User Manual

NOTE: The instructions given in this manual work only for linux-used machines, and might not work as expected on other operating systems such as Mac and Windows. the following installation instructions are carried out in accordance with the B-vEPC user manual used for the development of our grade work and can be found here. However, they have been adapted for the execution of our elastic scalability mechanism.

I. Download

1. Download elastic_vEPC repository, which includes **`Datastore`** , **`Entities`** , **`Install`** and **`Load Balancer`** folders. here.

2. in the repository you will find the configuration files for each vEPC module as shown in the table 1.

DOCUMENT/FOLDERS		CONTENT	CONTENT DESCRIPTION
User_Manual.pdf		User manual for installation and execution	This document contains the installation and execution instructions for each of the vEPC modules and entities.
elastic_vEPC	Datastore	Contains subfolders with *.cpp.sh.h* files for the execution of the Data store	Contains subfolders with the configuration files for the installation and execution of the KeyValueStore based data stores..
	Entities	Contains subfolders with *.cpp.sh.h* files for the execution each entity of vEPC	This folder contains the subfolders with configuration and execution files of all the entities and replicas of the vEPC and the RAN module.
	Install	find_bw.sh install.sh	This folder contains the two installation files of the tools necessary for the execution of the different entities and modules of the vEPC. These files are executed in all the virtual machines.
	LoadBalancer	contiene subcarpetas con archivos *.conf*	this folder contains subfolders with examples of load balancer configuration for each cluster of entities in the vEPC architecture scaled horizontally.

Table 1: Content of GitHub repository

II. Setup LTE components

1. our vEPC have a elastic design. It can contain many parallel replicas of LTE components. We describe here a three replica system. More replicas can be setup using the same procedure. Figure 1 shows virtual machine placement and interconnection of LTE components in the distributed setup. The system as per the Figure 1 contains two replicas for MME, SGW and PGW one HSS. Each of the MME replicas is identical to each other and are placed behind a load balancer. SGW and PGW also follow similar setup. A shared data store is also used for state sharing among the replicas. The setup also contains a ran-simulator which generates load for the vEPC. to setup a two replicas system you will be requiring 17 virtual machines.

