Using DPDK to build 5G Data plane - UPF and RAN emulator

A Project Report Submitted

in the Partial Fulfilment of the Requirements

for the Degree of

Master Of Technology

by

Prateek Agarwal

Computer Science and Engineering

Indian Institute of Technology Bombay

June, 2021

Contents

List of Figures

Chapter 1

Introduction

1.1   5G Data Plane

5G technology aims to satisfy the increasing requirements of the customers of mobile service in terms of higher throughput and lower processing latency. The major difference in the processing of data and signalling packets between 5G and earlier standards is control-user plane separation and the use of network function virtualization. Forwarding of packets, authentication of mobile devices, session establishment and management are some of the network functions required in the core of a telecommunication network. These network functions run on different or same physical machines as virtual machines for easier migration/scaling.

This project is mainly concerned with the data forwarding plane. The network functions in our setup run as separate processes. The high speed data plane is required to meet the ever increasing demands of higher bandwidth and lower latency. The Linux kernel stack cannot meet this demand as it is a general purpose stack catering to different protocols and there is a packet copy overhead from kernel space to the user space for reading the packet payload. The technqiues used to overcome this limitation of the Linux stack is the use of kernel bypass frameworks like Data Plane Development Kit (DPDK) [2], and the use of programmable NICs to offload processing functions in the hardware. DPDK runs in software and there is a single copy of packets from NIC buffer to the user space. DPDK libraries are optimized for high speed data processing. The use of superpages, lockless data structures, better memory management and customized device drivers for various NICs are
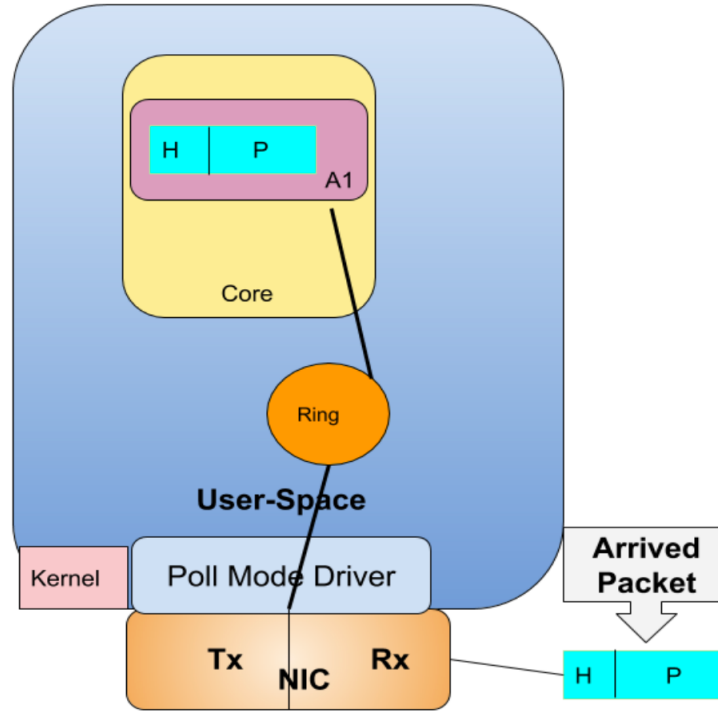
Figure 1.1: DPDK kernel bypass

among some of the salient features of DPDK framework which enables high speed processing of data.

The main network functions in the data plane are

- **Radio Access Network (RAN)** RAN is the main point of contact for all the user equipments and is an entry point to Non Access Stratum in the 5G network.

- **User Plane Function (UPF)** UPF is responsible for forwarding the incoming packets from the RAN to the public internet. UPF ensures QoS guarantees, collects usage statistics useful for charging purpose, buffers packets (if required).

- **Data Network Name** DNN is the gateway to the public internet. DNN forwards the uplink packets from UEs to the public internet and all the incoming packets for the UEs (downlink) are received at the DNN.
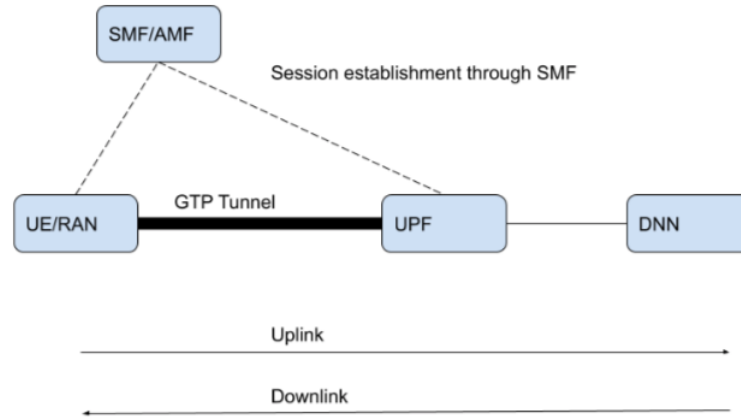
Figure 1.2: 5G Forwarding Plane.

