Table of Contents

Abstract

It is envisioned in the future that not only smartphones will connect to cellular networks, but also all kinds of different wearable devices, sensors, vehicles, home appliances, VR headsets, and robots etc. However, since the characteristics of these different devices differ largely, people argue that future 5G communication systems should be designed to elastically accommodate these different scenarios. We propose a reconfigurable core network called RECO that demonstrates how to implement customized virtual network entities efficiently to suit for different types of users with different characteristics. We then implement a reconfigurable MME called RECO MME which verifies our proposed RECO architecture. Besides, we particularly focuses on the dynamic linking framework used in our RECO MME.

1. Introduction

1.1. Architecture Overview

Reconfigurable Core (RECO)

Figure 1 shows the architecture to build a flexible 5G core efficiently, and we call it **Re**configurable **Co**re (RECO). RECO has three key components:

Modularized virtual network entities (VNFs) designed by object-oriented programming language:

Inside each virtual network entity or so-called virtual network function (VNF), we split the code into modules and compile them into shared libraries (.so). We then separate the modules into two groups: (i) ***common modules*** which are the same among different types of users, and (ii) ***customized modules*** which differ between different types of users. As shown in Figure 1, the MME common libraries are the common modules and GTP.so, NAS.so, S1AP.so, and S6A.so are the customized modules. The common modules can be implemented in any programming language since they are only treated as libraries for customized modules. Also, different types of customized VNFs share the same copy of common modules. For example, the human MME VNF, the IOT MME VNF and the vehicle MME VNF all use the same copy of common modules stored inside memory and disk. On the other hand, customized modules differ between different user types and should be implemented in object-oriented programming languages. The major benefit of implementing customized modules like this is that it can highly reuse already written code and enhance the process of creating a new customized VNF. For example, suppose we have already implemented a human MME VNF. To implement a high-mobility vehicle MME VNF, we can directly inherit most of the classes within the human MME VNF and just override particular mobility related member functions to build a high-mobility vehicle MME VNF.

Dynamic linking framework which links customized modules during run-time:

For each virtual network entity, there is a dynamic linking framework inside it. The framework's major job is to link customized modules during run-time according to the configuration file the identifier provides to form a customized virtual network entity. The reason we choose to use dynamic linking is because it highly increases the flexibility to create a new network slice. When needing to form a new virtual network entity, all we need to do is to create new corresponding customized modules for the particular type of user and the dynamic linking framework will then compose everything together to form a new network entity. This is somewhat similar to adding a plugin into the Chrome browser to generate a customized browser.

An identifier which generates a configuration file for a particular network slice:

When a new user first tries to attach to the core network, the identifier inside the load balancer will try to identify the user's type by parsing some "user type tag" in the attach request packet. If this kind of user type has never attached to the core network before, the identifier will generate a ***configuration file*** including information about what customized modules it needs to form a particular network slice and pass it to the dynamic linking framework. Then the dynamic linking framework will compose these modules together and form customized virtual network entities to serve the user. The identifier also records a ***hash table*** with user types as the key and which VNFs are for that particular user type as the value. In this way, subsequent packets of the same user type can go through the hash table and find out which VNFs it should route its packets to.

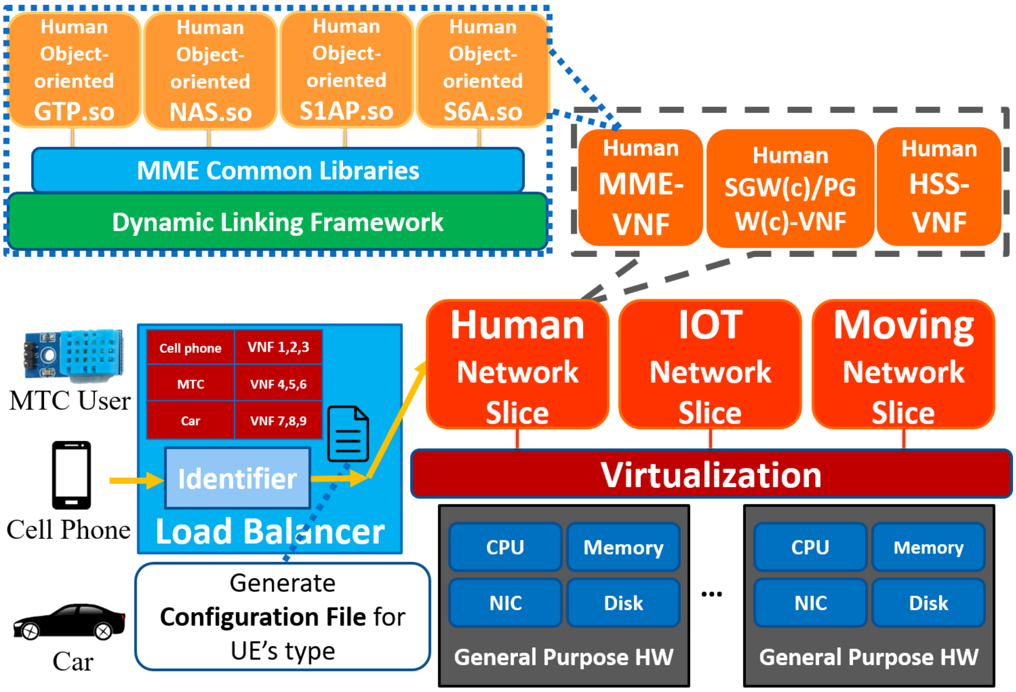


Figure 1. Reconfigurable Core

1.2. Dynamic Linking Framework for RECO MME

We choose first to implement the MME entity so that we can verify that our reconfigurable architecture is feasible and has the benefits we expect. In the following subsections, we first give an overview of our reconfigurable virtual MME (RECO MME) and show its architecture, then we focus on how the dynamic linking framework is implemented in RECO MME.

1.2.1. RECO MME

We built a reconfigurable virtual MME to demonstrate the proposed reconfigurable architecture. It is mainly modified from the MME inside openair-cn, a simple core network developed by EURECOM. We followed the code architecture of openair-cn but to separate the highly bundled mme executable linked with many static libraries into a dynamic linking framework linked with shared libraries. Figure 3 shows the architecture of our RECO MME. It is composed of four main components listed below:

Common modules

We recompiled the static libraries used in openair-cn's MME into two kinds of shared libraries: (i) *load-time dynamic linking shared libraries* which share among different types of users, and (ii) *run-time dynamic linking shared libraries* which differ between different types of users. The *load-time dynamic linking shared libraries* are common modules which can be shared among different customized MME. When the mme executable (the dynamic linking framework) is executed and loaded into memory, these common modules will also be loaded into memory. We purposely built these common modules into *load-time dynamic linking shared libraries* because when there are tons of different customized MMEs (human MME, eHealth MME, high mobility MME etc.) serving different types of users simultaneously on a machine, the storage and memory device will only need to store one copy of these common modules. This highly reduces disk space and memory usage.

In practice, we compiled the static libraries (.a) 3GPP_TYPES, BSTR, CN_UTILS, HASHTABLE, SECU_CN, UDP_SERVER, SCTP_SERVER, GTPV2C, ITTI inside openair-cn into *load-time dynamic linking shared libraries* for our RECO MME to get the benefits of saving storage and memory space.

Customized Object-oriented modules

We refactored MME_APP, NAS, S1AP, S11_MME and S6A inside openair-cn from C-based static libraries into C++ based *run-time dynamic linking shared libraries*. Building these customized modules into *run-time dynamic linking shared libraries* enables the MME to load and unload shared libraries during run-time. By doing so, the dynamic linking framework can load different customized modules according to a configuration file and form a customized MME. For example, to form a high-security MME that serves eHealth users, the dynamic linking framework would load customized high-security modules according to the configuration file during run-time and link them into a customized high-security MME used particularly by eHealth users.

In addition, the source code inside these five modules (MME_APP, NAS, S1AP, S11_MME and S6A) are all refactored by C++ object-oriented programming language. This was done by composing related functions and variables inside each module into classes. By doing so, when a programmer wants to customize an already written module (base module) into a totally new module (such as a high-security module or a high-mobility module), he/she does not need to rewrite the whole module again. Instead, he/she can inherit classes inside the base module and override particular member functions with new functionalities into a new customized module. This highly reuses already written code and increases the development process.

Pseudo Identifier

Currently, we simply implemented a checkbox list shown in Figure 3. The checkbox list enables the programmer to manually choose particular modules for a type of user and then generate the corresponding configuration file and pass it to the dynamic linking framework.

The configuration file generated by the pseudo identifier is shown in Figure 2. We can see that for a human user, we should choose MME_APP, NAS, S11_MME , S1AP and S6A as the customized modules to be loaded by the dynamic linking framework. And for a high-mobility user such as a user taking the high-speed rail, we should choose high-mobility modules HMB_MME_APP, HMB_NAS and HMB_S11_MME with customized high mobility classes implemented inside these modules and S1AP and S6A which are the same modules as the human users since these modules do not differ from a human user. As for an eHealth user which requires special high-security authentication methods, we should choose high-security modules HS_NAS and HS_S6A which are implemented with new security algorithms and the other three modules the same as a human user.

```
/* Module list for human users */
MOD_LIST :
{
   MODNAME = ["MME_APP","NAS","S11_MME","S1AP","S6A"];
};
/* Module list for high-mobility required users */
MOD_LIST :
{
   MODNAME = ["HMB_MME_APP","HMB_NAS","HMB_S11_MME","S1AP","S6A"];
};
/* Module list for high-security required users */
MOD_LIST :
{
   MODNAME = ["MME_APP","HS_NAS","S11_MME","S1AP","HS_S6A"];
};
```

Figure 2. Configuration file for RECO MME

Dynamic linking framework

The dynamic linking framework is used for linking customized modules at run-time according to the configuration file the pseudo identifier provides to form a customized MME. Its main functionalities include provide an interface for customized modules, parse the configuration file, load and initialize corresponding modules according to the configuration file, and help resolve dependency relationships among customized modules. We will describe each of these functionalities clearly in the next subsection.

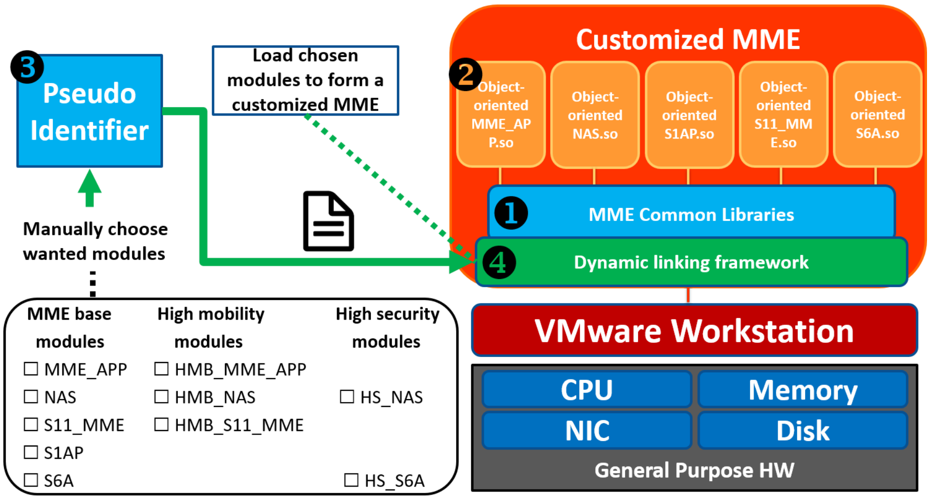


Figure 3. RECO MME Architecture

Figure 4 illustrates an example to form a high-speed rail (HSR) high-mobility MME inside our RECO MME. (1) We manually choose the high mobility modules HMB_MME_APP, HMB_NAS and HMB_S11_MME[1] and base modules S1AP and S6A. (2) The pseudo identifier will generate a corresponding configuration file for the dynamic linking framework. (3) The framework will load and initialize the corresponding modules to form a customized MME for HSR. (4) The MME for HSR along with other core network entities form a HSR network slice.

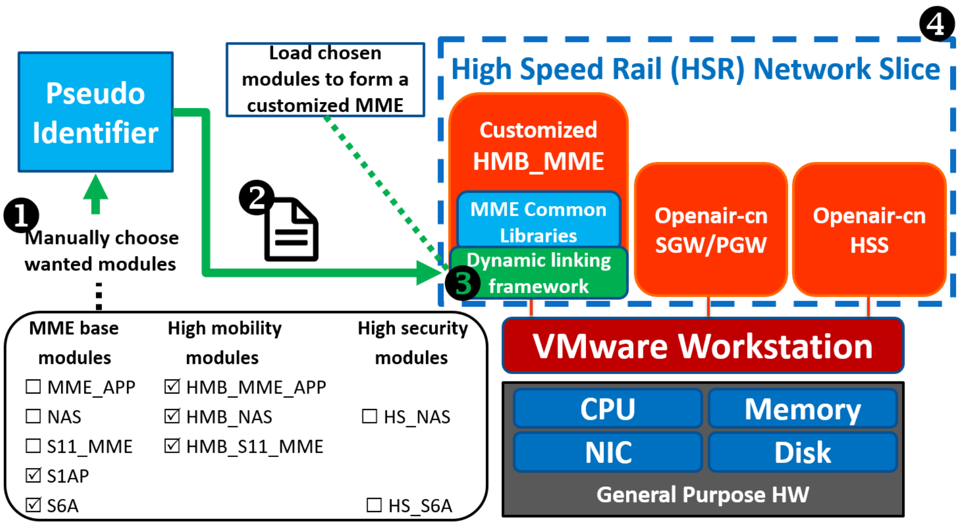


Figure 4. High mobility MME example

1.2.2. Dynamic Linking Framework

In this subsection, we describe how the dynamic linking framework is implemented in our RECO MME. The framework has four main functionalities, (1) provide an interface for customized modules (2) parse the configuration file (3) load and initialize corresponding modules according to the configuration file, and

[1] Note that we did not modify the source code for mobility related modules. We just renamed these modules' names to HMB_MME_APP, HMB_NAS and HMB_S11_MME to demonstrate our reconfigurable concept. The same is for high security modules.

