Scaling Telecom Core Network Functions in Public Cloud Infrastructure

Dinesh Kumar
dinesh.kumar@intel.com

Somnath Chakrabarti
somnath.chakrabarti@intel.com

Ashok Sunder Rajan
ashok.sunder.rajan@intel.com

Jim Huang
jimhuan@amazon.com

***Abstract*—Telecommunication networks, especially 5G, typify high performance network infrastructure with very high data rates and throughput. Traditionally, high data rates meant line rate in telecom networks. As link capacities increase to 100 Gbit/s, 64byte packet rates will need to reach 150 MPPS. Technologies that attempt to deliver this high packet rate take liberties with proprietary hardware and application-enhanced networking stacks, pushing complexity and cost. Our approach challenges the need to deliver 64byte packet at line rate and instead addresses delivering good enough packet processing capacity with conventional networking stacks natively in the Public Cloud. We explore the deployment of operational telecom core network packet processing on Public Cloud infrastructure and empirically demonstrate this is in fact completely feasible. Using compute and networking available on Public Cloud as is, we realize an end-to-end operational telecom packet core delivering over 2 MPPS, ~20Gbps for 250K users, representing the network load of a Mobile Edge site. This paper re-thinks telecom core network capacity planning, addresses scale-up vs. scale out thresholds, details leveraging of Linux kernel advancements for compute and I/O efficiencies and provides a reference architecture for deployment of telecom core on Public Cloud.**

Keywords— Telecom, 5G, Public Cloud, Evolved Packet Core, Performance, Scalability, Testing

I. INTRODUCTION

The fundamental motivation of Network Function Virtualization (NFV) [1] is to have telecom Core Network Functions run on commodity servers, in the same manner as other large-scale applications. Being able to deliver telecom network functions on commodity servers meant transformative reduction of Total Cost of Ownership (TCO) for telecom operators. Traditionally core network functions were realized on customized hardware highly optimized to deliver packet processing at the line rate achievable by the physical links at maximum throughput. This approach leads to specialized hardware for each of the network functions increasing cost and complexity, restricting the benefits of commodization.

Our earlier research and engineering efforts resulted in the open sourcing of a comprehensive set of telecom core network functions at the Open Mobile Evolved Core (OMEC) [2], a Linux Foundation project. The high-performance packet processing core functions were deployed on production networks [3] using commodity servers. This success of deploying core network high performance packet processing stacks we had written on production networks established that packet processing does not need to happen at the line rate achievable by the physical link [4, 5, 6].

In this paper, we re-think capacity planning of telecom core networks for deployment in Public Cloud. The conventional capacity planning methods are based on number of users per cell tower, number of cell towers connected to the core network and the total number of users that are authorized for admittance to the network in a deployment region. The required network capacity is estimated in Erlangs, a statistical unit of capacity utilization [7], to maximize link capacity utilization. We propose telecom core network capacity planning based on cloud infrastructure resources, technologies and services, in addition to maximizing link capacity for massive 5G core scale out.

We aim at Public Cloud to deliver the scale and capacity of high-performance core networks with commodity servers, elasticity, high deployment velocity, and economy of scale.

We deploy an end-to-end 5G core network data path, also called the User Plane, at operational scale on Amazon Web Services (AWS), develop and experiment methods for 5G infrastructure capacity planning and performance scaling. This paper focuses on the following topics.

- We layout a reference architecture for realizing capacity scaling of 5G core network on AWS Cloud infrastructure end to end. (Section III)
- We form an approach to scaling up and scaling out the User Plane and performance capacity respectively. We develop methods for scaling up the User Plane performance and obtain insights in networking and compute performance alignment and implications. We define a threshold for packet processing performance per core beyond which we propose scaling out units of compute and networking capacity. (Section IV)
- Through analyzing packet processing mechanisms and code, we compare SR-IOV technology with native Linux kernel features that enable pre-filtering and targeting flows at a set of packet processing application cores utilizing device queues. (Section V)
- We detail multiple experimental setups to profile scaling up and scaling out of packet processing capacity at operational scale on AWS Cloud and a lab network infrastructure. Our empirical results demonstrate that it is feasible to deliver the 5G core network on AWS Cloud with a minimum of 2 MPPS, ~20Gbps for 250K users, which represents the network load of a Mobile Edge site [3, 4]. (Sections VI & VII)

The contributions of this paper are:

- Through empirical study with mobile traffic workloads, production-grade core network infrastructure stacks, and AWS Cloud resources, we establish that performance-efficient packet processing capacity for operational telecom infrastructure can be realized using

2330-2186/20/$31.00 ©2020 IEEE
DOI 10.1109/CloudCom49646.2020.00006

native Linux networking stacks. We develop a simplified VNF software approach that capitalizes on native Linux in Public Cloud infrastructure away from hardware-specific or proprietary solutions.

- We develop a reference architecture for deploying telecom core network infrastructure on AWS Cloud. The architecture supports telecom application use cases and core network performance optimization from telecom on-premises edge to the cloud region, end to end. The telecom core network infrastructure is modularized in the public cloud so that telecom operators will be able to deploy Virtualized Network Functions (VNFs) on Virtual Machines (VM) right-sized by our scale-up method within performance-efficiency thresholds.

