vEPC 1.1 User Manual

NOTE: The instructions given in this manual work only for linux-used machines, and might not work as expected on other OSes such as Mac and Windows. Please use online references in such cases. For details on understanding the code and various procedures involved in vEPC, please look at the `developer_manual.pdf` under `doc` folder.

Installation

1. Download vEPC 1.1 repository, which includes `doc`, `scripts` and `src` folders.
2. Navigate to `scripts` folder and run `install.sh` file.

   ```
   $ bash install.sh
   ```

   This will install all the software modules/tools required for proper compilation and working of vEPC.
3. Navigate to `src` folder and run `makefile` to obtain binary executables for the individual software modules: MME, HSS, SGW, PGW, RAN, SINK.

   ```
   $ make
   ```

4. The following binary executable files would have been generated: `mme.out`, `hss.out`, `sgw.out`, `pgw.out`, `ransim.out`, `sink.out`. With these executables, you can now proceed to setup our vEPC, where individul modules will be hosted on separate Virtual Machines (VMs).

Note: If there were any installation errors thrown by the compiler while running makefile, please install the corresponding dependencies. There should not be any other errors in this set of steps. Please note that the above procedure is provided only to understand the installation instructions, and it can be skipped to move directly to the Setup section.

Setup

1. You will be requiring six VMs to setup our vEPC. Assign one VM for each software module (MME, HSS, SGW, PGW, RAN, SINK). 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.
2. Distribute the source code files into six sets according to the table 1 below. Eact set of `.cpp/.h` files correspond to one of the EPC modules. Note that in table 1, each file name (e.g., `diameter`) corresponds to both `.cpp` and `.h` files (`diamter.cpp` and `diameter.h`).

Table 1: Setup: Source code distribution.

MODULE	**MME**	**HSS**	**SGW**	**PGW**	**RAN**	**SINK**
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			✓			
sink						✓
sink_server						✓
sync	✓	✓	✓	✓	✓	
telecom	✓				✓	
tun					✓	✓
udp_client	✓		✓	✓	✓	✓
udp_server			✓	✓	✓	✓
utils	✓	✓	✓	✓	✓	✓

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

   (a) Open the main **`.cpp`** file for the module (e.g., **`mme.cpp`** for MME VM) and add the required IP addresses at the top of the file. For e.g., the set of IP addresses information required for SGW module is given below.

```
string g_sgw_s11_ip_addr = "192.168.1.74";
string g_sgw_s1_ip_addr = "192.168.1.74";
string g_sgw_s5_ip_addr = "192.168.1.74";
```

(b) Copy the **`install.h`** file from **`scripts`** folder and run it to have all the required packages/tools in the VM. Install any additional tools that might be required (e.g., **`gedit, emacs`**)

```
$ bash install.h
```

(c) Copy the **`makefile`** file from **`scripts`** folder and run it for the corresponding module to have the updated binary executable.

Sample run:

```
$ make mme.out
```

4. **HSS:** Create MySQL database **`hss`** in the VM assigned for HSS.

```
$ CREATE DATABASE hss
```

5. **HSS:** Copy the **`hss.sql`** file from **`scripts`** folder and run it to upload the necessary tables in MySQL database.

```
$ mysql -p -u root hss < hss.sql
```

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 dotted lines given in figure 1.

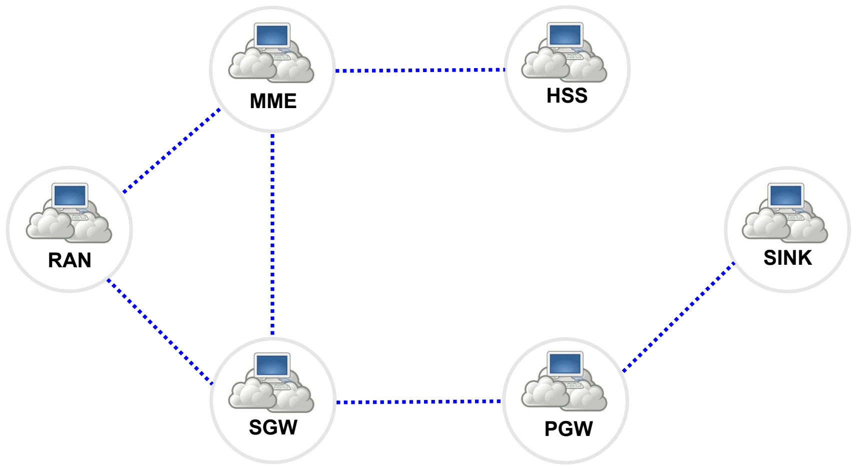

Figure 1: vEPC Setup.