(4) help resolve dependency relationships among customized modules.

<u>Module-framework interface</u>

First, we provide an interface class named *module* inside the dynamic linking framework shown in Figure 5. This interface class includes a pure virtual function named *init* which forces every class that inherits *module* to implement its own initialization function.

```
class module {
public:
 virtual int init(mme_config_t *) = 0;
};

/* type definition of the class factory functions */
typedef module* create_t();
typedef void destroy_t(module*);
```

Figure 5. Interface class "module" in the dynamic linking framework

Next, in every customized module, we implement a class named after its module name and inherits the "module" interface class. Besides, we create two *class factory functions* which helps the module to create/destroy an instance of its own. For example, Figure 6 shows the implementation of a class named *nas* inside the NAS.so module. We can see that class *nas* inherits interface *module* and implements the *init* virtual function. In addition, it implements two class factory functions named *create* and *destroy*. Function *create* helps create a *nas* instance, and function *destroy* frees the created *nas* instance. The dynamic linking framework will use *dlsym* to access the symbol address of *create* and *destroy* and use these addresses to create or destroy the *nas* instance.

```
#include "module.hpp"

class nas : public module {
 public:
   //call nas_init to create a nas module thread
   virtual int init(mme_config_t * mme_config_p) {
      return nas_init (mme_config_p);
   }
};

/* the class factory functions */
extern "C" module* create() {
   return new nas;
}
extern "C" void destroy(module* p) {
   delete p;
}
```

Figure 6. nas.cpp

Parse configuration file

The dynamic linking framework is also responsible for parsing the configuration file for the MME. The configuration file includes a module list which lists the customized modules that the framework should load and link during run-time as listed in Figure 2. Line 10 in Figure 7 shows the function called to parse the configuration file. After parsing the configuration file, the framework will store the list of customized modules in a global variable named *mod_list* shown in line 5 in Figure 7. This variable will be used later for loading and initializing customized modules.

Load and initialize customized modules

We can see in line 18 ~ line 34 in Figure 7 that how the dynamic linking framework loads and initializes customized modules. It uses a *for* loop to iterate through the *mod_list.* Moreover, for every customized module, the framework uses *dlopen* to load the customized modules into memory and then uses *dlsym* to access the symbol address of the class factory function *create* and calls it to create an instance of that module. Later in line 32, it calls the *init* function of every module to initialize that module.

Resolve dependency relationships among customized modules

Note that in line 20 in Figure 7, we set the *RTLD_GLOBAL* flag in *dlopen*. This flag tells the dynamic linker to merge the symbol table of each module into a global symbol table which enables subsequently loaded shared libraries to access symbols defined in previously loaded shared libraries. In other words, by setting *RTLD_GLOBAL,* customized modules can access symbols defined in other modules easily as if they were defined in their own module.

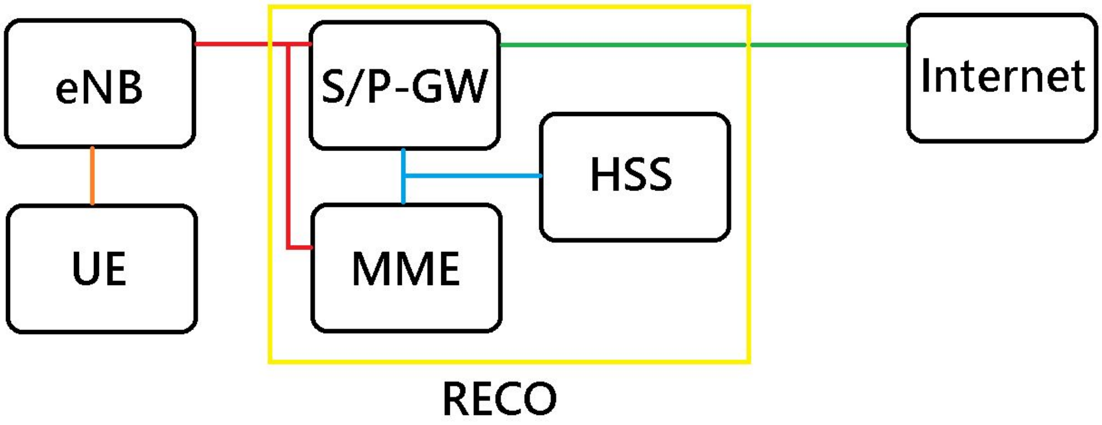
```
1 #include "module.hpp"
2
3 /* The list of modules to load while executing mme
4    same list as the MOD_LIST in mme's configuration file */
5 std::vector<std::string> mod_list;
6
7 int main (int argc, char *argv[]) {
8
9   /* Parse mme's configuration file and store the attributes in
mme_config structure */
10  mme_config_parse_opt_line (argc, argv, &mme_config);
11
12  /* Array of pointers pointing to the address of the modules
loaded by dlopen() */
13  void *handle[mod_list.size()];
```

```
/* Array of pointers pointing to the address of the created module's instance */
module *mod[mod_list.size()];

/* Load mme modules according to MOD_LIST in mme's configuration file */
for (int i = 0; i < mod_list.size(); i++) {
  // open the module's shared library
  handle[i] = dlopen(mod_list[i].c_str(), RTLD_LAZY|RTLD_GLOBAL);

  //load the "create" class factory symbol
  create_t* create_mod =
      reinterpret_cast<create_t*> (dlsym(handle[i], "create"));

  //create an instance of the class
  mod[i] = create_mod();
}

/* Initialize and start all modules */
for (int i = 0; i < mod_list.size(); i++) {
  mod[i]->init(&mme_config);
}

/*** wait for created threads to exit ***/

/* Destroy(free) each module's instance */
for (int i = 0; i < mod_list.size(); i++) {
  //load the "destroy" class factory symbol
  destroy_t* destroy_mod =
      reinterpret_cast<destroy_t*> (dlsym(handle[i], "destroy"));
//destroy instance of the class
  destroy_mod(mod[i]);
}

/* Close(unload) shared libraries */
for (int i = 0; i < mod_list.size(); i++) {
  dlclose(handle[i]);
}
}
```

Figure 7. "main" function for the dynamic linking framework

2. Environment Setup

The example of network architecture is like the picture below. We can follow it when setting IP addresses to run the RECO. And there is an example of how to set IP addresses in chapter 2.1.1.



Notice that:
HSS and MME still need the Internet when installing some application tools.

And there are some particular colors to show different meanings:

Commands
Items
Important points

2.1. Core Network

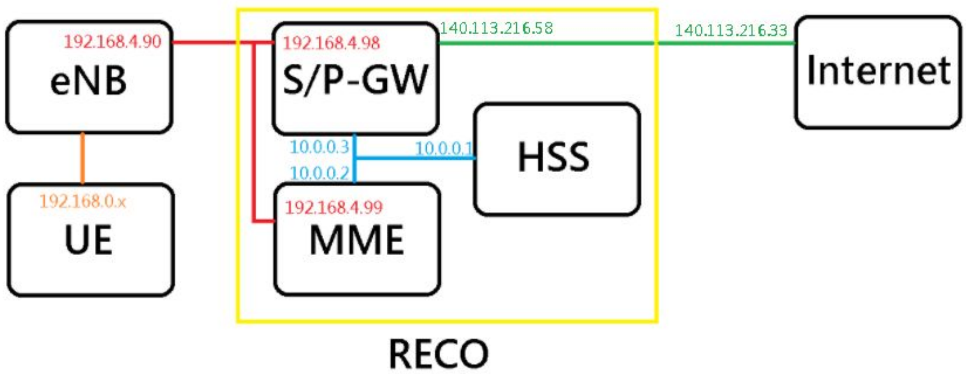
Notice that:
The S/P-GW require a Linux kernel version equal to 3.19 or greater than 4.7.

2.1.1. Setup IP Architecture

We use ubuntu 16 with VMware Workstation 12 for example:

Notice that:
We have to snapshot the VMs before shutdown. Then we have to load the snapshot every time to avoid the problems occurred when rebooting.

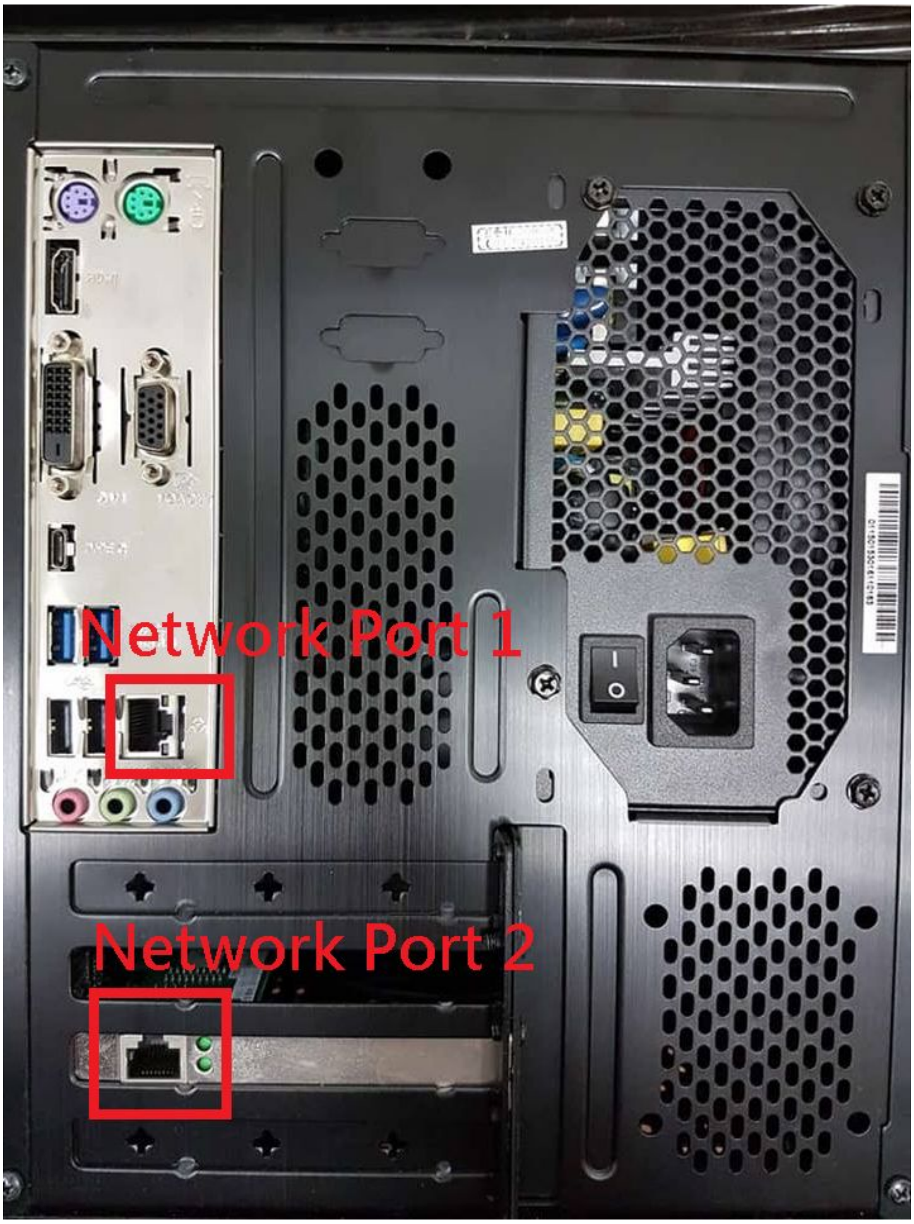
The architecture is like the picture below.

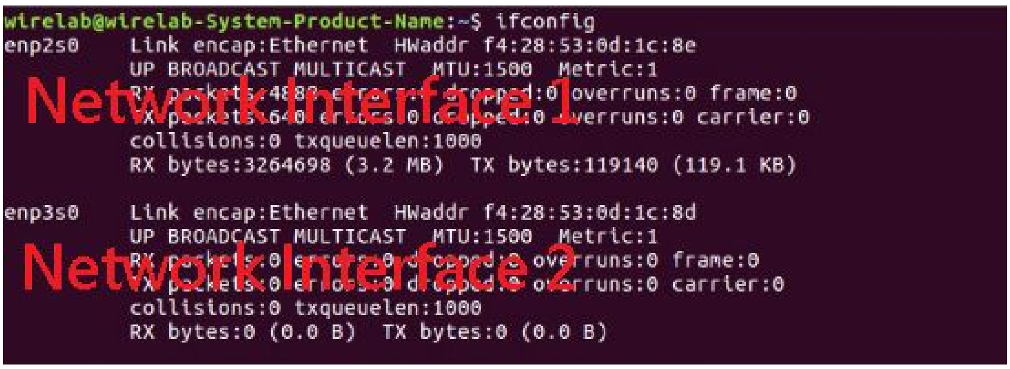


Notice that:
We need at least two NICs (Network Interface Card) on the host device.