1.2 Problem Statement

- **User Plane Function: Implement the packet steering offload mechanism in the UPF.**

  The packets coming from RAN are tunneled in the application layer of an outer packet.The tunneled application level header contains the information associating the data packet to a corresponding user. The outer IP/UDP headers are same for all the packets between a set of network functions. The key for the standard hash computation in the hardware are the outer packet headers. The goal here is to implement the UPF design in which the packets are redirected in the hardware to the multiple cores for parallel processing based on the inner packet and application level headers. The redirection based on inner packet fields ensure that packets of the same flow are directed to the same core resulting in better spatial locality.

- **Radio Access Network** Extend the existing DPDK based RAN by adding new features and refactor, clean the code to make RAN extensible in the future. The new features are aimed at emulating the real-time traffic and to benchmark various designs of UPF.

1.3 Major Contributions

- The UPF design which with the hardware based redirection on the inner packet headers was implemented.

- Two new features were added in the RAN - measurement of control plane latency and simultaneous transfer of control plane and the data plane traffic.

- The source code of the existing RAN was entirely refactored and restructured.

- Conducted experiments for benchmarking different models of the UPF for measurement of throughput and latency metrics under different load conditions.

Chapter 2

Background

2.1 Interaction of Network Functions

The network functions in the data plane - RAN, UPF and DNN are defined in section 1.1. Following steps describe the interaction between various network functions
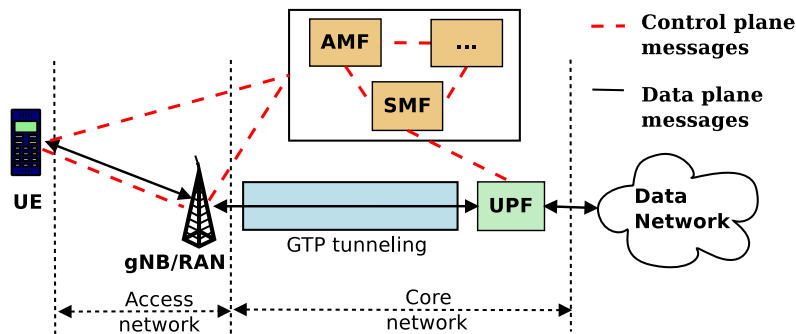
Figure 2.1: 5G Architecture.

during a session-

1. UE interacts with the Access and Mobility function (AMF) to request for intitiation of a new session.

2. AMF authenticates the UE and forwards the request to Session Management Function (SMF).

3. On receiving the request from AMF and after getting the UE details from other NFs (UDR etc.), SMF initiates an session establishment procedure to User Plane Function. PFCP protocol (section 2.2) is used for the interaction between UPF and SMF.

5

4. Once the session related details are received and stored at the UPF, a confirmation is sent to the UE signalling the permission to forward data packets.

5. Data packets are sent by the UE through RAN to the UPF.

6. GTP-U (section 2.3) - a tunneling based protocol is used between RAN and UPF. GTP header has an identifier to map data plane packets to a particular session.

7. UPF de-encapsulates the GTP packet and based on the stored session related information provides various services like QoS, forwarding to a DNN etc.

8. The downlink packets are received by the UPF from the data newtork (DNN) and performs GTP encapsulation before forwarding them to the RAN/UE.

9. Once the required data is forwarded, UE sends session termination request to AMF which forwards it to the SMF. SMF sends session release request to the UPF. The UPF deletes the session related information at itself.
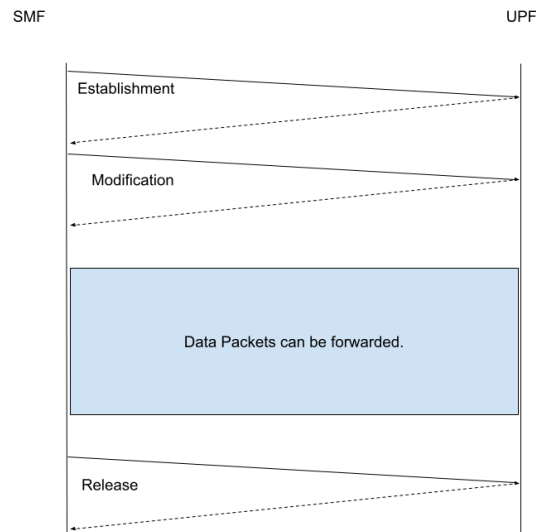
2.2 PFCP Protocol

Figure 2.2: PFCP Session Messages

PFCP stands for Packet Forwarding Control Protocol ([3]). There are two types

of messages sent using PFCP - node related and session related. The discussion here describes session related messages.

Session Management Function (SMF) interacts with the User Plane Function (UPF) to setup sessions related information at the UPF. This information enables UPF to identify data packets of different sessions coming from RAN and provide forwarding, usage report, charging, buffering, QoS related service to the sessions. Each session is identified by a unique session ID which helps in differentiating among different UEs/sessions at the UPF.

There are three kinds of messages sent from SMF to the UPF in PFCP session related messages-

- **Establishment Request** This message has all the forwarding, usage report, QoS etc. related information for a new UE session.

- **Modification Request** Once the establishment response is received from the UPF, the UPF sends the modification request. The important field in this message is the intimation of identifier that will be used for data plane packets coming from the UPF. The data forwarding can not start before the modification response is received.

