User Plane Function (5G): Optimizing Run-to-Completion model and its comparison with Pipeline model.

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

December, 2020

Contents

List of Figures **v**

1 Introduction **1**

1.1 5G Forwarding Plane 1

1.1.1 Radio Access Network (RAN) 1

1.1.2 User Plane Function (UPF) 2

1.1.3 Data Network Name (DNN) 2

1.2 GTP Protocol 3

1.3 Organization 3

2 UPF Architecture **4**

3 Receive Side Scaling **6**

3.1 What is Receive Side Scaling or RSS? 6

3.2 Standard RSS 7

3.3 Alternatives 7

3.4 Dynamic Device Personalization 8

3.5 Limitations of Dynamic Device Personalization 9

4 Models of Execution **10**

4.1 Run-to-Completion (RTC) 10

4.2 Pipeline 11

4.3 Comparison 11

5 Experiments and Results **13**

5.1 Experimental Setup 13

5.2 Experiments 14

5.2.1 Throughput v/s. Payload Size 15
5.2.2 Throughput v/s. Number of active sessions/UEs 17
5.2.3 Scaling of throughput with number of cores 17
5.2.4 Handling Skewed Traffic 20
5.2.5 Dynamic Scaling 22
5.3 Summary 23

Bibliography **27**

List of Figures

1.1 5G Forwarding Plane 2
1.2 GTP-U header [1] 3

2.1 UPF Architecture 4

3.1 Standard RSS input Fields 7
3.2 DDP RSS input Fields 8
3.3 Extension Header Limitation 9

4.1 Run-to-Completion 10
4.2 Pipeline 11

5.1 Experimental Setup 13
5.2 Throughput v/s Payload Size. Throughput calculation in Gbps includes header size. 16
5.3 Throughput (in Mpps) v/s number of active UEs. 17
5.4 Throughput v/s Number of Cores (64 Byte payload). Throughput calculation in Gbps includes header size. 18
5.5 Throughput v/s Number of cores (IMIX payload). Throughput calculation in Gbps includes header size. 19
5.6 Handling of skewed UE traffic in pipeline and RTC models 21
5.7 RTC: Figure (a) shows throughput and end to end latency during the whole run. Figure (b) shows the output during one of the core launches (the first one). 24

5.8 Pipeline: Figure (a) shows throughput and end to end latency during the whole run. Figure (b) shows the output during one of the core launches (the first one). 25

Chapter 1

Introduction

5G forwarding plane and the relevant network functions are briefly reviewed in 1.1. GTP protocol and its significance is discussed in 1.2. The layout of the report is outlined in 1.3.

1.1 5G Forwarding Plane

The major difference in data packet processing between 5G and earlier standards is control-user plane separation and the use of network function virtualization. Forwarding of packets (user plane), authentication of mobile devices (control plane), session establishment and management (control plane) 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 (preferably) for easier migration/scaling.

This project is mainly concerned with data forwarding plane. The network functions in our implementation will run as separate processes. The network functions relevant for forwarding plane are described further.

1.1.1 Radio Access Network (RAN)

RAN is a point of contact for all the user equipments (UEs) like handsets, IOT devices, industrial machine controllers etc. RAN runs on all the mobile towers and UEs communicate with the one in their vicinity. RAN is responsible for talking to Access Mobility Function for authenticating the UEs, registering the new session. The

session establishment request is further forwarded to Session Management Function (SMF) which establishes a new session and forward session information to the User Plane Function (UPF).

1.1.2 User Plane Function (UPF)

User plane function (UPF) is responsible for forwarding packets from user equipments to the Internet and vice versa. The uplink direction is defined as the flow of the packets from user equipments to the Internet. The downlink direction is defined as the traffic coming from the Internet to the user equipments/RAN. The main tasks of UPF are

- **GTP Encapsulation** in the downlink direction.
- **GTP Decapsulation** in the uplink direction.

The GTP protocol is discussed in section 1.2.

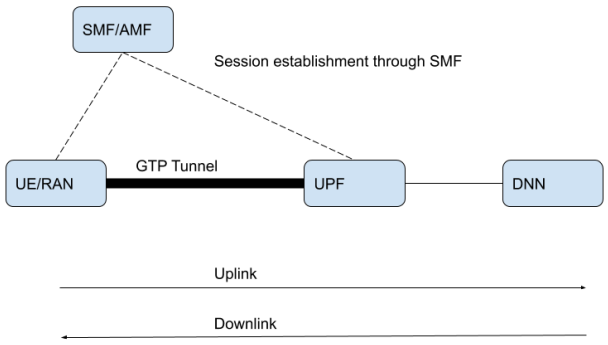


Figure 1.1: 5G Forwarding Plane

1.1.3 Data Network Name (DNN)

This network function is the gateway to the public Internet. All incoming packets from outside the local network are received by this NF and are subsequently forwarded to the user equipment through the UPF and the RAN.

1.2 GTP Protocol

The raw packets meant for communicating outside the netowrk are encapsulated with GTP application level header. GTP runs over UDP/IP stack. This GTP header (shown in 1.2)contains various fields for regulating the session parameters. The relevant fields are

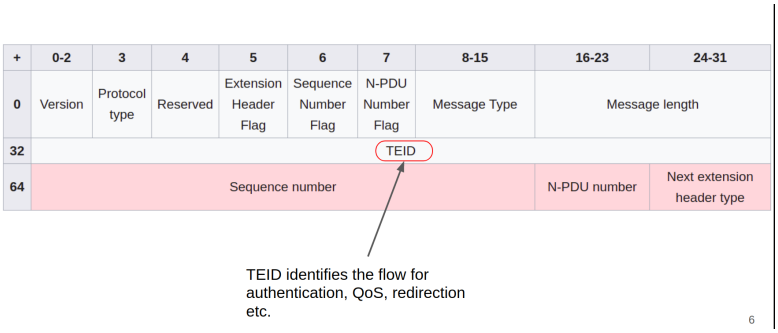


Figure 1.2: GTP-U header [1]

- **Tunnel Endpoint Identifier (TEID)** identifies the packet with a particular session which may have different Quality of Service parameters.
- **Extension Headers** These are required for providing differentiated services to a given packet in the same or different session.

Note that this refers to the user plane packet. This protocol is also called as GTP-U protocol. GTP-C protocol meant for control plane messages is not discussed here.

1.3 Organization

The chapter 2 discusses the architecture of User Plane function's implementation. Chapter 3 discusses the hardware based redirection of the packets to different cores. The issues with the standard RSS and alternative solutions are also discussed. Chapter 4 discusses the different models of execution used in the implementation of the UPF. Chapter 5 discusses the experiments performed and results obtained.