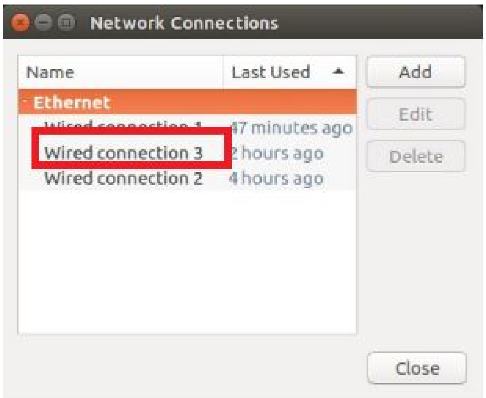
2.1.1.1. Network Ports and NICs

We have to know the relationship between network ports and NICs.
This step is on the host device.

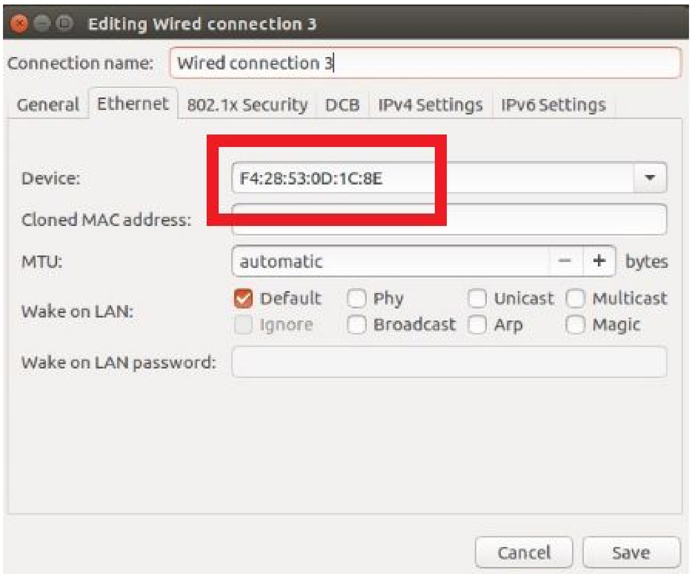




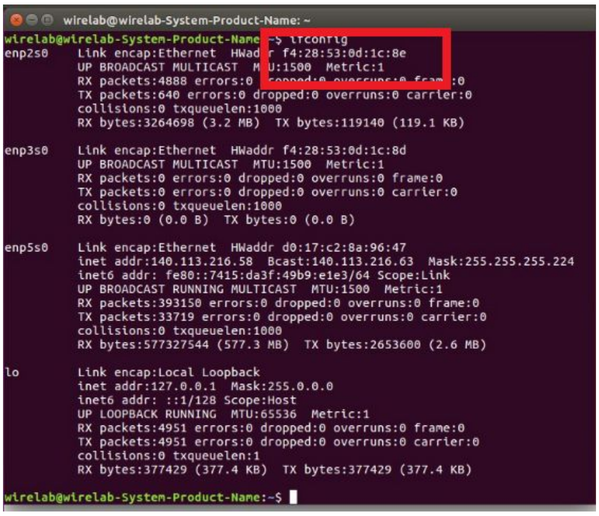
The simplest way to ensure the relationship is that attach one of the network ports to the Internet through an Internet line. After that, there will be a 'wired connect', click the 'Edit connections'.



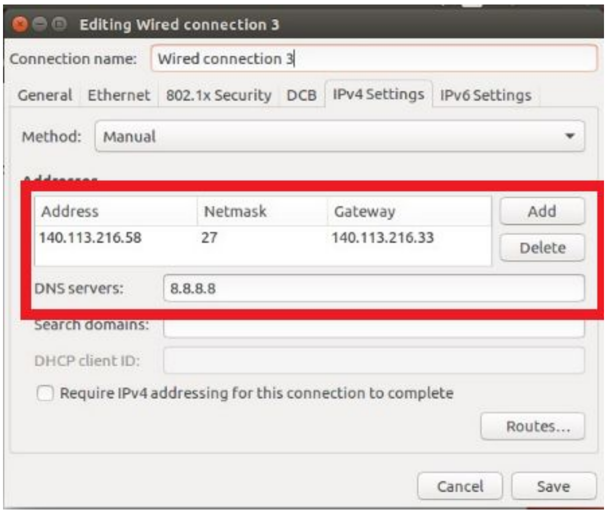
Click the wired connection we find in the previous step. See the 'Ethernet' page, remember the device address.



Then command 'ifconfig', find the NIC which has the same physical address, that is the one the network port mapped.



Finally, set the internet well.

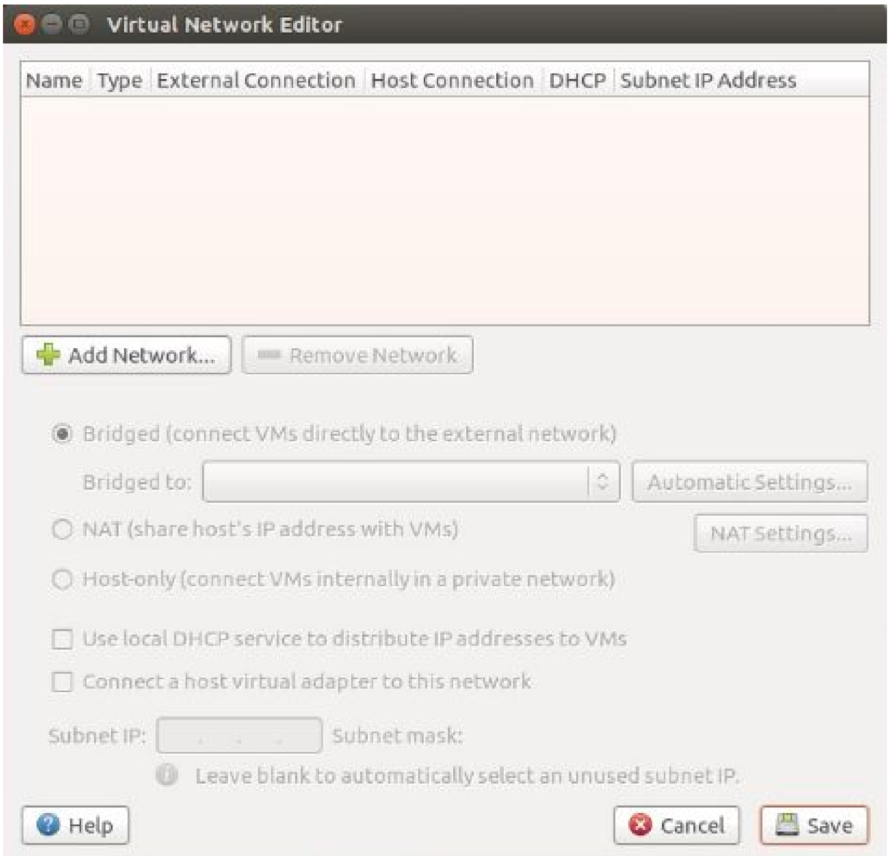


So, in this step, we know the relationship between NICs and network ports!

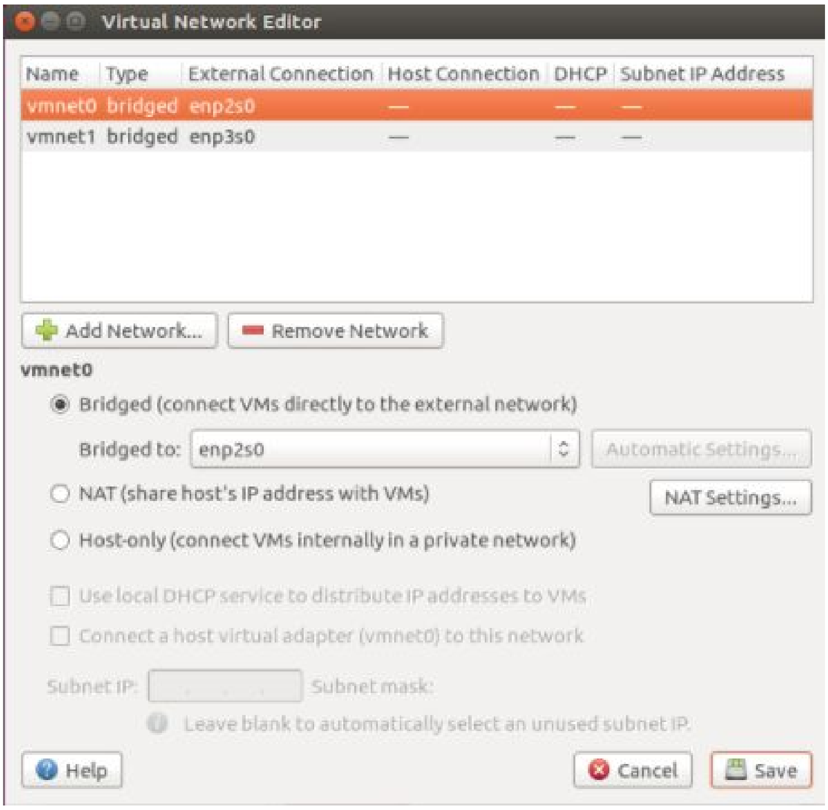
2.1.1.2. NICs and Virtual NICs

We have to know the relationship between NICs and virtual NICs.
This step is between the host device and VMware.

To use the custom mode in the VMware, we have to bridge NICs to virtual NICs. Choose Edit -> Virtual Network Editor -> key in the host password -> remove all settings.



Then Add Network -> choose bridge mode -> Add -> select the particular NIC to bridge, and that is the NIC we bridge to the virtual NIC.

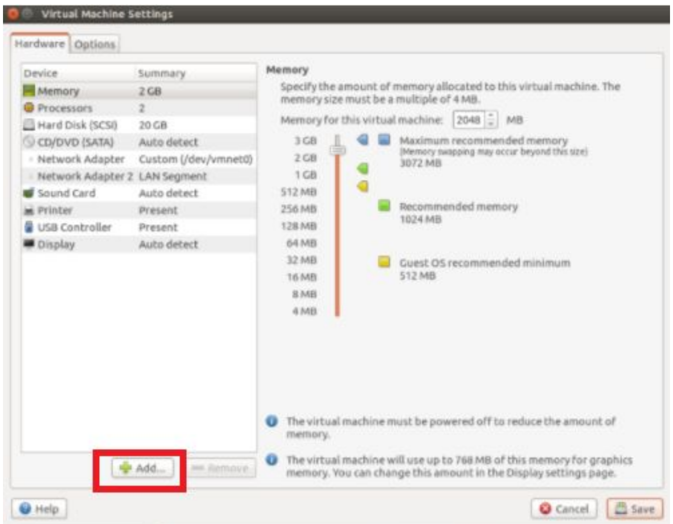


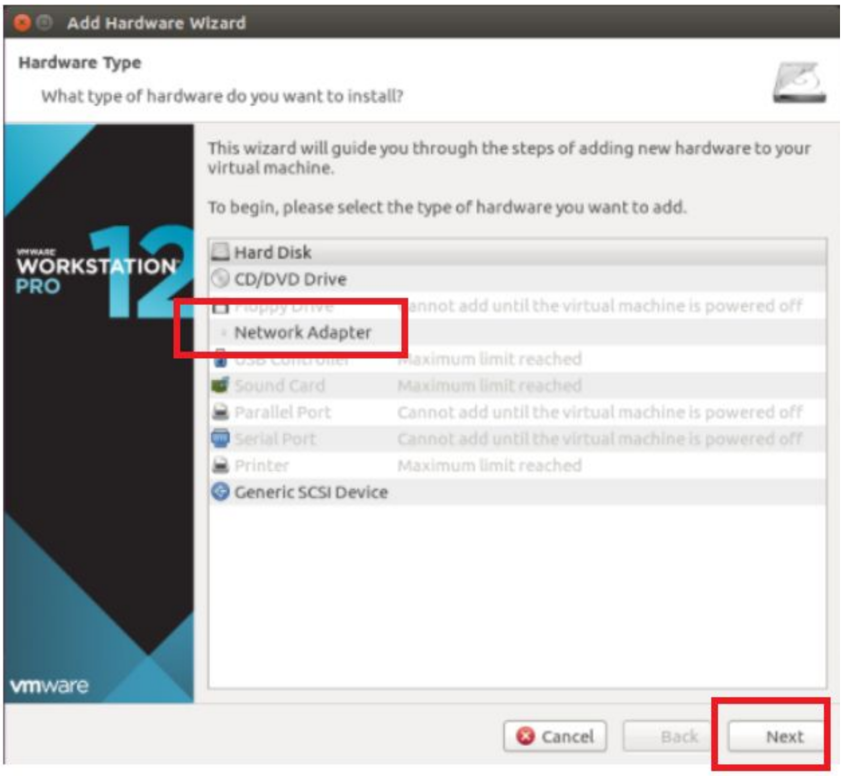
So, in this step, we know the relationship between NICs and virtual NICs!

