COSSIM SIMULATION FRAMEWORK

User Guide

COSSIMUG (v1.2) – March 10, 2023

COSSIM

COSSIM Simulation Framework is the outcome of the EU Research Project H2020-644042-COSSIM (www.cossim.org). This document details how to install and use the framework.

COSSIM Simulation Framework extends and interconnects established simulators and open source packages. Please check licenses for all packages distributed by COSSIM as well as packages that are not directly distributed by COSSIM.

The following table shows the revision history for this document.

Date	**Ver.**	**Author/Editor/Contributor**	**Summary of main changes**
March 30, 2017	1.0	Nikolaos Tampouratzis	First completed version of the user guide (COSSIM internal)
January 15, 2018	1.0.1	Nikolaos Tampouratzis	Adaptation for public release
January 30, 2018	1.1	Andreas Brokalakis	Document redesign and release
May 10, 2023	1.2	Nikolaos Tampouratzis	Update COSSIM with latest GEM5 and CERTI 4.0.0

Table of Contents

CHAPTER 1

1. COSSIM Installation Guide

This chapter provides instructions for installing the whole COSSIM framework. The current version of COSSIM has been tested and verified in the following Linux distributions:

- **Ubuntu 20.04 LTS**

1.1. Installing the Prerequisite Packages

COSSIM requires several packages to be installed on the computer. These packages include the C++ compiler (gcc or clang), the Java runtime, and several other libraries and programs. These packages can be installed from the software repositories of your Linux distribution. COSSIM requires at least 25GB of available disk and 8GB of RAM.

1.1.1. Ubuntu distribution 20.04

Before starting the installation, refresh the database of available packages. Type in the terminal:

- `$sudo apt-get update`

To install the required packages, verify that you have installed compatible C/C++ compiler and type in the terminal:

- `$sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python3-dev python3-six python-is-python3 libboost-all-dev pkg-config`
- `sudo apt install cmake bison flex libxml2-dev libstdc++-12-dev`
- `sudo add-apt-repository ppa:rock-core/qt4 (https://ubuntuhandbook.org/index.php/2020/07/install-qt4-ubuntu-20-04/)`
- `sudo apt install openjdk-8-jdk openjdk-8-jre tcl-dev tk-dev qt4-qmake libqt4-dev libqt4-opengl-dev openmpi-bin libopenmpi-dev clang`

1.2. Manual installation

Please read the following steps to install manually the required COSSIM subparts.

1.2.1. Clone the COSSIM Project

Create a folder in `$HOME` directory (with name: *COSSIM*) and put all COSSIM components there.

- `$mkdir $HOME/COSSIM`
- `$cd $HOME/COSSIM`
- `$git clone https://github.com/H2020-COSSIM/cgem5`
- `$git clone https://github.com/H2020-COSSIM/cCERTI`

- `$git clone https://github.com/H2020-COSSIM/OMNETPP_COSSIM_workspace`
- `$git clone https://github.com/H2020-COSSIM/cMcPAT`
- `$git clone https://github.com/H2020-COSSIM/COSSIM_GUI`

Move the cMcPAT in the correct location

- `$mv -f $HOME/COSSIM/cMcPAT/ $HOME/COSSIM/cgem5/McPat`

1.2.2. cCERTI & Our SynchServer Installation

This subsection provides instructions for installing CERTI HLA as well as our SynchServer implementation.

- Prepare a separate build directory and run cmake
  - `$cd $HOME/COSSIM/cCERTI`
  - `$mkdir build_certi`
  - `$cd build_certi`
  - `$cmake -DCMAKE_INSTALL_PREFIX=$HOME/COSSIM/cCERTI/build_certi $HOME/COSSIM/cCERTI`
- Compile & install the CERTI
  - `$make`
  - `$make install`
- Install SynchServer
  - `$cd $HOME/COSSIM/cCERTI/SynchServer`
  - `$./build.sh`
- Copy the Federation file to appropriate directory
  - `$cd $HOME/COSSIM/cCERTI`
  - `$cp Federation.fed $HOME/COSSIM/cCERTI/build_certi/share/federations`
- Export the cCERTI variables in .bashrc file
  - `$echo "#cCERTI exports" >> ~/.bashrc`
  - `$echo "export CERTI_HOME=$HOME/COSSIM/cCERTI/build_certi" >> ~/.bashrc`
  - `$echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/COSSIM/cCERTI/build_certi/lib" >> ~/.bashrc`
  - `$echo "export PATH=$HOME/COSSIM/cCERTI/build_certi:$PATH" >> ~/.bashrc`
  - `$echo "export PATH=$HOME/COSSIM/cCERTI/build_certi/bin:$PATH" >> ~/.bashrc`
  - `$echo "export CERTI_SOURCE_DIRECTORY=$HOME/COSSIM/cCERTI" >> ~/.bashrc`
  - `$echo "export CERTI_BINARY_DIRECTORY=$HOME/COSSIM/cCERTI/build_certi" >> ~/.bashrc`
  - `$echo "#change CERTI_HOST if you want to use HLA Server (rtig) and SynchServer from another machine" >> ~/.bashrc`

- `$echo "export CERTI_HOST=127.0.0.1" >> ~/.bashrc`

1.2.3. cgem5 Installation

This subsection provides instructions for installing COSSIM-GEM5 with our images & libraries.

- Build the gem5 for both ARM & X86 ISA (-j<number> is the *number* of physical cores of your machine)
    - `$cd $HOME/COSSIM/cgem5`
    - `$source ~/.bashrc`
    - `$scons build/ARM/gem5.fast -j4`
    - `$scons build/X86/gem5.fast -j4`

- Download GEM5 images and kernels from here: http://kition.mhl.tuc.gr:8000/f/ee638f25d1/
- Untar kernels and images in `$HOME/COSSIM` directory
    - `$cd $HOME/COSSIM`
    - `$tar -xf kernels.tar.xz`

- Export the cCERTI variables in .bashrc file
    - `$echo "#GEM5 exports" >> ~/.bashrc`
    - `$echo "export GEM5=$HOME/COSSIM/cgem5" >> ~/.bashrc`
    - `$echo "export M5_PATH=$HOME/COSSIM/kernels" >> ~/.bashrc`

1.2.4. cMcPAT Installation

- `$cd $HOME/COSSIM/cgem5/McPat/mcpat`
- `$make all`
- `$cd $HOME/COSSIM/cgem5/McPat/Scripts`
- `$chmod +x GEM5ToMcPAT.py`
- `$chmod +x print_energy.py`

1.2.5. cOMNET++ with COSSIM WORKSPACE Installation

Download OMNeT++ 5.0 from http://omnetpp.org. Make sure you select to download the generic archive, `omnetpp-5.0-src.tgz`.

- Untar it in the $HOME directory & execute the following commands:
    - `$cd $HOME`
    - `$tar xvfz omnetpp-5.0-src.tgz`
    - `$cd omnetpp-5.0`
    - `$export PATH=$PATH:$HOME/omnetpp-5.0/bin`
    - `$./configure && make`

- Create a simulations folder in HLANode

    - `mkdir $HOME/COSSIM/OMNETPP_COSSIM_workspace/OMNET_WORKSPACE/HLANode/simulations`

- Move the GUI files (2 .jar files) in `$HOME/omnetpp-5.0/ide/dropins` directory
    - `$cp -R $HOME/COSSIM/COSSIM_GUI/* $HOME/omnetpp-5.0/ide/dropins`

- Export the OMNET++ variables in .bashrc file
    - `$echo "export PATH=$PATH:$HOME/omnetpp-5.0/bin" >> ~/.bashrc`
    - `$echo "export OMNETWP=$HOME/COSSIM/OMNETPP_COSSIM_workspace/OMNET_WORKSPACE" >> ~/.bashrc`
    - `source ~/.bashrc`

- Execute OMNET++ and import the COSSIM-OMNET workspace (it contains INET-3.2.4 version)
    - `$omnetpp`
    - `Select Browse -> $HOME -> COSSIM -> OMNETPP_COSSIM_workspace -> OMNET_WORKSPACE -> Press "OK"`

- Build OMNET_WORKSPACE
    - Select Project -> Clean -> Select “INET” -> Select “Start a build immediately” -> Select “Build only the selected projects” -> Press “OK”
    - Select Project -> Clean -> Select “HLANode” & “test” -> Select “Start a build immediately” -> Select “Build only the selected projects” -> Press “OK”

1.3. COSSIM installation for distributed systems (Optional)

To execute gem5 instances in different physical machines (to extract parallelism from distributed systems), the user should install the cGEM5 (Section 1.1.4) as well as the CERTI & SynchServer (Section 1.1.3) in node with **static IP** so that it is accessible from cOMMET++ (and may from cGEM5 localhost instances). Finally, the user should define the static IP in *CERTI_HOST* variable in *.bashrc* file instead of localhost (127.0.0.1).