II. Background and Related Work

A. Background

Deployed 4G core network infrastructure in the United States spans about 250K cell towers across the country [8] that connect approximately 500M devices to the Internet backbone and telecom services. These 250K cell towers connect to roughly 150 switching centers that land the packet processing load on about 50 Evolved Packet Core (EPC) processing functions.

5G workload projects a 100x increase in the number of cell towers on account of a 10x improvement in spectral efficiency and 10x increase in spatial density [9]. This means that with 5G the number of cell towers will grow from the current 250K to about 25M cell towers. The number of devices per cell tower will be lower by an order of 10, with each 5G cell supporting approximately 200 devices compared to the roughly 2000 devices supported by the existing 4G cell sites. The number of EPC sites will need to increase 100x to 5000. The reduced coverage of the 5G cell sites will however push wider geographical distribution of the packet processing core. The challenge of 5G core networks is therefore to realize a 100x increase not just by increasing the number of physical packet processing locations, but in having these functions distributed closer to the cell sites.

The table below compares current 4G topology with 5G.

Infrastrucutre	#of devices	# of Cell Towers	#of Switching Centers	# of EPC sites	# of devices/EPC site
4G (current)	500M	250K	150	50	10M
5G	5Bn	25M	15K	5K	1M

Table 1: Telecom Core Network Workload

Traditionally telecom core network capacity is dimensioned as the probability of availability of a circuit i.e. Erlang. Given a 4G device can sustain a data rate of 20Mbps [10], arithmetically a cell tower with 2000 devices connected would be required to sustain 40Gbps and each EPC site required to handle 200Tbps. Handling this high data rate at the cell tower and even more so at the EPC site is practically impossible. As a reference, the total backhaul capacity in the United State is ~25Tbps [11].

Telecom network capacity planning employs statistical methods to dimension offered network capacity based on estimated capacity required at each cell site [12]. The statistical planning of the network capacity allows available capacity to be orders lower at around 500Gbps.

The need for statistical planning of capacity is driven by cost of link, requiring the entire link capacity to be used. This translated to the need to have the packet processing functions at the network core perform at line rate of the physical link.

With 5G, the traffic aggregation at each segment of the network from the device to the packet processing core changes, challenging the traditional network capacity planning methods.

The scope of this paper is on performance scaling of the core network user plane. Figure 1 provides a high-level view of a typical telecom core network packet processing VNF or User Plane Function (UPF). The cell towers connect to the S1U network interface (Uplink) of the UPF. The SGI interface of the UPF (Downlink) connects to services delivered from within the telecom network or from the Internet.

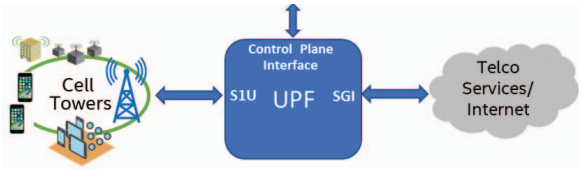

Figure 1: UPF Network Interfaces

B. Related work

Researchers and practitioners have been conducting cloud performance studies. S. Lehrig et al. [13] provides systematic literature review with definitions of key cloud performance concepts such as capacity, scalability, elasticity and efficiency. In [14], Amro Al-Said Ahmad and Peter Andras propose volume scalability and quality scalability metrics in the context of cloud elasticity. A cloud performance study conducted by Foivos Michelinakis et al. [15] aimed at determining the most dominant Cloud Service Providers (CSPs) enabling the mobile Internet and the performance of CSP services when accessed from commercial Mobile Network Operators (MNOs). The authors performed a holistic analysis of the complex ecosystem formed by mobile applications, CSPs, and MNOs, characterized their relationships and dynamics, and measured their performance as perceived by end-users.

Google presents its design and experience with Google Cloud Platform's network virtualization stack, named Andromeda [16]. It consists of Andromeda data plane optimized with Fast Path for packet processing per host and control plane providing scale up and down of compute and rapid provisioning of large numbers of VM hosts. A notable design principle is a software-oriented data plane to allow a uniform feature set for customers running on heterogeneous hardware with different Network Interface Cards (NICs) and hardware offloads. It states a full SR-IOV hardware approach requires dedicated middleboxes to handle new features or to scale beyond hardware table limits. Such middleboxes increase latency, cost, failure rates, and operational overhead.

While there have been quite a few studies on cloud performance, the uniqueness of this paper lies in several aspects. First, it provides an end-to-end reference architecture for deploying a telecom infrastructure in AWS Cloud. Second, it

develops a method of scaling up the User Plane Function performance with native Linux kernel mechanisms. Third, it shows empirical data from experimentation in the AWS cloud with mobile traffic workloads and OMEC software that has been deployed in telecom production core networks.

III. Telecom Network Core on Public Cloud Infrastructure

The work presented in this paper builds on a reference architecture we developed for deploying and operating Virtualized Network Functions (VNFs) in Public Cloud toward 5G realization. This reference architecture aims at evolving Evolved Packet Core (EPC) [17] toward the 5G system architecture specified by 3GPP [18]. It reflects on several Cloud and mobile infrastructure design principles and methods to address 5G use cases:

- Control plane and User Plane Separation (CUPS) [19] to scale the control plane and user plane resources independently.
- Multi-access Edge Computing (MEC) [20] to support high-performance applications with low latency (less than tens of milliseconds round trip time) requirements
- Resource rightsizing for performance scale up
- Resource scaling for performance scale out
- Automation of the telecom infrastructure and VNF deployment in the cloud

A high-level view of the reference architecture is shown in Figure 2.