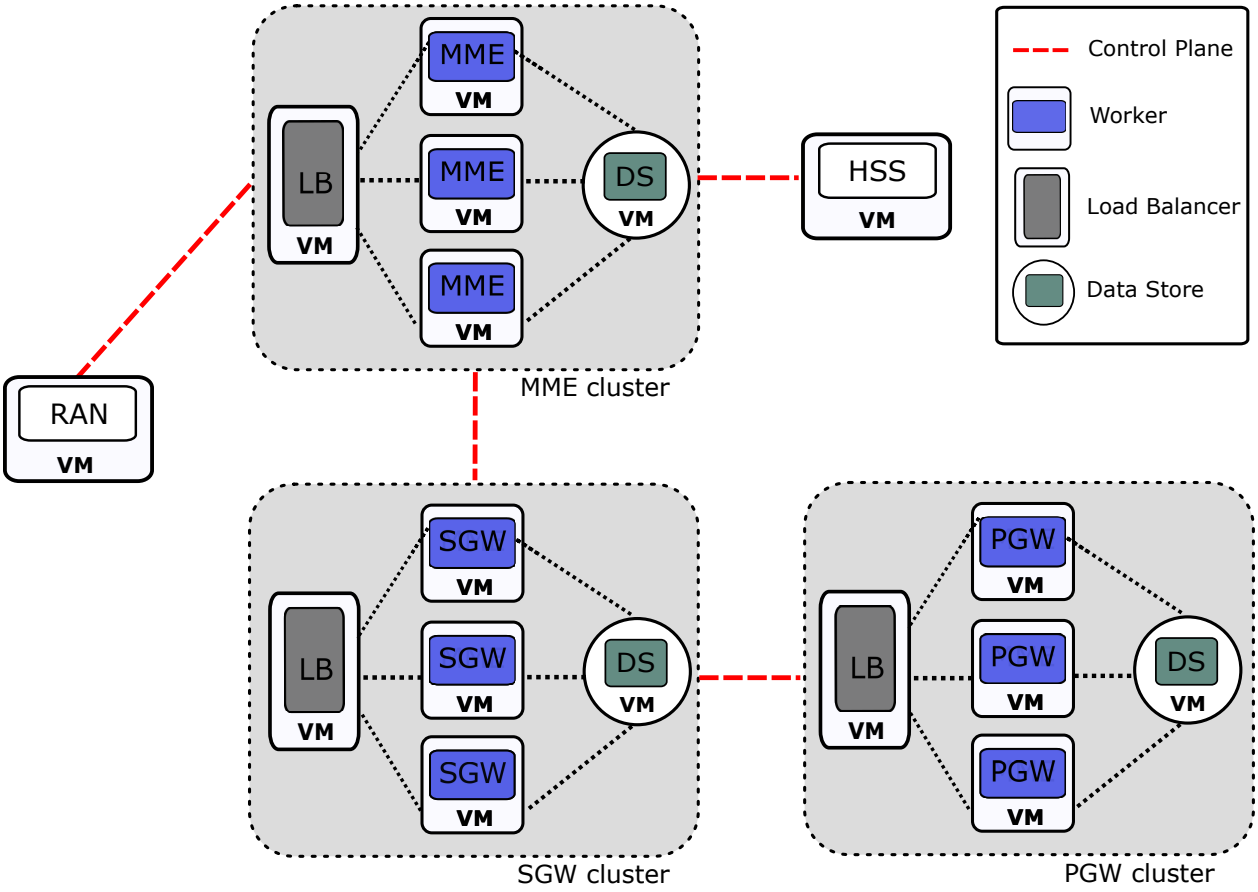

Figure 1: vEPC Setup.

2. The VM details of a standard experiment setup of our vEPC is listed in Table 2. Specification for MME, SGW and PGW are for a single instance. A two replica system having two MMEs, two SGWs and two PGWs will require each of the instances to have the mentioned specification.

COMPONENT	CPU CORES	RAM	DISK	OS
RAN, SINK	4	4 GB	10 GB	Ubuntu 14.04
MME, SGW, PGW (per instance)	1	1 GB	10 GB	Ubuntu 14.04
HSS	1	2 GB	10 GB	Ubuntu 14.04
MME LOAD BALANCERS SGW LOAD BALANCERS PGW LOAD BALANCERS	1	2 GB	10 GB	Ubuntu 14.04
LEVELDB CLUSTER	2	2 GB	15 GB	Ubuntu14.04

Table 2: Individual component specification

Before proceeding, ensure proper communication among all VMs through the use of **`ping`** command. Also,

note down the **`eth0`** IP addresses of VMs for future references.

3. Distribute the source code files into five folders for each MV according to the table 3below. Eact set of **`.cpp/.h`** files correspond to one of the EPC modules. Note that in table 3, each file name (e.g., **`diameter`**) corresponds to both **`.cpp`** and **`.h`** files (**`diamter.cpp`** and **`diameter.h`**). the function of each of the scripts is described here.

MODULE	MME	HSS	SGW	PGW	RAN
diameter	✓	✓	✓	✓	✓
gtp	✓	✓	✓	✓	✓
hss		✓			
hss_server		✓			
mme	✓				
mme_server	✓				
mysql		✓			
network	✓	✓	✓	✓	✓
packet	✓	✓	✓	✓	✓
pgw				✓	
pgw_server				✓	
ran					✓
ran_simulator					✓
s1ap	✓	✓	✓	✓	✓
sctp_client	✓				✓
sctp_server	✓	✓			
security	✓				✓
sgw			✓		
sgw_server			✓		
sync	✓	✓	✓	✓	✓
telecom	✓				✓
tun					✓
udp_client	✓		✓	✓	✓
udp_server			✓	✓	✓
utils	✓	✓	✓	✓	✓

Table 3: Setup: Source code distribution.

4. Once source code files are segregated for each module (MME, HSS, SGW, PGW, RAN), place each set of files in its corresponding VM. Perform the following steps for each VM.