Chapter 2

UPF Architecture

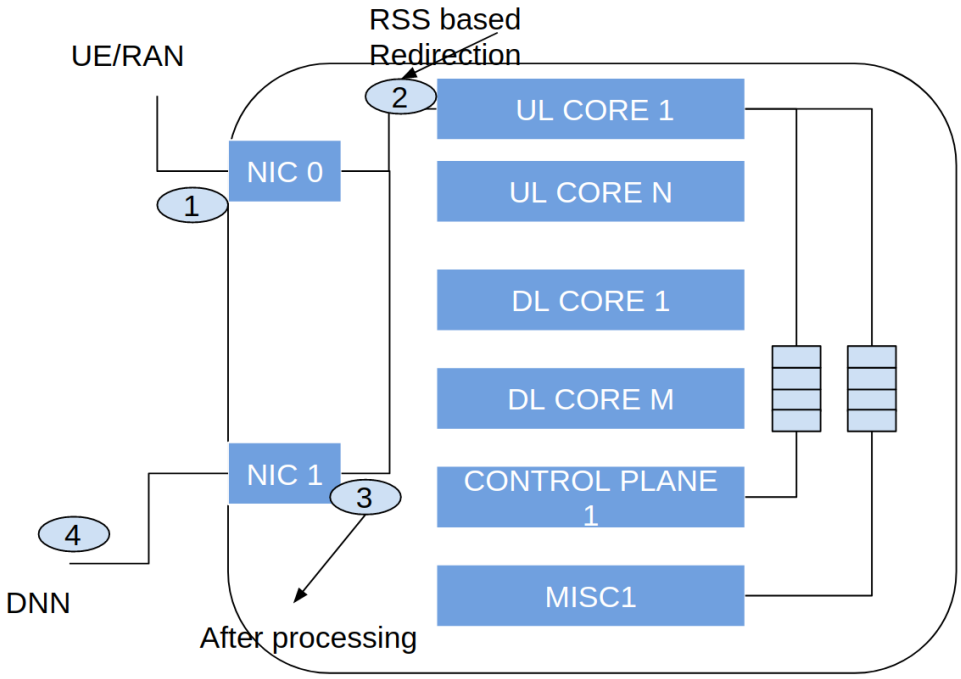


Figure 2.1: UPF Architecture

- **Connections** The UPF server is connected to Radio Access Network (RAN) on Port 0 and to the Internet through Data Network Name (DNN) on Port 1.
- **Uplink Cores** Uplink cores are the cores dedicated for handling uplink traffic.
- **Downlink Cores** Downlink cores are the cores dedicated for handling downlink traffic.

- **Control Plane Core** This core handles the session establishment messages coming from the Session Management Function (SMF). These messages arrive on port 0 and are redirected to uplink cores. Uplink core transfer these messages to control plane core through a circular queue.

- **Miscellaneous Core** This core is responsible for sending heartbeat messages and for handling ARP requests for MAC address (as raw packets are received in user space and no kernel-level functions available).

Figure 2.1 shows the UPF architecture and packet flow in uplink direction.

Chapter 3

Receive Side Scaling

3.1 What is Receive Side Scaling or RSS?

The multiple cores present on the current hardware can be used to scale the processing capability of a single node. A packet arriving on the NIC is redirected to different cores for independent processing. This redirection may occur either in software or hardware.

- **Software** A master core receives all the packets and redirects it to multiple worker cores. This redirection can happen in round robin or hash computation with header fields as key for the hash function (see Figure 4.2). The hash computation technique is preferred as packets of the same flow go to the same core giving better memory locality. This model is referred to as the pipeline model of execution.

- **Hardware (on the NIC)** The hash computation on header fields happen on the NIC itself (see Figure 4.1). This is known as Receive Side Scaling (RSS). The rest of the packet processing happens in software. The RSS computation in the context of 5G UPF uplink packet handling will be discussed.

These alternative models of execution are discussed in Chapter 4.

3.2 Standard RSS

The RSS hash for an incoming packet is computed on the transport and network layer headers. In the case of IPv4-UDP packet, the input to the hash function are IP and UDP headers. The main fields in the headers for hash computation for a GTP encpasulated packet are

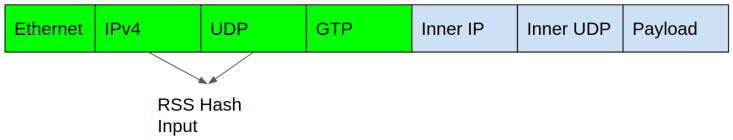


Figure 3.1: Standard RSS input Fields

- **Outer Source IP** is the RAN IP - same for all the uplink packets.
- **Outer Source Port** In the earlier setup, this port was randomized to get unique hash key for RSS computation. This is not a good idea as source ports are not in the control of the UPF.
- **Outer Destination IP** IP of the UPF - same for all the uplink packets.
- **Outer Destination Port** is 2152 - same for all the uplink packets.

So the RSS hash key computation based on these headers will not work in our context.

3.3 Alternatives

The following alternatives were explored:

- **Flow Director** The 40Gbps Intel X710 NIC provides an option of inserting flow rules in the hardware itself (see [2]). These flow rules can be used to match different fields (including the inner payload) of the packet to make forwarding decisions. It is a better alternative than processing headers in software. However, there are two major limitations found with this approach.

- **Limited number of entries** Only 8k entries can be made in the flow table. The 5G use case requires us to handle UE sessions which are much larger in number. The experiments are performed for a maximum of 65536 live UE sessions.
- **No wildcards** Wildcard matching is not supported by the NIC. The number of flow rules could have been reduced drastically with mapping a range of UEs to a fixed core.

Due to the above limitations, the idea of using flow director was dropped. The 40Gbps NICs may support these features in future.

- **Dynamic Device Personalization for RSS** This feature provides the capability to parse fields of inner packet in a GTP-U packet for hash computation. This feature was exploited to get hardware based redirection and is explained in further sections.

3.4 Dynamic Device Personalization

Intel 40 Gbps NIC (XL710) provides the capablility to customize RSS hash key input fields [3]. RSS hash is still computed in hardware. However the fields parsed for RSS hash computation includes

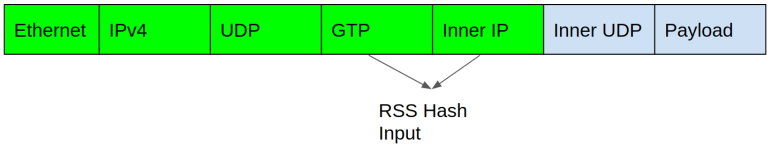


Figure 3.2: DDP RSS input Fields

- **Inner Packet IP fields** Every UE has different source IP. It may not be unique for multiple sessions by the same UE. Still it has good randomness.
- **Tunnel ID in GTP header** This field is unique for every session.

