Implementation Details and design of DPDK based RAN-emulator

Prateek Agarwal

May 2021

Contents

1 Introduction

This document explains the implementation details of DPDK based RAN-emulator. The main purpose of this report is to ensure that any future developer can understand RAN code and should be able to further modify or extend it as per requirement.

This network function acts as the load generator for the testing of user plane forwarding function. The load needs to be generated in various modes for the testing of UPF forwarding capabilities. Section **??** suggests some of the tools that should be learnt to handle large code base easily and be productive while doing so.

2 Tools

Learn the tools mentioned in this document. Some of them are necessarily required to move through a large code base such as the 5G test bed project. Others are strongly advised to increase your productivity and save you a whole lot of time.

2.1 Visual Studio Code

Please use Visual Studio code to read/write code. VScode is an excellent and a basic IDE for writing C/C++ code. Most of us use VS Code for our development work. The use of VS code will help in better communication with the team members. Here are some of the important features that we routinely use:

- **Remote SSH** You can login remotely to the server from the editor itself. This implies that you do not need GUI applications like team-viewer etc. to use an editor or login on a graphical interface. These applications are heavy and affects your overall productivity and development experience.

- **Refactoring, Formatting etc. Shortcuts** See what F2, F12, Shift + F12, ctrl + shift + I are used for. You will find them very useful.

- **Terminal Support** Terminal support is available inside the editor. I have frequently used for building the executable after making some changes and while working on further changes simultaneously. Bash commands like grep etc. can also be used from inside.

2.2 Vim

This is not necessarily required but strongly advised. If you learn vim (which has initial steeper learning curve), you will thank me later. This point may look contradictory to the first point. But it is not. You **must** use Vim key bindings in Vscode. Resources:

- A nice place to get started is `https://www.youtube.com/watch?v=a6Q8Na575qc&t=2s` .

- Practice multiple times on terminal with the help of **vimtutor**.

2.3 SSH keys

Install SSH key on the remote server as soon as possible for smoother login into the remote machine. Keys if installed do not require you to type password again and again and will save you from a lot of unnecessary repeated work. When the key is installed on the remote machine, the server identifies your machine. Use aliases or scripts to login through SSH.

2.4 Git

It is required. Learn it properly from experts, review it multiple times. Learn the internal working of Git rather than copying commands from stack overflow without understanding what they mean. There is no other alternative. Not knowing Git well can create a lot of stress for you and nuisance for others. Do not push commits on other person's branch (I have done it). Keep updating the gitginore file so as not to push binary executables or other files not meant to be pushed on the remote repo (I have also done this).

- **Important Commands**: Add, reset, commit, branch, merge (three-way and ff), difference between pull and fetch, git log, stash. Vscode is good for handling merge conflicts as well. Learn how to write gitginore.

- **Book** `https://git-scm.com/book/en/v2` Chapter 1,2,3 are compulsory reading. Read other chapters if you are interested or feel the need. Read them frequently.

- **Video** There are many good videos on the web with hardly any difference in the quality. Some of them are so basic that you lose interest midway. `https://www.youtube.com/watch?v=2sjqTHE0zok&t=3s` is one of the videos that suits the M.Tech second year students at IIT Bombay.

2.5 Terminal Multiplexers

Learn how to use tmux or screen for command line work. Network functions are compiled and executed through the command line only. I use tmux. It is also good for pair programming if you are working with someone remotely. tmuxinator can be used to make config files for saving the session configurations. The .tmux.conf file on turing 4, turing 5 has all the configurations one need. Kindly understand these config files. See the tmuxinator configurations in ~/.config/tmuxinator on turing 5 to learn how to write tmuxinator configurations. Resources:

- The configuration files are taken from `https://www.amazon.in/tmux-2-Brian-P-Hogan/dp/1680502212/ref=sr_1_1?dchild=1&keywords=tmux&qid=1619933598&sr=8-1`

- See the missing semester video on tmux.