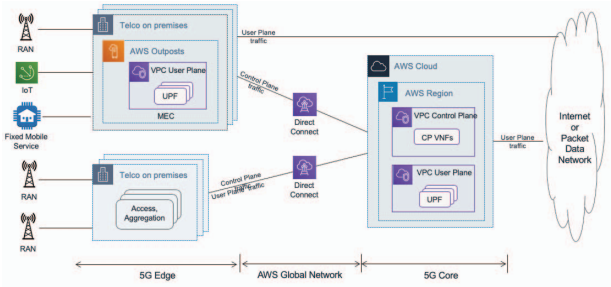

Figure 2: High-level View of the Reference Architecture

The reference architecture consists of AWS Cloud, in a region, and on-premises telecom infrastructure interconnected through AWS Direct Connect links. The AWS cloud region hosts Control Plane and User Plane VNFs and the telecom on-premises may host AWS Outposts which runs additional User Plane Function (UPF).

The reference architecture is designed to support three types of use cases as the EPC evolves toward the eventual 5G architecture.

(1) Both User Plane and Control Plane VNFs run in the public cloud. The user plane and control plane traffic from telecom on premises connect to the Control Plane and User Plane Functions in the AWS cloud region. Traffic from the Internet is processed by the UPF on AWS Cloud. This serves most traffic patterns today.

(2) User plane VNFs run on AWS Outposts on premises and control plane VNFs run on the AWS cloud region. This serves ultra-latency-sensitive applications executing at the 5G network edge.

(3) A hybrid of (1) and (2) where user plane VNFs run on both AWS Outposts at the edge and AWS Cloud to serve all types of user plane traffic.

This reference architecture can be generalized as a public cloud comprised of inter-connected regional infrastructure and local infrastructure on premises, where VNFs are deployed and run in the cloud region and/or cloud on-premises environment.

Our approach to the performance scaling of VNF is end-to-end. First, across the AWS cloud infrastructure, for latency-sensitive or high-throughput applications, telecom operators can use AWS Outposts on premises. Second, when a set of the user plane VNFs run in an AWS cloud region, we allocate the complete VNF set and connecting resources in the same Availability Zone to increase packets per second rate. Third, we deploy VNFs in the Amazon EC2 Placement Group to further minimize the traffic processing latency across VM instances. Finally, we optimize the VNF performance at the VM instance granularity – scaling up VNF performance through benchmarking and rightsizing of EC2 instances and scaling out by increasing the number of EC2 instances to match the load on the network.

With performance scale up and scale out schemes, the reference architecture enables telecom operators to conduct infrastructure capacity planning, select and deploy resources in the cloud or at the edge, optimally. This methodology and empirical results are detailed in the following sections.

To automate the core infrastructure and VNFs deployment, we modularize the core infrastructure and cloud resources such as Amazon VPC, Subnets to host VNFs, AWS Transit Gateway and VPC Peering to interconnect VPCs, Internet Gateway, NAT, and Route Table for routing packets to different destinations. We use AWS CloudFormation templates to implement the reference architecture and deploy a telecom core network infrastructure environment in minutes. Furthermore, we use AWS Machine Image to capture individual VNF EC2 instances that are right sized by the scale up method described in the following sections. We then launch the VNFs through the CloudFormation stack into the core environment.

In the rest of this paper, we focus on our methods, mechanisms, experimentation, and analysis of the scale up and scale out of the VNF performance with UPF for operational realization of the telecom core.

IV. Telecom Core Network Capacity Planning For Cloud Infrastructure

Given that the number of 5G packet processing core sites is 100x that of 4G while the number of devices per cell site is actually 10x lower, we can base the sizing of the 5G core network on the number of devices that need to be supported by a core packet processing site. Taking from the 4G studies on the actual traffic per device [4, 6] we can infer the packet rate per device for different deployment scenarios or markets.

We can therefore have the simple relation below:

$$PPS = N \times p \quad (1)$$

where PPS is packets per second; 'N' is the number of devices in a deployment area and 'p' is the packet per second per device. 'p' will be generally a constant for each market. For example, for Internet of Things (IoT), 'p' will be ~1 packet per second per device. For Smart Phones, 'p' can be as high as 20 [6].

Fixing 'p' for a given market we can then dimension the packet processing capacity on 'N', the total number of devices to be sustained in the given deployment area. 'N' would be proportional to the geographical area.

$$N = k \times A \quad (2)$$

where 'k' is the density of active devices in the deployment area 'A'. For example, higher density of devices in active urban areas vs. low density in smaller towns and communities. We can envisage a finite matrix space of 't' market types and 'r' active device densities:

$$p \subset \{p1, p2, \ldots, pt\} \times k \subset \{k1, k2, \ldots, kr\} \quad (3)$$

We can then build out the core network as modular 'p x k' packet processing capacities. This approach to core network packet processing capacity dimensioning would allow us to right-size compute and networking resources for the lowest value in the 'p x k' matrix and add additional capacity units based on market segment or deployment area density. For example, we define compute and networking resources required for a small city with kc devices per square mile addressing the IoT market delivering 'kc x pi' packets per second as a unit of capacity. We can scale up capacity with additional units of 'kc x pi'.