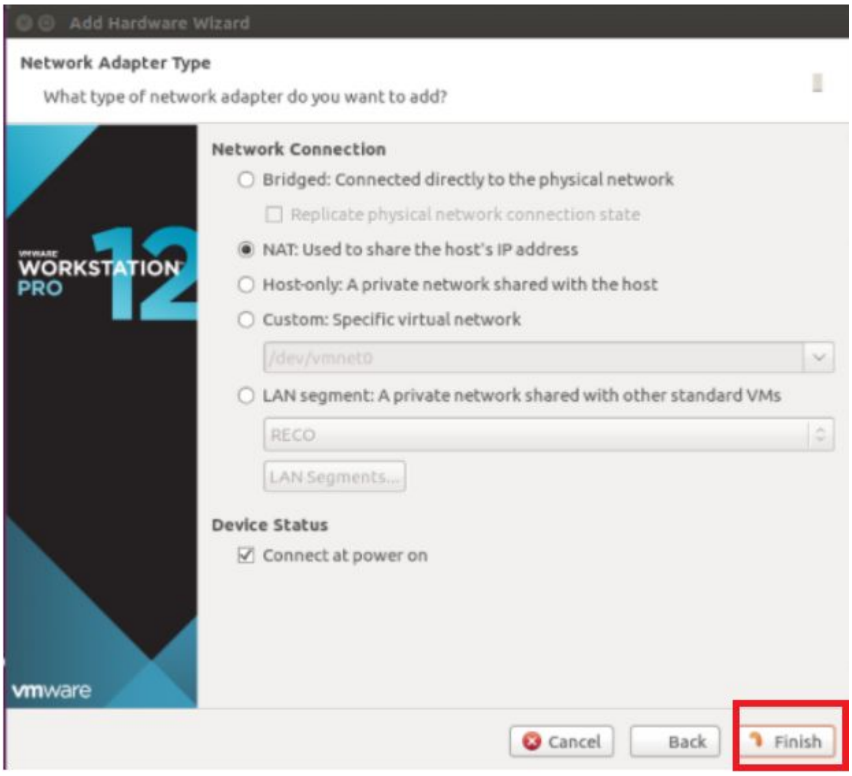
2.1.1.3. Virtual NICs and NICs in VMs

We have to know the relationship between virtual NICs and NICs in VMs. This step is between VMs and VMware.

To bridge NICs in the VM to the particular host NICs, we have to use the custom mode. Right-click the VM -> Settings -> Add -> Network Adapter -> Next -> Finish , that is the way to add a NIC into the VM. Use this way to fit the request of each VM.



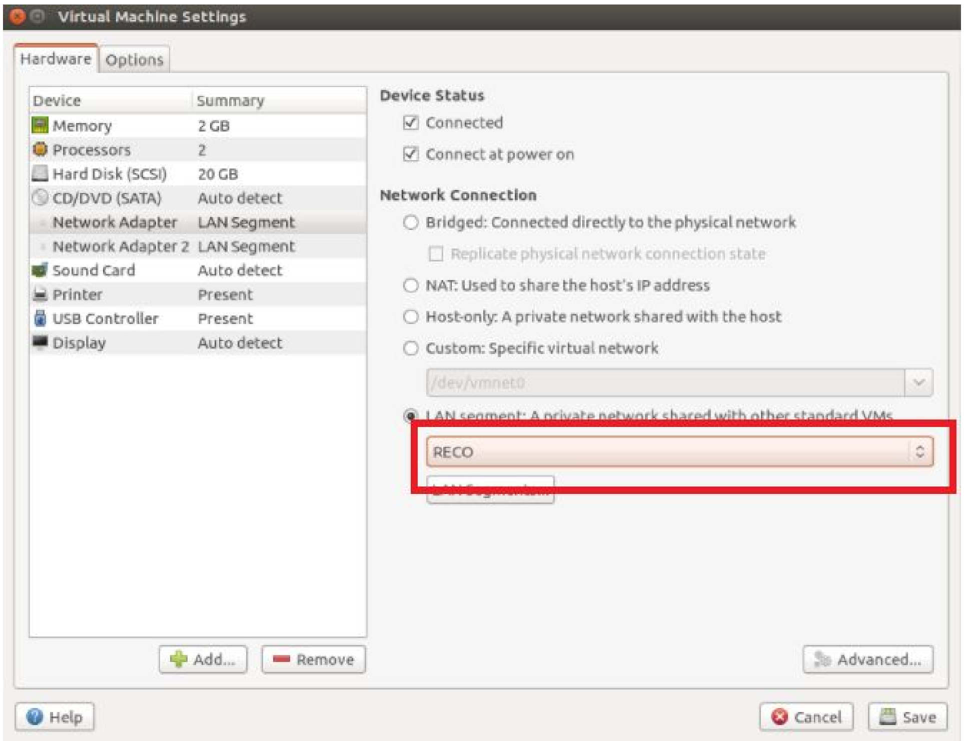




HSS requires two NICs in the VM.

Choose the first NIC in the VM to connect to the Internet, and it needs the real IP because we use the custom mode. Right-click the VM -> Settings -> click the first NIC in the VM-> Custom mode -> choose virtual NIC (Which one maps the host NIC that can attach the Internet) -> Save.

Choose the second NIC in the VM as the LAN interface, and set it at the first time. Right-click the VM -> Settings -> Click the second NIC in the VM-> LAN mode -> (for the first time we set LAN: Add -> Rename it as you want (For example is RECO) -> Close -> click the choose list ->) Save.



Notice that:
If we only have one or two real IPs can be used, we have to remove it after all the HSS installation process.

The way to remove it is that right click the VM -> Settings -> click the first (which one connect to the Internet) NIC in the VM -> Remove -> Save.

MME requires three NICs in the VM.

Choose the first NIC in the VM to connect to Internet, and it needs the real IP because we use the custom mode. Right-click the VM -> Settings -> click the first NIC in the VM -> Custom mode -> choose virtual NIC (Which one map the host NIC that can attach the Internet) -> Save.

Choose the second NIC in the VM as the LAN interface, and set it at the first time. Right-click the VM -> Settings -> Click the second NIC in the VM -> LAN mode -> (for the first time we set LAN: Add -> Rename it as you want (For example is RECO) -> Close -> click the choose list ->)Save.

Choose the third NIC in the VM to connect to the eNB. Right-click the VM -> Settings -> click the third NIC in the VM -> Custom mode -> choose virtual NIC (Which one map the host NIC that can attach the eNB) -> Save.

Notice that:
If we only have one or two real IPs can be used, we have to remove the NIC in the VM after all the HSS installation process.

The way to remove it is that: right-click the VM -> Settings -> click the first (which one connect to the Internet) NIC in the VM -> Remove -> Save.

S/P-GW requires three NICs in the VM.

Choose the first NIC in the VM to connect to Internet, and it needs the real IP because we use the custom mode. Right-click the VM -> Settings -> click the first NIC in the VM -> Custom mode -> choose virtual NIC (Which one map the host NIC that can attach the Internet) -> Save.

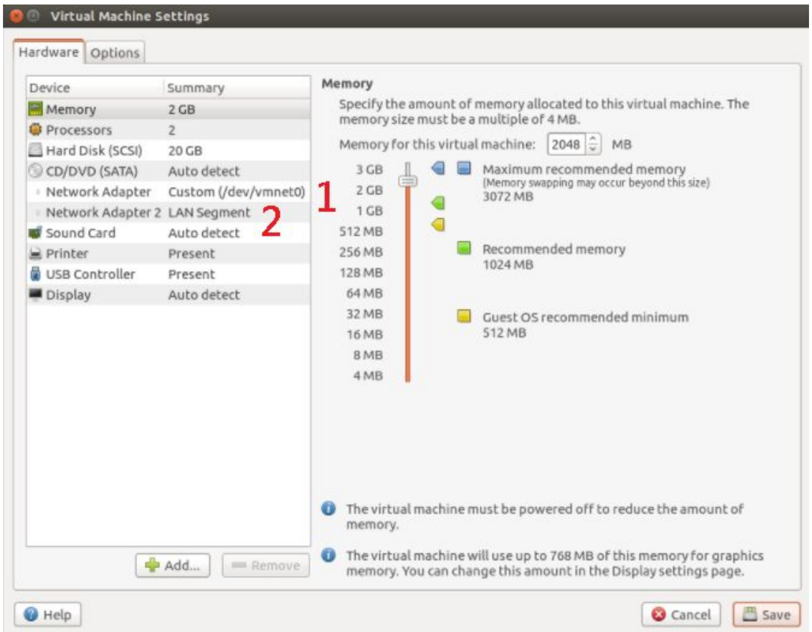
Choose the second NIC in the VM as the LAN interface, and set it at the first time. Right-click the VM -> Settings -> Click the second NIC in the VM -> LAN mode -> (for the first time we set LAN: Add -> Rename it as you want (For example is RECO) -> Close -> click the choose list ->) Save.

Choose the third NIC in the VM to connect to the eNB. Right-click the VM -> Settings -> click the third NIC in the VM-> Custom mode -> choose virtual NIC (Which one map the host NIC that can attach the eNB) -> Save.

2.1.1.4. NICs in VMs and IPs

We have to know the relationship between NICs in VMs and IPs.
This step is at VMs.

The relationship between NICs in the VM and settings of the VM will follow the same order. In other words, the first NIC in the VM maps to the first VM interface, the second NIC in the VM maps to the second VM interface.



```
reco@ubuntu: ~
reco@ubuntu:~$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:16:a7:b8
          inet6 addr: fe80::1e63:265d:6feb:27c9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:689 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1660 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:133324 (133.3 KB)  TX bytes:290699 (290.6 KB)

ens37     Link encap:Ethernet  HWaddr 00:0c:29:16:a7:c2
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1618 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:281432 (281.4 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:29716 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29716 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2199776 (2.1 MB)  TX bytes:2199776 (2.1 MB)
```

After that, it is not a problem to set the IP now. Set the real IP to the VM interface that finally connects to the network port which attaches to the Internet. Set the IP to eNB to the VM interface that finally connects to the network port which attaches to the eNB, and set the IP to the LAN we defined.

The problem now is that what is the command to set the IP to the VM interface? The command is that:

$ sudo ifconfig <VM interface> <IP you want to set to it>

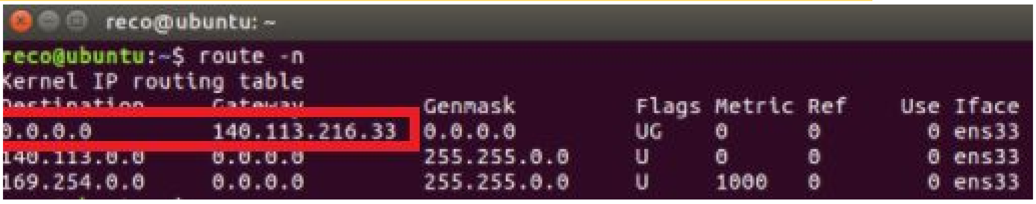
```
reco@ubuntu: ~
reco@ubuntu:~$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:16:a7:b8
          inet addr:140.113.216.58  Bcast:140.113.255.255  Mask:255.255.0.0
          inet6 addr: fe80::1e63:265d:6feb:27c9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:755 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1715 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:147045 (147.0 KB)  TX bytes:300879 (300.8 KB)

ens37     Link encap:Ethernet  HWaddr 00:0c:29:16:a7:c2
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1618 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:281432 (281.4 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:29876 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29876 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2211616 (2.2 MB)  TX bytes:2211616 (2.2 MB)
```

Notice that:
We have to stop the auto connection because it will remove IPs we have set.

Edit connections -> Wired connection x -> General -> delete the check of "Automatically connect to this network when it is available" -> Save.

The final problem is that: Why we do not connect to the Internet even we have set the IP to the VM interface? It is because we have to set more things like the default gateway and the DNS server.

The command to set the default gateway is that:
$ sudo route add -net 0.0.0.0 netmask 0.0.0.0 gw <default gateway>.



The command to set the DNS server is that:
$ sudo chmod 777 /etc/resolvconf/resolv.conf.d/base
$ sudo echo "nameserver 8.8.8.8" > /etc/resolvconf/resolv.conf.d/base
$ sudo resolvconf -u

It should connect to the Internet now!

Notice that:
If the LAN mode NIC in VMs not work, reboot all the VMs.

2.1.2. Auto Installation Script

Notice that:
If we can not install RECO by this script successfully, we can follow the manual installation steps in the later chapters to install RECO.

2.1.2.1. MME

2.1.2.1.1. Download RECO and Other Tools

Download the git tool to download RECO.
$ sudo apt-get update -y
$ sudo apt-get install subversion git -y

Then download RECO.
$ git clone https://github.com/RECONet/RECO.git

2.1.2.1.2. Run The Script

Get into the scripts file.
$ cd ./RECO/SCRIPTS

Run the auto installation script.
$ sudo ./install_RECO MME

Key in the IP architecture. (We use the example IP architecture for example)

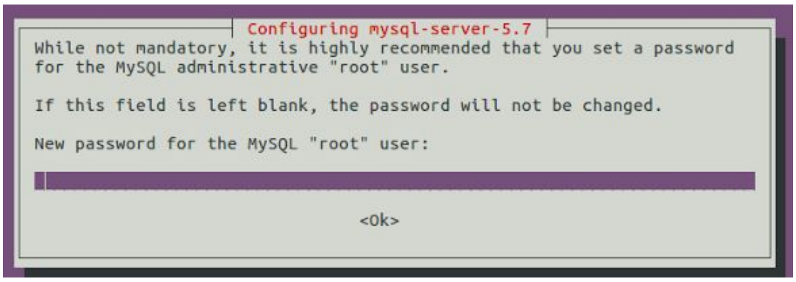
For the connection to eNB
NIC name of MME: ens38
IP address of MME (with mask): 192.168.4.99/24

For the connection to SPGW
NIC name of MME: ens37
IP address of MME (with mask): 10.0.0.2/8
IP address of SPGW (with mask): 10.0.0.3/8

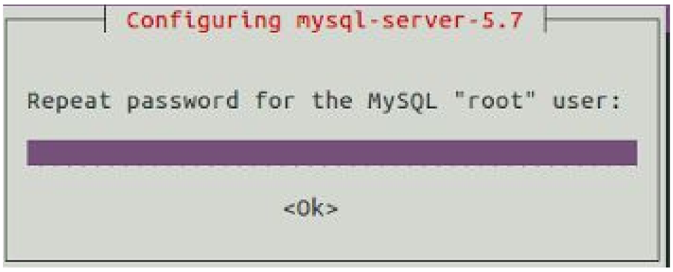
For the connection to HSS
IP address of HSS (without mask): 10.0.0.1

Then press 'Enter' to go to the next state.

