LTE PROJECT REPORT

5GControlCoreLite: Scalable & Cost-Effective Packet Core Architecture for Heterogeneous M2M devices in 5G

LTE Stateless Packet Core Architecture

Vasudevan Nagendra

WINGS LAB
STONY BROOK UNIVERSITY
Department of Computer Science

Contents

Chapter 1

Introduction

1.1 What is this project and setup all about?

This project aims to create an LTE EPC-core testbed built in a virtualized environment where all components are light-weight microservice-based NFs (Network Functions). This project was created with the long term goal in mind to be able to customize (add/remove) functionalities, scale rapidly to accommodate the dynamic traffic needs.

Chapter 2

Setup Details

2.1 Overall setup

- For basic setup, below is the diagram. The Connections between UE and 30; 30 and 70 is used.

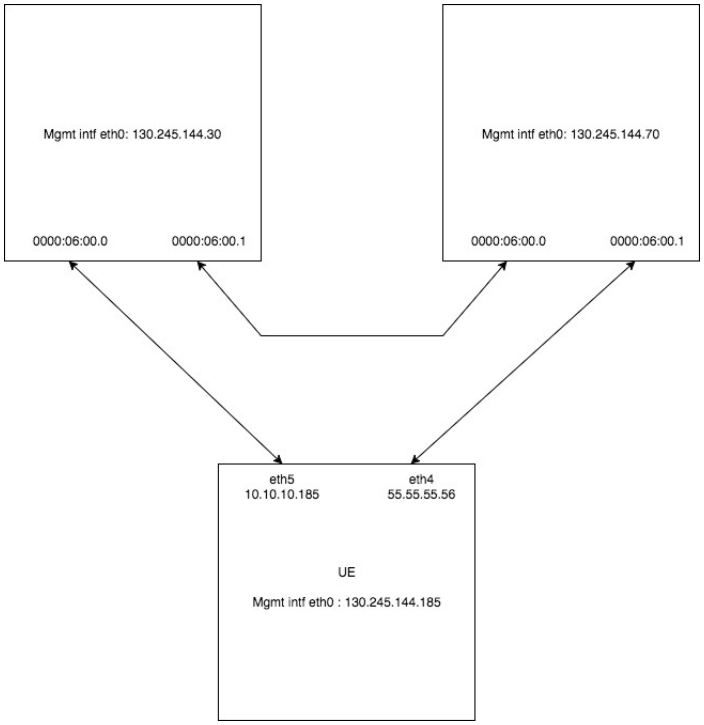

Figure 2.1: Basic setup details

2.2 Hard-coded mappings in the setup

- In MME/HSS/SPGW, all frame_packet∗ functions have the following hard-codings
  - MAC addresses
  - checksum
  - destination IP address
- In UE following hardcodings are present
  - In ue.h, OTHER_IP, US_IP and iplist, need to be set to appropriate values
  - In ue.c, all packets which are constructed are hardcoded. Whenever a certain offset in the packet is being modified, that is known beforehand that it is the exact byte from where the modification needs to be made, for eg. the IMSI goes in from the 24th byte onwards. When in doubt, refer to pcap captures of packets in openair setup

Chapter 3

Using Infrastructure

3.1 Setup Environment Variables

set environment variables in the .bashrc file
export PYTHONPATH=/usr/local/lib/python:$PYTHONPATHexportPATH =$
/bin :PATH export RTE_SDK=/home/mallesh/openNetVM/dpdk export KVS_SDK=/home/mallesh/ope
export RTE_TARGET=x86_64-native-linuxapp-gcc export ONVM_NUM_HUGEPAGES=15240
export ONVM_HOME=/home/mallesh/openNetVM

3.2 How to run complete setup

1. setup 130.245.144.30
   (a) git clone https://github.com/vasu018/openNetVM
   (b) git checkout experimental/shared_cpu
   (c) under openNetVM directory:
   git clone https://github.com/vasu018/dpdk
   (d) inside dpdk run:
      i. make config T=x86_64-native-linuxapp-gcc
      ii. make install T=x86_64-native-linuxapp-gcc
   (e) under openNetVM directory: git clone https://github.com/vasu018/kvs
   (f) inside kvs run:
   make
   (g) inside dpdk/usertools run:
      i. ./dpdk-devbind.py –status
      ii. if 0000:06:00.0 and 0000:06:00.1 are not bounded to dpdk drivers, then run:
      sudo ./dpdk-devbind.py -b igb_uio 0000:06:00.0

(h) go to openNetVM/onvm and run:
make

(i) go to openNetVM and run:

i. "ps aux | grep -ie onvm | awk {'print $2'} | xargs sudo kill -9" to ensure that onvm related processes aren't running

ii. ./onvm/go.sh 0,1,2,3 3 -s stdout

(j) run MME as per guidelines given later in the doc.