A. *Scale Up*

There are primarily three stages in the journey of a packet in either Uplink (UL) or Downlink (DL) direction inside the UPF:

- Receive (rx) of the packet from the S1U or SGI network interface
- Processing (c) the packet by looking up applicable rules
- Transmitting (tx) the processed packet out the SGI or S1U interface

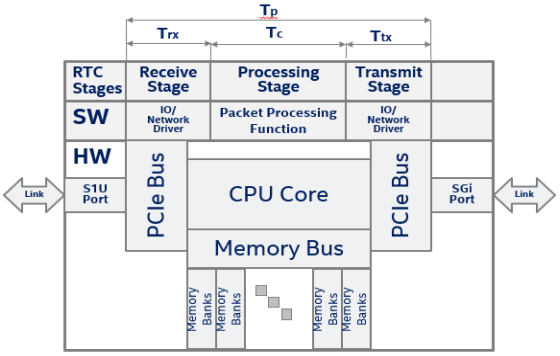

Figure 3: Packet Processing Unit and Stages

Scale up refers to increasing the size of a packet processing unit of compute and networking in the p x k space. The theoretical scale up limit would be the reciprocal of the maximum time per packet, taken by the network, compute and the underlying interconnects in the cloud infrastructure.

Taking the unit of networking to be a single physical port to which a link will connect and a unit of compute to be a CPU Core to which the physical port is bound, the packet processing rate is limited by:

$$\text{Scale-Up limit} = 1 / (Tn + Tc) \quad (4)$$

where 'Tn = Trx + Ttx'; 'Trx' is the time taken by the I/O drivers and the networking stack to make a packet received on the S1U or SGI link available to the CPU core; 'Ttx' is the time to deliver packet out on the link; Tc is the time taken by the application running on the CPU core to process the packet.

$$Tp = Tn + Tc \quad (5)$$

represents the end-to-end packet processing time.

UPFs designed in a Run-To-Complete (RTC) loop, run all three stages on the same CPU core, time-sharing CPU cycles among the three stages above.

A lot of research work, including ours [5, 6] has been focused on reducing 'Tn + Tc'. Generally, the approach has been to have the application processing the packet in an RTC model. The application sidesteps the network driver and pulls the packet directly from the port, making 'Tn' close to zero.

As link speeds increase, 'Tp' or the scale up limit starts to get compute bound. Also, we see from operational requirements in real world deployments, the functions provided by the networking stack need to be replicated within the application, increasing 'Tp'. Furthermore, staying away from arguments on cache coherence, memory bandwidth limits with increasing 'N' [4], we would also see an increase in the flow tables, rules and rate of provisioning by the control plane elements in the core network. Our earlier research examines these limits of rate of rule processing in the core network. [6]

As detailed in Section V, we take an approach to scale up the packet processing unit by aligning and balancing network and compute processing capacities while reducing 'Tn' through native Linux advances.

B. *Scale Out*

After a Scaled-Up module 'M' of packet processing capacity 'p x k' for a given market type 't' and deployment area 'A' is defined, packet processing capacity can then be Scaled-Out in units of 'M'. For example, in AWS Cloud, a unit of 'M' is an EC2 instance which runs the packet processing VNF with required interfaces connected to VPC subnets. The scale out takes place by simply increasing the number of EC2 instances.

'M' can be dimensioned to optimally align 'Tn + Tc' to the market and deployment area. 'Tn' is completely dependent on the underlying networking stack and predicates 'M'.

In the subsequent sections we determine the value of 'Tn' attainable with different networking technologies. We then scale-out 'M' to demonstrate:

- Public Cloud infrastructure can deliver operational packet processing capacity of a Mobile Edge site.

- Public Cloud infrastructure can allow scale-out boundaries to cover very large deployment areas 'A' for regional packet processing capacity.
- Scale Out provides optimal utilization of compute and networking resources in Public Cloud infrastructure
- Public Cloud infrastructure can benefit from native Linux kernel advances to further increase the value of 'M', the module of packet processing capacity.

V. UPF Performance Scaling for Native Deployment

To enable realization of telecom core on Public Cloud infrastructure, we now explore the highest possible value of a module of packet processing capacity 'M' that we can get with a native Linux networking stack based UPF.

A. Scaling up UPF packet processing capacity

From Section II we learnt that telecom core networks have traditionally focused on saturating the link to deliver line rate. Technologies such as Data Plane Development Kit (DPDK) [21] help achieve these goals by providing a direct path for a user mode application to transmit and receive the packets from and to application space memory, bypassing Linux kernel stack, avoiding system calls and context switches. Basically, these technologies attempt to make 'Tn' close to zero with 'Tn << Tc'.

However, since kernel is bypassed, key kernel functionality such as routing and standard network management becomes a challenge. To deploy a DPDK application, the administrator needs to configure system hardware/hypervisor level details, such as PCI bus number, network interface hardware MAC address, huge page memory. DPDK bound PCI device is invisible to Linux system so troubleshooting and debugging of network issues becomes a major challenge as system will not respond to standard network discovery protocols such as, OSPF, ARP and ICMP and tools like ethtool, ip link etc. do not work.

The cost of making 'Tn' close to zero severely complicates operational deployment of the UPF and the increase in performance may not justify undertaking configuration complexity of deploying a DPDK-based application. All these deployment problems are automatically resolved when using native Linux networking stack making scale out seamless for cloud deployments.