CHAPTER 2

2. COSSIM User Guide

This section provides instructions on how to build a simple application and extract statistics using COSSIM environment.

2.1. Setup the HLA Server & SynchServer

If all components of the COSSIM Framework run on localhost, you should open two terminals and execute the following commands to initiate the CERTI HLA Server as well as the SynchServer:

```
1. rtig
2. SynchServer
```

Figure 1 illustrates the normal execution of CERTI HLA Server & SynchServer.

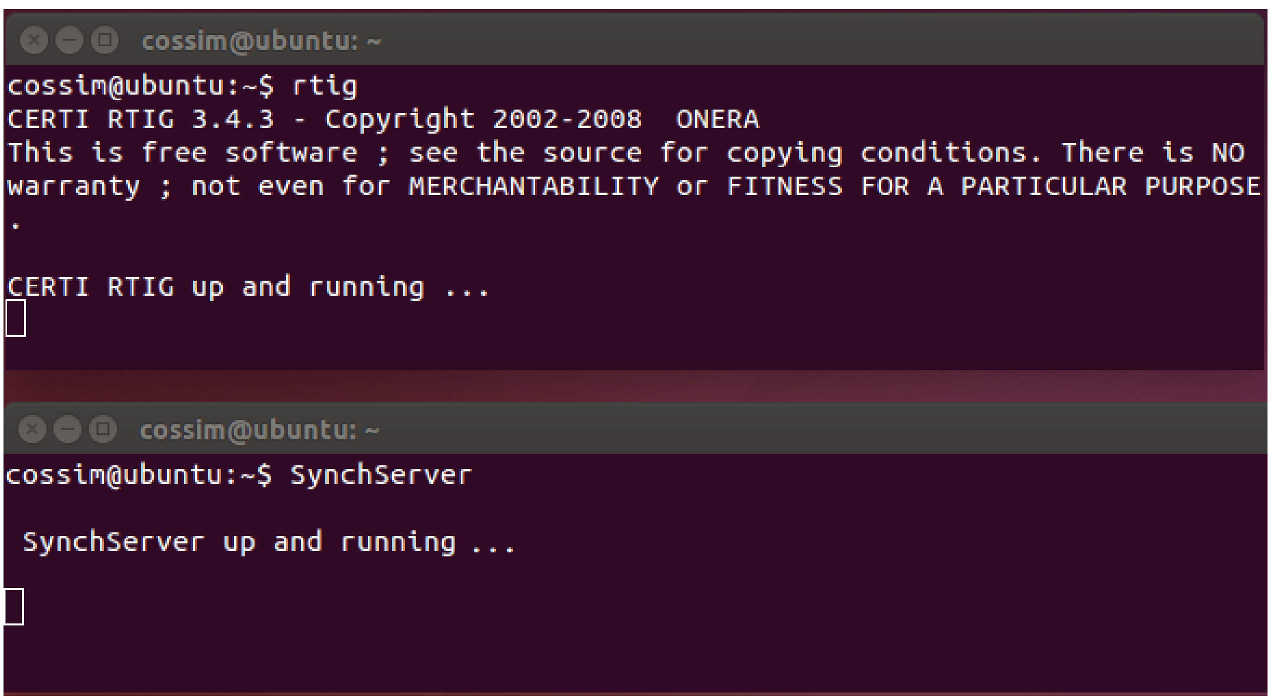

Figure 1. Normal Execution of CERTI Server (rtig) & SynchServer

The following Section (2.2) describes analytically the four steps to configure and execute an example in COSSIM simulation framework. The example consists of two clusters of one X86 node and three ARM32 nodes respectively. Specifically, this example includes three end nodes (clients) that are based on ARM devices and a server node that is based on an x86 processor. The four nodes are connected through an Ethernet network where two nodes are operating in the same Class-C network and they are communicating through a router with the other two nodes that reside in a distant similar subnetwork as illustrated in Figure 2 (described in detail in Section 2.2.2). Each of the three ARM end nodes sends a number of ping requests to the x86 node that assumes the role of a server (described in detail in Section 2.2.3).

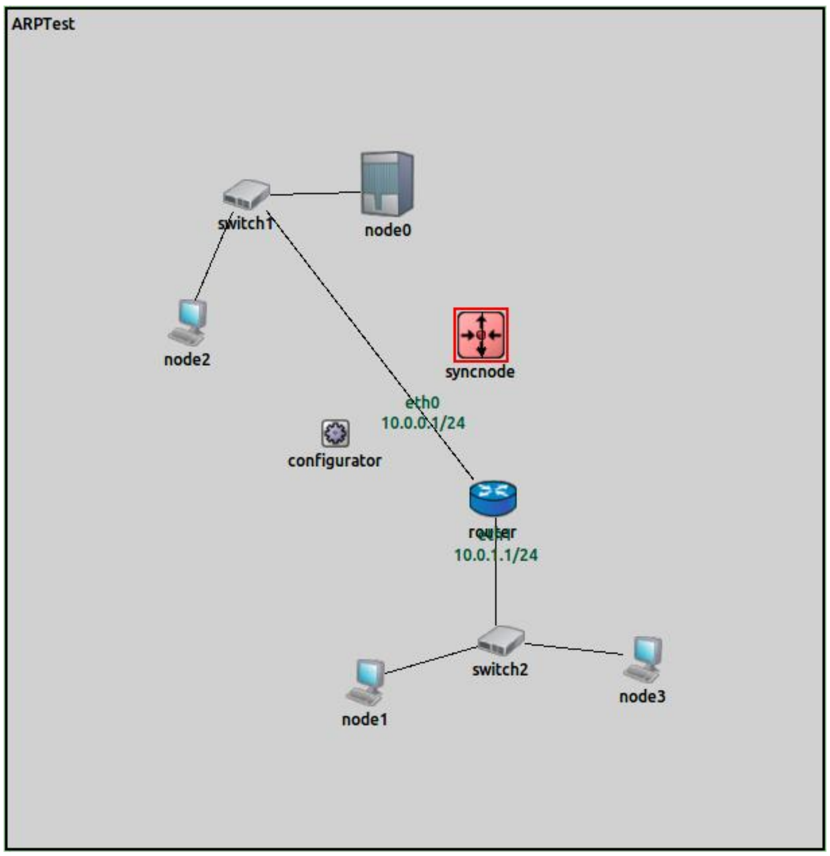

Figure 2. The Network Topology of our example

2.2. A simple COSSIM configuration

First of all, you should open one new terminal and execute the following command in order to execute the OMNET++, while you should select the correct Workspace as described in Section 1.5 (illustrated in Figure 3).

```
1. omnetpp
```

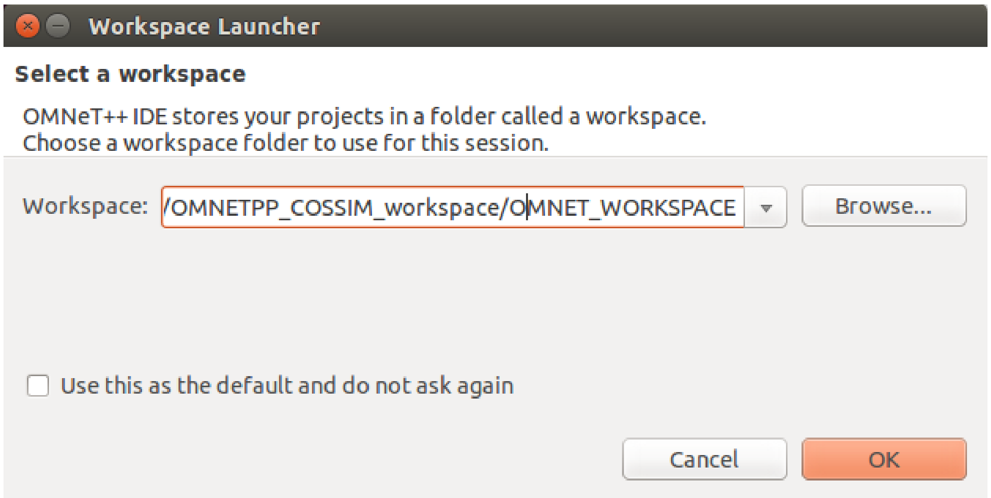

Figure 3. COSSIM-OMNET Workspace selection

2.2.1. Step1: COSSIM-Wizard configuration

In addition, the user should select the COSSIM-Wizard to configure the whole system as illustrated in Figure 4.