So a combination of the above two tuples generates enough randomness for redirection of packets on different cores. As the name suggests, this feature/profile can

be dynamically loaded and removed from the NIC during the execution of the program. There are no permanent changes made to the NIC configuration. Note that this feature can also be used with Flow Director (discussed in section 3.3) which was not used because of the reasons mentioned.

3.5 Limitations of Dynamic Device Personalization

This feature requires exact matching of all the headers for hash computation. If a field does not follow the protocol format, the packets are directed to the first queue configured on the NIC.

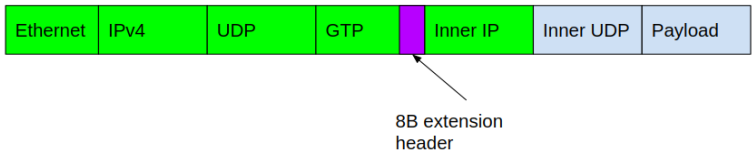


Figure 3.3: Extension Header Limitation

The extension header is used in our case for providing QoS aggregate maximum bit rate (AMBR) that is allowed per session. The parsing of the GTP packet fails due to the presence of the extension header and all packets are directed to the first queue. This is the limitation of the hardware and may be addressed in future by programmable switches or customized DDP feature prvided by hardware vendor (Intel in this case). The experiments performed during this project are not providing different AMBRs for different sessions. AMBR is set above the incoming rate. As a workaround, no extension header is used and AMBR is implicitly assumed. The pipeline model does not have this limitation and the extension headers are used.

Chapter 4

Models of Execution

Run-to-Completion and Pipeline are two models of execution for packet processing and forwarding at User Plane Function. Comparison of these models in the different use cases/scenarios is one of the major objectives of our work. This chapter will explain these models of execution.

4.1 Run-to-Completion (RTC)

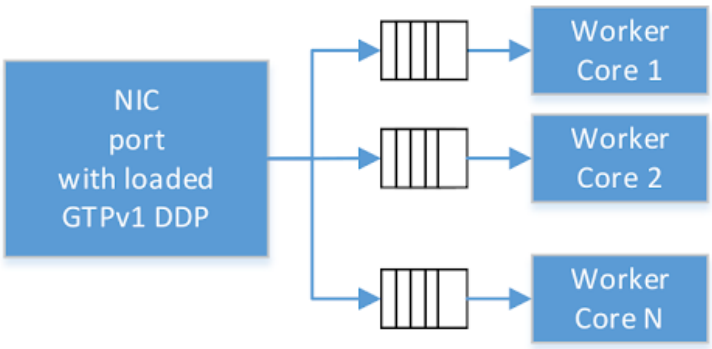


Figure 4.1: Run-to-Completion

Keeping processors independent with minimal or no communication between them is the key idea of this model. Each processor should be able to completely recieve, process and forward/transmit the packet if required. Redirection of packets on different core requires receive side scaling (RSS) discussed in chapter 3. Packets of a session are redirected to the same core as RSS value for packets of the same session is same.

4.2 Pipeline

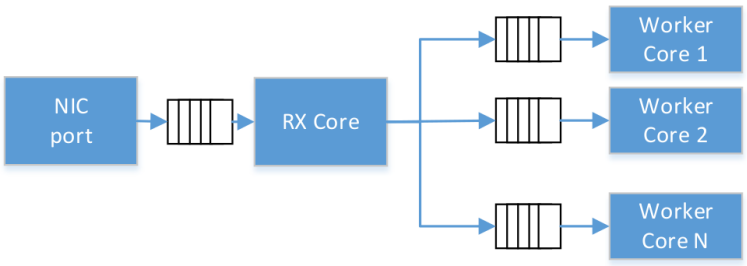


Figure 4.2: Pipeline

Hardware based redirection is either not possible or not used in this model. Complete packet processing including redirection occurs in software. Packets are redirected to a master core which processes the outer headers to redirect packet on one of the worker cores. Generally packets of a session are redirected to the same core to maintain better spatial and temporal locality. However packets may be redistributed if a particular core gets overloaded. It requires inter-core communication. This communication happens through a software ring/circular queue for each worker core. The master core keeps incoming packets on the corresponding software ring of the worker core. The processed packets are then received by master core (through another ring) to transmit them over the wire. The pipeline model can also redistribute load by shifting load from high load core to least loaded loead. Our pipeline implementation handles this by mapping UEs from heavy loaded core to lightly loaded core. The UEs are remapped as long as there is no uniform distribution of load among different cores.

4.3 Comparison

The RTC has some significant advantages over the pipeline model.

- **Better Throughput** The per core processing capability is expected to be higher in RTC model. This is due to no inter core communication and hardware support for hash computation. The computation in the hardware is generally faster than in software. This also keeps CPUs meant for hash computation free for other tasks.

- **Lower Latency**The end to end latency in pipeline also takes a hit due to intercore communication between master and worker core.

- **Easy to implement and understand** The RTC implementation's source code is easy to understand as it is same on all the cores. Pipeline has different logic and complex interaction between master and worker cores.

However, pipeline is better suited to change operational behavior dynamically. This hypothesis is tested for two use cases.

- **Dynamic Scaling** The key idea here is to start user plane function with minimum number of cores and vary the number of cores according to the current traffic load. This helps in better energy efficiency as the cores may remain idle when there is less traffic. It is easy to reconfigure the number of cores in pipeline than RTC. RTC requires stopping of ports before reconfiguring them as the number of RSS queues needs to be updated in hardware. A large number of packets are dropped during this reconfiguration. Pipeline requires the launch of cores in software which is relatively light weight.

- **Redistribution of traffic** Once flows are mapped to cores by hash output in RTC, it is not possible to redirect them to another core. This may be required when the load is highly skewed towards some cores and the rest of the cores are idle. In such cases, a large number of packets are dropped by heavily loaded core in RTC model. Pipeline can effectively handle such scenarios by redirecting certain flows from heavily loaded cores to less loaded cores. This reduces the number of packets dropped and better core utilisation.

Experiments (discussed in chapter 5) were performed to test and evaluate these claims in 5G use case context.

Chapter 5

Experiments and Results

5.1 Experimental Setup

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

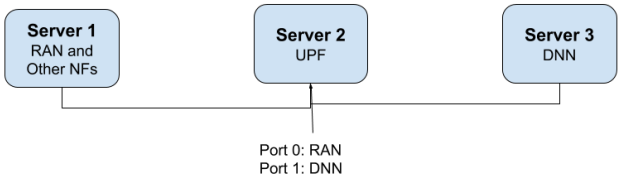


Figure 5.1: Experimental Setup

- **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). The UPF architecture is discussed in detail in Chapter 2. Two ports on this server are used to communicate with RAN (server 1) and data network name (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. The use of superpages reduces the number of TLB misses. Fragmentation due to large page sizes is dealt by DPDK libraries (see [4]) internally while allocating memory buffers storing the incoming packets.
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** Intel XL710 NIC controller wih QSFP+ cables that can handle upto 40 Gbps traffic.