5. Copy the **`install.sh`** file from **`Install`** folder and run it to have all the required packages/tools in all the VMs that will be used for EPC operations. Install any additional tools that might be required (e.g., **`gedit`**, **`emacs`**)

   ```
   $ bash install.sh
   ```

6. Open the utilsh file for the module.

7. Set RAN as the current IP of the ran simulator system.

8. Set MME1 to the ip address of the MME entity.

9. Set SGW1 to the ip address of the SGW entity.

10. Set PGW1 to the ip address of the PGW entity.

11. Set MME to the ip address of the MME load balancer.

12. Set SGW to the ip address of the SGW load balancer.

13. Set PGW to the ip address of the PGW load balancer.

- **Set up RAN Simulator**

  Note: Perform the following steps with the ran simulator code module only

  (a) compile using the command

  ```
  $ make ransim.out
  ```

  (b) run using the command for control plane operation:

  ```
  ./ransim.out <num of parallel UEs> <Duration of experiments>
  ```

  the maximum number of concurrent users is 200 and there is no time limit.

- **Set up MME**

  Note: Perform the following steps with the MME code module only and for each of the parallel MME replicas.

  (a) Open the utilsh file for the module.

(b) set MME1 to the ip address of the current MME system. In case of multiple replicas for the MME cluster, please specify the next replicas IP address (assuming 2 more parallel replicas) as MME2 and MME3.

(c) Set INIT_VAL TO 1000000 / 4000000 / 7000000 for different replicas to maintain a disjoint range for UE tunnel id assignment.

(d) Set REP to 1 to enable synchronization with sibling replicas.

(e) compile using the command

```
$ make mme.out
```

(f) run using the command for control plane operation:

```
./mme.out <num of worker threads for control plane>
```

- **Set up SGW**

Note: Perform the following steps with the SGW code module only and for each of the parallel SGW replicas.

(a) Open the utilsḣ file for the module.

(b) set SGW to the ip address of the current MME system. In case of multiple replicas for the SGW cluster, please specify the next replicas IP address (assuming 2 more parallel replicas) as SGW2 and SGW3.

(c) Set REP to 1 to enable synchronization with sibling replicas.

(d) compile using the command

```
$ make sgw.out
```

(e) run using the command for control plane operation:

```
./sgw.out <threadCount1> <threadCount2> <threadCount3>
```

- **Set up PGW**

Note: Perform the following steps with the PGW code module only and for each of the parallel PGW replicas.

(a) Open the utilsḣ file for the module.

(b) set PGW1 to the ip address of the current PGW system. In case of multiple replicas for the PGW cluster, please specify the next replicas IP address (assuming 2 more parallel replicas) as PGW2 and PGW3.

(c) Set REP to 1 to enable synchronization with sibling replicas.

(d) compile using the command

```
$ make pgw.out
```

(e) run using the command for control plane operation:

```
./pgw.out <threadCount1> <threadCount2>
```

- **Set up HSS**

Note: Perform the following steps with the HSS code module only.

(a) Open the utilsḣ file for the module.

(b) set HSS to the ip address of the current HSS system.

(c) compile using the command

```
$ make hss.out
```

(d) run using the command for control plane operation:

```
./hss.out <servingthreadCount>
```

(e) Setting up HSS state store

i. Set up a levelDB based key value store **`hss`** in the VM assigned for HSS. A levelDB server client implementation is included as part of the release in directory datastore. Any other datastore also can be used as long as it follows identical setup. We describe below the procedure to setup a levelDB datastore.

ii. Copy the setup_ds directory from the **Datastore** directory.

iii. Place the directory in the virtual machine for HSS.

iv. cd to setup_ds and execute

```
$ bash install_server.sh
$ bash run_server.sh <ip of the vm>:8090
```

v. HSS has a details of authenticated users who can perform attach. To configure data, open the **setup_hss** directory from **HSS** folder replace the target ip address of levelDB in **fill_hss.cpp** and run the following steps to load data.

```
$ run load_data.sh
```

The command should end with a success message if everything worked correctly.

Setup Datastore for MME/ SGW/ PGW

Refer to instruction provided at LINK. here.

Setup load-balancers

The system has been well tested with LVS-DR based load balancer. However, you are free to choose any other layer-4 load-balancer as long as you follow similar configuration. The following section contains configuration of an LVS-DR based load balancer.

Setup three virtual machines for MME/SGW/PGW load balancers as per the specification provided in Table 2.

