A Comparison of SDN and NFV for Re-designing the LTE Packet Core

Aman Jain, Sadagopan N S, Sunny Kumar Lohani, Mythili Vutukuru
Department of Computer Science and Engineering, Indian Institute of Technology, Bombay
Email: {amanjain, nss, slohani, mythili}@cse.iitb.ac.in

Abstract*—With an increase in the number of mobile users and traffic, mobile network operators are finding it difficult to scale their radio and core networks. Further, hardware network appliances are expensive to procure and upgrade, and are difficult to adapt and program for new services. These trends have recently spurred several efforts to redesign various components of mobile networks, including the LTE Evolved Packet Core (EPC). Software Defined Networking (SDN) and Network Functions Virtualization (NFV) are two popular emerging networking paradigms that aim to increase network flexibility and scalability, while reducing the overall cost. With SDN, the control and data planes of the packet core can be separated, enabling cheaper packet gateways in the data plane, and an intelligent core network controller to handle the signaling and management functions. With NFV, the various hardware components that comprise the packet core can be virtualized and run as software on a cloud, enabling benefits such as elastic scaling and quick innovation. While several proposals exist to use SDN and NFV to redesign the EPC, there is no common framework to compare the new designs on various performance metrics. This paper presents the design and evaluation of two open-source implementations of the LTE EPC, one based on SDN principles and the other based on NFV, and presents a performance comparison of the two approaches. Experiments with our prototype show that an NFV-based implementation is better suited for networks with high signaling traffic, because handling the communication with the SDN controller quickly becomes the bottleneck at the switches in the SDN-based EPC. On the other hand, an SDN-based design of the EPC is better suited for networks with high data plane traffic, because SDN switches are often more optimized for packet forwarding than virtualized software appliances. We believe that our framework can be used to develop and compare several such design alternatives, and can serve as a guide for future redesigns of mobile data packet core networks.

I. Introduction

Mobile cellular networks have witnessed an abrupt increase in the volume of data traffic due to growing user demands and an almost pervasive presence of modern gadgets such as smart phones and tablets. Cisco [1] predicts that mobile data traffic will increase eightfold between 2015 and 2020. Current telecom network architectures are expected to face difficulties scaling to this load, causing mobile operators to look for new alternatives to overcome this challenge. Let us consider the particular case of the popular mobile packet core architecture in use today, the 4G/LTE EPC (Long Term Evolution Evolved Packet Core). The well-known problems with the traditional EPC architecture (Figure 1) are:

- *Cost:* The EPC components such as MME (Mobility Management Entity), HSS (Home Subscriber Server), SGW (Serving gateway) and PGW (Packet data network gateway) are proprietary devices, with a number of complex functionalities packed into a single box. This makes them expensive to procure, maintain, and upgrade with increasing traffic demands.
- *Flexibility:* Most components of the EPC have all the control and data plane logic programmed into the hardware. Any modifications in the functionality to add new services requires replacement of the entire appliance. Further, it is usually hard to integrate and incorporate functions from different vendors into a single box.
- *Scalability:* Hardware appliances typically scale vertically, i.e., the complete appliance must be replaced by a more powerful one in order to handle higher load. However, telecom operators would prefer architectures that scale horizontally, in order to elastically scale the network in response to load.
- *Signaling overheads:* The current architecture involves a lot of control signaling among the EPC components, incurring substantial amounts of overhead in terms of bandwidth and processing time.

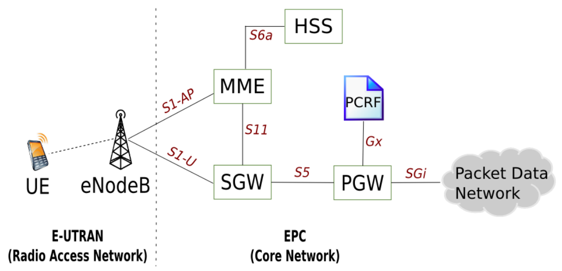

Fig. 1: Traditional LTE Architecture.

Given the limitations above, there is an immediate need to re-design the LTE packet core network to make it more scalable, flexible, and cost effective in the coming years. Several new architectures for the LTE EPC have been proposed recently (see Section II for a detailed discussion), incorporating the emerging trends of Software Defined Networking (SDN) and Network Functions Virtualization (NFV). SDN is a new paradigm which decouples the network control functions (control plane) and the forwarding functions (data plane), enabling network administrators to program the network in a dynamic

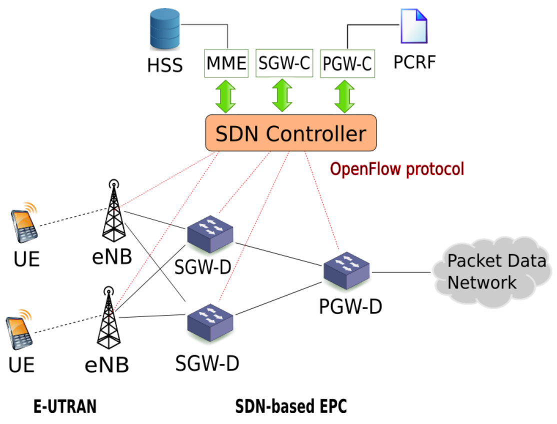

Fig. 2: A typical SDN-based LTE EPC.