- **Release Request** Once the session is not required, th release request is sent to the UPF signaling the tear down of the session and UPF may remove this session's related information.

UPF gives response to each of the three messages.

2.3   GTP Protocol

The raw packets meant for communicating outside the network are encapsulated within GTP application level header. GTP ([4]) runs over UDP/IP stack. This GTP header (shown in 2.3)contains various fields for regulating the session parameters. The relevant fields are

- **Tunnel Endpoint Identifier (TEID)** identifies the packet with a particular session which may have different Quality of Service parameters.

+	0-2	3	4	5	6	7	8-15	16-23	24-31
0	Version	Protocol type	Reserved	Extension Header Flag	Sequence Number Flag	N-PDU Number Flag	Message Type	Message length	
32						TEID			
64				Sequence number				N-PDU number	Next extension header type

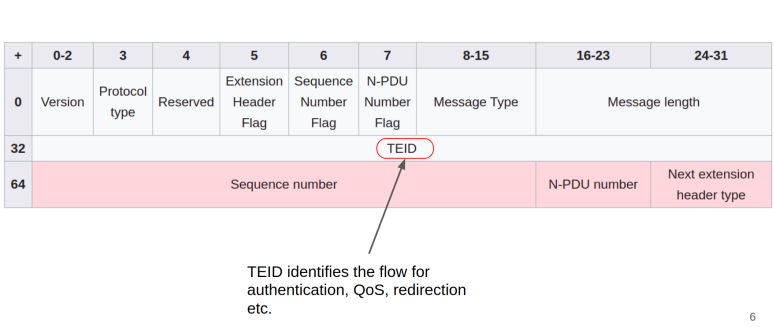TEID identifies the flow for authentication, QoS, redirection etc.

6

Figure 2.3: GTP-U header [1]

- **Extension Headers** These are required for providing differentiated services to a given packet in the same or different session.

Chapter 3

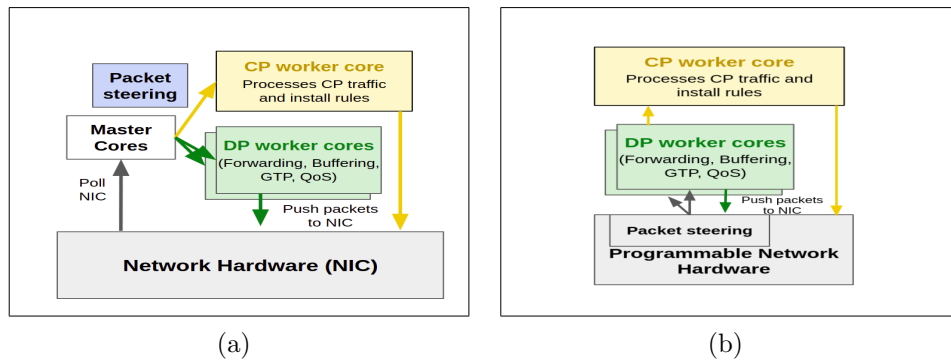# Stage 1 Summary

3.1 DPDK based UPF designs

Figure 3.1: (a) Software UPF (b) SteerOffloaded Design

There are two main alternatives for DPDK based UPFs -

- **Software UPF** - All the processing happens in software. The sole task of the NIC is to transfer packets into the buffers of the primary core. The primary core then redirects packets onto secondary cores which perform the entire processing of data packets.

- **Packet Steering Offload** - The packet redirection to different cores happens on the NIC using RSS based hash computation of the inner packet headers of a GTP packet. The standard RSS computation on the outer packet IP and UDP headers is not sufficient for solving redirection issues. This is due to the fact that outer headers correspond to the RAN and UPF end points- they are

same for all the sessions emanating from a single RAN. This leads to poor randomization and all the packets are redirected to a single core for the same 4-tuple. The inspection of GTP header is made possible by NICs which can be configured dynamically for different protocols. Intel provides Dynamic Device Personalization [5] feature for 40Gbps XL710 NICs. This feature was used to implement the packet steering offload design of the UPF.
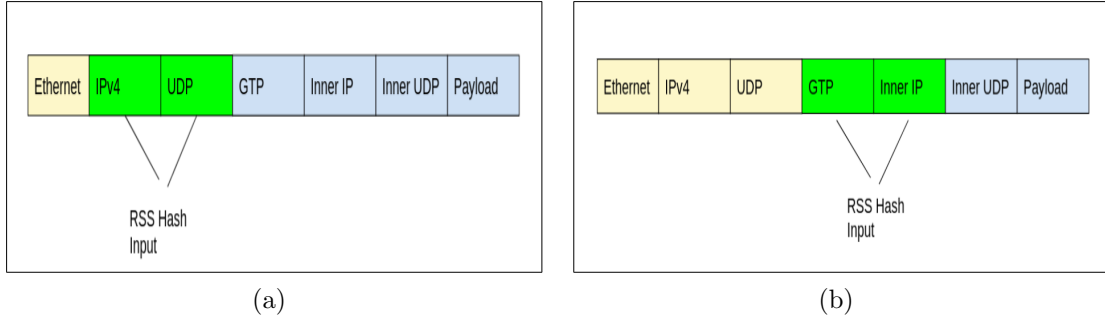

Figure 3.2: (a) Standard RSS (b) DDP RSS

3.2 Comparison of the two Models

Packet Steering offload has the benefit of hardware acceleration. The hash computation is offloaded to the hardware. The amount of processing on the CPUs is reduced. Elimination of inter-core communication between the primary and the secondary core in the case of Software UPF is also a great advantage. This makes the steering offload design better performant and provides higher throughput and lower latency for the same number of forwarding cores.