5.2 Experiments

After deploying Dynamic Device Personalization feature (discussed in Chapter 3) in the RTC model, the pipeline and RTC models were compared on the parameters described in the section 4.3. The main questions addressed are

1. How does the transmit throughput for a single core vary with **increasing packet sizes** in both uplink and downlink directions?

2. How does the transmit throughput for a single core vary with the increase in **number of active UEs/sessions** ?

3. How does throughput scale with the **number of cores**?

4. **Skewed Traffic** How do Pipeline and RTC handle skewed traffic with a few UEs as heavy hitters?

5. **Dynamic Balancing** What is the impact on latency and packet drop rate or throughput during the transient phase of dynamic increase in number of worker cores to handle the increasing traffic?

5.2.1 Throughput v/s. Payload Size

A single core is active in both uplink and downlink directions to handle incoming traffic. The load generated at RAN/DNN is more than what a single core can handle and the actual transmit throughput is measured.

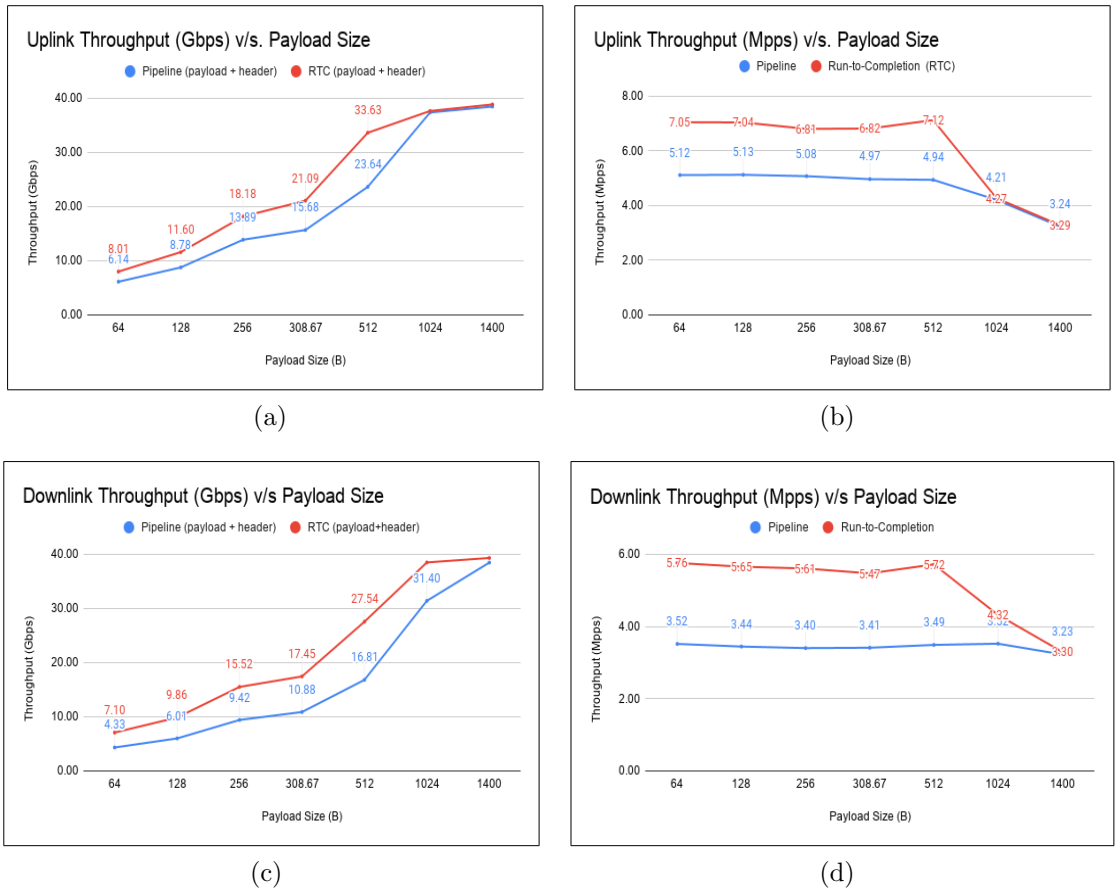
It is expected that the number of packets processed per second should remain constant irrespective of packet size as header processing is the main task and size of the header is independent of packet size.

Results

Figure 5.2 shows the results. The measurements are made in both uplink and downlink direction for different payload sizes. Each run is of the duration of 150 seconds. The number of sessions maintained in the UPF is equal to 100000. A total of 65536 UEs are kept active while making measurements. It was observed that there was negligible variation in the packets processed per second when readings were taken every second. The main results are

- Packets processed per second is independent of the packet size. The packets processed per second for 1024 Byte packet decreases due to the saturation of link/NIC.

- RTC gives 30-40% higher throughput than pipeline for different payload sizes as expected (see also section 4.3).



(a) (b)

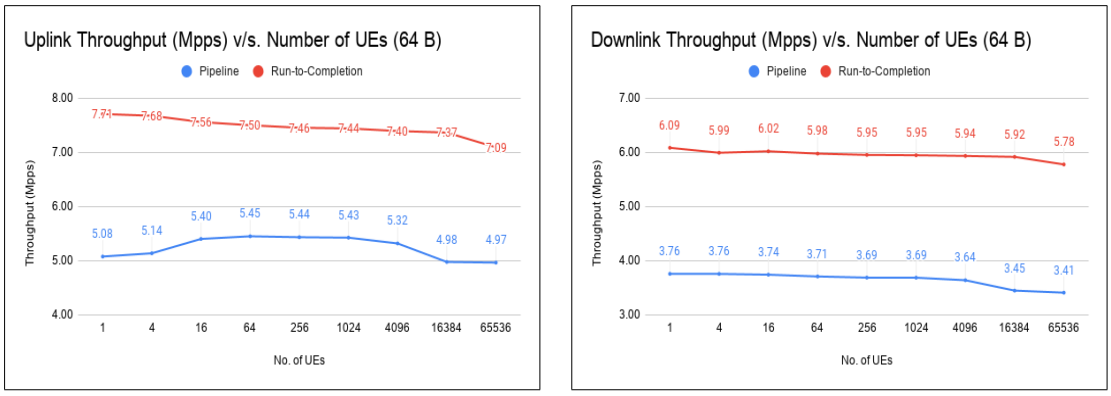
(c) (d)

Figure 5.2: Throughput v/s Payload Size. Throughput calculation in Gbps includes header size.

- Uplink performance is better than downlink performance. This is because of **GTP encapsulation overhead** in downlink direction.