(k) run forwarder as per guidelines given later in the doc.

2. setup 130.245.144.70

(a) git clone from https://github.com/vasu018/openNetVM

(b) under openNetVM git clone from https://github.com/vasu018/dpdk

(c) inside dpdk run

i. make config T=x86_64-native-linuxapp-gcc

ii. make install T=x86_64-native-linuxapp-gcc

(d) under openNetVM git clone from https://github.com/vasu018/kvs

(e) inside kvs run make

(f) inside dpdk/usertools run

i. ./dpdk-devbind.py –status or ./dpdk-devbind.py -s

ii. if 0000:06:00.0 and 0000:06:00.1 are not bounded to dpdk drivers, then run sudo ./dpdk-devbind.py -b igb_uio 0000:06:00.0

(g) go to openNetVM/onvm and run make

(h) go to openNetVM and run

i. "ps aux | grep -ie onvm | awk 'print $2' | xargs sudo kill -9" to ensure that onvm related processes aren't running

ii. ./onvm/go.sh 0,1,2,3 3 -s stdout

(i) run HSS as per guidelines given later in the doc.

(j) run SGW as per guidelines given later in the doc.

(k) run forwarder as per guidelines given later in the doc.

3. run UE from 130.245.144.180 after cloning from https://github.com/vasu018/UE_SIM

3.3 How to run UE

3.3.1 compile

run make.
This makes *ue* and *ue_no_write*.
ue_no_write is just like *ue* but it doesn't write anything in the log files.

3.3.2 running steps

Usage : *./ue < pkt_type >< num_par >< num_ser >< Thread IMSI seed >*

Usage : *./ue < pkt_type >< num_par >< num_ser >< Thread IMSI seed >< burst interval >*

Usage : *./ue < pkt_type >< num_par >< num_ser >< Thread IMSI seed >< burst interval >< max_retry_co*

- **pkt_type**
  0 ATTACH
  1 SERVICE
  2 DETACH
  3 ATTACH+DETACH (TAU)
- **num_par, num_ser** = The number of parallel x serial connections required.
- **Thread IMSI seed** = the seed value to offset parallel value. For eg. if you run ./ue 0 10 10 1 ; then the next set of distinct connections should be ./ue 0 10 10 11.
- **burst_interval** = the amount of time in usec between each parallel burst. This helps emulate serial connections. So for a 10x3 run with burst_interval 5, there will be 3 bursts of 10, with 5 usec gap in between each burst. Default(1)
- **max_retry_count** = Max allowed number of retries for each connection. Default(10)
- **timeout_between_retries** = Timeout in ms after which a retry is sent. Default(100)

3.3.3 Output

3.3.3.1 console output

Sample output → ./ue 0 100 3 1
********** System Global Stats **********
ATTACH_ATTEMPT = 300
ATTACH_AUTH_REQ_RECV = 300
ATTACH_AUTH_RESP_SENT = 300
ATTACH_NAS_REQ_RECV = 300
ATTACH_NAS_RESP_SENT = 300
ATTACH_ACCEPT_RECV = 300
ATTACH_ACCEPT_COMPLETE_SENT = 300

********** Exit counters **********
Request Successful Fail
Attach 300 300 0
Service 0 0 0
Detach 0 0 0

Retries: 8

********** Time Global Stats **********
Thread ID 1 Serial ID 0 ran for 0.962000 ms
Total Time from start to end :111.348000 ms

Mean Stats
Attach Total = 3.068193
Attach_initiate->auth_recv = 0.167660
Auth_recv->Nas_recv = 0.105453
Nas_recv->accept = 2.795080

Min attach time = 0.265000
Max attach time = 100.361000
Median:
Time: 0.342000

- **System Global Stats** = These stats will give you how many packets were sent or received for each step.
- **Exit counters** = Total successful and failures for the mode used. Number of retries show the number of connections for which at least one retry was done.
- **Time Global Stats** =
  - First line shows the time taken by the first connection which got successful
  - Total time shows the total time for the experiment from the beginning of sending first packet to receiving the last packet of the last successful connection.
  - Min and Max times show the time taken by the connections which took the max and min time to complete.

– Median show the median completion time of all connections. Sometimes when mean is not reliable due to huge spikes, this is more useful.