Linux kernel enhancements and new features are constantly being added to optimize the packet traversal path, perform I/O with (almost) zero copy and with reduced number of system calls. Examples of such technologies are PACKET_MMAP, PF_RING, RDMA, PACKET FANOUT, ebpf and AF_XDP [22]. We therefore focus on traditional Linux network I/O mechanisms that aligns with cloud native technologies.

B. Design considerations for Scaling Up UPF Capacity

The graph in Figure 4 plots 'Tn', 'Tp' and the Packets Per Second (PPS) processed by the UPF based on Open Mobile Evolved Core (OMEC), that was deployed in a production telecom core using DPDK, DPDK with Kernel Network Interface (KNI) and Linux networking stack.

For Linux based experiments, we used Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz servers with Ubuntu 18.04 and kernel 5.3 and captured performance numbers as shown in Table 2 below. Unless we add algorithmic improvements, we see that 'Tc' i.e. the number of cycles taken by the UPF to process each packet takes about 259 ns, while 'Tn' takes about 3147 ns with standard networking functionality of the Linux stack. The end-to-end packet processing time Tp is mostly dependent on Tn and drives the UPF PPS at around 270 KPPS with Linux I/O calls.

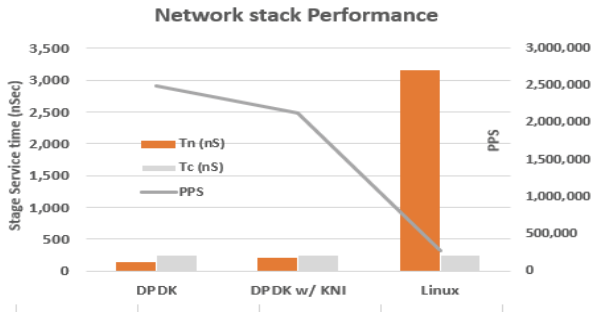

Figure 4: Network stack performance

Note that 'Tn' is the per packet combined time taken by recvfrom() and sendto() system calls. For larger packets, 'Tn' will likely increase but 'Tc' stays mostly constant as packet processing scope in the UPF is restricted to packet headers only and not a function of the payload. The table below shows how 'Tn' is spent in terms of recvfrom() and sendto() overheads.

recvfrom (Trx) nSec	sendto (Ttx) nSec	Total I/O Trx + Ttx (Tn) nSec	Packet processing (Tc) nSec	malloc, free, stats, charging etc. nSec	Total Processing (Tp) nSec
1,554	1,594	3,147	259	288	3,694

Table 2. Avg. per packet processing overhead (128 Byte packets)

The effective packet processing rate is calculated as

$$1/ (Tp / (\text{CPU speed in KHz})) = 270 \text{ KPPS}$$

The table also shows that the network I/O call 'Tn' accounts for more than 80% of the total processing time 'Tp'. Thus, the most effective way to reduce 'Tp' is to reduce 'Tn' as much as possible using efficient I/O mechanisms [23]. For example, Linux I/O calls like sendmmsg()/recvmmsg() [24] and advanced Network Interface Card (NIC) technologies can reduce the cost of 'Tn' to a significant extent as described below.

If we can stretch packet processing performance with Linux to 500 KPPS, a single UPF supporting 50K devices will be able to deliver a p value of 10 PPS per device, which is good enough for most markets. We can further reduce the coverage area 'A' for a reduced number of devices to increase 'p', scaling the capacity along the matrix space defined in (3) above.

For our experiments, we set the scale-up module 'M' of packet processing capacity as:

k = 50K devices, p= 10 PPS/device, PPS= 250 KPPS;

where compute and networking resource footprint for 'M' is 2 CPU cores and 2 network interface resources.

C. Design considerations for Scaling Out UPF Capacity

To scale out UPF capacity, we examine two mechanisms of sharing the network resource across several instances on a host:

- With SR-IOV, we can create multiple Virtual Functions (VFs) and assign each UPF separate VFs.
- Split the Receive and Transmit Queues of a single network interface across UPF instances.

1) Scaling Out with SR-IOV

We take single Scaled-Up module 'M' and run 'R' such Modules on the same host to realize 'M x R' capacity.

Using SR-IOV technology we create multiple Virtual Functions (VFs) and assign each UPF a separate VF for S1U and SGI ports, respectively. Each UPF sees the VF as an independent Network Interface Card (NIC) port with its own IP address. The NIC resource isolation is provided by NIC hardware at PCI bus level. Network then can be provisioned such that available user traffic is distributed across these multiple UPFs targeting different S1U IP addresses.

Figure 5 below shows the detailed packet processing path with SR-IOV for 'R' Modules and total capacity of 'M x R'.

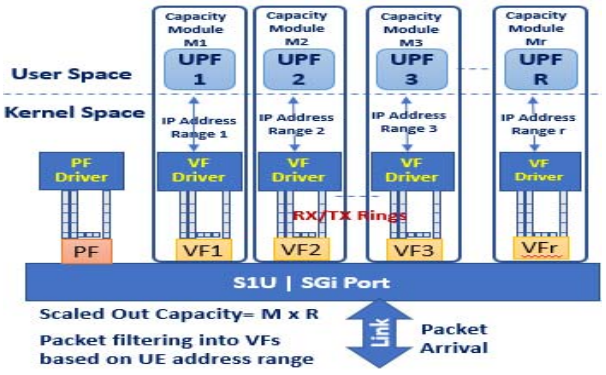

Figure 5: SR-IOV Scale Out Schematic

Scaling out with SR-IOV has its own shortcomings. It requires creation and management of VFs with each UPF needing its own IP address to be provisioned for directing traffic from the cell tower.