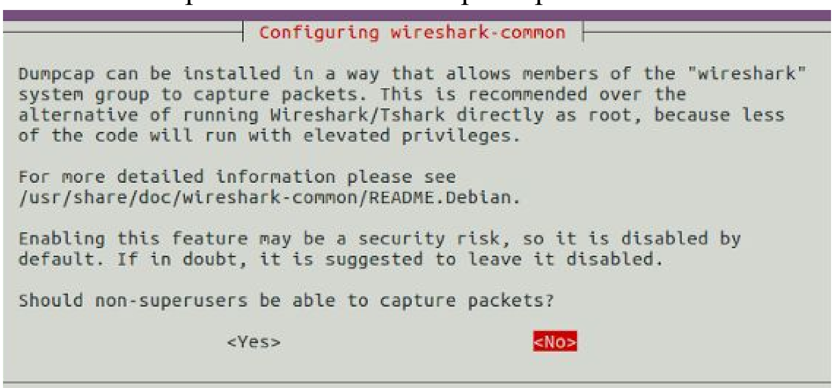
Key in the MySQL password, but it is useless in MME.



Key in the password again.



Should non-superusers be able to capture packets? Yes

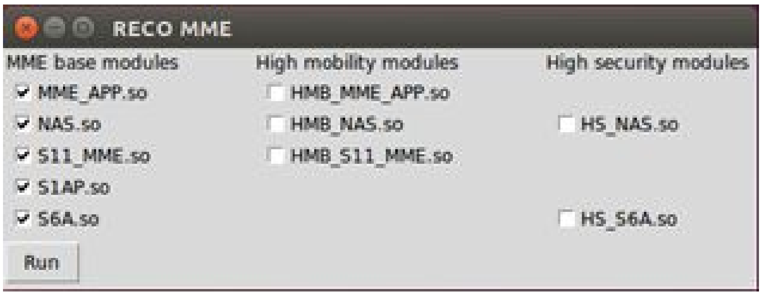


2.1.2.1.3. Run The MME

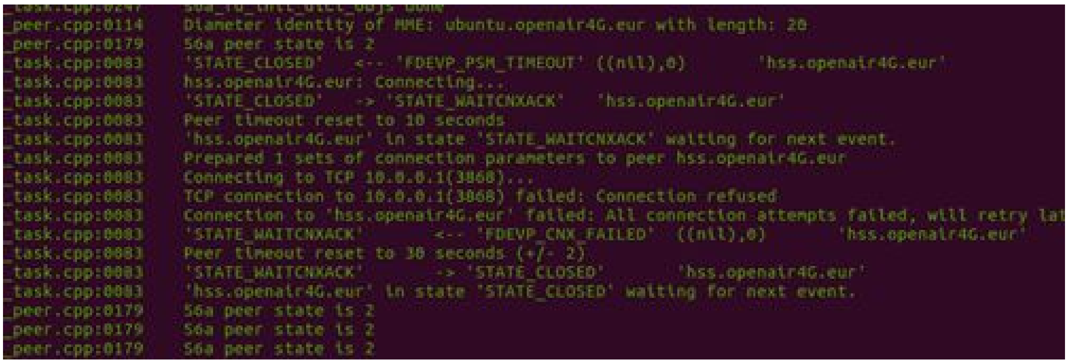
Run the pseudo identifier to start the MME.

```
$ sudo python ./pseudo_identifier.py
```

Check the items in the first column as below and click ‘run’.



Then the MME is running! Congratulations!



If the MME connects to the active HSS, there will be a log as below.

2.1.2.2. HSS

2.1.2.2.1. Download RECO and Other Tools

Download the git tool to download RECO.

$ sudo apt-get update -y
$ sudo apt-get install subversion git -y

Then download RECO.

$ git clone https://github.com/RECONet/RECO.git

2.1.2.2.2. Run The Script

Get into the scripts file.

$ cd ./RECO/SCRIPTS

Run the auto installation script.

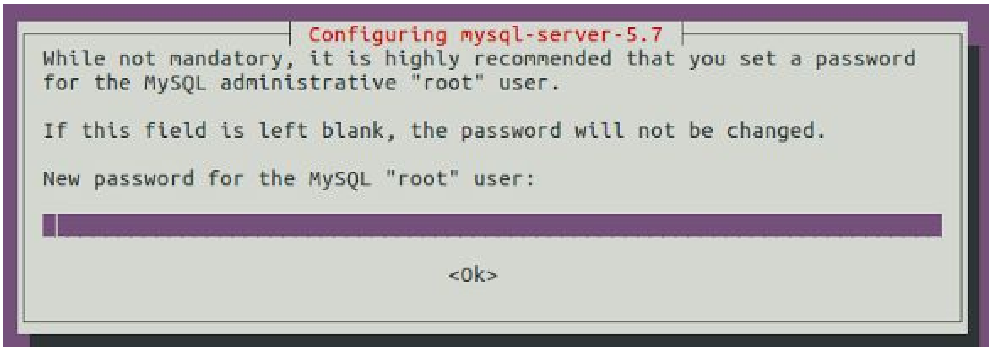
$ sudo ./install_RECO HSS

Key in the MySQL password you want to use. (We use ‘123’ for example)

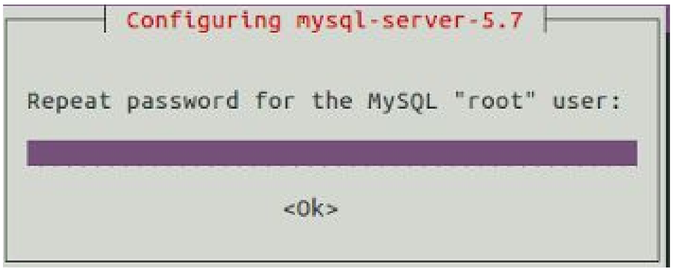
For MySQL database
Password: 123

Then press 'Enter' to go to the next state.

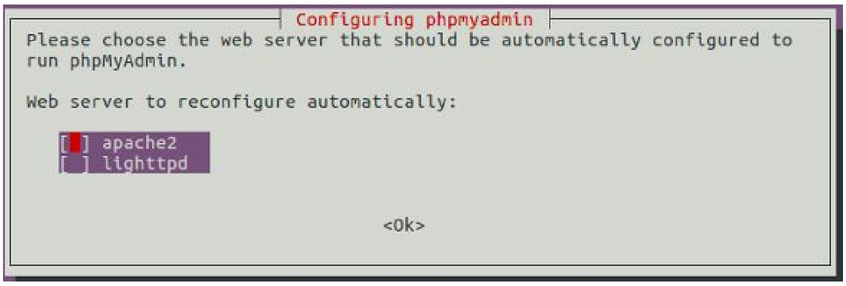
Key in the MySQL password as same as the previous one ('123' for example). This password is important in HSS.



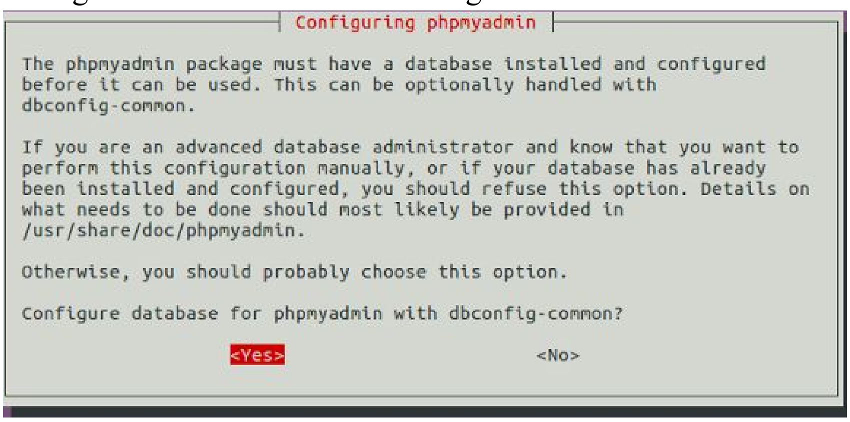
Key in the password again.



Select the web server: apache2



Configure the database with dbconfig-common? No

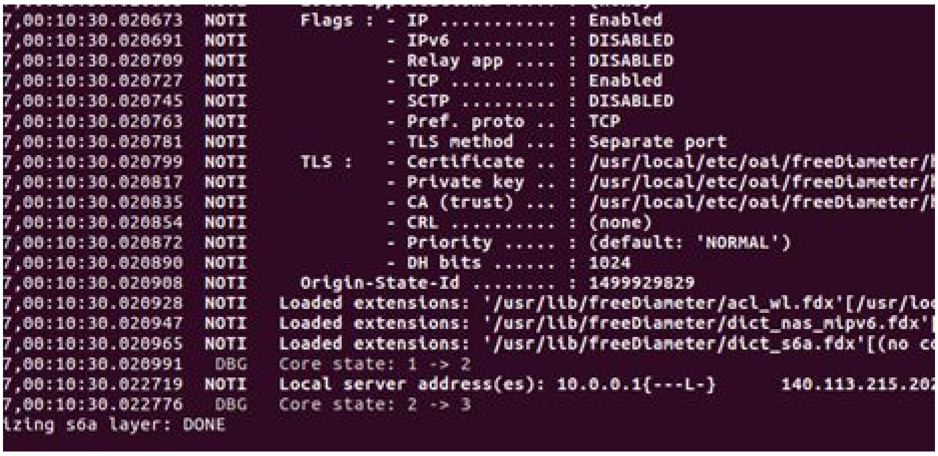


2.1.2.2.3. Run The HSS

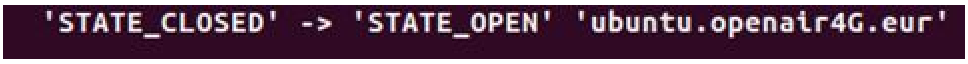
Run the script to start the HSS.

$ sudo ./run_hss

Then the HSS is running! Congratulations!



If the HSS connect to the active MME, there will be a log.



2.1.2.3. S/P-GW

2.1.2.3.1. Download RECO and Other Tools

Download the git tool to download RECO.

$ sudo apt-get update -y
$ sudo apt-get install subversion git -y

Then download RECO.

$ git clone https://github.com/RECONet/RECO.git

2.1.2.3.2. Run The Script

Get into the scripts file.

$ cd ./RECO/SCRIPTS

Run the auto installation script.

$ sudo ./install_RECO SPGW

Key in the IP architecture. (We use the example IP architecture for example)

For the connection to MME
NIC name of SPGW: ens37
IP address of SPGW (with mask): 10.0.0.3/8

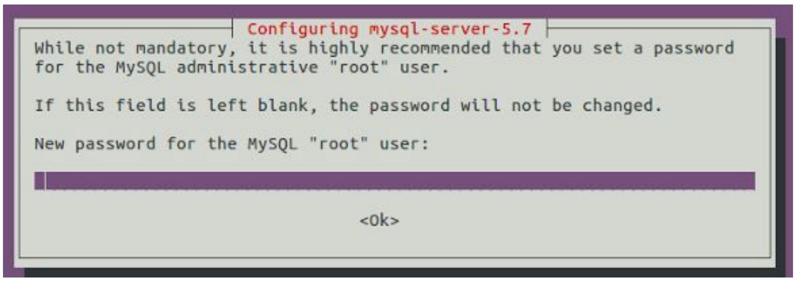
For the connection to eNB
NIC name of SPGW: ens38
IP address of SPGW (with mask): 192.168.4.98/24

For the connection to Internet
NIC name of SPGW: ens33

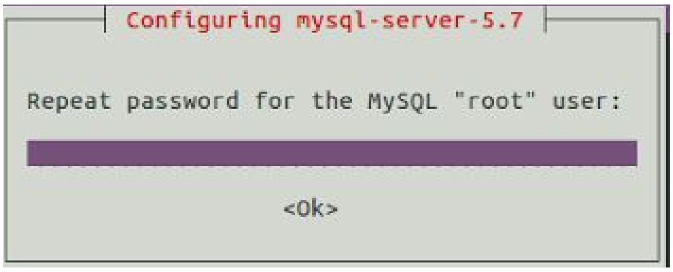
For UE
IP address of UE (with mask): 192.168.0.0/16

Then press 'Enter' to go to the next state.

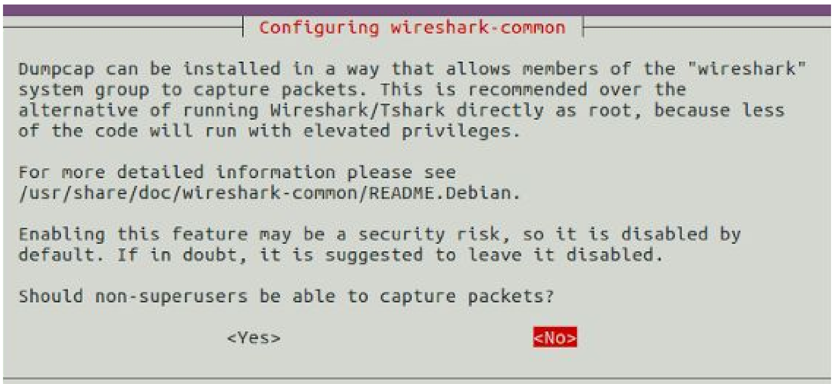
Key in the MySQL password, but it is useless in SPGW.