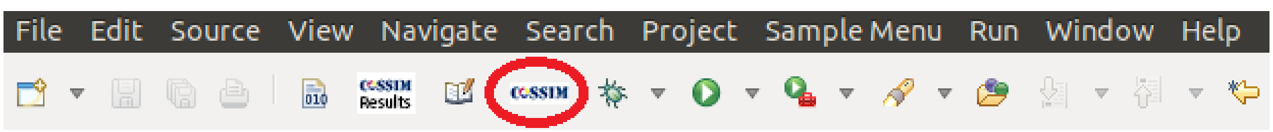

Figure 4. COSSIM-Wizard selection

The wizard offers the ability to create a configuration from scratch, open an existing configuration from a saved file, or open a sample configuration as illustrated in Figure 5.

If there are many cGEM5 nodes in the configuration it is very time consuming for a user to set the parameters for each node separately. We assume that there would be sequential nodes with the same parameters and we consider them as a cluster of same nodes. The first option of the initial wizard page "*Create New Configuration*" leads to the second wizard page where the user is asked to define the clusters of those same nodes as illustrated in Figure 6. The number of nodes and clusters are selected from dropdown lists where the maximum number of clusters that a user can select is equal to the number of nodes that have been selected in the first dropdown list. In this step of the wizard the user must also define the path of the main configuration file and the prefix of the cGEM5 script file that will be used. This prefix will be followed by the number of node in the configuration. The predefined values are *"$GEM5/configs/example/fs.py"* for the configuration file and *"$GEM5/configs/boot/COSSIM/script"* for cGEM5 scripts. Moreover, the user should define the Global Synchronization time (Appendix A). In this example we set a configuration with 4 nodes and 2 clusters of same nodes and Global Synchronization 10ms.

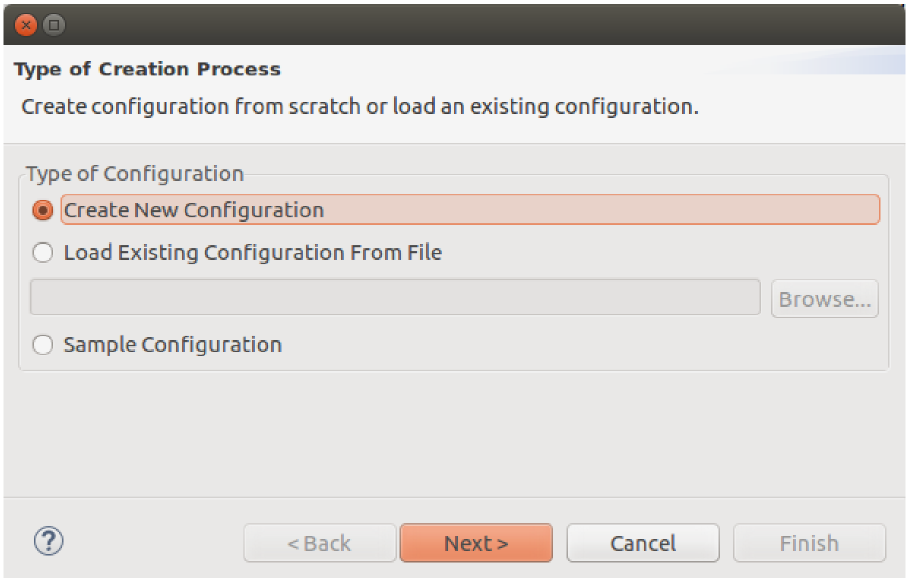

Figure 5. Type of configuration creation process (from scratch or existing configuration)

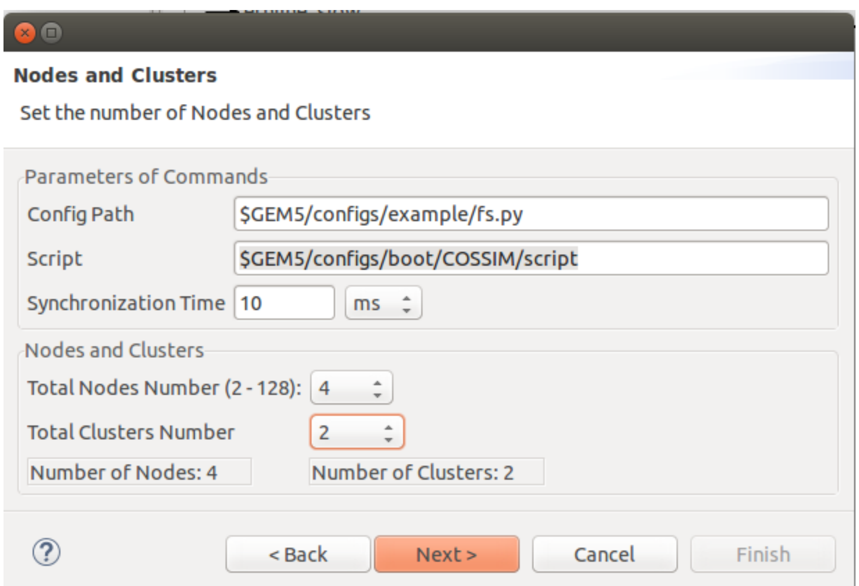

Figure 6. Definition of paths, number of nodes and number of clusters

In the next page the user defines the end node and the type of processor for each cluster. There are three options for the processor: ARM-32, ARM-64 and x86 and the selection will define the options of the next wizard page. The start node is defined automatically: (i) zero for the first cluster, (ii) the next node of the selected ending node of the previous cluster. In this step of the wizard the user defines if there are *"cGEM5 scripts"* and *"etherdump file[1]"* and if the nodes in the cluster are remote (it is executed in distributed physical machine) or local. In Figure 7 we have set the two clusters of the example. Specifically, node0 is x86 architecture (representing an X86 server), while the remainder nodes (1-3) are based on ARM32 architecture. Additionally, we select GEM5 "*script file*" and "*etherdump file"* to store the exchange Data Packets as described in the next subsections, while we have unchecked the "remote" because we want to execute the whole COSSIM in the local machine.