The main advantage of Software UPF lies in its flexibility. The steering offload cannot move the packets of a specific flow/session from one core to another easily. This requires reconfiguration of the queue-to-core mapping. This movement of the session among cores may be required in the case of skewed load from few sessions. This sessions may be from heavy hitter UEs overloading the same core. This will lead to packet drops on the heavily loaded core in the offloaded steering design. When the load on the UPF is highly variable across the time, it may be a good idea to switch off some cores under low load conditions and to increase the number of data forwarding cores when the load is high. This dynamic scaling can be easily

achieved in software UPF without appreciable loss of performance. Steering Offload design requires the shutting down of port and reconfiguring the registered number of queues at the NIC for receiving the data. This leads to a loss of performance in terms of the number of packets dropped during reconfiguration.

3.3   Evaluation

3.3.1   Experimental Setup

Refer to Figure 5.1. UPF and DNN run on a different physical machine. RAN and other Network functions run on the a single machine. For these experiment, Intel XL710 40 Gbps NICs were used.

3.3.2   Performance

Figure 3.4 shows the latency and throughput performance of the two designs. Offloaded packet steering has 45% higher throughput and 37% lower latency when compared to Software UPF. This is expected due to the reasons mentioned in Section 3.2.

3.3.3   Flexibility

Two experiments were conducted to show the flexibility of software UPF design over offloaded steering design.

Dynamic Scaling The load was scaled up in steps after every 60 seconds. The average queue length was considered as a criterion for measuring the load. When the average queue length increased beyond a certain threshold, a new core was started / allocated for forwarding data packets. This reconfiguration was quite fast in software UPF design. The offloaded steering design required restart and reconfiguration of NIC Rx-queues for increasing the number of data forwarding cores. This lead to substantial packet losses during the time (around 500 ms) (see Figure 3.3).

Heavy Hitters In an unfavorable scenario, a set of UEs or a single UE mapped to a single core in the offloaded Steering design can send unusually high traffic. It is not possible to redistribute load among cores in the case when the steering

is offloaded in the hardware. The software UPF can keep a tab of packet losses/ increased latency to redistribute load among cores. The traffic from RAN was sent in such a way that one of the core was heavily loaded. A mapping of the sessions to the core was obtained before the experiment was conducted. Figure 3.5 shows the difference in the impact on latency between the two designs.
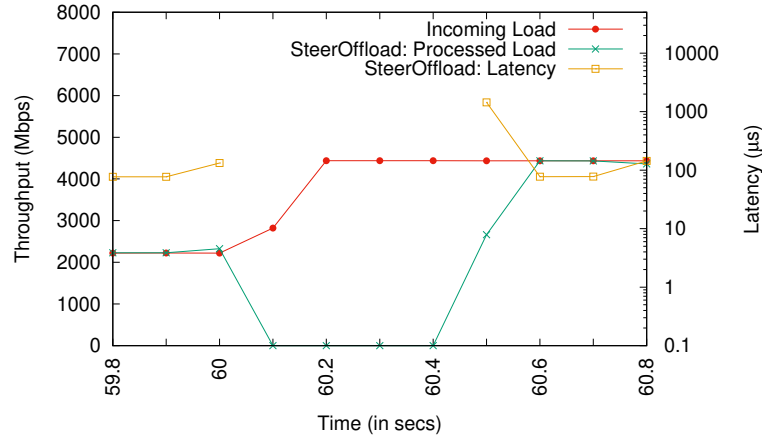
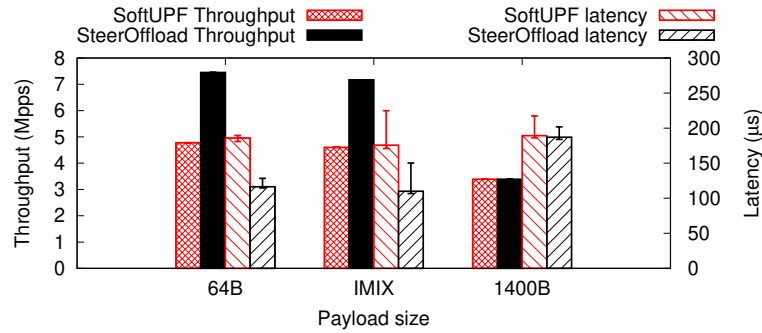Figure 3.3: SoftUPF vs. SteerOffload: Dynamic Scaling