5.2.2 Throughput v/s. Number of active sessions/UEs

The experiment with same conditions as section 5.2.1 was conducted. However the payload size is kept fixed at 64 Bytes and the number of active UEs are varied. Performance degradation is expected with the increasing number of active sessions due to the increase in lookup time. A substantial degradation is expected when the data structure is poorly chosen or due to poor implementation. Per core hashmap is used in our implementation.



(a) (b)

Figure 5.3: Throughput (in Mpps) v/s number of active UEs.

Results

There is slight/negligible degradation of performance for RTC model with the increasing number of sessions. Downlink throughput decreases for pipeline model with increase in number of sessions. No discernible pattern is observed for pipeline uplink.

5.2.3 Scaling of throughput with number of cores

The throughput is measured for 64 Byte payload and IMIX traffic (see [5]). The results are illustrated in Figure 5.4 and Figure 5.5. The IMIX traffic models the standard internet traffic. The average payload size for IMIX distribution comes out to be 309 bytes. Each run is of the duration 150 seconds. The number of

active sessions are 65536. The load generated at RAN/DNN exceeds the processing capacity of the UPF and the link is always saturated. The number of cores are varied across the runs to measure the total transmit throughput of the UPF in both uplink and downlink direction.

RTC is expected to take fewer cores to saturate the transmit throughput than pipeline as per core throughput is higher for RTC (see section 5.2.1). The number of cores required to saturate a given load in Gbps increases with decreasing payload size because the number of incoming packets increases with the decreasing payload size.

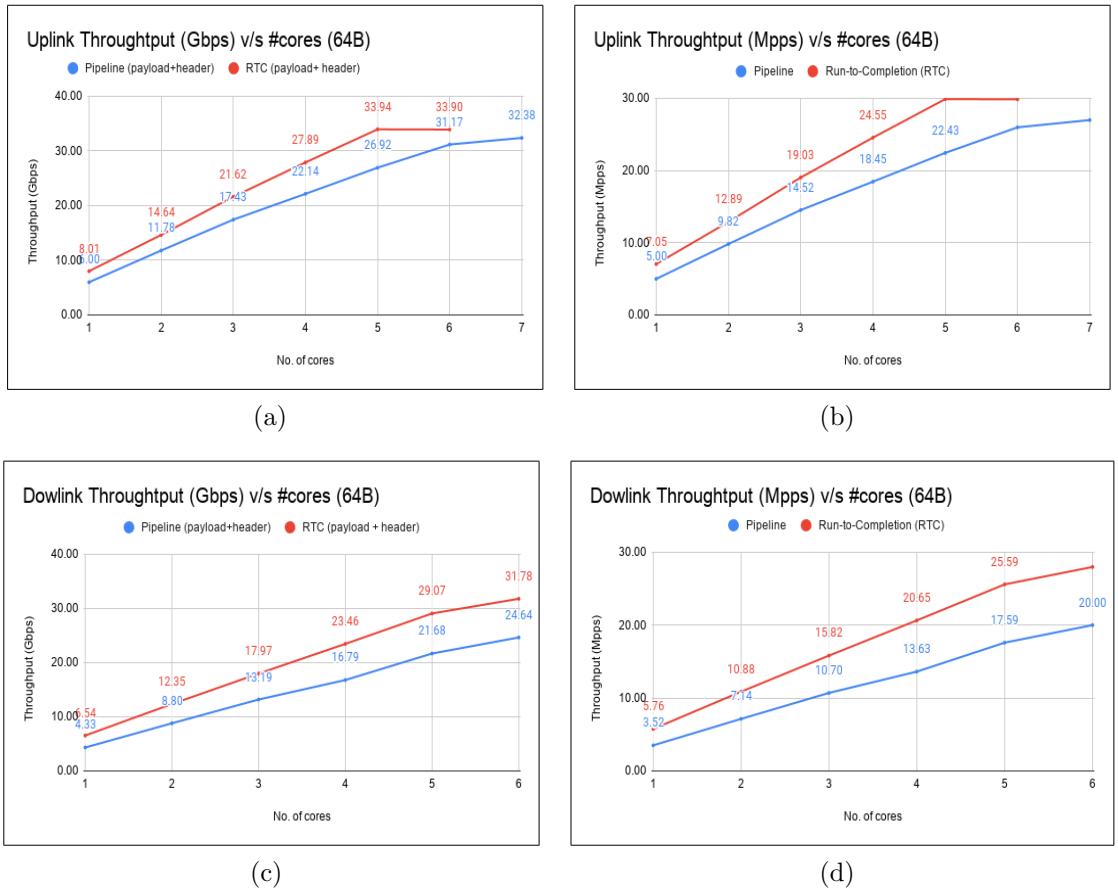


Figure 5.4: Throughput v/s Number of Cores (64 Byte payload). Throughput calculation in Gbps includes header size.

Results

- For smaller sized packets (64 B in our case), the number of packets to be processed is quite large. Uplink RTC throughput is saturated in 5 cores.