1. Follow the following steps to setup each of the load-balancers:
2. Edit the ipvsadm configuration as shown below

```
$ vi /etc/default/ipvsadm
# change
AUTO="true"
# change
DAEMON="master"
# change to the interface to use
IFACE="eth0"
```

3. Create a new interface with a virtual ip that is different from the ip address of the load balancers. A different IP can serve the purpose of a virtual director to handle a master slave load balancer configuration in case of failure of the load balancer.

```
vi /etc/network/interfaces
# add to the end
auto eth0:0
iface eth0:0 inet static
address <VIP> (VIP that is exposed to clients)
network 10.0.0.0  (your network address)
netmask <subnetmask> (your subnet mask)
root@user:~# ifup eth0:0
```

4. With this procedure the laod balancer network setup is complete. LVS load balancing can be referred from ipvsadm man pages. Load balancing configuration for a certain setup with three replicas is given in the script

folder. Next section shows one of the LVS load balancing configuration as an illustration. [LVS configuration of SGW]

```
ipvsadm -A -u <SGW_VIP>:7000 -s rr
ipvsadm -a -u <SGW_VIP>:7000 -r <Replica 1 IP>:7000 -g -w 1
ipvsadm -a -u <SGW_VIP>:7000 -r <Replica 2 IP>:7000 -g -w 1
ipvsadm -a -u <SGW_VIP>:7000 -r <Replica 3 IP>:7000 -g -w 1

ipvsadm -A -u <SGW_VIP>:7100 -s rr
ipvsadm -a -u <SGW_VIP>:7100 -r <Replica 1 IP>:7100 -g -w 1
ipvsadm -a -u <SGW_VIP>:7100 -r <Replica 2 IP>:7100 -g -w 1
ipvsadm -a -u <SGW_VIP>:7100 -r <Replica 3 IP>:7100 -g -w 1

ipvsadm -A -u <SGW_VIP>:7200 -s rr
ipvsadm -a -u <SGW_VIP>:7200 -r <Replica 1 IP>:7200 -g -w 1
ipvsadm -a -u <SGW_VIP>:7200 -r <Replica 2 IP>:7200 -g -w 1
ipvsadm -a -u <SGW_VIP>:7200 -r <Replica 3 IP>:7200 -g -w 1
```

5. The load balancer configuration also requires an iptables entry at the MME/SGW/PGW replicas.

   ```
   iptables -t nat -A PREROUTING -d <SGW_VIP> -j REDIRECT
   ```

   More information on LVS setup can be found here.

The vEPC setup would be now complete and you can proceed to experimenting with our vEPC using different types of traffic. A sketch of the implemented setup is given in Figure 1. Communication among modules will take place according to the blue/red lines given in Figure 1.

Generating Control traffic

For experiments with control traffic, we simulate a number of concurrent UEs in the RAN simulator and make the UEs continuously perform attach and detach procedures with the EPC, to create a continuous stream of control traffic. We increase or decrease load on the vEPC by varying the number of concurrent UE threads loading the vEPC. A detailed description of the procedure is given below.

1. Once the setup is ready, run each vEPC binary executable in its own VM. Usage format of each executable is given in table 4 below.

Table 4: Usage format of binary executables.

MODULE	USAGE
MME	**./mme.out <#S1-MME threads>**
HSS	**./hss.out <#S6a threads>**
SGW	**./sgw.out <#S11 threads> <#S1 threads> <#S5 threads>**
PGW	**./pgw.out <#S5 threads> <#SGi threads>**

A sample run of vEPC modules is given below.

MME:

```
$ ./mme.out 50 (to be done at each mme instance)\\
```

HSS:

```
$ ./hss.out 50
```

SGW:

```
$ ./sgw.out 50 50 50 (to be done at each instance)
```

(to be done at each sgw instance)

PGW:

```
$ ./pgw.out 50 50 (to be done at each instance)
```

By doing this, you are simulating **RAN** objects that generate only control traffic.

2. **RAN:** Rerun the **makefile** for the RAN module to make the binary executable for generating only control traffic.

   ```
   $ make ransim.out
   ```

3. **RAN:** Start RAN simulator with appropriate parameters as given below. This will generate the required amount of control traffic for the given time duration.

   ./ransim.out <#RAN threads> <Time duration>

   A sample run is as follows.

   ```
   $ ./ransim 10 100
   ```

Performance results

Control traffic

Two performance metrics would be reported at the end of control traffic experimentation. These parameters are given below. No additional scripts/coding would be required to produce these parameters as they are computed along with the generation of control traffic.

1. **Throughput**, the number of registration requests successfully completed by the EPC per second
2. **Latency**, the time taken by a registration to complete