2.6 Important bash utilities/commands

- **grep** Know the flags. Useful flags : -i, -n, -r, -l , -A, -B , -C .

- **cut** This command is very useful for dissecting the log files for the cleaning of data. Flags: -d, -f

- Other useful commands Ctrl-R, less, cat, sudo !! etc.

- Please do **NOT** use sudo for everything. It is a poor practice on a multi user system and is against the principle of least privilege. Many students do this and keep doing this for the entire year. It messes up file permissions besides being disaster prone.

2.7 gdb

I found gdb to be very useful in debugging segmentation faults. It is good to have a hands on practice on gdb with the help of a cheat sheet available online. Although **bt** or **backtrace** command is all you need for handling segmentation faults.

2.8 C++

C++ is not only C with STL. It is an ocean in itself to learn and I am still learning it. There are certain idioms and ways of the language which do not make sense but they are still there. Only relying on stack overflow like all of

us do for python, bash and other scripting languages will not work here. I am listing down some ideas which will be helpful and will save you a lot of time with C++.

- **extern C directive** - This helps in keeping both C and C++ code in the same source code and must be used while incuding C only headers. This is required in using some of the DPDK headers. Otherwise you may get undefined reference error.

- Avoid **using namespace std;** and other such directives in writing the new code. This pollutes the namespace and creates confusion for other developers. I did not remove this until now as this is deeply entrenched and used multiple times in headers/source files.

- Use **auto** keyword wherever you can. It is foolish to write std::vector < int>::iterator when using auto is enough and the type can be inferred.

- Other items may be added as required.

3   Later

3.1   Session Related Messages

In a 5G-conforming RAN, the session related messages are sent by SMF to UPF. This RAN-emulator's primary role is that of a load generator. Session related messages - session establishment, modification and release messages are directly sent by RAN to the UPF. There are however some functions defined before the start of the load testing mode in the main function that interacts with AMF and to SMF through AMF. These are not removed yet and can be deleted after testing.

3.2   Cores Layout

NUMA- node 1 with starting core 12 is currently used by the RAN.

- **CORE_RX_POLL**: Receives all the incoming packets from UPF. These includes the latency packets or data packets in the downlink direction.

- **CORE_TX_START - CORE_TX_END** These cores are used to send data packets in the uplink direction i.e. from RAN to UPF.

- **CORE_RTT** This core sends the data plane latency packets in the uplink direction. These packets are reflected back by the DNN so that end to end latency can be measured.

- **CORE_CP_TRAFFIC** This core sends the control plane traffic. A separate core was used so that a call back can be registered for storing timestamps. These timestamps are used for calculating control plane latency.

- **CORE_MISC and CORE_STAT** These cores handles miscellaneous functions like ARP handling, timer and logging of stats.

3.3 Callback functions

Certain call back functions are registered on receive and transmit queues for the purposes of latency calculations.

- **dataPlaneStoreTimestampCallback** This callback is registered on CORE_RTT for storing the timestamp of outgoing latency packets.

- **controlPlaneStoreTimestampCallback** This callback is registered on CORE_CP_TRAFFIC for storing the timestamp for outgoing control plane packets.

- **calculateLatencyCallback** This callback is registered on CORE_RX_POLL for inspecting the incoming packets. These packets are either response packets to control plane packets or the data plane packets reflected back by DNN. The difference in the current time and the timestamp stored in the transmit callbacks gives the latency values.

3.4 Latency Calculation Algorithms

3.4.1 Key Concept

When packets are transmitted from RAN, the timestamp when the packet is forwarded is stored for the outgoing packet. When the response for the packet arrives at the receiving side of RAN, the difference in time values of the current time and the stored timestamp is calculated and then reported in the log file.

3.4.2 Storing of Timestamp

- **Data Plane** The packet identification field in the IP header is used to identify the packet. These latency packets are sent from the core CORE_RTT. All the established sessions/UEs are used for forwarding the traffic. When the packet is transmitted, a callback function stores the outgoing packet timestamp in a hashmap with packet id as the key. This field is generally used in the fragmentation and reassembly of packet data. The data plane latency packet throughput is substantially reduce by introducing sleep commands. The option of disabling the calculation of latency is removed as latency packets do not significantly affect any other metrics. Data plane latency is independently logged in a different column in the log file.