3.3.3.2 Log output in file summarised

attach_test_values.csv and service_test_values.csv contain all stats taken from output console in csv order. Total 11 entries in each row.

1. Total Experiment time (msec)
2. Attach 1st Successful Latency
3. Avg Latency
4. Attempted
5. Successful
6. Fail
7. Attach_initiate->auth_recv
8. Auth_recv->Nas_recv
9. Nas_recv->accept
10. Max Latency
11. Min Latency

3.3.3.3 Log output in file for each connection

Logs are generated under logs/ which contain output of time taken for each connection. They are in csv format with each row containing output for one burst. The name of the log files are like attach_100_3.txt for attach of 100x3. There will be 3 rows in the file.

3.4 How to run MME

3.4.1 compile

run make under examples/dummydmme

3.4.2 run

./go.sh 5 4 1 -k mme -t 1
In above command the following means

- 5 means run on core number 5
- 4 means this NF's service ID is 4
- 1 means this NF's default destination is service ID 1
- *k* option is to assign a name for the NF just to identify in onvm console
- *t* option is to assign additional delay for Attach requests. Default(0)

3.5 How to run LB

3.6 How to run FW

make under examples/forwarder_slice
./go.sh 4 1 2

3.7 How to run SGW

make under examples/dummysgw
./go.sh 6 3 1 -k sgw

3.8 How to run HSS

make under examples/dummyhss
./go.sh 5 2 1 -k hss

Chapter 4

Kalyan Experiments List

4.1 Experiment: Statelessness NF Performance

4.2 Experiment: Decomposition

4.3 Experiment: Static Bindings Impact

4.4 Experiment: Fault Tolerance Experiments

Chapter 5

Amos Experiments List

5.1 Experiment: LB/Forwarder NF Performance

Integrating the LB logic to our MME Infrastructure and experimenting with LB logic. For this we need to test it with these 3 different approaches i.e., RR, CH and Our-CH. You need to send traffic with all these three cases and plot CDF. Along with that you might need to plot the violations i.e., taking 100 msec or some time as threshold we need to how much percent of traffic is violating their latency requirements. [Coding is completed.]

5.1.1 Code for LB

Code for load balancer [1]:
https://github.com/vasu018/openNetVM/blob/queue_status/examples/loadbalancer_chash_modifie

Code for sending status information to the load balancer [2]:
https://github.com/vasu018/openNetVM/tree/queue_status/onvm
[Changes are in multiple files. You can get the complete set of changes using the history at git]
All code changes are present in the queue_status branch.

5.1.2 Running the modified onvm manager

1) Added a new lcore for sending the sync messages. ./onvm/go.sh 0,1,2,3,4 3 -s stdout

2) Make sure when you compile ONVM manager, specify the address of the load balancer in nf_queue_status_main @ [3]
https://github.com/vasu018/openNetVM/blob/queue_status/onvm/onvm_mgr/main.c

Running the load balancer: 1) I am using 1 more lcore to receive and process the sync messages from the ONVM manager of the host.
./go.sh 5,6 1 2

5.2 Experiment: Load Balancer Resource and Performance

5.3 Experiment: State Migration Performance

Plotting the effect of service requests on TAU and state migration:
Setup:
Normal setup. Make sure that ue.c is compiled with file logging in append mode as multiple runs of need to be compiled to the same log.

Script running:
@.185 you can run the following script from the UE_SIM.
usage: ./final_tau.sh <ser_load> <tau_load> <tau_batches>
The latency logs will be present in
logs/amos/tau_agg_<tau_load>_<tau_batches>_<load type>_mme.txt
Each running iteration of ue is captured in
logs/amos/tau_agg_<tau_load>_<tau_batches>_<load type>.txt

Plotting the effect of State migration and TAU on Attach and service requests

Setup for finding out performance of State migration:

Normal setup in both .30 and .70. Apart from that run examples/ue_state_server2 on both .30 and .70.

Script running:
@.185 you can run the following script from the UE_SIM.
usage: ./final_tau_reverse_migration.sh <src/dst> <migration_size> <run_number>
This will capture the Effect of state migration on Attach and service requests. The logs will be stored in logs/tau_reverse_logs/tau_migration_logs
Example logname: attach_10000_1_dst.txt attach_10000_1_src.txt
Setup for finding out performance of TAU:
Normal setup in both .30 and .70.
Script running:
@.185 you can run the following script from the UE_SIM.
Before running edit the script to change the TAU size.
usage: final_tau_reverse_tau.sh <run_number>

This will capture the Effect of state migration on Attach and service requests.
The logs will be stored in logs/tau_reverse_logs/tau_reverse_logs

5.4 Experiment: Nice Value NF Performance

Experiment with Nice values:

(a). I wanted you to try out different nice values with MMEs (4) in the same lcore and send traffic equally to all NFs and plot their CDFs.