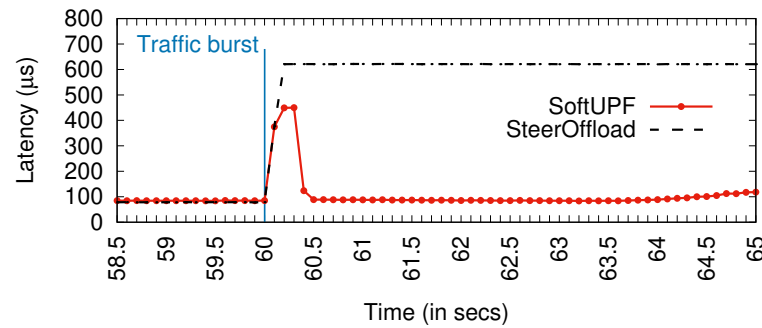Figure 3.4: SoftUPF vs. SteerOffload: Performance

Figure 3.5: SoftUPF vs. SteerOffload: Heavy Hitters

Chapter 4

RAN Design and Implementation

4.1 Design Objectives

The RAN-emulator which acts as load generator in both the data plane and control plane has to satisfy the following objectives -

- **High Throughput** The data plane packets should be generated at a sufficient pace to saturate 40 Gbps link.

- **Flexible Modes** The data should be sent at a varying rate from a very low load to the saturation throughput either in a single run or across different runs. It should also be possible to support various modes e.g. both the data plane and control plane packets are sent together, correctness of QoS at UPF can be checked, find a mapping between inter-batch delay and throughput.

- **Latency Measurement** The measurement of latency for both the data plane and control plane packets should be possible.

- **Logging** The latency and throughput should be measured and logged at the every fixed interval (parameter for a given run).

4.2 Prior Work

4.2.1 Kernel-Based RAN

The major limitation of this RAN is that it is not able to saturate max line rate of 40 Gbps. This is due to the fact that there are multiple copies of packet when the packet is received or transmitted - from user space to kernel space and from kernel space to the NIC buffers when the packet is forwarded.

4.2.2 DPDK-based RAN

This RAN is based on DPDK. This RAN satisfied the requirements of high data throughput and low processing latency. The current work extends on this emulator. This emulator had satisfied most of the objectives mentioned in 4.1 except for the following -

- Measurement of Control Plane Latency.

- Sending Control Plane and Data Plane Packets simultaneously.

4.3 RAN Architecture

4.3.1 Threading Model

DPDK pins one thread to each core for avoiding the overhead of context switch. This is especially important for both the data forwarding cores and the polling cores to get the maximum throughput per core. The auxillary threads like ARP messages handler, heart beat messages, timers etc. can run on available free cores. The cores which send data and control plane latency packets have to handle the callbacks associated with the storage of timestamps of outgoing packets. Running them on separate cores is preferred if the sufficient number of cores are available.

The NIC queues/buffers for storing the outgoing (TX-queue) and the incoming (Rx-queue) packets are one-to one mapped to the data forwarding/polling cores. The threads pinned to these cores process the incoming packets and prepare the packets for forwarding.

4.3.2   Core Layout

There are 24 cores on the available machines. 12 cores are available on each NUMA socket. Although there are 2 CPUs (2 hardware threads) available on each core, hyperthreading is kept off for reducing the non deterministic behavior attributed to contention of hardware units and getting repeatable results.

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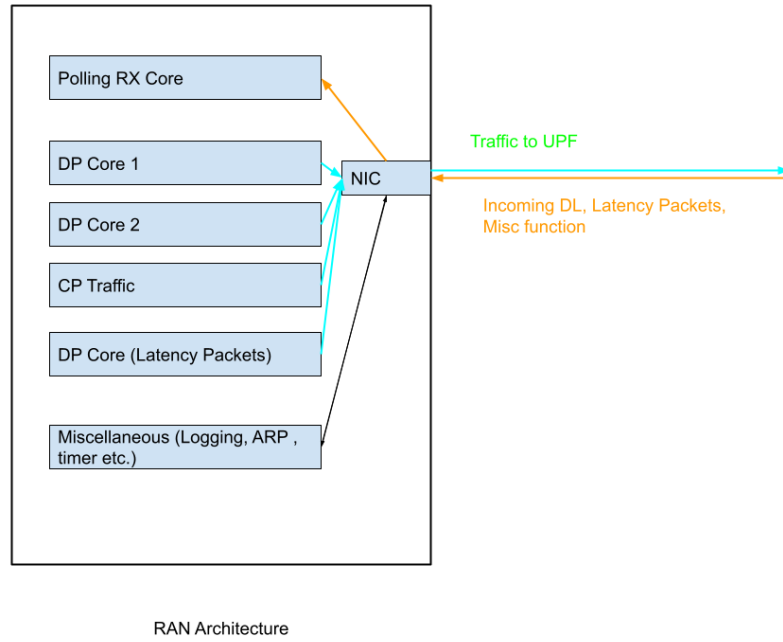

RAN Architecture

Figure 4.1: RAN Architecture

- **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.

Using this layout, a maximum of 7 cores are available for the data forwarding on the same numa node. To increase this number, the functions running on CORE_MISC and CORE_STAT can be run parally on the same core. Although 7 cores have been found sufficient till now to saturate 40 Gig line with 64 byte packets.