- **Control Plane** This requires deep packet inspection of control plane packets. These packets are sent from the core CORE_CP_TRAFFIC. There are three types of control plane packets - session establishment, session modification and session release packets - all of them are used for

measuring control plane latency. These packets have session ids stored at different offsets in the payload. A hashmap is maintained in which the key (64 bits length) is composed of :

- 16 MSB bits store the message type i.e. establishment (50), modification (52), and release (54).
- The next 48 bits store the session ids.

A callback function is registered on the transmit queue corresponding to CORE_CP_TRAFFIC which stores the timestamp in the hashmap.

3.4.3 Calculation of Latency

A single callback function is registered on the CORE_RX_POLL which receives the incoming packet.

- **Data Plane** The same outgoing packet is reflected by the DNN packet and stored timestamp is retrieved from the hashmap and the difference is the end to end latency of each packet.

- **Control Plane** Every control plane packet has a response packet - establishment response (51), modification response (53), release response (55). The control plane latency is the time period from when the request packet is sent and the response packet is received. The response packets have message type and either session IDs in their payload. For modification and deletion responses, the session ids have a difference of 3000. So key is retrieved from the response packet and difference in the current time and the stored timestamp is calculated.

3.4.4 Why do we need different techniques for control plane and data plane?

- **Why can't we use packet identifier for control plane packets?** Both types of packets are received on the same rx queue/core. So, only the packet identifier in ip header is not enough to differentiate control plane and data plane packets. And you will need deep packet inspection anyways to differentiate among the two packets.

- **Why don't we inspect packet payload for data plane packets?** Data plane packet has no useful payload and when the ip header identification field is unused till now, we can use it. :

- **Development Sequence** Data plane latency was implemented earlier when the packet id field was unused. Control plane latency calculation is done later.

3.5   DPDK APIs used

List of all DPDK APIs with their description, list of arguments and the return value is available at `https://doc.dpdk.org/api/index.html`. Although the list on documentation is according to the latest version of DPDK, function signatures do not change across implementations. There is a deprecated functions list on the documentation page to check for the list of deprecated functions. This implies that do not be hesitant in looking at the documentation on the webpage irrespective the versions. The only issue with the documentation is that its search functionality is not upto the mark. It does not account for spelling mistakes etc. Moreover searching on google for DPDK apis is also not very effective. I shall list down the most frequently used APIs and what I think any new developer who extends the code needs to know. Kindly refer to the documentation on the url for correct use and complete understanding.

- rte_eal_remote_launch,rte_eal_wait_lcore
- rte_pktmbuf_mtod_offset
- rte_eth_tx_prepare, rte_rth_rx_burst
- rte_be_to_cpu_(64, 32, 16)
- rte_eth_add_(tx,rx)_callback
- rte_delay_(ms,us), rte_rdtsc, rte_timer_init, rte_get_timer_hz, rte_get_timer_cycles
- struct rte_(ipv4,udp)_hdr
- rte_lcore_id, rte_lcore_count
- rte_mempool_get_bulk, rte_pktmbuf_reset_headroom

4   Modes of Operation

The load generator can be used in different modes to characterize the different behaviors of the user plane function. The various modes are

- Setup Sessions and Send Data.
- Send only control plane traffic.
- Send Control Plane and Data Plane traffic simultaneously.
- Helper functions.
- QoS functions.

4.1 Setup sessions and send data

Initially sessions are setup by sending session establishment and session modification messages directly from the RAN to the UPF. Once the sessions are setup the data packets are forwarded from CORE_TX_START- CORE_TX_END. Data Plane latency options while forwarding are

- **n** - Send latency packets from all the established sessions.

- **y** - Send latency packets from a specified UE or session.