Some details:

Nice values can be set using the command renice.
In this experiment we are starting 4 instances of MME with different service id's and the same lcore value.
After this using renice command we can set the nice values of these processes.
You can run htop and filter the processes to find the pid.
I have used nice values of -5,0,5,5 for the 4 MMEs.

Setup:

I have modified the forwarder to route the requests based on the slice ID to different MME's.
Enable the flag LOG_SLICE so that logs for each slice id is segregated.

Running the experiment:

Run the script test_nice.sh and collect the logs in logs/nice_testing/varied/

Plots present in the following folder:

https://drive.google.com/open?id=1Ihup05hi3ForlclGmUPOJYnz9XulIy48

(b). Now swap or changes the values of nice values assigned to each of the MME NFs and plot their CDFs. Now you should see the difference in their performance.

Chapter 6

Quick Run Steps

6.1 Steps to create and build repos

cd /home/mallesh/
git clone https://github.com/vasu018/openNetVM.git
cd openNetVM git checkout experimental/shared_cpu
git clone https://github.com/vasu018/kvs.git
git clone https://github.com/vasu018/dpdk.git
Set these environment variables (in .bash_profile or .bashrc).
export RTE_SDK=/home/mallesh/openNetVM/dpdk
export KVS_SDK=/home/mallesh/openNetVM/kvs
export RTE_TARGET=x86_64-native-linuxapp-gcc
export ONVM_NUM_HUGEPAGES=15240
export ONVM_HOME=/home/mallesh/openNetVM

6.2 Build things in the following order

DPDK:
cd dpdk
make config T=x86_64-native-linuxapp-gcc
make install T=x86_64-native-linuxapp-gcc

KVS:
cd kvs
make

OPENNETVM:
cd onvm
make

6.2.1 Initial one-time steps

Set arp on 185 for destination 30. Make sure that the interface on 185 is up
Make sure that the interfaces on 70/30 are binded to dpdk driver and not the kernel driver.

6.2.2 Final Steps to run setup:

For ATTACH Procedure:

On Host: 130.245.144.30

Kill all onvm processes:
1. ps aux | grep -ie onvm | awk '{print $2}' | xargs sudo kill -9

Start onvm:
cd /home/mallesh/openNetVM>
2. ./onvm/go.sh 0,1,2,3 3 -s stdout

Start Forwarder:
3. examples/forwarder_slice> ./go.sh 4 1 2

Start mme:
4. examples/dummysmme> ./go.sh 5 2 1 -k mme

After experiment:
5. Take perf output
6. Start MME again since it must have died. Sometimes entire onvm needs to be started again since we run out of onvm IDs.

On Host: 130.245.144.70

1. Kill all onvm process
ps aux | grep -ie onvm | awk '{print $2}' | xargs sudo kill -9

2. Start onvm
cd /home/mallesh/openNetVM>
./onvm/go.sh 0,1,2,3 3 -s stdout

3. Start Forwarder
examples/forwarder_hss_sgw> ./go.sh 4 1 2

4. Start hss
examples/dummyhss> ./go.sh 6 2 1 -k hss

5. Start SPGW
examples/dummysgw> ./go.sh 5 3 1 -k sgw

On host: 130.245.144.185
wingslabserver1:/home/mallesh/UE_SIM>
./ue_nonblock_serial 0 2 2 1 1

For SERVICE Request:

Only changes required to run service requests are that a separate MME should be run on service ID 3. Apart from that, everything is the same. the first option in ue program will be 1 of course. An obvious thing is that service request will only work for range of IMSIs which have already had attach successful.

Chapter 7

Code Documentation

7.1 OpenNetVM

Code for NF Manager and Flow director is present here.
/home/amos/workspace/openNetVM/onvm/onvm_nflib/onvm_flow_dir.c
You could take a look at these examples to understand it better
/home/amos/workspace/openNetVM/examples/flow_table /home/amos/workspace/openNetVM/

References

[1] Forwarder Code. Nov 2017. https://github.com/vasu018/openNetVM/blob/queue_status/examples/loadbalancer_chash_modified/forward.c.

[2] Load Balancer Code for Queue Status:. Nov 2017. https://github.com/vasu018/openNetVM/tree/queue_status/onvm.

[3] ONVM LB:. Nov 2017. https://github.com/vasu018/openNetVM/blob/queue_status/onvm/onvm_mgr/main.c.