4.4 Latency Calculation Mechanisms

4.4.1 Key Idea

The sample packets that are used to measure latency are sent from a specific core. When these packets are copied into a NIC buffer, a callback function is called. This callback function stores the timestamp of the outgoing packet in an array. The location at which the timestamp is stored is identified by an identifier present in the outgoing packet. When the response to a sample packet arrives at the receiving core, a callback function is called to measure the end to end latency or response time of the UPF. This callback function extracts the timestamp stored previously and subtracts it from the current time to get the latency measure.

4.4.2 Defintion of Control Plane Latency

For control plane, the measured latency is the difference between two events:

- Forwarding of PFCP requests - session establishment, modification, and deletion requests.

- Receipt of PFCP responses to the establishment, modification and response requests.

This time period includes the following major delays -

- **Processing Delay** This includes the processing of request packets, updation of local data structures at UPF and preparation of response packets.

- **Queueing Delay** This is the time when the PFCP packets remain queued in the NIC or any userspace buffers present in the UPF and the RAN.

4.4.3   Implementation Issues

As discussed in 4.4.1, the main issues to address for measuring control plane latency are-

1. **Identify the identifier** All kinds of PFCP messages have a session identifier field in Forwarding Action rules that is used by the UPF to classify the data plane packets across different UEs/sessions. This field is present at different offsets for different PFCP messages. Each message is also identified by a unique number, for example - 51 for establishment request, 52 for establishment response etc. So we have 6 different kinds of messages for a given session id. A combination of both the message type and session identifier uniquely identifies the packet.

2. **Manage the Callback Overhead** It becomes difficult to process all the callbacks if the packets are sent at a sufficiently high rate. The array is a performant data structure in the sense that it is random access and data is stored contiguously. The access pattern in our use case an excellent spatial locality and temporal locality i.e. all the consecutive entries are accesssed one after the other (session IDs increase sequentially).

4.4.4   Details

A fixed length array is used to store timestamps - **TSArray** . The array size is $65536 * 3$ and stores the value of timestamp register **rdtsc** for the outgoing packets. The first 65536 values are used to store establishment packet related timestamps, next 65536 modification packet related timestamps and then release packet timestamps are stored. The bitwise operations can be easily used as 65536 is a power of 2.

When the corresponding response packets are received, the matching entry is retrieved from the array and latency is calculated.

4.5 Control Plane + Data Plane Traffic

$n1$ 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 $n1 + n2$.

$t1$ is the total duration of the experiment. $t2$ is the duration of the establishment, modification and release cycle of each of the $n2$ sessions.

The duration $t1$ for which all the static sessions $n1$ and the dynamic sessions $n2$ are used is also asked to the user.

Data forwarding starts from the $n1$ sessions at the start. After sleeping for a time (currently 5 seconds), $n2$ threads are started. The role of each thread is to sleep for a random amount of time $t3$ ($< t2$), establish and modify the session, and then sleep for some time $t4$ and release the session. The invariant is $t3 + t4 = t2$. The session is available for data forwarding in the period $t4$. The threads with new session ids are started once all the previous threads have joined i.e. have finished their task. Each of the data packet forwarding cores use all the existing established sessions/UEs to forward the data. Note that this is different from the case when sessions were partitioned among cores.

4.6 Summary

The control plane latency calculation and simultaneous transfer of the control and data plane messages were added in the code. The inherited code had various issues which were also addressed -

- **Refactoring** The main routine was 2000 lines long and had no function calls for various modes. Repeated snippets of code were also refactored for better understanding of the code.

- **Data Plane Latency** 100 Mbps of traffic (64 B payload) was sent for sampling data plane latency alone. This number was reduced to 0.4 Mbps as

the numbers become quite stable even for lower values. This also reduces the callback pressure on the core.

- **Redundant and Dead Code** The same data forwaring mode code supported through kernel functions were removed. The functions which were never called , the branches which were never taken were also removed.

- **Directory Structure** The RAN function was defined earlier inside AMF to avoid writing CMake file. The logs were also produced in the parent folder itself. These issues were also addressed (see Figure 4.2).

- Deprecated forwarding modes, stale comments, TODOs, functions declared but not used were also removed. Global variables were defined in a separate namespace.
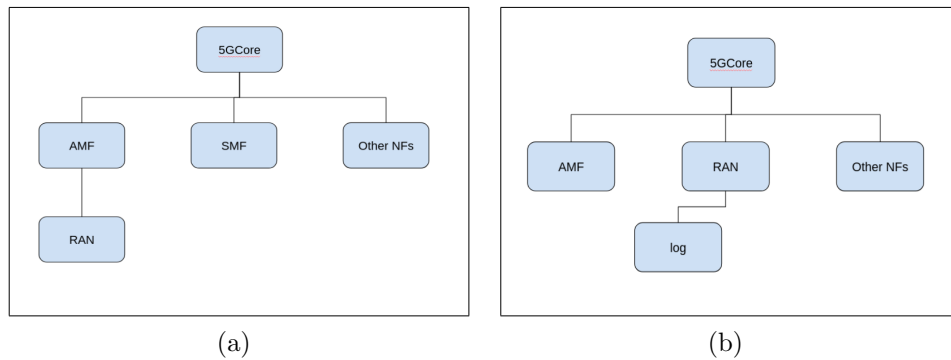

(a)

(b)