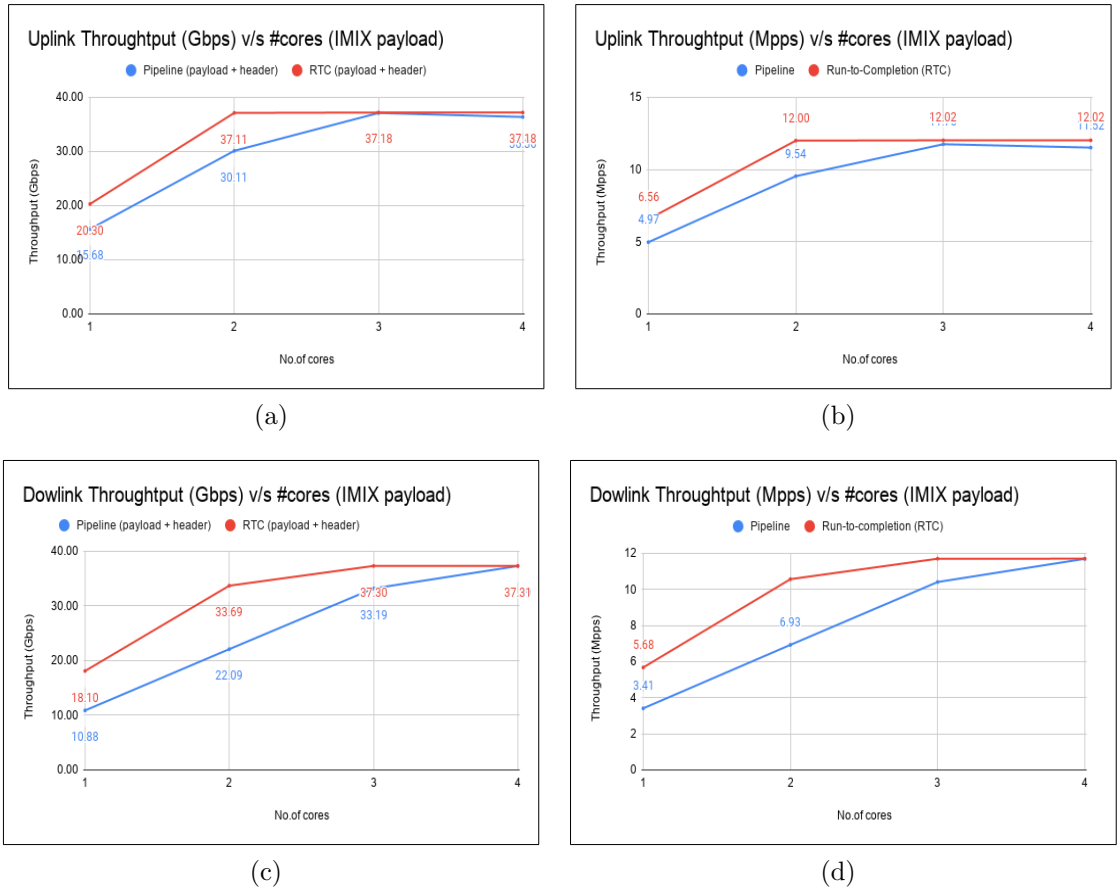


Figure 5.5: Throughput v/s Number of cores (IMIX payload). Throughput calculation in Gbps includes header size.

Uplink Pipeline is not saturated in even 6 cores. Downlink RTC throughput is saturated in 6 cores. Downlink Pipeline is not saturated.

- For IMIX Uplink, RTC saturated in 2 cores. Pipeline saturated in 3 cores. For IMIX Downlink, RTC saturates in 3 cores. Pipeline saturates in 4 cores.

The difference in uplink and downlink perfomance is due to the GTP encapsulation overhead in the downlink direction.

5.2.4 Handling Skewed Traffic

The section 4.3 discusses the limitations of RTC model in handling skewed traffic. As the packets are redirected in hardware, RTC model has no flexibilty to redirect traffic from one core to another. The pipeline model can redistribute traffic (see Section 4.2).

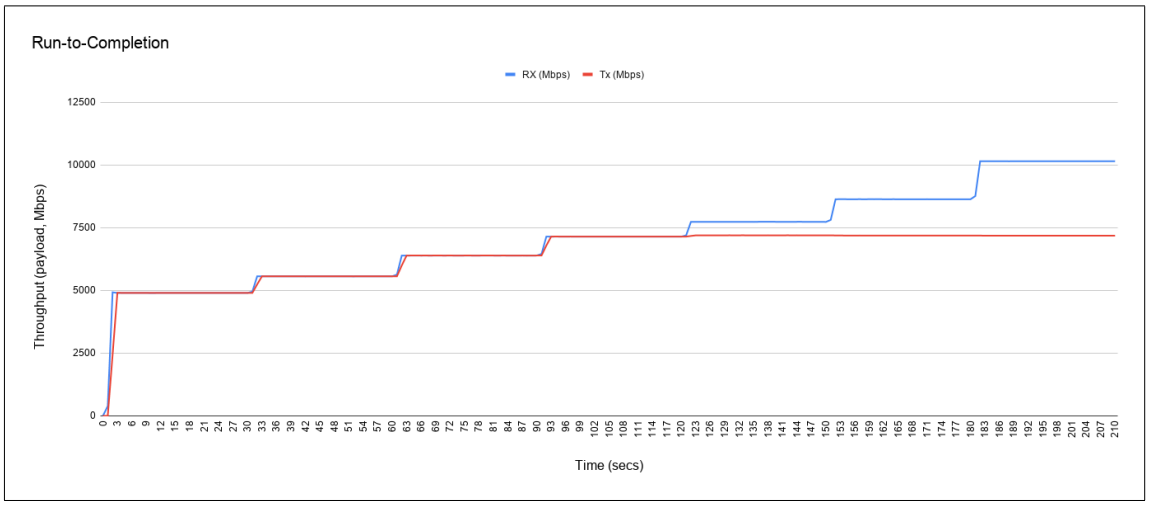
Experimental Setup

The number of worker cores is kept fixed at 4 in both the models (excluding master core in pipeline model). Initially a uniform load is applied on all the worker cores. The load on a single UPF worker core is increased in steps while load on the remaining cores is kept fixed. As RSS based packet redirection is random and uniform across cores, it is not possible to know apriori which session will be mapped to which core. Initially , packets were sent from 1024 different UE sessions and IP-Session ID to UPF core mapping was precomputed and stored in a file. This mapping was used to send packets on differential rate to the different UPF cores. For overloading a particular UPF core, the inter packet delay was reduced for the packets coming from RAN side. The rate limiting APIs were not available for the 40 Gbps NIC. Sessions-IP were range partitioned for pipeline model.

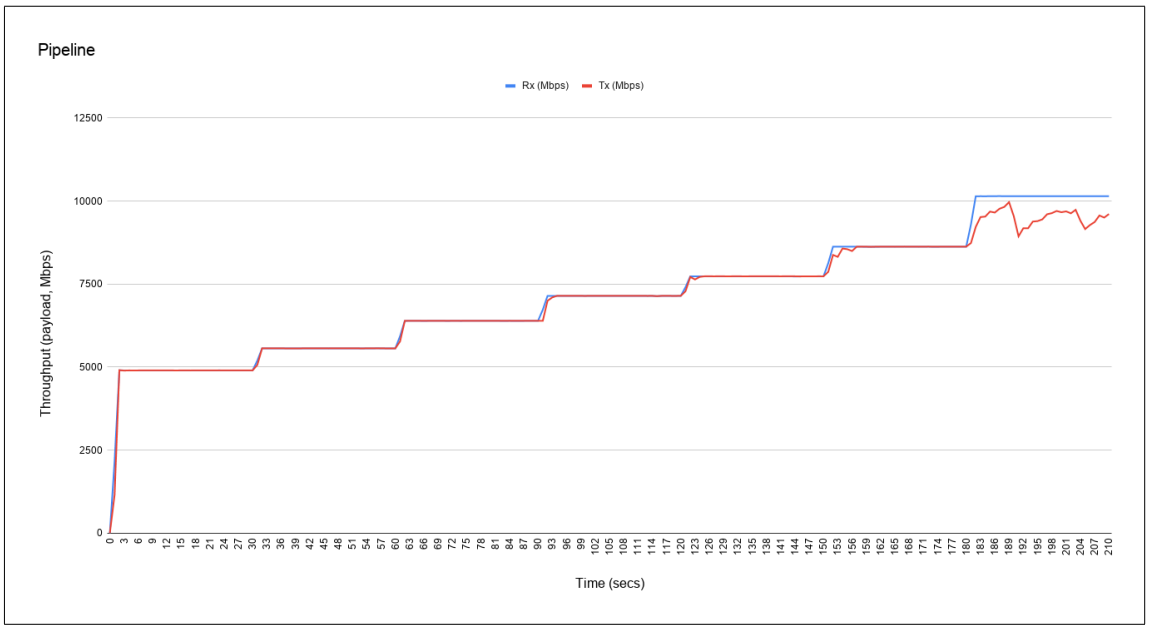
Results

The results are illustrated in Figure 5.6.