2) Scaling Out Aligning Network Queues to Cores

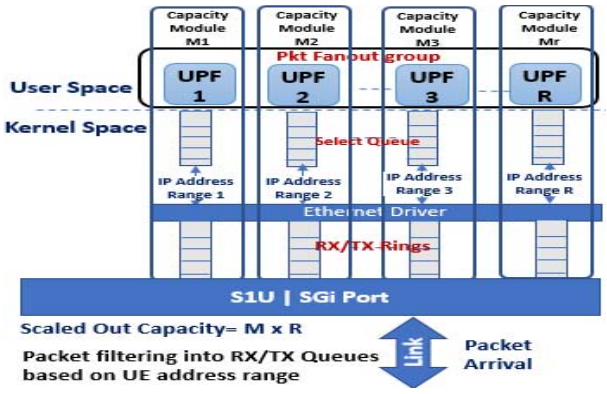

Figure 6: Schematic aligning Queues to Cores

The second method shown in Figure 6 splits network traffic among the UPFs using PACKET_FANOUT_QM mechanism. We configure the NIC to filter the packets to different queues based on user device IP address ranges. Each queue is then serviced via a different UPF. All UPFs in the system now share the same physical NIC and the same host IP address. Scaling UPFs in a system can now be simplified by spinning up another instance and adding NIC filters for queues associated UPFs.

VI. Design of Experiment and Test Infrastructure

We empirically establish Scaling Out of 'R' UPF modules, of capacity 'M' each, in a lab infrastructure and AWS Cloud infrastructure, respectively. Our experiments prove that 'R' modules of UPF capacity scale to deliver operational capacity for a telecom edge site on native AWS cloud. The Scaling Out experiments in the lab infrastructure allow us to demonstrate network and compute resource utilization and technology improvements possible with the state-of-the-art native Linux technologies: SR-IOV and PACKET_FANOUT, described in the schematics of the previous section.

To simulate real world deployments for telecom system architecture [3], we use OMEC based version of NGIC user plane functions [3], modified version of OMEC traffic generator [25] to emulate traffic on the S1U interface. The emulation allows us to emulate loads representative of up to a million users. An instance of the packet generator is mirrored on the SGI interface as a responder, allowing us to measure scaling performance of the UPF instances on S1U and SGI interfaces, simultaneously in both directions. Packet rate in PPS on both S1U and SGI interfaces is measured.

The emulation generates GTP-U [26] tunnel packets with varying configurations of number of devices, packets per seconds, packet size, number of cell towers, number and range of application servers, user IP address ranges and test duration. Packet size of 128 bytes, representative of a fully voice call saturated telecom core network is set across all the experiments. The traffic generator sends the user plane packets with equal distributions of number of packets per device and number of flows per cell tower to UPF.

As we focus our study on user plane performance and capacity planning, we use simulated control plane interfaces. The user sessions and tunnels are pre-populated at the UPF endpoints, so the generator is ready to generate the user plane traffic on the S1U and SGI interfaces, without having to explicitly prime the system with session creation messages.

The setup in the lab environment is composed of Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz Servers with Intel® Ethernet Network Adapter X710-T2L dual port 25G NICs. The lab infrastructure setup is detailed in Figure 7 below.

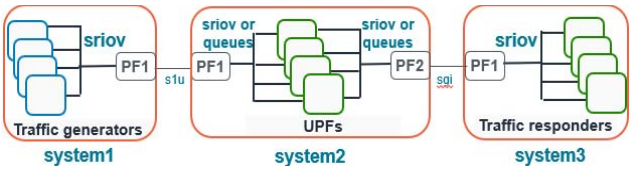

Figure 7: Lab Infrastructure Setup

For the scale out environment, SR-IOV functions are enabled on the Intel® Ethernet network adapter X710-T2L and multiple instances of trafficgen, responder and UPFs are instantiated on separate SR-IOV Virtual Functions (VFs).

For the next experiment, we replaced the X710-T2L with Intel® Ethernet 800 Series Ethernet network adapter with support for packet filtering based on user device IP Address range. PACKET_FANOUT configuration was used to assign separate device queues from same physical function to different UPF instances. The underlying physical network interface is configured to filter and direct ingress packets to target UPF's queue. UL ingress filtering is done based on source IP address range for GTP-U tunneled packets. DL filtering is based on destination user IP address.

This setup is then migrated to AWS Cloud, where we deployed our environment in the AWS us-west-2 region using AWS CloudFormation templates which are available on GitHub [27]. To maintain the same deployment environment in the AWS cloud and to align it to telecom deployments, we deployed traffic generator, traffic responder, and UPFs in separate VPCs connected with AWS Transit Gateways.

Each UPF runs on a dedicated AWS EC2 instance. This is because it is not feasible to create additional VFs on top of an existing VF exposed by the instance. To scale out, we use multiple EC2 instances with one UPF per instance. This aligns to running multiple instances of UPFs with the SR-IOV functions in the lab infrastructure.

We use EC2 instance type c5n.2xlarge for the UPFs. Each UPF instance is assigned two ENA adapters, one for UL and other for DL. An additional network interface is used for management traffic. All VPC resources and subnets connecting to the Transit Gateways are created within the same Availability Zone. Figure 8 below shows the AWS environment setup for our experiment.