Figure 4.2: (a) Previous Directory Layout (b) Current Layout

Chapter 5

Evaluation

5.1 Experimental Setup

The setup is illustrated in Figure 5.1. The main features of the setup are:
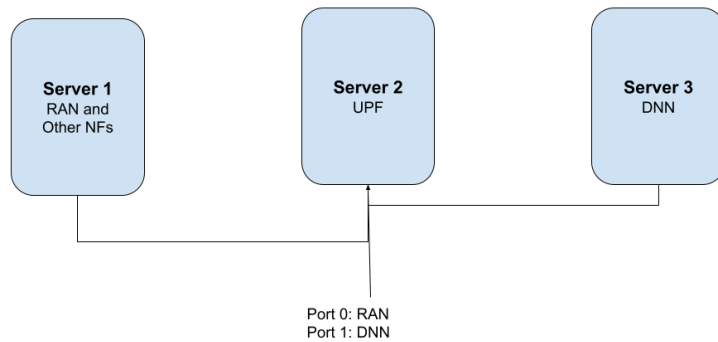

Figure 5.1: Experimental Setup

- **Server's Layout**

  1. **Server 1** The radio access network (RAN) and other network functions responsible for authentication, session establishment, charging functions are simulated on server 1. The user equipments are also simulated on the same machine. The load generation in the uplink direction is the main function of this server in the data forwarding plane.

2. **Server 2** This server hosts the user plane function (UPF). Two ports on this server are used to communicate with RAN (server 1) and DNN (server 3).

3. **Server 3** This server simulates the data network name (DNN) network function. This network function is the gateway to public Internet for 5G telecommunication. This server is used to generate downlink traffic. This server is also used to mirror packets received from uplink and forwarded back in the downlinj direction. The mirroring of latency packets help in measuring end-to-end latency (Round Trip Time) at the RAN.

- **Hardware Configuration**

    1. **CPU** Intel Xeon Core i5 @2.20GHz. 12 cores on every NUMA node. Only a single NUMA node is used in the experiments. Hyperthreading is kept off to facilitate repeatability of results.

    2. **Memory** 8192 superpages of size 2MB are reserved initially for the whole run.

    3. **Cache** 32 KB L1i-cache, 32 KB L1d-cache, 256 KB L2 cache per core. 30 MB L3 cache per NUMA node.

    4. **NIC** Netronome Agilio CX 2x10GbE smartNIC [6].

5.2 UPF

The primary role of user plane function is described in 1.1. UPF is the main network function which takes the centre stage in the core network. The RAN load generator is used to test the processing capability of UPF. The metrics used are throughput and latency of data plane and control plane traffic.

Different designs of the UPF are studied:

- **Software UPF** This design uses kernel bypass framework DPDK for userspace processing of the incoming packets. All the control plane and data plane packets are received on the master core which redirects these packets into different worker cores (discussed in 3.1).

- **Offloaded Data Plane** The complete data plane processing is offloaded on the smartNIC. The control plane messages are still processed in the software. Once a session is established (released), the rules are inserted (removed) in the smartNIC (see Figure 5.2).

- **Offloaded Control Plane** In this design, the control plane handling is also offloaded to the NIC.
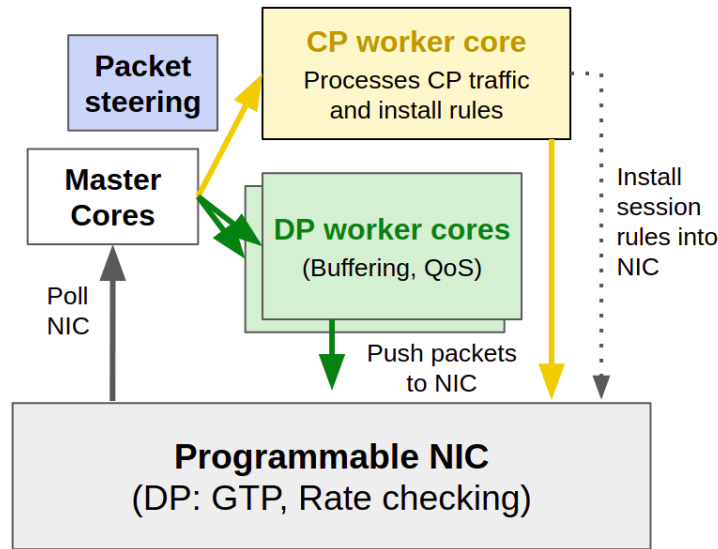
Figure 5.2: Data Plane Offload Design

The detailed description of these designs are available in [].

Performance metric	SoftUPF	DPOffload	CPOffload
Throughput (messages/sec)	5.1K	666	2.08M
Latency (μs)	113	1646	24

Table 5.1: Control plane performance.