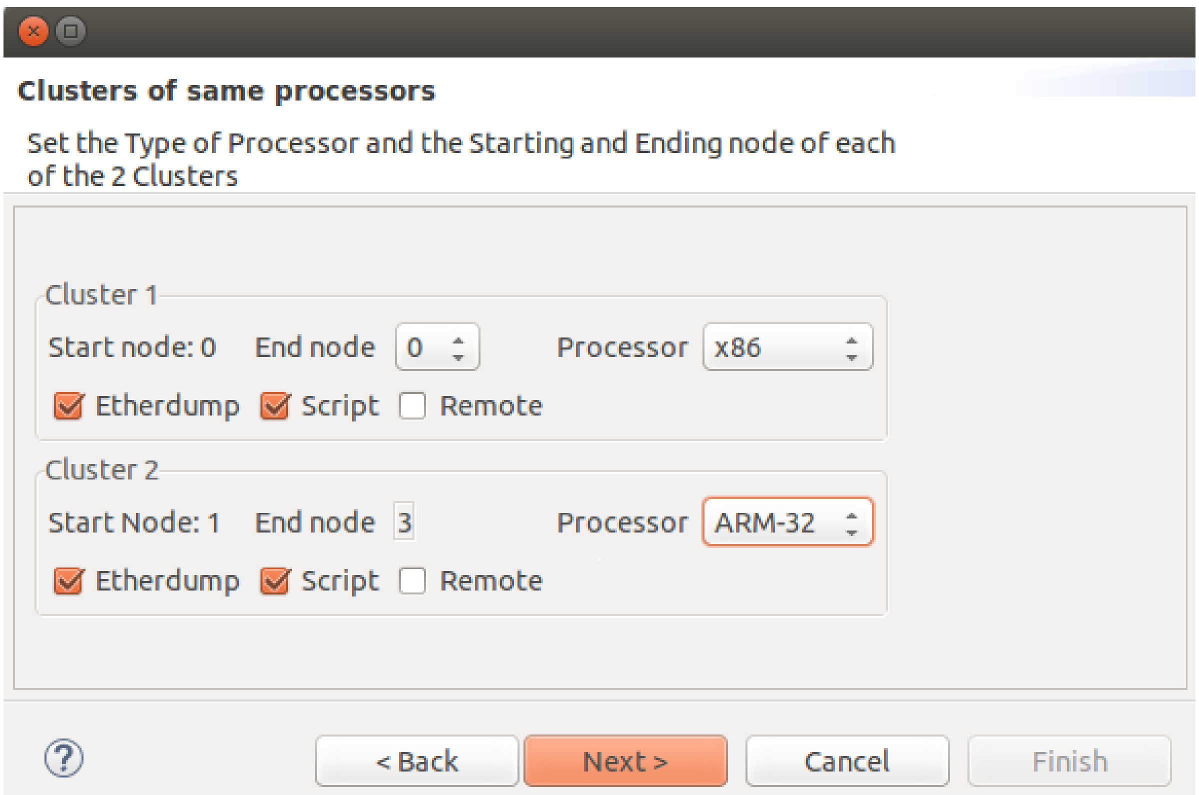

Figure 7. Selection of cluster limits, processor type, remote or local and etherdump and GEM5 script existence

The values of the parameters of each cluster are defined in the next page of the wizard (Figure 8). For each cluster, only the controls that are related to the selections of the previous page are

[1] Etherdump is a network file (it can be accessed using e.g. wireshark)

activated. Specifically, the user should define the kernel[2], disk-image, mem-size as well as the RxPacketTime (*Synchronization per node*).

Our version of cGEM5 can support:

- 2 lightweight Linux distributions
  - Gentoo Base System (v1.12.11.1) for X86 processors (x86_64root.img)
  - BusyBox (v1.15.3) for ARM processors (linux-aarch32-ael.img & linaro-minimal-aarch64.img)
  - In both systems Ubuntu-minimal package and JRE7 are installed so as to enable execution of C, C++ and Java applications

- Ubuntu 12.04 Linux distribution are integrated for both X86 (ubuntu-12.04.img) & ARM (aarch32-ubuntu-natty-headless.img & aarch64-ubuntu-trusty-headless.img)
  - Ubuntu-minimal & Ubuntu-essential packages are installed
  - Apt-get is enabled for easy installation
  - Takes ~20 min & 25min to boot for X86 & ARM respectively

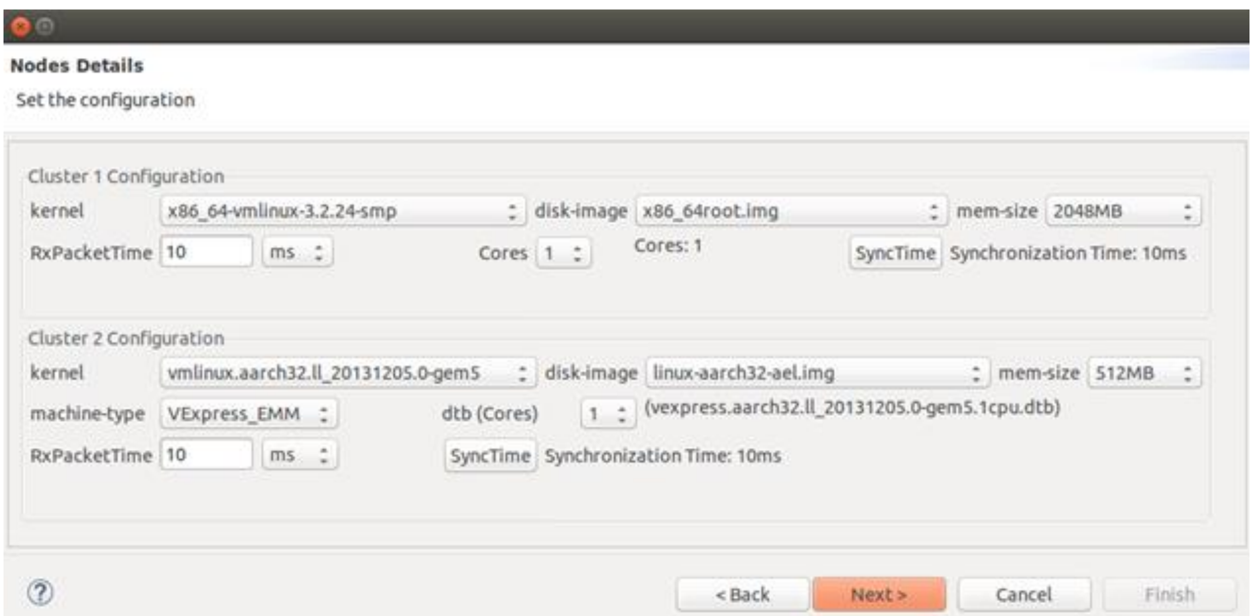

Figure 8. Parameters of each cluster

The next page shows the complete configuration in a tree. Each node of the tree is a cGEM5 node created in the previous pages. A user can expand the nodes in order to see the values of each parameter as illustrated in Figure 9. In the scope of this example we select the "*Done*" button (bottom-left) to finalize the configuration.

[2] Linux Kernel 3.2.24 & 3.13.2 are configured for X86 & ARM respectively instead of Linux Kernel 2.x

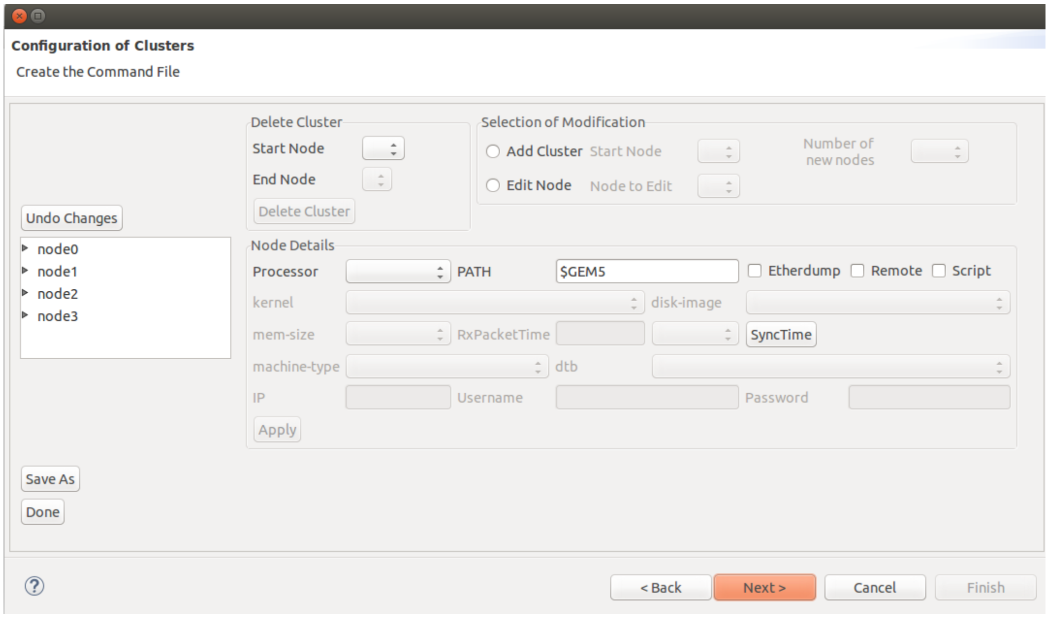

Figure 9. Final configuration with delete, add and edit nodes & clusters

2.2.2. Step2: Network Topology configuration

After wizard configuration, the user should define the description of the network topology in file **ARPTest.ned.** This example includes three end nodes (clients) that are based on ARM devices and a server node that is based on an x86 processor. The four nodes are connected through an Ethernet network where two nodes are operating in the same Class-C network and they are communicating through a router with the other two nodes that reside in a distant similar subnetwork[3]. Specifically, in this file, a situation where an x86 Server (node0) is connected through the switch1 with an ARM node2 is described. Both belong to the same Class-C subnet as shown by their configuration files. They also have the same gateway (router) in the address 10.0.0.1. A similar setup is found on the other side of the network (nodes 1 and 3). Those nodes also share the same Class-C subnet network using the switch2 with base IP the 10.0.1.1 which is their gateway address to the rest of the network. Code-Segment I shows the corresponding configuration file, while Figure 2 represents the actual description of the network topology.

```
Network ARPTest{
  types:
    channel ethline_slow extends DatarateChannel{
      delay = 10ms;
      datarate = 100Mbps;
```