Key in the password again.



Should non-superusers be able to capture packets? Yes

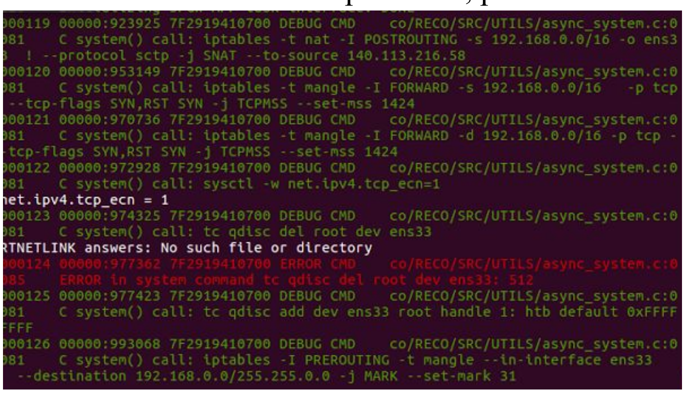


2.1.2.3.3. Run The S/P-GW

Run the script to start the S/P-GW.

$ sudo ./run_spgw

There seems to have some problems, press <control + c> to stop it.



$ <control+c>

Then rerun the script.

The S/P-GW is running! Congratulations!

2.1.3. Manual Installation

2.1.3.1. MME

2.1.3.1.1. Update

$ sudo apt-get update
$ sudo apt-get upgrade

2.1.3.1.2. Download RECO and Other Tools

Download RECO source code from 'github'.

$ git clone https://github.com/RECONet/RECO.git

Download some tools will be used later.

$ cd ./RECO/SCRIPTS
$ sudo ./build_hss -i
$ sudo ./build_mme -i

Notice that:
If there is any asking during the process, choose 'yes' for safety.

2.1.3.1.3 Copy configuration files

Copy configuration files to the particular locations.

$ cd ..
$ cd ./ETC
$ sudo cp mme.conf /usr/local/etc/oai
$ sudo cp mme_fd.conf /usr/local/etc/oai/freeDiameter
$ sudo chmod 777 /usr/local/etc/oai/mme.conf
$ sudo chmod 777 /usr/local/etc/oai/freeDiameter/mme_fd.conf

2.1.3.1.4. File settings

Modify the hostname to 'ubuntu'.

$ sudo vim /etc/hostname

Modify hosts as the picture below.
$ sudo vim /etc/hosts

```
127.0.0.1       localhost
127.0.1.1       ubuntu.openair4G.eur ubuntu
127.0.0.1       hss.openair4G.eur hss

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Set the 'mme_fd.conf' file.
$ sudo vim /usr/local/etc/oai/freeDiameter/mme_fd.conf

Set 'ubuntu.openair4G.eur' to 'Identity'.

```
# -------- Local ---------

# Uncomment if the framework cannot resolv it.
Identity = "ubuntu.openair4G.eur";
Realm = "openair4G.eur";
```

Set 'ConnectTo' as the IP of HSS.

```
ConnectPeer= "hss.openair4G.eur" { ConnectTo = "10.0.0.1";
```

Notice that:
If we have no idea about how to set the IP addresses, see the example in chapter 2.1.4.

Set the 'mme.conf' file.
$ sudo vim ./mme.conf

Notice that:
This file will be copied to '/usr/local/etc/oai' by running 'pseudo_identifier.py'.

Set 'MME_INTERFACE_NAME_FOR_S1_MME', which is the network interface that MME used to connect to the eNB, and set 'MME_IPV4_ADDRESS_FOR_S1_MME', which is the IP with the mask of the network interface.
Set 'MME_INTERFACE_NAME_FOR_S11_MME', which is the network interface that MME used to connect to the S/P-GW, and set 'MME_IPV4_ADDRESS_FOR_S11_MME', which is the IP with the mask of the network interface.

```
NETWORK_INTERFACES :
{
    # MME binded interface for S1-C or S1-MME  communication (S1AP),
    MME_INTERFACE_NAME_FOR_S1_MME           = "ens33";
    MME_IPV4_ADDRESS_FOR_S1_MME             = "192.168.4.99/24";

    # MME binded interface for S11 communication (GTPV2-C)
    MME_INTERFACE_NAME_FOR_S11_MME          = "ens38";
    MME_IPV4_ADDRESS_FOR_S11_MME            = "10.0.0.2/8";
    MME_PORT_FOR_S11_MME                    = 2123;
};
```

Set 'SGW_IPV4_ADDRESS_FOR_S11', which is the IP with the mask of S/P-GW.

```
S-GW_LIST_SELECTION = (
      {ID="tac-lb01.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb02.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb03.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb04.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb05.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb06.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb07.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb08.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb09.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb0a.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb0b.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb0c.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb0d.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb0e.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";},
      {ID="tac-lb0f.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ; SGW_IPV4_ADDRESS_FOR_S11="10.0.0.3/8";}
);
```

2.1.3.1.5. Install libgtpnl

$ cd ..
$ cd ..
$ sudo apt-get install libmnl-dev
$ git clone git://git.osmocom.org/libgtpnl
$ cd ./libgtpnl
$ autoreconf -fi
$ ./configure
$ make
$ sudo make install

2.1.3.1.6. Build the MME

$ cd ..
$ cd ./RECO/SCRIPTS
$ sudo ./build_mme -c

2.1.3.1.7. Check the certification

$ sudo ./check_mme_s6a_certificate /usr/local/etc/oai/freeDiameter/ ubuntu.openair4G.eur

```
wirelab@ubuntu:~/NctuReco_Demo_CXX/SCRIPTS$ sudo ./check_mme_s6a_certificate /usr/local/etc/oai/freeDiameter/ ubuntu.openair4G.eur
File /usr/local/etc/oai/freeDiameter//mme.cert.pem not found
MME S6A: Did not find valid certificate in /usr/local/etc/oai/freeDiameter/
MME S6A: generating new certificate in /usr/local/etc/oai/freeDiameter/...
Creating MME certificate for user 'ubuntu.openair4G.eur'
Generating a 1024 bit RSA private key
.........******
.............******
writing new private key to 'mme.cakey.pem'
-----
Generating RSA private key, 1024 bit long modulus
...............................******
..........................******
e is 65537 (0x10001)
Using configuration from /usr/lib/ssl/openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Jul 13 08:15:59 2017 GMT
            Not After : Jul 13 08:15:59 2018 GMT
        Subject:
            countryName               = FR
            stateOrProvinceName       = PACA
            organizationName          = Eurecom
            organizationalUnitName    = CM
            commonName                = ubuntu.openair4G.eur
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                48:E8:B5:2B:91:C0:FB:6F:FB:20:15:C3:4A:CA:6D:1C:99:76:AD:38
            X509v3 Authority Key Identifier:
                keyid:85:8A:19:F3:D0:38:AA:24:A3:1E:DD:D4:2A:B7:AF:8D:49:98:4E:D3

Certificate is to be certified until Jul 13 08:15:59 2018 GMT (365 days)

Write out database with 1 new entries
Data Base Updated
/home/wirelab/NctuReco_Demo_CXX/SCRIPTS
MME S6A: Found valid certificate in /usr/local/etc/oai/freeDiameter/
wirelab@ubuntu:~/NctuReco_Demo_CXX/SCRIPTS$
```

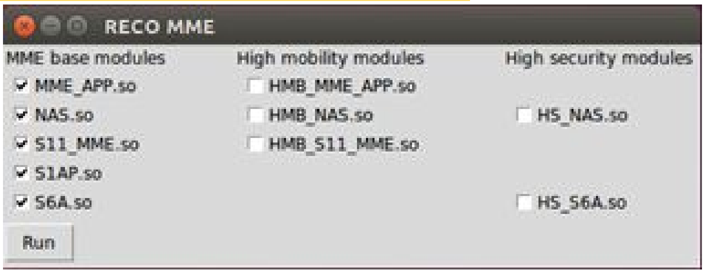
2.1.3.1.8. Install packet python-tk

$ sudo apt-get install python-tk

2.1.3.1.9. Run the MME

Run the script to simulate the identifier to perform dynamic linking.

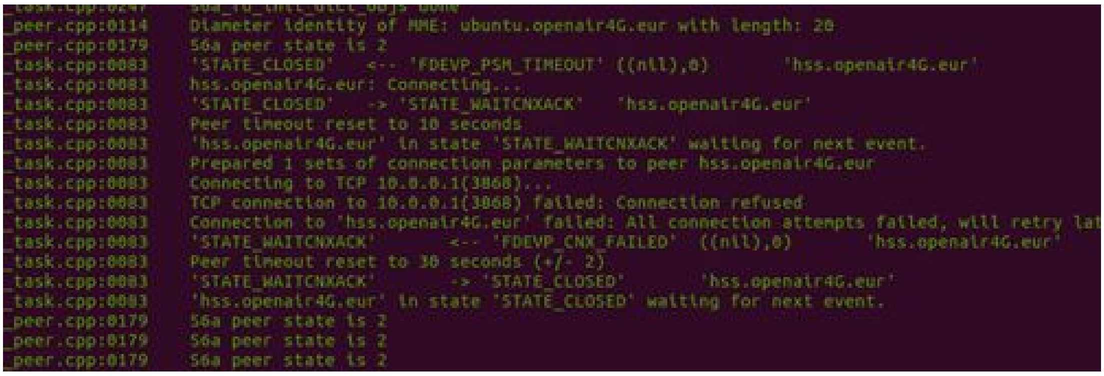
$ sudo python ./pseudo_identifier.py



Notice that:

In the latest version, we suggest checking the items in the first column.

Click 'Run' to start the MME.



When the MME connects to the HSS, there will be a log.

Notice that:

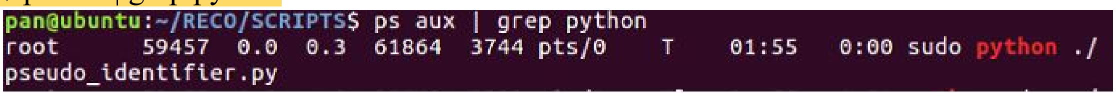
The HSS must be active to connect to the MME.

Notice that:

If we want to rerun the MME, type commands below to release sources.

Find the 'pid' of the process. It is '59457' in the picture below for example.

$ ps aux | grep python



Then kill the process with the 'pid' we have found.

$ sudo kill -9 <pid>

2.1.3.2. HSS

2.1.3.2.1. Update

$ sudo apt-get update
$ sudo apt-get upgrade

2.1.3.2.2. Download RECO and other tools

Download RECO source code from 'github'.
$ git clone https://github.com/RECONet/RECO.git

Download some tools will be used later.
$ cd ./RECO/SCRIPTS
$ sudo ./build_hss -i

Notice that:
The password we set when running the 'build_hss -i' at the first time will be used to log in the MySQL database, and we set '123' to it for example.

Notice that:
If there is any asking during the process, choose 'yes' for safety.

2.1.3.2.3 Copy configuration files

Copy configuration files to the particular locations.
$ cd ..
$ cd ./ETC
$ sudo cp hss.conf /usr/local/etc/oai
$ sudo cp hss_fd.conf /usr/local/etc/oai/freeDiameter
$ sudo cp acl.conf /usr/local/etc/oai/freeDiameter
$ sudo chmod 777 /usr/local/etc/oai/hss.conf
$ sudo chmod 777 /usr/local/etc/oai/freeDiameter/hss_fd.conf
$ sudo chmod 777 /usr/local/etc/oai/freeDiameter/acl.conf

2.1.3.2.4 File settings

Modify the hostname to 'ubuntu'.
$ sudo vim /etc/hostname

Modify hosts as the picture below.
$ sudo vim /etc/hosts

```
127.0.0.1       localhost
127.0.1.1       ubuntu.openair4G.eur ubuntu
127.0.0.1       hss.openair4G.eur hss

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Set the 'hss.conf' file.
$ sudo vim /usr/local/etc/oai/hss.conf

Set 'MYSQL_user' and 'MYSQL_pass' as same as the database.
Set serial '1's to 'OPERATOR_key'.

```
HSS :
{
## MySQL mandatory options
MYSQL_server = "127.0.0.1";      # HSS S6a bind address
MYSQL_user   = "root";  # Database server login
MYSQL_pass   = "123";  # Database server password
MYSQL_db     = "oai_db";         # Your database name

## HSS options
#OPERATOR_key = "1006020f0a478bf6b699f15c062e42b3"; # OP key matching your database
OPERATOR_key = "11111111111111111111111111111111"; # OP key matching your database
```

Notice that:
The 'MYSQL_user' is 'root', and the 'MYSQL_pass' is the password we set in the previous step, for example, that is '123'.

2.1.3.2.5 Database import

Create database 'oai_db'.

$ mysql -u root -p
mysql > CREATE DATABASE oai_db;
mysql > exit

Notice that:
The password is the one we set in the previous step, for example is '123'.

Import data to 'oai_db'.