- **RTC** Beyond the processing capability of a core, the increase in incoming traffic has no effect on the transmit throughput.The packets are dropped thereafter.



(a)



(b)

Figure 5.6: Handling of skewed UE traffic in pipeline and RTC models

- **Pipeline** Pipeline handles the increasing load quite well by remapping some of the UEs to least loaded core. However, it can be observed that pipeline Tx-throughput does not match the RX-throughput on the last step increase. This is because the total processing capability of all the worker cores is reached.

5.2.5 Dynamic Scaling

The section 4.3 discusses why dynamic scaling is a useful concept for energy efficiency and how pipeline may effectively handle dynamic reconfiguration with dropping of fewer packets and insignifcant impact on latency.

Experimental Setup

The maximum number of cores are kept at 4. Initially a single worker core is started (and one master core for pipeline). The payload size is kept at 64 Bytes. The experiment is run for 300 seconds. The load is increased in step size of 5 Gbps (payload + header) every 60 seconds. The overload detection algorithm for different models is

- **RTC** The idea of exponential weighted moving average (EWMA) (see [6]) was used for calculating core load factor. When the core load factor reaches above a threshold, port is stopped, hardware queues and RSS hash is reconfigured, the port is restarted and a new core is launched for forwarding of packets. The core load factor is calculated after every polling event.

  $$averageCLF = 0.999 * averageCLF + 0.0001 * (numberofpacketsReceived)/32$$

  A maximum of 32 packets can be received on every polling event. 0.001 is the weight assigned to the current load factor. EWMA is resilient to transient overload and is influenced by both the current and previous value of the load factor.

- **Pipeline** The EWMA technique was not feasible for Pipeline model. The software rings were used for communication between master and worker cores. A function call was required for calculating the existing number of elements in

the ring. The performance is degraded substantially if this function is called after every polling event on master core. As a workaround, core load factor was sampled every millionth packet. If the core overload is detected for a certain number of consecutive time, the core is deemed to be overloaded and a new core is launched.

Results

The results are illustrated in Figures 5.7 and 5.8.

- **RTC**: Figure 5.7(a) shows the entire run of the experiments for pipeline model. The spike in latency and sharp drop in throughput is observed whenever a new core is launched (see Figure 5.7(b)). The maximum latency observed during the reconfiguration is around 100 ms. The reconfiguration time is around 400 ms.

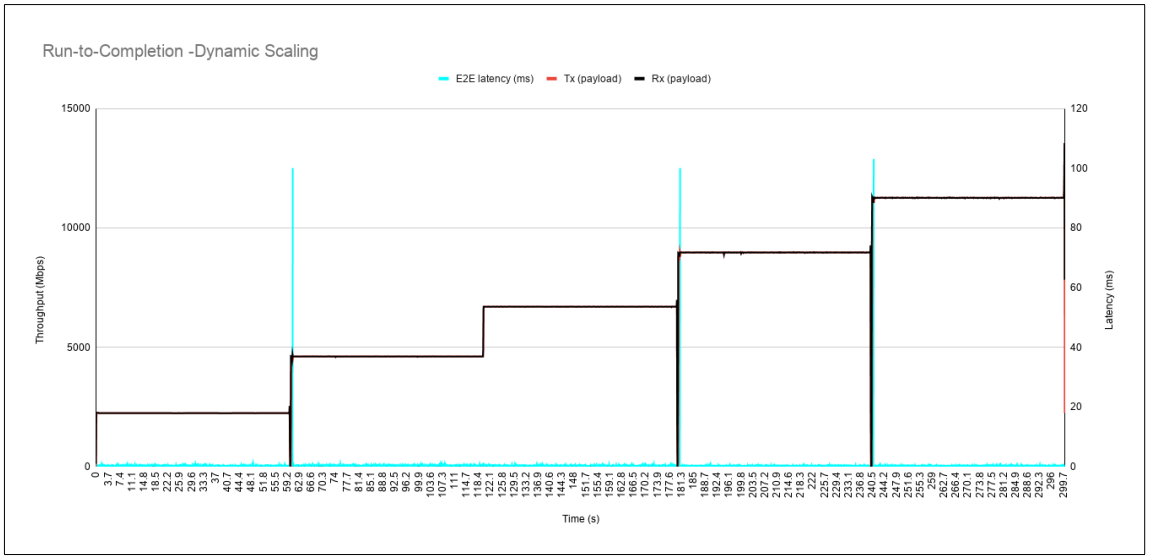
- **Pipeline**: Figure 5.8(a) shows the entire run of the experiments. No appreciable change is observed in throughput or latency (see Figure 5.8(b)). The reconfiguration time is around 130 us.

5.3 Summary

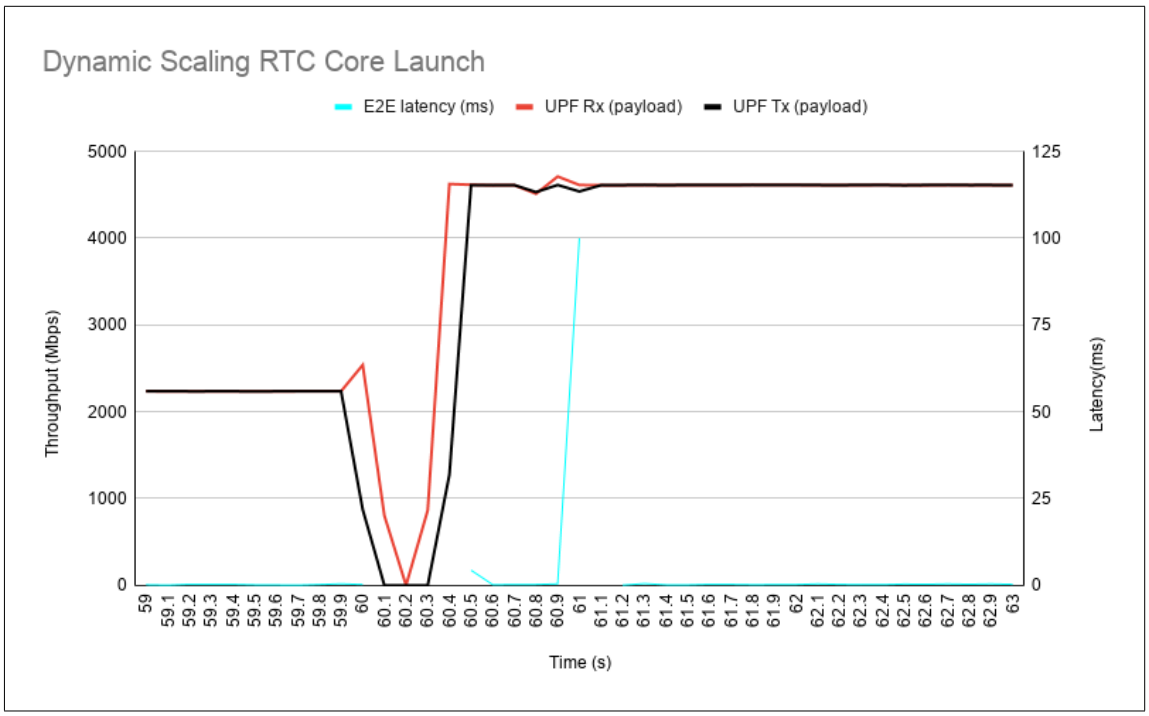
The main question explored in these experiments was to define and evaluate different scenarios in which one model of execution is better than the other among the Pipeline and RTC models (also see section 4.3). The effect of the number of cores and the increase in the number of active sessions on throughput was also observed. The important findings are