5.3 Results

5.3.1 Control Plane Performance

The latency v/s. throughput graphs for different UPF designs are illustrated in 5.3, 5.4, 5.5. The summary of the results is shown in 5.1. The table 5.1 shows the latency around the control plane saturation. The control plane throughput is defined in terms of sessions handled per second - the whole establishment, modification and response cycle for a given session is counted once.
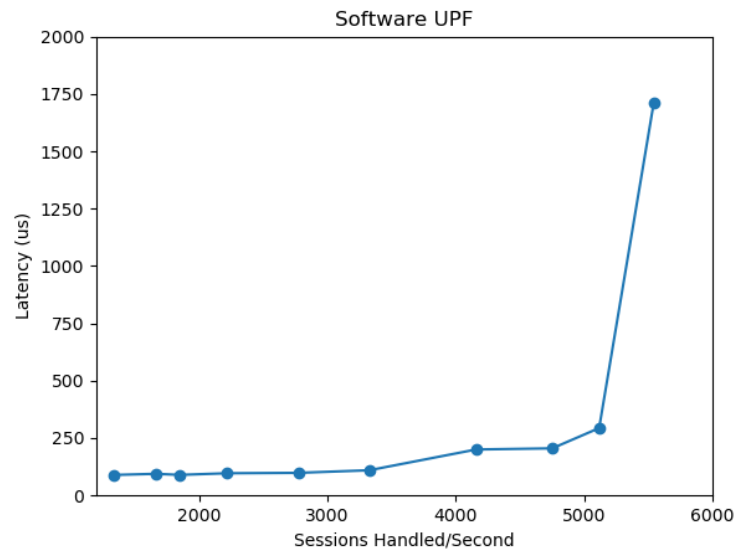

Figure 5.3: Control Plane Performance for Software UPF

When the control plane is offloaded, we can see the best performance in terms of latency and throughput. With the control plane offload, the latency is only around 24 us and 2.08 M sessions can be handled per second. The performance deteriorates for data plane offload when compared to software UPF. This is attributed to the bottleneck created between offloaded data plane and user space control plane - once the control plane messages are processed in the user space, the rules are installed in smartNIC.
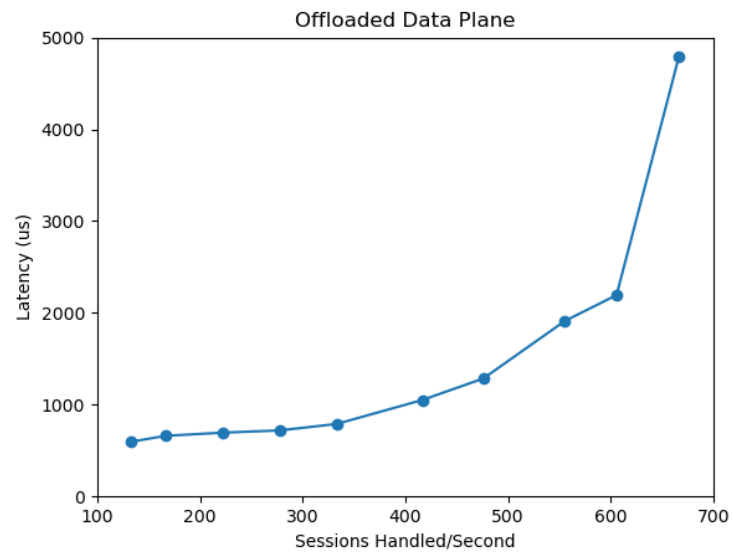
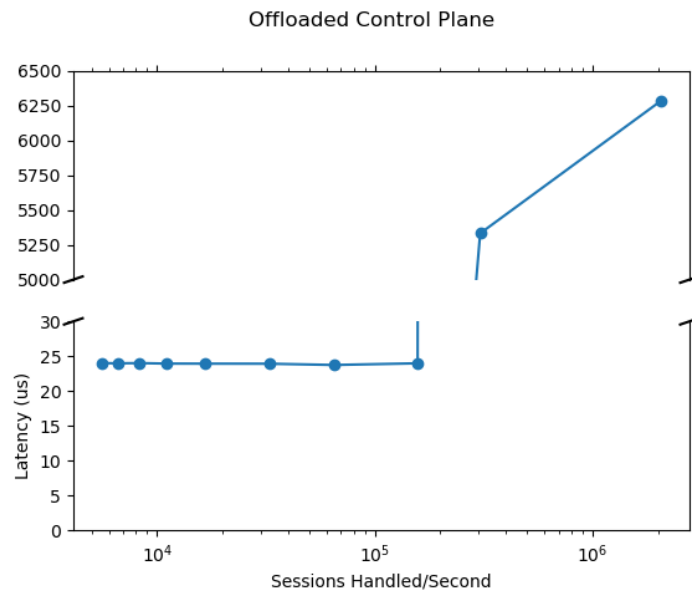Figure 5.4: Control Plane Performance for Offloaded Data Plane

Figure 5.5: Control Plane Performance for Offloaded Control Plane

Bibliography

[1] `https://en.wikipedia.org/wiki/GPRS_Tunnelling_Protocol`.

[2] `https://www.dpdk.org/`.

[3] "3gpp ref #: 29.244. 2017. System architecture for the 5G System (5gs)," 2017.

[4] G. 29.060, "General Packet Radio Service (GPRS); GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface," 2015.

[5] `https://builders.intel.com/docs/networkbuilders/`
`intel-ethernet-controller-700-series-gtpv1-dynamic-device-personalization.`
`pdf/`.

[6] N. systems, "Agilio CX 2x10GbE SmartNIC," 2020.