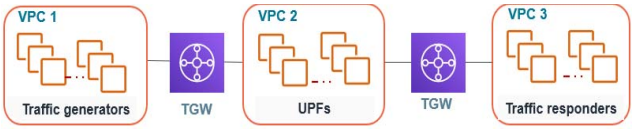

Figure 8. AWS Test Environment

Test automation scripts are used to run concurrent tests with varying number of instances for scale out, both in the AWS cloud and in the lab setup. The automation scripts then collect, aggregate and analyze the results.

VII. Results and Discussions

In this section, we present the results of our experiments in the lab infrastructure and the AWS cloud environment.

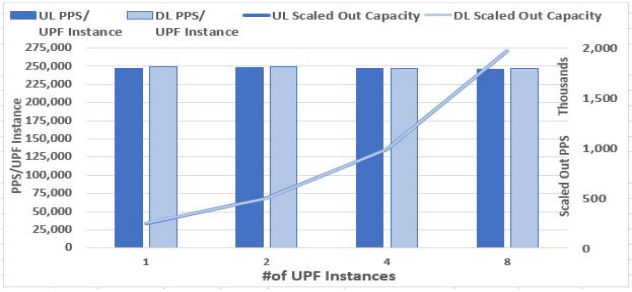

Figure 9: UPF Scale Out Performance with SR-IOV in Lab

The scale out performance of the SR-IOV setup is plotted in Fig. 9 above. Each UPF is assigned two SR-IOV VFs, one for UL, the other for DL and two CPU cores. Each UPF handles about ~250 KPPS in each direction using native Linux networking stack. The total capacity scales out linearly with an aggregate capacity of 2 MPPS in each direction using 8 UPF instances, equivalent to '8 x M' modules of capacity.

Next, we present our results with PACKET_FANOUT. *Intel*® Ethernet 800 Series Ethernet Adapter enables configuring filters to match on user device IP address range and forward these to specific queue associated to a UPF.

Using dedicated queues for each UPF enables UPF to busy poll for packets as and when it is ready for network I/O. This helps reducing the system interrupts and context switches and free up kernel cores from processing interrupts [28].

As shown in Figure 10, we see that the performance per UPF instance remains same between 240 KPPS to 250 KPPS. The total capacity scales out linearly with an aggregate capacity of 2 MPPS in both UL and DL directions using 8 UPF instances, again equivalent of '8 x M'.

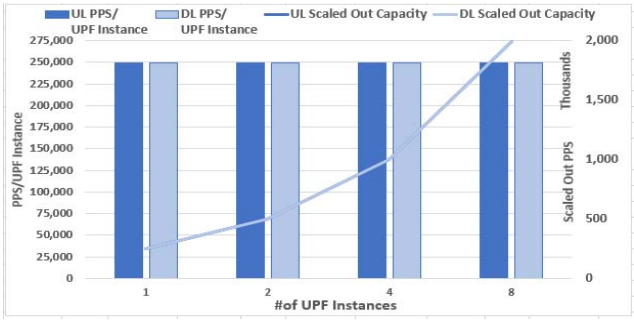

Figure 10. UPF Scale Out Performance with Interface PACKET_FANOUT in Lab

These results show the packet fanout scheme can achieve the same performance as SR-IOV without extra cores to process the system interrupts to receive and transmit packets.

Next in the AWS cloud, performance for single UPF instance is observed as ~200K PPS using two CPU cores per UPF same as in the earlier experiment. Performance scales out to approximately 2 MPPS with 10 UPF instances i.e. the equivalent of '10 x M' units of capacity, as shown in Figure 11. This shows that the data plane performance in AWS cloud linearly scales out in units of M.

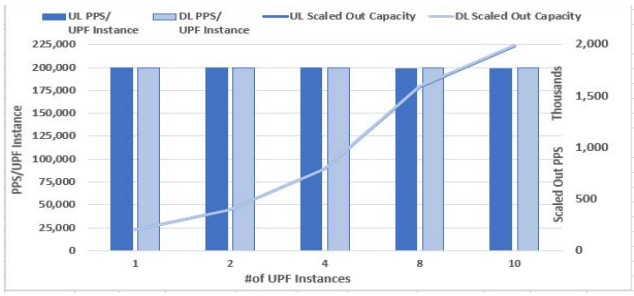

Figure 11. UPF Performance Scale Out on AWS Cloud

We expect the performance to scale further upto ~250K PPS per instance to achieve 2 MPPS with '8 x M' modules of capacity on AWS cloud as part of our future work. These results further demonstrate there are no intrinsic VPC or inter-VPC networking capacity bottlenecks with increasing number of instances to scale out capacity.

VIII. Conclusions and Future Work

In this paper, we provide a novel approach to capacity planning for telecom core networks in Public Cloud. We establish a packet processing capacity model for scaling up and scaling out VNF performances on a per VNF unit basis. We explore native Linux kernel technologies to scale up and scale out the UPF performance. Our experiments conducted on a lab testbed indicate that the performance of Linux SR-IOV and the interface fanout are comparable. Our empirical results from AWS Cloud proves that multiple EC2 instances are equivalent to SR-IOV and interface fanout schemes and deliver the same level of UPF performance i.e. 2MPPS, ~20Gbps for 250K user devices, representing the network load of a Mobile Edge site.

