 21600
Scenario.nrofHostGroups = 7

#Interface
AsymmetricInterface.type = SimpleAsymmetricInterface
AsymmetricInterface.transmitSpeed = 500k
AsymmetricInterface.transmitRange = 200

#Group (common)
#Router selection (alternatively OCGR, Prophet, or Epidemic; only
OCGR requires the contact plan):
Group.router = OpportunisticContactGraphRouter
ContactGraphRouter.ContactPlanPath =
example_settings/ocgr/hybrid/contact_plan_hybrid.txt
OpportunisticContactGraphRouter.epidemicDropBack = true
OpportunisticContactGraphRouter.preventCGRForward = false
OpportunisticContactGraphRouter.debug = false

#Group.router = ProphetRouter
#Group.router = EpidemicRouter

#Group (specific)
#Group of nodes definition (6 groups for the 6 fixed hosts, one
group for 10 cars). It is worth stressing that nodes inserted in
the contact plan
#must have the same radio interface in position 1.
```

```
Group1.nrofHosts = 1
Group2.nrofHosts = 1
Group3.nrofHosts = 1
Group4.nrofHosts = 1
Group5.nrofHosts = 1
Group6.nrofHosts = 1
Group7.nrofHosts = 10

Group1.groupID = h
Group1.nrofInterfaces = 1
Group2.groupID = h
Group2.nrofInterfaces = 1
Group3.groupID = h
Group3.nrofInterfaces = 1
Group4.groupID = h
Group4.nrofInterfaces = 1
Group5.groupID = h
Group5.nrofInterfaces = 1
Group6.groupID = h
Group6.nrofInterfaces = 1
Group7.groupID = c

Group1.interface1 = AsymmetricInterface
Group2.interface1 = AsymmetricInterface
Group3.interface1 = AsymmetricInterface
Group4.interface1 = AsymmetricInterface
Group5.interface1 = AsymmetricInterface
Group6.interface1 = AsymmetricInterface
Group7.interface1 = AsymmetricInterface

#Node movement or position; note that in order to locate 6 fixed
nodes on different point of the map (cross-roads), it was
necessary to define
#6 different groups, which is clearly cumbersome; new ONE methods
should be defined to make simpler the fulfilment of this basic
need:
Group1.movementModel = StationaryMovement
Group1.nodeLocation = 1535,353
Group2.movementModel = StationaryMovement
Group2.nodeLocation = 1484,748
Group3.movementModel = StationaryMovement
Group3.nodeLocation = 1146,1489
Group4.movementModel = StationaryMovement
Group4.nodeLocation = 2029,1805
Group5.movementModel = StationaryMovement
Group5.nodeLocation = 2540,1622
Group6.movementModel = StationaryMovement
Group6.nodeLocation = 2495,995
Group7.movementModel = ShortestPathMapBasedMovement
Group7.okMaps = 1
```

```
Group7.speed = 2.7, 13.9

#Events
Events.nrof = 2
#Determinitic contacts
Events1.class = CPExternalEventsQueue
Events1.filePath =
example_settings/ocgr/hybrid/contact_plan_hybrid.txt
#Random message generators
Events2.class = MessageEventGenerator
Events2.interval = 40,60
Events2.size = 100k
Events2.hosts = 6,16
Events2.prefix = M

#Reports
Report.report1 = MessageStatsReport
#Report.report1 = OCGRStatsReport
```

5.2.3 Test: pocgr_hybrid.txt

The pocgr_hybrid test is a variant of the previous one. There are two main differences: first, the use of the Priority version of the OCGR router; second, the presence of deterministic message generators (provided in an external message plan. As before, contacts are planned for hosts, which are fixed, (given by mans of an external contact plan) and random for cars, which are moving. We can also have opportunistic contacts between cars and hosts. Below we report only the parts that are changed.

```
#Scenario
Scenario.name = Simulation_PriorityOCGR_hybrid

#Group (common)
#Router selection (alternatively POCGR, Prophet, or Epidemic; only
POCGR requires the contact plan):
Group.router = PriorityOpportunisticContactGraphRouter
…
#Events
Events.nrof = 3
#Determinitic contacts
Events1.class = CPExternalEventsQueue
Events1.filePath =
example_settings/ocgr/hybrid/contact_plan_hybrid.txt
#Determinitic messages
Events2.class = DMExternalEventsQueue
Events2.filePath =
example_settings/ocgr/hybrid/message_plan_hybrid.txt
Events2.prefix = D
```

```
#Random messages (for cars only)
Events3.class = PriorityMessageEventGenerator
Events3.interval = 40,60
Events3.size = 10k
Events3.hosts = 6,16
Events3.prefix = M
Events3.priority = 1
```

6 List of modifications to The ONE original code

All changes to be introduced into the ONE package are listed below. For the user convenience, all modifications can be done at once by applying the patch file "the-one_v1.6.0-cgr-jni.patch, in /cgr-jni. The script apply_patch.sh in the same directory helps the user in the patch application. It only requires the path of the one parent directory (e.g. ~/mySources/one)

6.1 Adding paths in the file one.sh

To allow the JVM to find the CGR-JNI Java classes, it is necessary to modify a command in one.sh.

In the file one.sh, add the parts in bold in the right order (note that the character ":" works as a separator)