$ mysql -u root -p oai_db < ~/RECO/SRC/OAI_HSS/db/oai_db.sql
$ mysql -u root -p
mysql > USE oai_db;
mysql > select * from mmeidentity;
mysql > UPDATE mmeidentity SET mmehost = 'ubuntu.openair4G.eur' WHERE idmmeidentity = '4';
mysql > exit

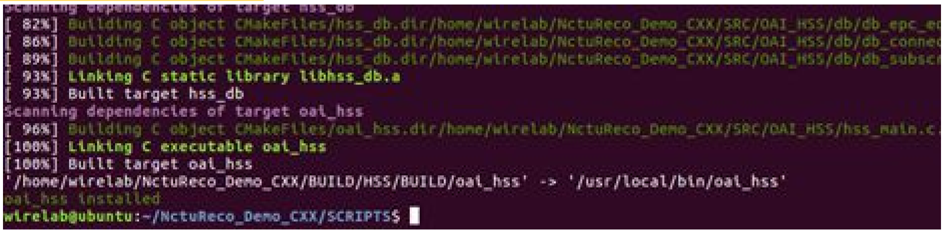
2.1.3.2.6 Check the certification

$ cd ..
$ cd ./SCRIPTS
$ sudo ./check_hss_s6a_certificate /usr/local/etc/oai/freeDiameter/hss.openair4G.eur

```
        countryName               = FR
        stateOrProvinceName       = PACA
        organizationName          = Eurecom
        organizationalUnitName    = CM
        commonName                = hss.openair4G.eur
 X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        Netscape Comment:
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            B5:6A:FC:9C:A8:AF:78:26:04:7D:9F:7A:07:CC:C1:37:FC:F1:E1:24
        X509v3 Authority Key Identifier:
            keyid:F6:30:7A:AD:55:C0:D0:5F:6F:13:A2:47:F8:0E:3D:94:80:25:DF:06

cate is to be certified until Jul 13 06:53:12 2018 GMT (365 days)

ut database with 1 new entries
se Updated
irelab/NctuReco_Demo_CXX/SCRIPTS
: Found valid certificate in /usr/local/etc/oai/freeDiameter/
@ubuntu:~/NctuReco_Demo_CXX/SCRIPTS$
```

2.1.3.2.7 Build the HSS

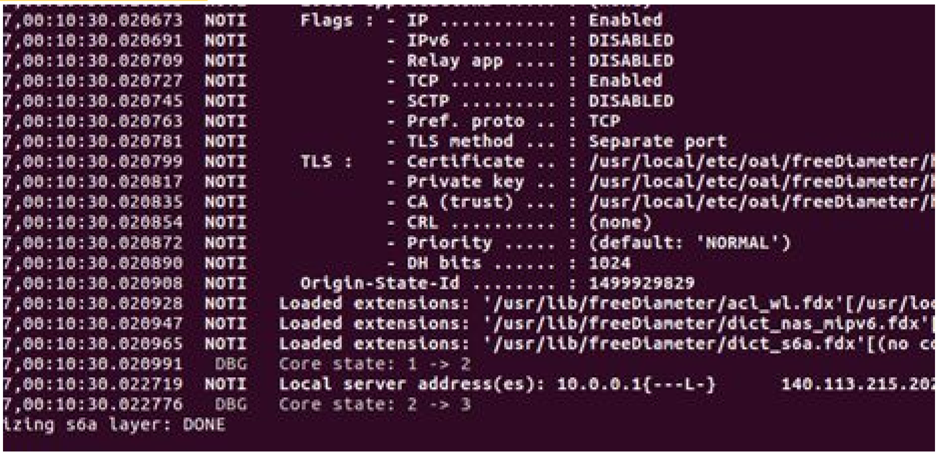
$ sudo ./build_hss -c



```
[ 82%] Building C object CMakeFiles/hss_db.dir/home/wirelab/NctuReco_Demo_CXX/SRC/OAI_HSS/db/db_epc_e
[ 86%] Building C object CMakeFiles/hss_db.dir/home/wirelab/NctuReco_Demo_CXX/SRC/OAI_HSS/db/db_conne
[ 89%] Building C object CMakeFiles/hss_db.dir/home/wirelab/NctuReco_Demo_CXX/SRC/OAI_HSS/db/db_subsc
[ 93%] Linking C static library libhss_db.a
[ 93%] Built target hss_db
Scanning dependencies of target oai_hss
[ 96%] Building C object CMakeFiles/oai_hss.dir/home/wirelab/NctuReco_Demo_CXX/SRC/OAI_HSS/hss_main.c
[100%] Linking C executable oai_hss
[100%] Built target oai_hss
'/home/wirelab/NctuReco_Demo_CXX/BUILD/HSS/BUILD/oai_hss' -> '/usr/local/bin/oai_hss'
oai_hss installed
wirelab@ubuntu:~/NctuReco_Demo_CXX/SCRIPTS$
```

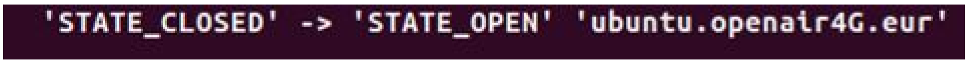
2.1.3.2.8 Run the HSS

$ sudo ./run_hss



```
7,00:10:30.020673  NOTI     Flags : - IP ........... : Enabled
7,00:10:30.020691  NOTI             - IPv6 ......... : DISABLED
7,00:10:30.020709  NOTI             - Relay app .... : DISABLED
7,00:10:30.020727  NOTI             - TCP .......... : Enabled
7,00:10:30.020745  NOTI             - SCTP ......... : DISABLED
7,00:10:30.020763  NOTI             - Pref. proto .. : TCP
7,00:10:30.020781  NOTI             - TLS method ... : Separate port
7,00:10:30.020799  NOTI     TLS :   - Certificate .. : /usr/local/etc/oai/freeDiameter/
7,00:10:30.020817  NOTI             - Private key .. : /usr/local/etc/oai/freeDiameter/
7,00:10:30.020835  NOTI             - CA (trust) ... : /usr/local/etc/oai/freeDiameter/
7,00:10:30.020854  NOTI             - CRL .......... : (none)
7,00:10:30.020872  NOTI             - Priority ..... : (default: 'NORMAL')
7,00:10:30.020890  NOTI             - DH bits ...... : 1024
7,00:10:30.020908  NOTI     Origin-State-Id ........ : 1499929829
7,00:10:30.020928  NOTI   Loaded extensions: '/usr/lib/freeDiameter/acl_wl.fdx'[/usr/lo
7,00:10:30.020947  NOTI   Loaded extensions: '/usr/lib/freeDiameter/dict_nas_mipv6.fdx'
7,00:10:30.020965  NOTI   Loaded extensions: '/usr/lib/freeDiameter/dict_s6a.fdx'[(no c
7,00:10:30.020991   DBG   Core state: 1 -> 2
7,00:10:30.022719  NOTI   Local server address(es): 10.0.0.1{---L-}      140.113.215.20
7,00:10:30.022776   DBG   Core state: 2 -> 3
izing s6a layer: DONE
```

When the HSS connect to the MME, there will be a log.



```
'STATE_CLOSED' -> 'STATE_OPEN' 'ubuntu.openair4G.eur'
```

Notice that:

MME must be active to connect to HSS.

2.1.3.2.9 Phpmyadmin

It provides the GUI for database operations.

$ sudo apt-get install phpmyadmin

$ sudo ln -s /etc/phpmyadmin/apache.conf /etc/apache2/conf-available/phpmyadmin.conf

$ sudo a2enconf phpmyadmin

$ sudo /etc/init.d/apache2 reload

$ sudo reboot

We can operate data by accessing 'http://127.0.0.1/phpmyadmin'.

Insert data of the SIM card into the database 'oai_db'.

Notice that:
If we do not insert SIM card data, we will fail to connect to the Internet.

2.1.3.3. S/P-GW

2.1.3.3.1. Update

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

2.1.3.3.2. Download RECO and other tools

Download RECO source code from 'github'.

```
$ git clone https://github.com/RECONet/RECO.git
```

Download some tools will be used later.

```
$ cd ./RECO/SCRIPTS
$ sudo ./build_hss -i
$ sudo ./build_mme -i
$ sudo ./build_spgw -i
```

Notice that:
If there is any asking during the process, choose 'yes' for safety.

2.1.3.3.3 Copy configuration files

Copy configuration files to the particular locations.

```
$ cd ..
$ cd ./ETC
$ sudo cp spgw.conf /usr/local/etc/oai
$ sudo chmod 777 /usr/local/etc/oai/spgw.conf
```

2.1.3.3.4 File settings

Set the 'spgw.conf' file.

```
$ sudo vim /usr/local/etc/oai/spgw.conf
```

Set 'SGW_INTERFACE_NAME_FOR_S11', which is the network interface S/P-GW used to connect to the MME and set 'SGW_IPV4_ADDRESS_FOR_S11', which is the IP with the mask of the network interface.
Set 'SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP', which is the network interface S/P-GW used to connect to the eNB and set 'SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP', which is the IP with the mask of the network interface.

```
NETWORK_INTERFACES :
{
    # S-GW binded interface for S11 communication (GTPV2-C), if none sel
    SGW_INTERFACE_NAME_FOR_S11                = "ens38";
    SGW_IPV4_ADDRESS_FOR_S11                  = "10.0.0.3/8";

    # S-GW binded interface for S1-U communication (GTPV1-U) can be ethe
    SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP      = "ens33";
    SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP        = "192.168.4.98/24";
    SGW_IPV4_PORT_FOR_S1U_S12_S4_UP           = 2152;
```

Notice that:
If we have no idea about how to set the IP addresses, see the example in chapter 2.1.4.

Set 'PGW_INTERFACE_NAME_FOR_SGI', which is the network interface S/P-GW used to connect to the internet.
Set 'yes' to 'PGW_MASQUERADE_SGI' and 'UE_TCP_MSS_CLAMPING' to avoid failure.

```
 # P-GW binded interface for SGI (egress/ingress internet traffic)
 PGW_INTERFACE_NAME_FOR_SGI                 = "ens37";
 PGW_MASQUERADE_SGI                         = "yes";
 UE_TCP_MSS_CLAMPING                        = "yes";
```

Set 'IPV4_LIST', which is a scope of IPs distributed to UEs connecting to the S/P-GW.

```
IP_ADDRESS_POOL :
{
    IPV4_LIST = (
                  "172.16.0.0/12"
                );
};
```

2.1.3.3.5 Build the SPGW

$ cd ..
$ cd ./SCRIPTS
$ sudo ./build_spgw -c

```
git found: /usr/bin/git
NETTLE_VERSION_INSTALLED  = 3.2
NETTLE_VERSION_MAJOR = 3
NETTLE_VERSION_MINOR = 2
spgw compiled
'/home/wirelab/NctuReco_Demo_CXX/BUILD/SPGW/BUILD/spgw' -> '/usr/local/bin/spgw'
spgw installed
wirelab@ubuntu:~/NctuReco_Demo_CXX/SCRIPTS$
```

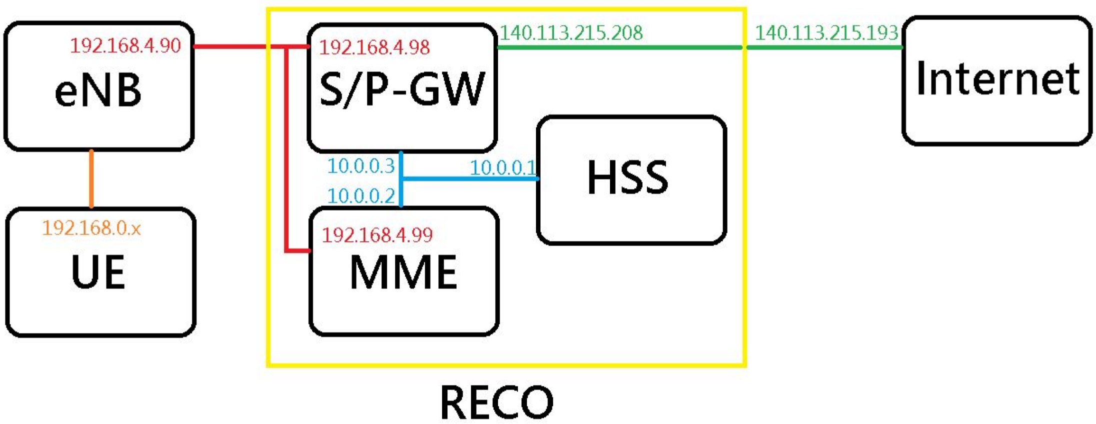
2.1.3.3.6 Run the SPGW

$ sudo ./run_spgw

```
*------------------------------------------------------------------------*
*                                                                        *
*                          n w - g t p v 2 c                             *
*   G P R S   T u n n e l i n g   P r o t o c o l   v 2 c   S t a c k    *
*                                                                        *
*                                                                        *
* Copyright (c) 2010-2011 Amit Chawre                                    *
* All rights reserved.                                                   *
*                                                                        *
* Redistribution and use in source and binary forms, with or without     *
* modification, are permitted provided that the following conditions     *
* are met:                                                               *
*                                                                        *
* 1. Redistributions of source code must retain the above copyright      *
*    notice, this list of conditions and the following disclaimer.       *
* 2. Redistributions in binary form must reproduce the above copyright   *
*    notice, this list of conditions and the following disclaimer in the *
*    documentation and/or other materials provided with the distribution.*
* 3. The name of the author may not be used to endorse or promote products *
*    derived from this software without specific prior written permission. *
*                                                                        *
* THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR   *
* IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES *
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. *
* IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,       *
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, *
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  *
* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT    *
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF *
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.      *
*------------------------------------------------------------------------*

Tx UDP_INIT IP addr 10.0.0.3
Creating new listen socket on address 10.0.0.3 and port 2123
Inserting new descriptor for task 6, sd 31
Received 1 events
Initializing S11 interface: DONE
Initializing SPGW-APP  task interface
Initializing GTPV1U interface

Using the GTP kernel mode (genl ID is 27)
Setting route to reach UE net 172.16.0.0 via gtp0
GTP kernel configured
Initializing GTPV1U interface: DONE
Initializing SPGW-APP task interface: DONE
```

2.1.4. Example of IP Settings

2.1.4.1. Architecture Overview with IPs



Notice that:
'140.113.215.193' is the IP address of the default gateway.