[3] Apparently, the simulator is able to support arbitrary number of nodes in more complex topologies.

```
    }
  submodules:
    switch1: EtherSwitch {
   @display(“p=202,156”);
    }
    switch2: EtherSwitch { }
    router:  Router {
      @display(“p=411,414”);
    }
    configurator: IPv4NetworkConfigurator {
      config = default(xml("<config> <interface hosts='router*'
                     address='10.0.x.1' netmask='255.255.255.0'/> </config>"));
    }
    node0: Txc0 {
      parameters:
      @display("i=device/server_l");
    }
    ...
  connections:
    syncnode.out --> {  delay = 0ms; } --> syncnode.in;
    node0.gate <--> ethline_slow <-->  switch1.ethg++;
    node2.gate <--> ethline_slow <-->  switch1.ethg++;

    node1.gate <--> ethline_slow <-->  switch2.ethg++;
    node3.gate <--> ethline_slow <-->  switch2.ethg++;

    switch1.ethg++ <--> ethline_slow <--> router.ethg++;
    switch2.ethg++ <--> ethline_slow <--> router.ethg++;
```

Code-Segment-I: ARPTest.ned file

In the topology shown in Figure 2 apart from the simulated nodes and the network devices, there is another node shown, named “*syncnode*”. This node is not part of overall CPS system being simulated; it is created by the simulation framework in order to synchronize the cOMNET++ simulation environment and each cGEM5 instance that is connected through HLA to an cOMNET++ node. This way, it is ensured that all simulation processes – each cGEM5 instance and cOMNET++ - are synchronized at (user-specified) intervals and as such the overall simulation time proceeds in sync for the overall system (Appendix A).

2.2.3. Step3: GEM5 configuration script

During the last step of the configuration of the simulation, the user should define the cGEM5 configuration scripts for each cGEM5 node. This is an optional step because he can interact with the cGEM5 using m5term console. However, we recommend to use these scripts to get more accurate results[4]. The user can define these scripts in folder *"COSSIM/gem5/configs/boot/COSSIM"*. The application that is executed in this configuration is a simple ping application. Each of the three ARM end nodes sends a number of ping requests to the x86 node that assumes the role of a server. All four nodes boot the operating system, execute a script that configures the network parameters for their network interface cards and then initiate the application[5] that is to be executed. The code segment II demonstrates this script for one of the three ARM nodes (for the other nodes the script is identical except for the IP address of the node which must be different for each node).

This script makes a basic configuration for the nodes during boot in order to properly configure the network interface card (i.e network IP, Gateway and Subnet mask). Those parameters should be consistent with the cOMNET network parameters (i.e gateway/router parameters). Apparently, these parameters can be modified according to user requirements. To be noticed that the user needs to compile his application in the host machine[6] using *gcc/g++* compilers in case of X86 architecture, while in case of ARM32 or ARM64, ARM cross compilers are required (*arm-linux-gnueabi-gcc*, aarch64-linux-gnu-gcc, etc).

[4] In case of m5term, the user can interact with gem5 but the simulated time is running during typing commands in m5term console.

[5] The executable application must be stored inside the gem5 image.

[6] Lighter GEM5.imgs don't contain any compiler.

```
#!/bin/sh
ifconfig lo 127.0.0.1                         #Configure the localhost
ifconfig eth0 10.0.1.101                      #Configure the IP address
ifconfig eth0 netmask 255.255.255.0           #Configure the Netmask
route add default gw 10.0.1.1                 #Configure the Gateway
export PATH=/usr/lib/jvm/java-7-sun/bin:$PATH #Export the Java Path
m5 resetstats

# Here you can put your application
ping -c 10 10.0.0.100                         #Send 10 ping requests
# END Here you can put your application

m5 dumpstats
/sbin/m5 exit
```

Code-Segment-II: GEM5 script for ARM node

2.2.4. Step4: Execute COSSIM environment

The user can execute the COSSIM simulator selecting the ***omnetpp.ini*** file in module: test→simulations→ omnetpp.ini and then pressing the green arrow as illustrated in Figure 10. Subsequently, a new pop-up window will be displayed and the user should press OK in the OMNET++ information message about the inifile selection (Figure 11) to display the COSSIM network topology (similar to Figure 2). In addition, the user should press 2nd run button of the new pop-up window as illustrated by Figure 12 to start COSSIM simulation. In the right side of Figure 12, the user can see the global time of whole COSSIM environment, while he can observe simultaneously the executed commands by each cGEM5 node, typing the following commands in 4 terminals:

```
1. m5term 127.0.0.1 3000 #for node 0
2. m5term 127.0.0.1 3001 #for node 1
3. m5term 127.0.0.1 3002 #for node 2
4. m5term 127.0.0.1 3003 #for node 3
```

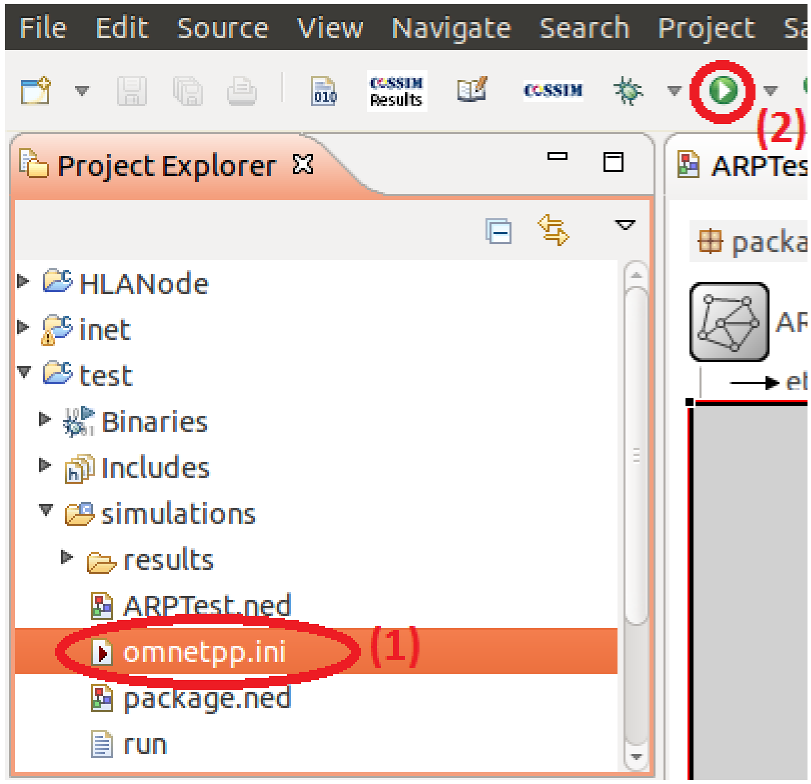

Figure 10. Start COSSIM execution

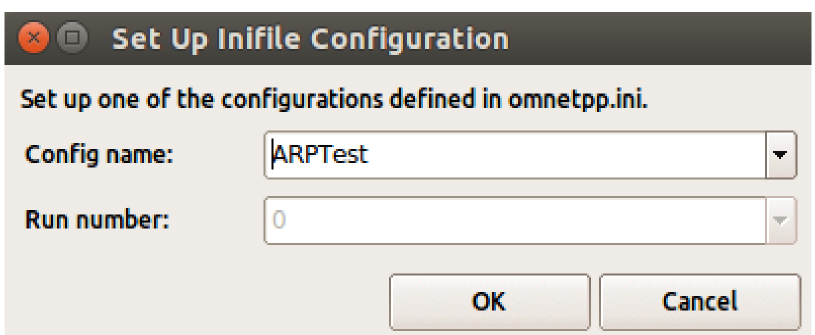

Figure 11. OMNET++ inifile configuration

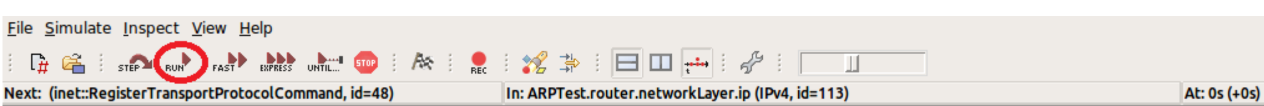

Figure 12. Start GEM5 executions through OMNET++

After COSSIM simulation, the user can find the results about the processing nodes in folders *$GEM5/nodeX*, where X is the node number. Finally, he can find the OMNET++ network statistics using the following path in OMNET++ simulator:

```
simulations-> results-> double click ARPTest-0.sca->finish (new analysis file)-
> choose Browse Data Tab in ARPTest.anf-> choose All tab->unfold list
ARPTest:#0-> unfold ARPTest.router.eth[0].mac
```