```
java -Xmx512M -Djava.library.path=$LD_LIBRARY_PATH -cp
bin:target:lib/ECLA.jar:lib/DTNConsoleConnection.jar:$CGR_JNI_CLAS
SPATH core.DTNSim $*.
```

6.2 Host numbering from 1

As in ION the node numbering starts from 1, it is necessary to shift ONE numbering, which starts from 0, of one position. To this end, the ONE code needs to be changed as follows:

```
class core.DTNHost
22   private static int nextAddress = 1; //Instead of 0
107  nextAddress = 1 //Instead of 0
```

Note that in practice ONE will continue to 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])

6.3 The getMoreInfo() method

To add further information to logs, it was necessary to add the getMoreInfo() method to the class report.MessageStatsReport. 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  }
```

6.4 Modifications related to contact or message plans

To enforce the Tx speed specified in an external contact plan, the following change is required:

- *class core.NetworkInterface.java*

  The new method setTransmissionSpeed must be added to this class in order to allow the CPConnectionEvent to set the transmission speed required by a specific contact in the contact plan (in ION format):

```
517  public void setTransmissionSpeed(int transmitSpeed) {
518        this.transmitSpeed = transmitSpeed
519  }
```

To enforce the reading of contact and message plans, the following change is required:

- *class input.EventQueueHandler.java*

  In the original version of this class an if clause force the use of the ExternalEvantsQueue if a path is specified. As for reading the contact or the message plans different event queues must be selected, this if clause must be deleted from the code by commenting the following lines (add parts in bold):

```
66   /*if (s.contains(PATH_SETTING)) { // external events file
67        int preload = 0;
68        String path = "";
69        if (s.contains(PRELOAD_SETTING)) {
70             preload = s.getInt(PRELOAD_SETTING);
71        }
72        path = s.getSetting(PATH_SETTING);
73   queues.add(new ExternalEventsQueue(path, preload));
74   }
75   else */ if (s.contains(CLASS_SETTING)) {
```

References

[1] A. Berlati, S. Burleigh, C. Caini, F. Fiorini, J. J. Messina, S. Pozza, M. Rodolfi, G. Tempesta, “Implementation of (O-)CGR in The ONE”, in Proc. of IEEE SMC-IT, Alcalá de Henares, Spain, Sept. 2017, pp. 1 - 4,

[2] CGR-JNI (Merge branch) download site: https://github.com/alessandroberlati/cgr-jni/tree/Merge

[3] G. Araniti, N. Bezirgiannidis, E. Birrane, I. Bisio, S. Burleigh, C. Caini, M. Feldmann, M. Marchese, J. Segui, K. Suzuki. “Contact Graph Routing in DTN Space Networks: Overview, Enhancements and Performance”, IEEE Commun. Mag., Vol.53, No.3, pp.38-46, March 2015.
[4] CCSDS “Schedule-aware bundle routing”, CCSDS White Book, May 2017, work in progress.
[5] N. Bezirgiannidis, C. Caini, D.D. Padalino Montenero, M. Ruggieri, V. Tsaoussidis, “Contact Graph Routing Enhancements for Delay Tolerant Space Communications”, in Proc. of ASMS 2014, Livorno, Italy, Sept. 2014, pp. 17-23. DOI: 10.1109/ASMS-SPSC.2014.6934518
[6] S. Burleigh, C. Caini, J. J. Messina, M. Rodolfi, “Toward a Unified Routing Framework for Delay-Tolerant Networking”, in Proc. of IEEE WiSEE 2016, Aachen, Germany, Sept. 2016, pp. 82 - 86, DOI:
[7] S. Burleigh, “Interplanetary overlay network design and operation V3.3.1,” JPL D-48259, Jet Propulsion Laboratory, California Institute of Technology, CA, May 2015. [Online]: http://sourceforge.net/projects/ion-dtn/files/latest/download
[8] A. Keränen, J. Ott, and T. Kärkkäinen, “The ONE Simulator for DTN Protocol Evaluation”, in Proceedings of the 2nd International Conference on Simulation Tools and Techniques. New York, NY, USA: ICST, 2009,
[9] The ONE web site: https://akeranen.github.io/the-one/
[10] "Epidemic Routing for Partially-Connected Ad Hoc Networks," Amin Vahdat and David Becker, Duke Technical Report CS-2000-06, July 2000.
[11] A. Lindgren, A. Doria E. Davies, and S. Grasic, “Probabilistic Routing Protocol for Intermittently Connected Networks”, Internet RFC 6693, Aug. 2012 .
[12] C. Caini, A. d’Amico and M. Rodolfi, “DTNperf_3: a Further Enhanced Tool for Delay-/Disruption- Tolerant Networking Performance Evaluation”, in Proc. of IEEE Globecom 2013, Atlanta, USA, December 2013, pp. 3009 - 3015. DOI: 10.1109/GLOCOM.2013.6831533
[13] M. Rodolfi’ “DTN discovery and routing: from space applications to terrestrial networks”, DISI, University of Bologna, March 2016 http://amslaurea.unibo.it/10361/1/DTN_discovery_and_routing.pdf
[14] S. Pozza, “New Routing Classes for the ONE (Opportunistic Network Environment)) Simulator”, DISI, University of Bologna, Oct. 2016, http://amslaurea.unibo.it/12150/1/pozza_simone_tesi.pdf