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 EPC by varying the number of concurrent UE threads loading the EPC. 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 2 below. Please look at the **`developer_manual.pdf`** under **`doc`** folder to understand more about the command line parameters used with each executable.

   Table 2: 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
   ```

   HSS:

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

SGW:

```
$ ./sgw.out 50 50 50
```

PGW:

```
$ ./pgw 50 50
```

2. **RAN:** Open `ran_simulator.cpp` and comment out the following code sections.

   - `data_transfer` section under the `simulate` function.
   - `tun` and `traffic_monitor` sections under the `run` function.

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

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

   ```
   $ make ransim.out
   ```

4. **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
   ```

Generating Data traffic

For experiments with data traffic, we attach a specified number of UEs to the EPC from the RAN simulator, and pump traffic from the RAN to the sink. A traffic generating tool called `iperf3` is employed, using which TCP data can be sent with a given bandwidth and time duration. To generate traffic, `iperf3` process is started in Server mode at the Sink module and the corresponding `iperf3` Client process is started at the RAN simulator with the required input data rate and time duration parameters. A detailed description of the procedure is given below.

1. Begin vEPC modules (MME, HSS, SGW, PGW) as explained before.

2. **SINK:** Start the Sink module with the required number of `iperf3` servers.

`./sink.out <#iperf3 servers>`

A sample run is as follows.

```
$ ./sink 10
```

3. **RAN:** Open **`ran_simulator.cpp`** and uncomment the following code sections: **`data_transfer, tun, traffic_monitor`**.

4. **RAN:** Open **`ran.cpp`** and set the duration of **`iperf3`** data transfer - **`dur`** under the **`transfer_data`** function. Note that this duration has to be smaller than the overall duration given as command line parameter while running RAN simulator. Input data rate is currently fixed at 1 Mbps. It can be modified as per requirements.

5. **RAN:** Rerun the **`makefile`** for the RAN module to reflect the updates in the binary executable.

   ```
   $ make ransim.out
   ```

6. **RAN:** Now begin the RAN simulator as explained before. This will simulate the required number of **`RAN`** objects, generating both control traffic and data traffic in sequence, for the given time duration. Note that the number of **`RAN`** objects created has to be equal or greater than the number of **`iperf3`** servers started at Sink module.

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

Data traffic

While experimenting with data traffic, the following steps need to be followed to obtain performance results.

1. **SINK:** Copy the **`find_bw.sh`** file from **`scripts`** folder. Just before starting the RAN simulator, run this script at the Sink module. This will measure the required throughput value. A file named **`bw.txt`** will be

created, which contains the various throughput values (in bytes/second): Uplink, Downlink, Total (Uplink + Downlink).

Format:

```
bash find_bw.sh -i <interface> -s <duration> -c <count> -n <#UEs>
```

Sample run:

```
$ bash find_bw.sh -i eth0 -s 100 -c 1 -n 1
```

2. **RAN:** Before starting the RAN simulator, open **`ran_simulator.cpp`** and uncomment the **`RTT`** code section under the **`simulate`** function. This will create a **`ping.txt`** file, which contains the Round Trip Time (RTT) latency values for the data transfer in milliseconds. Note that the time duration under **`ping`** function need to be set according to the time duration of the user data transfer.

With the procedure explained above, two performance metrics would be obtained at the end of data traffic experimentation. These parameters are given below.

1. **Throughput**, the amount of data traffic successfully forwarded by the EPC gateways to the Sink per second

2. **RTT Latency**, the amount of processing overhead added by the EPC gateways in the data plane, as measured by a ping command during load test