We also provide a reference architecture for deploying telecom core network infrastructure in AWS Cloud. The architecture enables telecom operators to deploy VNFs right-sized by our scale-up method within performance-efficiency threshold. The AWS CloudFormation templates are available in GitHub [27] to deploy the AWS test environment in Section VI.

In future, we plan to use packet fanout in AWS Cloud to deploy multiple UPF instances within a single EC2 instance that can vastly improve compute and I/O resources utilization for a given unit of capacity 'M'. we also plan to explore secure User Plane Functions in native cloud environments. This will require research in other UPF design patterns like pipeline and hybrid packet processing in a secure and trusted execution environment, e.g., Intel® SGX.

We will further develop and experiment dynamic scaling of user plane and control plane functions. This scheme will scale-out and scale-in resources per user sessions on demand.

Acknowledgment

We thank Sean Lion, Mesut Ergin, Yi Zou (Intel) for their reviews and guidance. Our sincere appreciation and thanks to Suresh Marikkannu (HCL America) and Arun Emanuel (HCL Technologies Ltd.) for their support in the experiments. We also thank Matt Lehwess, Gaurav Gupta, Ishwar Parulkar, and Young Jung of AWS for their feedback and guidance.

References

[1] ETSI, "Network Functions Virtualisation", The European Telecommunications Standards Institute white paper, October 22-24, 2012, "SDN and OpenFlow World Congress", Darmstadt-Germany.

[2] Open Mobile Evolved Core (OMEC), https://www.opennetworking.org/omec/ , accessed on 18 June 2020.

[3] T-Mobile Poland Achieves Production Roll-Out of OMEC https://www.opennetworking.org/news-and-events/press-releases/t-mobile-poland-achieves-production-roll-out-of-omec-onfs-open-source-mobile-evolved-packet-core/, accessed on 18 June 2020.

[4] A. S. Rajan, et al., "Understanding the bottlenecks in virtualizing cellular core network functions," Proceedings of the 21st IEEE International Workshop on Local and Metropolitan Area Networks, April 2015.

[5] B. Hirschman, et al., "High-performance evolved packet core signaling and bearer processing on general-purpose processor," IEEE Network, April-May 2015.

[6] R. Archibald, et al., "An IoT control plane model and its impact analysis on a virtualized MME for connected cars," Proceedings of the 22nd IEEE International Symposium on Local and Metropolitan Area Networks, June 2016.

[7] Erlang, https://www.erlang.com/calculator/erlb/, accessed on 18 June 2020.

[8] Cell Phone Tower Statistics, http://www.statisticbrain.com/cell-phone-tower-statistics/, accessed on 18 June 2020.

[9] NGMN 5G White Paper, https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf, accessed on 15 June 2020.

[10] 3GPP, "Feasibility study for evolved Universal Terrestrial Radio Access (UTRA) and Universal Terrestrial Radio Access Network (UTRAN)," TR 27.912, January 2015.

[11] GSMA, Mobile backhaul options, Spectrum analysis and recommendations https://www.gsma.com/spectrum/wp-content/uploads/2019/04/Mobile-Backhaul-Options.pdf, accessed on 10 June 2020.

[12] Y. Wang, C. Williamson, J. Doerksen, "CAC performance with self-similar traffic: simulation study and performance results," Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, October 1999.

[13] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," The 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA), 2015.

[14] Amro Al-Said Ahmad and Peter Andras, "Measuring and Testing the Scalability of Cloud- based Software Services," Proceedings of 2018 IEEE World Congress on Services, July 2018

[15] Foivos Michclinakis, et al., "The Cloud that Runs the Mobile Internet: A Measurement Study of Mobile Cloud Services," Proceedings of IEEE INFOCOM 2018, April 2018.

[16] Michael Dalton et al, "Andromeda Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization," Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation, April 2018.

[17] 3GPP, "Network Architecture," TS 23.002 V15.0.0, March 2018.

[18] 3GPP, "System architecture for the 5G System," TS 23.501 V16.4.0, March 2020.

[19] 3GPP, "Architecture enhancements for control and user plane separation of EPC nodes," TS 23.214 V6.0.0, June 2019.

[20] ETSI, "MEC in 5G networks," The European Telecommunications Standards Institute white paper, June 2018.

[21] Data Plane Development Kit https://www.dpdk.org/ , accessed on 18 June 2020.

[22] Address Family- Express Datapath https://www.kernel.org/doc/html/latest/networking/af_xdp.html , accessed on 10 June 2020.

[23] J Corbet, "Improving Linux networking performance" https://lwn.net/Articles/629155/ , accessed on 10 June 2020.

[24] T. Hruby, T. Crivat, H. Bos and A. Tanenbaum, "On Sockets and System Calls Minimizing Context Switches for the Socket API", Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, 2014.

[25] Open Mobile Evolved Core, il_trafficgen https://github.com/omec-project , accessed on 18 June 2020.

[26] 3GPP, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)," TS 29.281 V15.3.0, June 2018.

[27] AWS CloudFormation templates for UPF performance test, https://github.com/aws-samples/aws-intel-5g, 2020.

[28] Faster, More Predictable Ethernt with the Intel® Ethernet 800 Series Application Device Queues (ADQ) https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/application-device-queues-technology-brief.pdf, accessed on 10 June 2020.