Notice that:
The IPs may be different from the IP architecture at the chapter 2.1.1.

2.1.4.2. IP Settings of HSS

2.1.4.2.1. Network Interface Settings

Notice that:
We need at least one network interface card (NIC).

Find the name of the NIC.
```
$ ifconfig
```

Set IP Address to the NIC.
```
$ sudo ifconfig <name> 10.0.0.1
```

Notice that:
<name> is the name of the NIC.

2.1.4.3. IP Settings of MME

2.1.4.3.1. Network Interface Settings

Notice that:
We need at least two network interface cards (NICs).

Find the names of the NICs.
```
$ ifconfig
```

Set IP Addresses to the NICs.
```
$ sudo ifconfig <name_1> 10.0.0.2
$ sudo ifconfig <name_2> 192.168.4.99
```

Notice that:
<name_1> is the name of the first NIC.
<name_2> is the name of the second NIC.

2.1.4.3.2. Configuration File Settings

mme_fd.conf:
```
ConnectTo = "10.0.0.1"
```

mme.conf:
```
MME_INTERFACE_NAME_FOR_S1_MME  =  "<name_2>"
MME_IPV4_ADDRESS_FOR_S1_MME  =  "192.168.4.99/24"
MME_INTERFACE_NAME_FOR_S11_MME  =  "<name_1>"
MME_IPV4_ADDRESS_FOR_S11_MME  =  "10.0.0.2/8"
SGW_IPV4_ADDRESS_FOR_S11  =  "10.0.0.3/8"
```

Notice that:
Here we only set the part of IPs and NICs names. As for other settings, please follow the previous chapters.

2.1.4.4. IP Settings of S/P-GW

2.1.4.4.1. Network Interface Settings

Notice that:
We need at least three network interface cards (NICs).

Find the names of the NICs.
$ ifconfig

Set IP Addresses to the NICs.
$ sudo ifconfig <name_1> 10.0.0.3
$ sudo ifconfig <name_2> 192.168.4.98
$ sudo ifconfig <name_3> 140.113.215.2083

Notice that:
<name_1> is the name of the first NIC.
<name_2> is the name of the second NIC.
<name_2> is the name of the third NIC.

Notice that:
If ping google.com failed, type commands below to connect to the Internet.

$ sudo route add -net 0.0.0.0 netmask 0.0.0.0 gw 140.113.215.193
$ sudo chmod 777 /etc/resolvconf/resolv.conf.d/base
$ sudo echo "nameserver 8.8.8.8" >> /etc/resolvconf/resolv.conf.d/base
$ sudo resolvconf -u

Notice that:
'140.113.215.193' is the IP address of the default gateway.

2.1.4.4.2. Configuration File Settings

spgw.conf:

SGW_INTERFACE_NAME_FOR_S11 = "<name_1>"
SGW_IPV4_ADDRESS_FOR_S11 = "10.0.0.3/8"
SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP = "<name_2>"
SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP = "192.168.4.98/24"
PGW_INTERFACE_NAME_FOR_SGI = "<name_3>"
IPV4_LIST = "192.168.0.0/16"

Notice that:
Here we only set the part of IPs and NICs names. As for other settings, please follow the previous chapters.

2.2. Radio Access Network

2.2.1. SIM Card

The SIM card we use: **sysmoUSIM-SJS1 (with ADM keys)**
You can buy it at the following link:
http://shop.sysmocom.de/products/sysmousim-sjs1

We can use PySIM to program the SIM card.
Install packages:
$ sudo apt-get install pcscd pcsc-tools libccid python-dev swig python-setuptools python-pip libpcsclite-dev
$ sudo pip install pycrypto

Download PySIM from git:
$ git clone git://git.osmocom.org/pysim.git

Also need Pyscard:
Download from
https://sourceforge.net/projects/pyscard/files/pyscard/pyscard%201.9.5/pyscard-1.9.5.tar.gz/download

Install command:
$ cd <pyscard-path>
$ sudo /usr/bin/python setup.py build_ext install

Use the following command to check whether card reader is ready:
$ sudo pcsc_scan

If you see this picture, you are ready to program the SIM card:

```
        - Implicit DF selection
        - Short EF identifier supported
        - Record number supported
      Data coding byte: 21
        - Behaviour of write functions: proprietary
        - Value 'FF' for the first byte of BER-TLV tag fields: invalid
        - Data unit in quartets: 2
      Command chaining, length fields and logical channels: 13
        - Logical channel number assignment: by the card
        - Maximum number of logical channels: 4
    Tag: 6, len: 7 (pre-issuing data)
      Data: 43 20 07 18 00 00 01
+ TCK = A5 (correct checksum)

Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
3B 9F 96 80 1F C7 80 31 A0 73 BE 21 13 67 43 20 07 18 00 00 01 A5
        sysmoUSIM-SJS1 (Telecommunication)
        http://www.sysmocom.de/products/sysmousim-sjs1-sim-usim
^C
john@ubuntu:~/pyscard-1.9.5$
```

Read SIM card information:
$ cd <pysim-path>
$ ./pySim-read.py -p 0

```
john@ubuntu: ~/pysim
john@ubuntu:~/pysim$ ./pySim-read.py -p 0
Reading ...
ICCID: 8988211000000088347
IMSI: 208930000009487
SMSP: ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffff
ACC: 0080
MSISDN: Not available
Done !

john@ubuntu:~/pysim$
```

Program SIM card:
You need to prepare the following information:
-x MCC ： **Mobile Country Code,** the first 3 letter of IMSI
-y MNC ： **Mobile Network Code,** the 4th and 5th letter of IMSI

-i IMSI：**International Mobile Subscriber Identity,** presented as a 15 digit number
op OP：**Operator Code,** presented as a 32 digit number
-k KI：**Subscriber Authentication Key,** presented as a 32 digit number
-s ICCID：**Integrated Circuit Card Identifier,** presented as a 20 digit number
-a ADM1：The password uses to programming SIM card, presented as a 8 digit number

Programming commands:
$ cd <pysim-path>
$ ./pySim-prog.py -p 0 -x 466 -y 86 -t sysmoUSIM-SJS1 -i 466862054321003 --op=97A167DED889B6DFA92D985D77E5C088 -k 808182888485868788898A8B8C8D8E8F -s 8988211000000088313 -a 23605945

```
john@ubuntu: ~/pysim
john@ubuntu:~/pysim$ ./pySim-prog.py -n NCTU -p 0 -x 208 -y 93 -t sysmoUSIM-SJS1
 -i 208930000009487 --op=11111111111111111111111111111111 -k 8baf473f2f8fd09487c
ccbd7097c6862 -s 8988211000000088347 -a 85451664
Insert card now (or CTRL-C to cancel)
Generated card parameters :
 > Name     : NCTU
 > SMSP     : e1ffffffffffffffffffffffff0581005155f5ffffffffffff000000
 > ICCID    : 8988211000000088347
 > MCC/MNC  : 208/93
 > IMSI     : 208930000009487
 > Ki       : 8baf473f2f8fd09487cccbd7097c6862
 > OPC      : 8e27b6af0e692e750f32667a3b14605d
 > ACC      : None

Programming ...
Done !

john@ubuntu:~/pysim$
```

Verification:
$ ./pySim-read.py -p 0

If the IMSI changed, done!

```
john@ubuntu: ~/pysim
Generated card parameters :
 > Name     : NCTU
 > SMSP     : e1ffffffffffffffffffffffff0581005155f5ffffffffffff000000
 > ICCID    : 8988211000000088347
 > MCC/MNC  : 208/93
 > IMSI     : 208930000009487
 > Ki       : 8baf473f2f8fd09487cccbd7097c6862
 > OPC      : 8e27b6af0e692e750f32667a3b14605d
 > ACC      : None

Programming ...
Done !

john@ubuntu:~/pysim$ ./pySim-read.py -p 0
Reading ...
ICCID: 8988211000000088347
IMSI: 208930000009487
SMSP: ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffff
ACC: 0080
MSISDN: Not available
Done !

john@ubuntu:~/pysim$
```

2.2.2. eNodeB

2.2.2.1. Commercial eNodeB - Wistron NeWeb OSQ4G-01E2

First, connect to the eNB from the MME.

$ telnet <eNB IP>

Then log in the eNB as a root user.

$ root

Edit the configuration file. (Here are the examples)

$ vi /mnt/flash/etc/fsm/xml/provisioning.xml

```
S1-MME IP address:
<field name="s1SigLinkServerList" value="192.188.2.2"/>
PLMN:
<field name="plmnId" value="20893"/>
```

→ After rebooting the eNodeB, it can connect to MME through WAN port.

Notice that:
The 's1SigLinkServerList' is the IP of MME used to connect to the eNB.

Notice that:
The 'plmnId' is the same value as the PLMN ID of the SIM card.

2.2.2.2. OAI eNodeB

2.2.2.2.1. USRP B210

※It is recommend to use the USB3.0 port.

2.2.2.2.1.1. USRP driver installation

◎UHD binary installation

$ sudo add-apt-repository ppa:ettusresearch/uhd
$ sudo apt-get update
$ sudo apt-get install libuhd-dev libuhd003 uhd-host

◎Building and Installing UHD from source

$ sudo apt-get install libboost-all-dev libusb-1.0-0-dev python-mako doxygen python-docutils cmake build-essential
$ git clone --recursive git://github.com/EttusResearch/uhd.git
$ cd <uhd-repo-path>/host
$ mkdir build
$ cd build
$ cmake ../
$ make
$ make test
$ sudo make install
$ sudo ldconfig

2.2.2.2.1.2. Building OAI executables from source

```
$ git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
$ cd YOUR_openairinterface5g_DIRECTORY
$ source oaienv # Very important. It sets the correct environment variables
$ cd cmake_targets
$ ./build_oai -I -w USRP # Package installation + USRP Driver installation
$ ./build_oai --eNB -c -w USRP
```

2.2.2.2.1.3. Start the eNodeB with USRP B210

Here we use the configuration file: enb.band7.tm1.usrpb210.conf
(Remember to check the configuration of PLMN ID and the interface between MME and eNodeB.)

```
$ cd $OPENAIR_DIR/cmake_targets/lte_build_oai/build
$ sudo -E ./lte-softmodem -O
$OPENAIR_DIR/targets/PROJECT/GENERIC-LTE-EPC/CONF/enb.band7.tm1.usrpb210.conf
```

You can see some messages of s1ap_setup on your MME machine after the connection established.

2.2.2.2.2. ExpressMimo2

2.2.2.2.2.1. ExpressMimo2 card setup

Initialize express MIMO card

```
$ cd openairinterface5g
$ . oaienv
$ cd cmake_targets/tools/
$  . init_exmimo2
```

You should see the following output on the console

```
loading openair_rf
Using firware version 10
```

Running “dmesg”, you should see something ending with

```
[782979.116663] [openair][IOCTL] ok asked Leon to set stack and start execution (addr 0x40000000, stackptr 43fffff0)
[782979.116782] [LEON card0]: FWINIT: Will start execution @ 40000000, stack @ 43fffff0
[782979.228844] [LEON card0]: pcie_initialize_interface_bot(): firmware_block_ptr 3200100, printk_buffer_ptr 3240100, pci_interface_ptr 3240500, exmimo_id_ptr 3240700
[782979.229321] [LEON card0]: System Info:
[782979.229464] [LEON card0]: Bitstream: SVN Revision: 5307, Build date (GMT): Wed 2014-03-19 15:55:01,  User ID: 0x0001
[782979.229600] [LEON card0]: Software:  SVN Revision: 5541, Build date (GMT): Wed 2014-03-19 08:45:08
[782979.229691] [LEON card0]: ExpressMIMO-2 SDR! (Built on Nov  7 2014 15:14:44)
[782979.229819] [LEON card0]: Initialized LIME.
[782979.229935] [LEON card0]: Initializing RF Front end chain0 (to TVWS_TDD).
[782979.230209] [LEON card0]: ready.
```

2.2.2.2.2.2. Building OAI executables from source

```
$ git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
$ cd YOUR_openairinterface5g_DIRECTORY
$ source oaienv # Very important. It sets the correct environment variables.
$ cd cmake_targets
$ ./build_oai -I # Package installation + EXMIMO Driver installation
$ ./build_oai --eNB -w EXMIMO -c -s # eNodeB + EXMIMO + test
```

2.2.2.2.2.3. Start the eNodeB with ExpressMimo2

Here we use the configuration file: enb.band7.tm1. exmimo2.conf
(Remember to check the configuration of PLMN ID and the interface between MME and eNodeB.)

```
$ cd $OPENAIR_DIR/cmake_targets/lte_build_oai/build
$ sudo -E ./lte-softmodem -O
$OPENAIR_DIR/targets/PROJECT/GENERIC-LTE-EPC/CONF/enb.band7.tm1.exmimo2.conf
```

You can see some messages of s1ap_setup on your MME machine after the connection established.

3. Conclusion

We have presented a reconfigurable core network architecture called RECO to efficiently implement customized core network entities for a heterogeneous 5G environment. We also built a reconfigurable MME (RECO MME) to verify our RECO concept. We specifically introduced the implementation of how the dynamic linking framework is implemented in RECO MME and show that our RECO MME has the benefits of (1) reduce disk space and memory usage (2) easy to update and deploy (3) flexibility to link certain modules to form a customized MME (4) Object-oriented code structure which allows programmers to reuse code when forming a customized MME. Finally, we expect and hope the research community would like to join us in this research project.