Control plane latency is calculated during the initial setup for all the session related packets. Throughput is also logged in the log file. The number of UEs/sessions that are used for sending data per core is asked to the user. The sessions are partitioned in disjoint sets among the cores before forwarding.

4.2 Send only control plane traffic

This mode is used to send only control plane traffic and no data plane forwarding takes place. This is a synthetic traffic i.e. it does not represent any real world scenario. The main motive behind this mode was to saturate control plane and measure control plane handling capabilities of the different designs of the UPF. The latency and throughput values are logged in the file as earlier. This is an open load test in which session establishment, release and modification packets are sent one by one with a user provided inter packet delay (in us). These packets are sent from CORE_CP_TRAFFIC. For the uninitiated, the open load means that the RAN does not wait for response packets before sending the next packet.

4.3 Send Control Plane and Data Plane Traffic Simultaneously

$n1$ sessions are established before the data forwarding takes place. These sessions remain established throughout the run. $n2$ sessions are established, modified and released while the data forwarding is also taking place. The data packets are sent from all the currently established sessions. The minimum value of currently established sessions is $n1$. The maximum value is $n1 + n2$. The duration $t1$ for which all the static sessions $n1$ and the dynamic sessions $n2$ are used is also asked to the user. So the sessions are established, modified and then after waiting for the time $t1$, the $n2$ sessions are released. This establishment, modification and release cycle continues for the entire duration of the experiment. Each of the data packet forwarding cores use all the existing established sessions/UEs to forward the data. Note that this different from the case when sessions were partitioned among cores. Another implementation difference from the case when we only send

4.4　Helper Functions

There is only one helper function right now. This helper function - Delay Estimator - helps in mapping inter batch delay with the data forwarding throughput of the load generator for a given number of cores and sessions. This helps in limiting the rate of forwarding data packets. Intel 40Gbps NICs that we have does not have rate limiting APIs like 10Gbps NIC.

4.5　QoS Functions

These functions were developed to check the correctness of QoS algorithms deployed at the UPFs. These functions were tested on 10Gbps NIC systems and may require modification for other NICs.

4.6　Data Forwarding Options

- Number of cores for forwarding the data.

- The inter batch delay in microseconds. Larger the delay, slower the forwarding rate.

- Number of UEs per core. Per Core different UE count can be given by setting the macro ENABLE_DIFF_PER_CORE_UE_COUNT to 1. This option was required for producing skewed traffic.

- **ENABLE_GTP_EXTN_HEADER** : This option enables/disables the GTP Extension header. RTC model that uses customized hash function which parses the inner GTP headers requires extension headers to be disabled. For other models, kindly enable GTP extenstion header.

4.7　Payload Size

There are two available options:

- **Payload Size** Can be set to anything in the range 64-1400 bytes. The smaller payload size requires higher header processing per byte at UPF.

- **IMIX traffic** IMIX distribution resembles the real world traffic. The distribution mentioned on `https://en.wikipedia.org/wiki/Internet_Mix` is used to model IMIX traffic in out case.

4.8　Files and Folders

DPDK_RAN folder is created inside the 5GCore folder. This folder hosts all the relevant code, headers, build and log files. This section describes the relevant files that are present in this folder.

- **loadDPDK.sh** This script creates the dpdk binaries in the utility/dpdk/x86_64-linuxapp-gcc folder. This script also asks for the PCI addresses of the NIC ports that will be used by the DPDK Application. The generation of Makefile using CMake and the build using make is also performed by this script. This script is required to be run only once after every reboot. The ports need to be unbind from kernel driver before executing this script. Use make to rebuild after every source code/ header modification.

- **CMakeLists.txt** This file is used by CMake to generate make files. Going through this file will help in understanding various dependencies. Writing a new CMakeLists.txt may require the study of documentation present on their webpage.

- **include/dpdk**$_r$*an.h***include/dpdk**$_c$*onfig.h***src/ranMain.cpp**

- **src/ran.cpp**

- **src/dpdk**$_r$*an.cpp***src/dpdk**$_c$*onfig.cpp*