In order to integrate all COSSIM statistics, we have implemented a new Graphical User Interface (GUI) to visualize the most important cGEM5 & McPat results pressing the icon from cOMNET++ menu after COSSIM simulation. Figure 13 illustrates the "*select action*" window. *Select action* window have two capabilities; in the first one, the user can select the nodes which wants to visualize the statistics (pressing "Show selected nodes"); in the second one, the user can press the button "Compare selected nodes" to compare statistics of the nodes. Figure 14 presents an example of the statistics of Node0, while Figure 15 presents a comparison example of 4 Node COSSIM simulation.

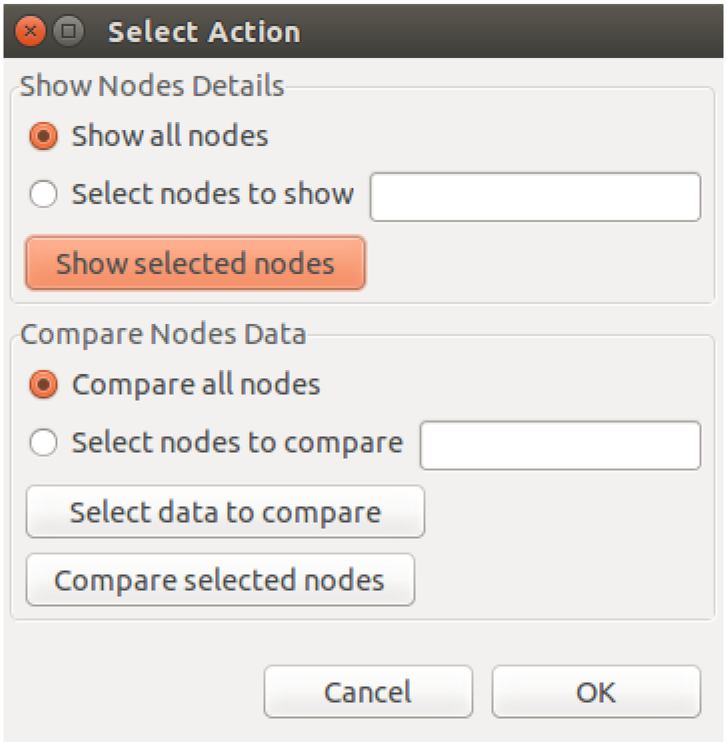

Figure 13. Select Action Window

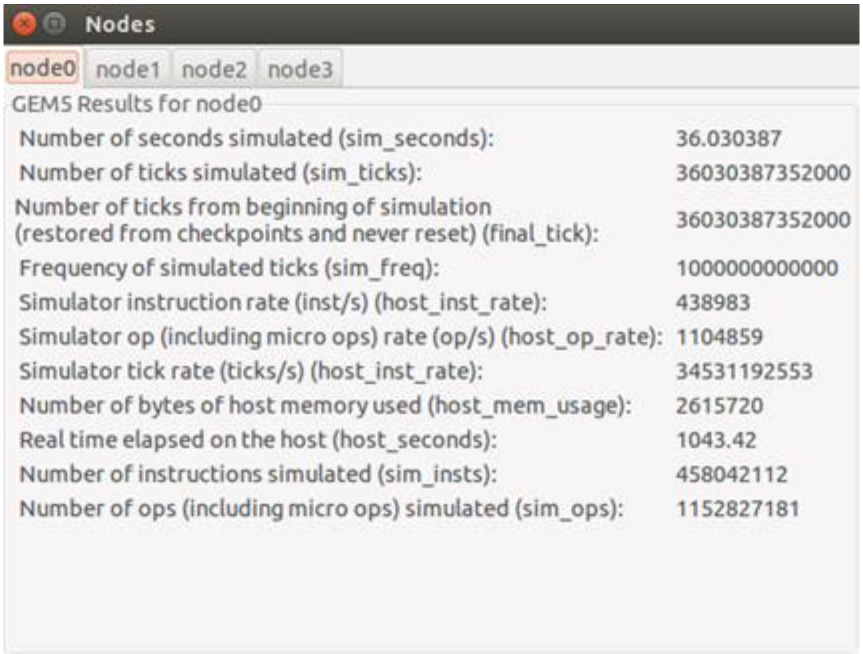

Figure 14. GEM5 statistics for Node0

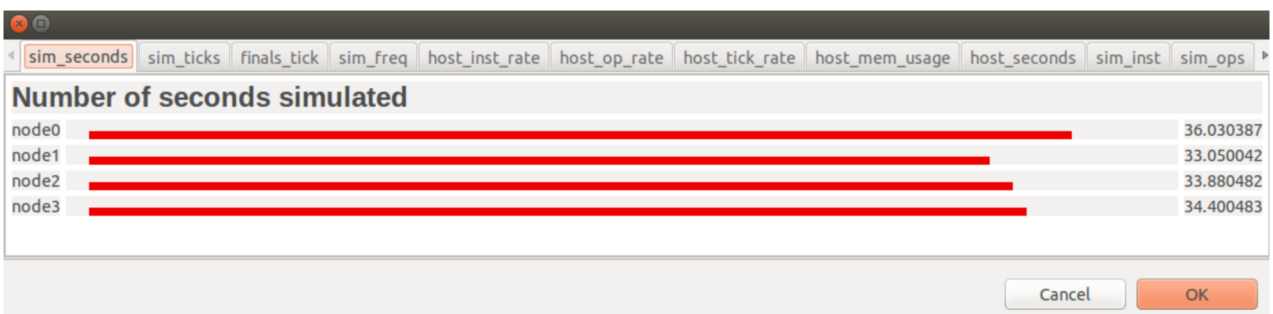

Figure 15. Comparison example of 4Nodes COSSIM simulation (Only Simulated Seconds are illustrated in this figure)

2.3. Configuration of wireless interfaces (Wifi, 3G, etc)

In this subsection we introduce the micro-routers concept, we describe the necessary steps the user should go through to extend the network topology, in order to support different kinds of interfaces. Everything that is described in this subsection it is applied as an extended version of "**Step2"** in the aforementioned, well described procedure. All the rest steps (Step1, Step3, Step4) are identical as analytically described in the aforementioned subsections.

More specifically we will focus on how to build HLA enabled nodes with wireless functionality as all the other network interfaces (e.g ppp, ethernet) are actually a subset of this one.

Similarly to the previous subsection, the user should define the description of the network topology in file **ARPTest.ned.** This example includes one HLA enabled node (node0) which is based on ARM architecture and one server node (node1) which is based on an x86 processor. Each of the nodes is hooked with a dedicated ***micro router*** using an Ethernet connection.

Apparently, network could be scaled to support more wireless clients or mixed wireless and wired clients with the corresponding micro-routers attached to them. For simplicity reasons though, we demonstrate a two-node example. In Figure 16, it is depicted that each HLA node is attached to a dedicated micro-router which is responsible to transparently change the default Ethernet interface to wireless interface. In this respect, GEM5 nodes always preserve the same configuration and we completely manage the simulated network interfaces only inside OMNeT++.