- RTC gives better per core throughput (30-40% higher) than Pipeline.

- Throughput scales linearly with the number of cores. RTC gets saturated earlier because of higher per core throughpu Throughput scales linearly with the number of cores. RTC gets saturated earlier because of higher per core throughput.

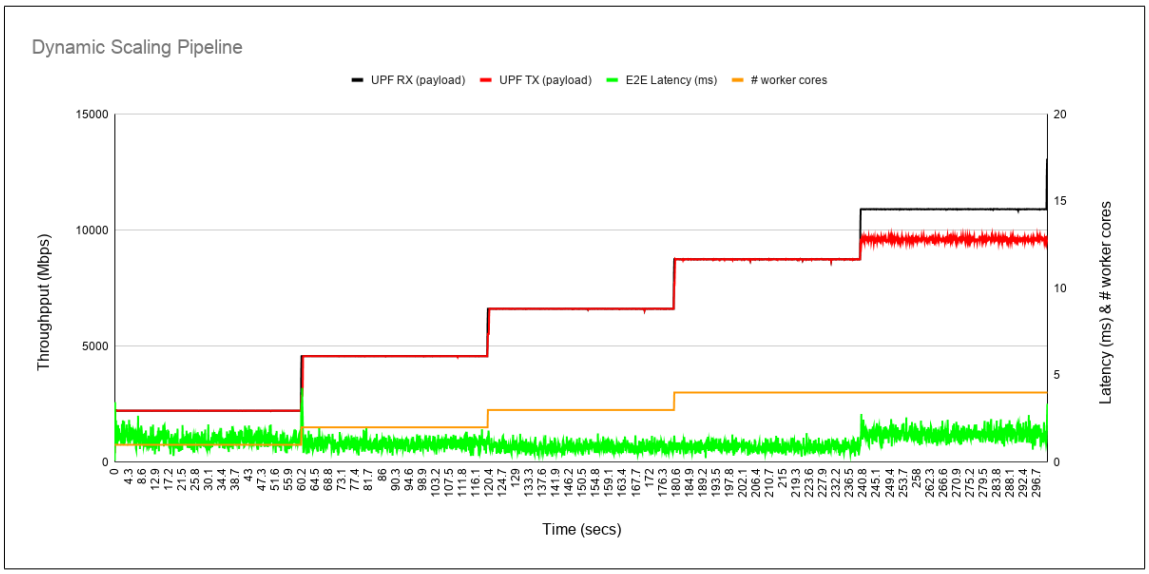


(a)

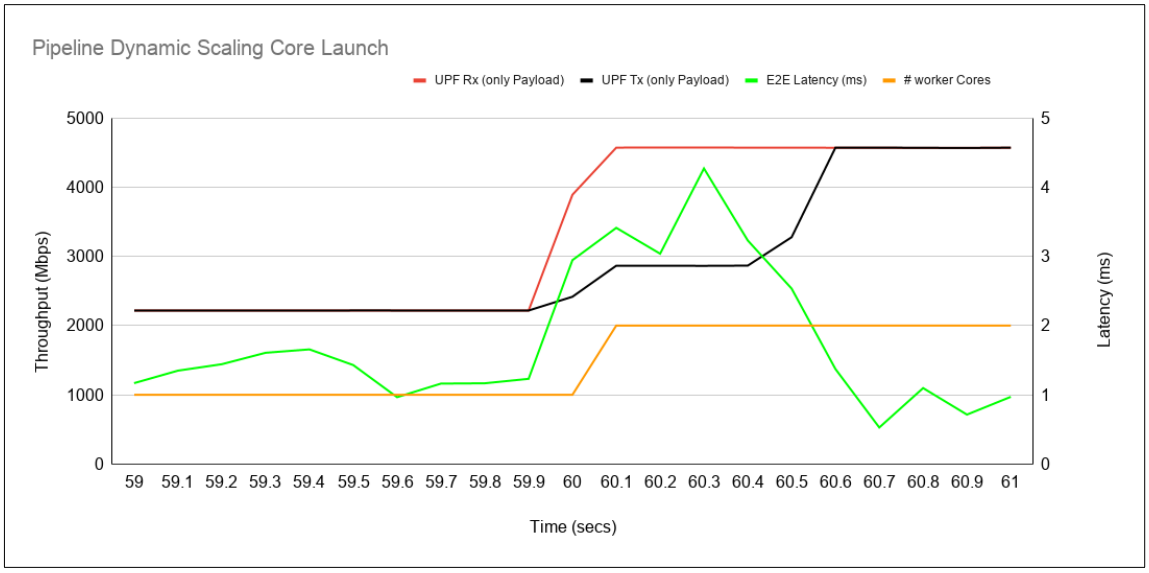


(b)

Figure 5.7: RTC: Figure (a) shows throughput and end to end latency during the whole run. Figure (b) shows the output during one of the core launches (the first one).



(a)



(b)

Figure 5.8: Pipeline: Figure (a) shows throughput and end to end latency during the whole run. Figure (b) shows the output during one of the core launches (the first one).

- The average throughput was not affected significantly by the increase in number of sessions.
- Pipeline model can handle skewed traffic effectively by redistribution of the traffic.
- The number of packets dropped and the impact on latency is quite low for pipeline model while dynamically increasing the number of cores to handle the higher incoming traffic.

Bibliography

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

[2] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/intel-ethernet-flow-director.pdf/.

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

[4] https://www.dpdk.org/blog/2019/08/21/memory-in-dpdk-part-1-general-concepts/.

[5] https://en.wikipedia.org/wiki/Internet_Mix.

[6] https://en.wikipedia.org/wiki/Moving_average.