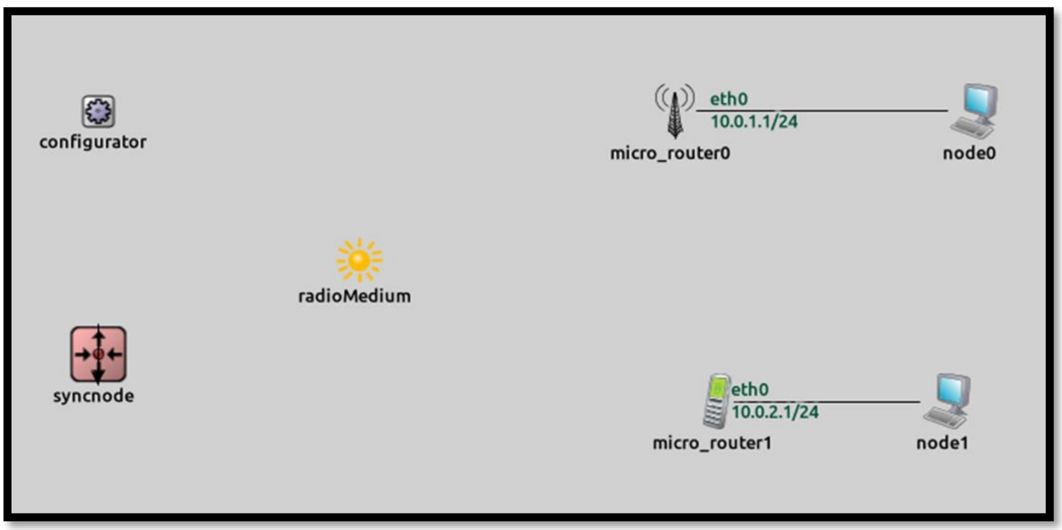

Figure 16. The Network Topology using wireless interfaces

As in the previous version of “Step2” configuration (subsection 1.2.2.2) we still assign a Class-C network for each of the HLA nodes in order to simplify the example, it is not restrictive though. Apparently in this configuration both micro-routers act as gateway for both HLA nodes and the configuration of them is performed like before inside GEM5. That is, to properly set the default gateway as the micro-router and a subnet mask appropriate for a Class-C network.

The complete description of the simulated network is presented in the Code-Segment-III, while Figure 16 gives a graphical depiction of the network topology. Apart from the Syncnode which is responsible for the global synchronization of the whole simulator, another addition to the former “Step2” is the **radioMedium** module which is necessary for any wireless communication inside OMNeT++. The corresponding initialization of this module is also presented in the Code-Segment-III: ARPTest.ned file.

```
network ARPTest
{
    @display("bgb=612,363");
    types:
        channel ethline_slow extends DatarateChannel
        {
            delay = 10ms;
            datarate = 100Mbps;
        }
    submodules:
        micro_router0: CossimWirelessHostTower {
            @display("p=383,90");
        }
        micro_router1: CossimWirelessHostMobile {
            @display("p=406,250");
        }
        radioMedium: Ieee80211ScalarRadioMedium {
            @display("p=209,172");
        }
        configurator: IPv4NetworkConfigurator {
                 config = default(xml("<config> <interface hosts='**'
                  address="10.0.x.x" netmask="255.255.255.0"/> </config>);
            @display("p=65,90");
        }

        node0: Txc0 {
            parameters:
                @display("i=device/pc;p=548,90");
        }
        node1: Txc1 {
            parameters:
                @display("i=device/pc;p=533,250");
```

```
        }
        syncnode: SyncNode {
            parameters:
                @display("i=,red;p=65,224");
        }
    connections:
        syncnode.out --> {  delay = 0ms; } --> syncnode.in;
        node0.gate <--> ethline_slow <--> micro_router0.ethg++;
        node1.gate <--> ethline_slow <--> micro_router1.ethg++;
}
```

Code-Segment-III: ARPTest.ned file (wireless support)

One final addition should be considered for the routing configuration of the micro-routers. That is the routing tables as described in the following two snippets of code. This applies only for the wireless micro-routers, as other wired types like *ppp* do not need any routing configuration at all.

node0: Routing table

```
route:
10.0.2.0  10.0.0.2   255.255.255.0    G  0   wlan0
10.0.1.0  *    255.255.255.0  H  0   eth0
 routeend.

```

node1: Routing table

```
route:
10.0.1.0  10.0.0.1   255.255.255.0    G  0   wlan0
10.0.2.0  *    255.255.255.0  H  0   eth0
 routeend

```

Lastly, in order for the above routing files to be loaded during simulation it is necessary to add the following lines in the simulation configuration file (i.e Code-Segment-IV: omnet.ini).

```
ARPTest.WirelessHost0.routingTable.routingFile = "whost0.mrt"
ARPTest.WirelessHost1.routingTable.routingFile = "whost1.mrt"
```

Code-Segment-IV: omnet.ini file (wireless support)

2.4. Execute COSSIM in Distributed Systems

2.4.1. Setup the .bashrc file

As mentioned in Section 1.1.5, you should define the **static** IP of the CERTI_HOST (Server) instead of localhost in .bashrc file.

```
export CERTI_HOST=IP
```

2.4.2. Setup the HLA Server & SynchServer

In case of some cGEM5 instances are executed in different physical machines, both CERTI Server & SynchServer should be installed[7] and executed in a **static** IP machine. For simplicity reasons, two scripts are implemented to start (startup.sh) and kill (killall.sh) the above servers in the remote machine. The user can start the above servers typing the following command:

```
1.  ./startup.sh IP username password
```

After the COSSIM execution (s), the user may kill the servers typing:

```
1.  ./killall.sh IP username password
```

2.4.3. A simple COSSIM configuration

First of all, the user should open one new terminal and execute the following command in order to execute the OMNET++, while he/she should select the correct Workspace.

```
2.  omnetpp
```

2.4.4. Step1: COSSIM-Wizard configuration

The steps of wizard configuration are similar with these in Section 1.2.2.1. The only difference is in Figure 7, in which the user may select the *Remote* box, in case of this cluster of cGEM5s will be executed remotely. Specifically, if the user has selected the “Remote” button (in Figure 7), he should define the IP, Username and Password in the next step of wizard as illustrated in Figure 17. To be noticed that, cGEM5 should be installed in the specific machine, and the simulated applications must be stored inside the **remote** gem5 image.

---

[7] See Section 1.1.3.

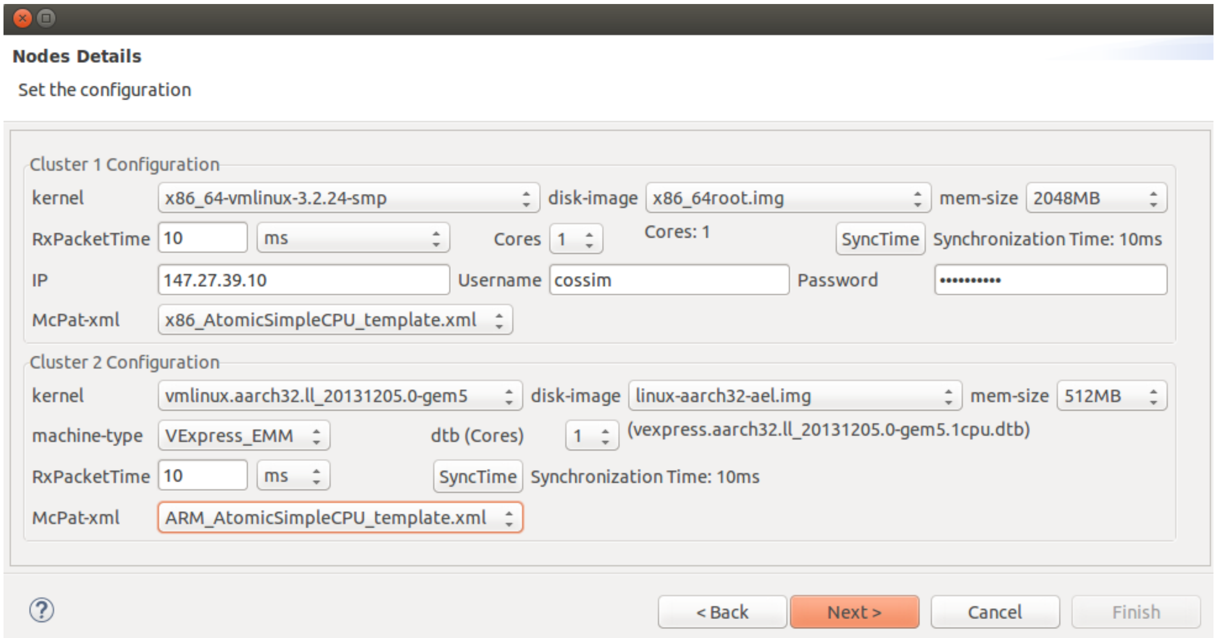

Figure 17. Parameters of each cluster (The first cluster will be executed remotely)

2.4.5. Step2: Network Topology configuration

Steps are similar with the Section 1.2.2.2.

2.4.6. Step3: GEM5 configuration script

Steps are similar with the Section 1.2.2.3.

2.4.7. Step4: Execute COSSIM environment

Steps are similar with the Section 1.2.2.4, while the user can observer simultaneously the executed commands by each cGEM5 node, typing the following commands in 4 terminals (the first node is remote):

```
1.  m5term IP 3000 #for node 0
2.  m5term 127.0.0.1 3001 #for node 1
3.  m5term 127.0.0.1 3002 #for node 2
4.  m5term 127.0.0.1 3003 #for node 3
```

After COSSIM simulation, the user can find the results about the processing nodes in folders *$GEM5/nodeX*, where X is the node number. In case of one cGEM5 is executed remotely, statistics are copied automatically through sftp in local cGEM5 folder *$GEM5/nodeX* (where X is the node number) so that our GUI visualizes them.

Appendix A. COSSIM simulator synchronization

COSSIM simulator synchronization is achieved through CERTI HLA in two stages:

1) *Synchronization per node.* Each node simulator needs to communicate, in a consistent way, with its representation in the network simulator to exchange data packets. This type of synchronized communication is necessary because network data between the two simulators must be exchanged while preserving the exact same time ordering. For this reason, one federation is created per node to achieve the same synchronization time as illustrated in the upper part of Figure 18. The user can define the minimum simulated time in which the two simulators can receive Data Packets.

2) *Global Synchronization.* The COSSIM simulator needs to periodically synchronize all nodes. This is because it supports different types of CPUs with potentially different clock cycles and/or different network protocols, all resulting in varying workload for the simulators engine. Therefore, the simulated time (the timing of the modelled system) in each node can be completely different given the same real-time (the simulation time). For this reason, a Global Synchronization federation is created to achieve a unified notion of time which contains all cGEM5 nodes and one OMNET++ helper Node (SynchNode) as illustrated in the lower part of Figure 18. The user can define the simulated time in which all COSSIM instances are synchronized periodically. The SynchNode is also a normal user-space instantiated node (as the rest of the HLA Enabled Nodes) inside the OMNeT++ simulator that follows the standard Node structure and as a result it is 100% compatible with OMNeT++.

The Synchronization time per node and the Global Synchronization time are two different entities that can be separately defined by the user. The first one is mostly defined by the latency of the network interface and it doesn’t constrain the simulation speed while the second is a trade-off between simulation speed and simulation accuracy.

The proposed global synchronization scheme also functions as a way to preserve the cycle-accurate notion of the simulation process. OMNET++ is natively an event driven simulator, however by employing the global synchronization scheme, it becomes hooked to the ”cycle-events” of each of the GEM5 simulated nodes. This not only prevents the clocks of all the nodes from any drift but also implicitly ”forces” the OMNET++ to act like a ”cycle-driven” event simulator. In this respect every component of the simulated CPS acts within the same notion time (i.e. clock cycles).

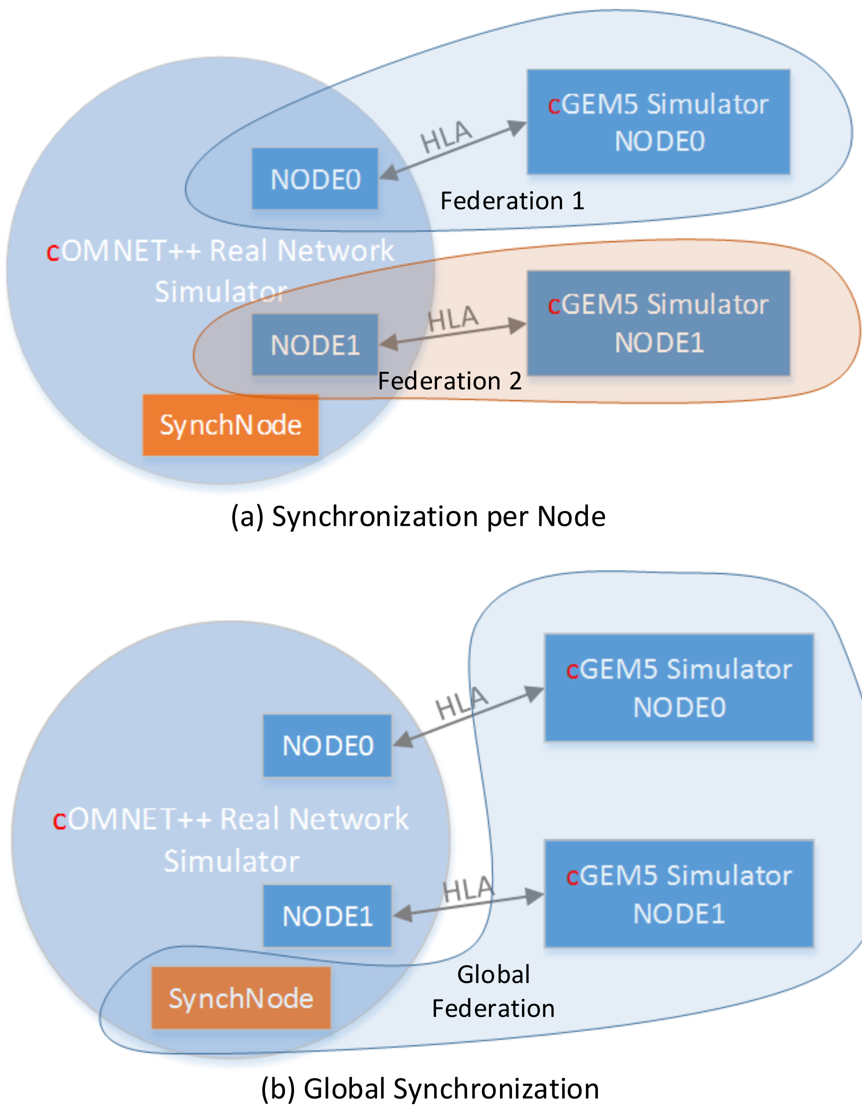

Figure 18. COSSIM HLA federations

Appendix B (Scripts for easy mount files on GEM5 images)

In order to mount the executables (and other files) on GEM5 images, two scripts are implemented; the first one (mount.sh) in case of local GEM5 execution; the second one (remote_mount.sh) in case of remote GEM5 execution.

2.5. mount.sh

The mount.sh script (http://kition.mhl.tuc.gr:8000/f/0a886ad25b/) takes exactly two parameters (e.g.: ./mount ARCH fullpath/filename), where *ARCH* is the Architecture (for image selection), while the *fullpath/filename* is the fullpath and filename of the file (or folder) which will be stored in the corresponding GEM5 image. In *ARCH* parameter, 3 types are supported:

i) X86
ii) X86-Ubuntu
iii) ARM32
iv) ARM32-Ubuntu
v) ARM64
vi) ARM64-Ubuntu

The following command line describes an example of usage (for *test_executable* application and *ARM32* ARCH):

```
./mount ARM32 $HOME/Desktop/test_executable
```

2.6. remount_mount.sh

The remote_mount.sh (http://kition.mhl.tuc.gr:8000/f/27b4eff559/) script takes 6 exactly parameters (e.g.: ./remore_mount.sh *ARCH fullpath/filename FILE_or_DIR IP Username Password*), where *ARCH* is the Architecture (for image selection), *fullpath/filename* is the fullpath and filename of the file (or folder) which will be stored in the corresponding GEM5 image, *FILE_or_DIR* is the *type* of filename, IP is the **static** IP in which the GEM5 will be executed, *Username* and *Password* are the credentials of this physical machine. In *ARCH* parameter, 6 types are supported:

i) X86
ii) X86-Ubuntu
iii) ARM32
iv) ARM32-Ubuntu
v) ARM64
vi) ARM64-Ubuntu

In *FILE_or_DIR* parameter, 2 types are supported:

i) FILE
ii) DIR

Specifically, this script upload the FILE or DIR in the machine with *IP* through sftp, and subsequently, it mounts it in the appropriate GEM5 image depending on *ARCH* parameter. The following command line describes an example of usage (for *test_executable* application and *ARM32 ARCH*):

```
./mount ARM32 $HOME/Desktop/test_executable FILE IP Username Password
```