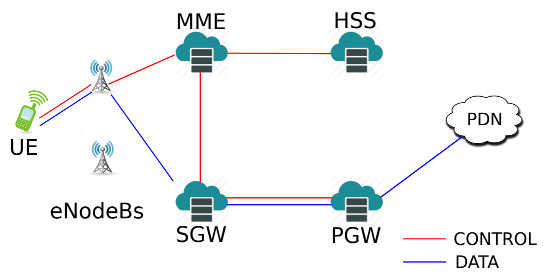

Fig. 3: A typical NFV-based LTE EPC.

and flexible manner. The control plane is built in software as applications running on top of a controller, whereas general purpose switches constitute the data plane. Both the planes communicate with each other via a standardized protocol such as OpenFlow. The principles of SDN can be applied to separate out the control and data plane functionalities of the EPC components as well, to reap the benefits of the new paradigm. For example, Figure 2 shows a typical SDN-based LTE EPC. Control plane entities like MME are implemented as part of the SDN controller, while components with both control and data functions are partitioned.

NFV is another network architecture concept that has gathered significant interest recently. NFV proposes that network functions (e.g., functionality of components such as MME, SGW, PGW) should migrate from custom hardware equipment to virtualized software appliances. NFV greatly reduces the cost and complexity of implementing network functions, by having network functions run on clusters of virtual machines (VMs) hosted on commodity high-end servers. Further, such virtualized network functions (VNFs) can be modified easily to incorporate new features, and elastically scaled on demand to handle increasing load. Figure 3 shows a typical architecture of NFV-based LTE EPC. The core functions of MME, HSS, SGW and PGW are deployed as software modules on VMs in datacenters, and the corresponding control and data traffic flow across these virtualized entities.

While there have been several proposals around these new EPC architectures, there does not exist a common framework to build and compare these architectures along dimensions such as performance, to the best of our knowledge. While commercial or licensed implementations exist for some new EPC architectures (e.g., [2], [3], [4]), there is no open-source implementation framework for researchers to experiment with these ideas. As a result, there are no published studies comparing and understanding the tradeoffs across the various EPC architectures for different types of networks.

This paper presents the design, implementation, and evaluation of two LTE EPC architectures, one based on the principles of SDN (Section IV), and the other based on the idea of NFV (Section V). While our implementations are not fully standards compliant, they carefully capture the various complex functionalities of the various LTE components (Section III) that impact performance, and can be used to make meaningful comparisons across metrics such as throughput and latency. Our entire source code is available on Github [5], [6] for other researchers to use and modify. We believe that an open-source codebase such as ours will spur research and innovation in the area of new EPC architectures, both in academia and industry.

We have run several experiments with our code to draw several interesting conclusions (Section VI). Our results show that an NFV-based architecture provides better performance than an SDN-based architecture on comparable hardware for control plane traffic, because the SDN-based EPC runs into performance bottlenecks on the path to the centralized controller. On the other hand, an SDN-based architecture performs better for data plane traffic, because the NFV architecture incurs several I/O stack overheads when transferring packets to/from VMs. Therefore, the relative merits of the two architectures depend on the network traffic characteristics among other things. We believe that our EPC framework can be used to build and evaluate several EPC architectures in this manner, and can inform design choices of mobile packet cores for 5G and future standards.

II. Related Work

There have been several recent proposals to redesign various components of mobile data networks using principles of SDN and NFV. Proposals that deal with the LTE EPC form a bulk of the work, given the significance of the packet core to the operation of mobile data networks. Basta et al. [7], [8] have proposed the use of SDN and NFV to redesign LTE EPC. Their work has discussed several tradeoffs between various futuristic EPC architectures. The authors also compare and evaluate the various designs using simulations along parameters such as network load and data plane delay. Several other researchers [9], [10], [11], [12] propose using the idea of SDN in different ways to redesign various components (or replace the entire architecture) of the EPC. These papers differ in the amount of work offloaded to a centralized controller, and the accruing benefits. However, none of these papers have a complete EPC implementation in order to compare with other alternatives. Our SDN-based EPC is inspired by these proposals, and takes their vision to completion by providing a complete EPC implementation based on the idea of SDN. In fact, our framework can be used to easily build and compare several such SDN-based EPC architecture proposals.

NFV-based EPC implementations have also seen significant interest from academia and industry. Authors in [13] evaluate a commercial NFV-based EPC, and demonstrate that its performance can match that of a hardware-based appliance. Other research proposals [14], [15], [16] show how to horizontally scale the MME component of a virtualized EPC in order to build a high-performance scalable distributed EPC. In [15], the central MME core node is distributed into multiple replicas and pushed closer to the access edge, resulting in reduced latency and better handover performance. SCALE [14] proposes dividing an MME element into a load balancer and packet processor components, and using consistent hashing at the load balancer to distribute incoming connections. [16] presents an MME architecture based on a stateless worker-thread and centralized state storage model. However, none of the prior work in this area make their code available for researchers. Our NFV-based EPC implementation is intended to be available as open-source code, in order to spur further research into scalable NFV architectures.

Open EPC [3] and Open Air Interface [17] are two recent efforts to develop software-based LTE network functions. While the former is licensed code available for commercial deployments, the latter has been designed with standards compliance (and not performance) in mind. For example, all EPC components of the Open Air Interface codebase are designed to run on a single machine. On the other hand, our EPC codebase has been developed with performance evaluation in mind, and standards compliance testing is a non-goal for our work. Given these differences, we believe that our work is complementary to existing open-source EPC software, and can be useful to conduct research on high performance, scalable NFV designs of the LTE EPC.

III. Background: LTE EPC Procedures

We now briefly describe the basic procedures of LTE EPC that are implemented by our NFV and SDN implementations. An LTE network is composed of two parts: a Radio Access Network (RAN) and an Evolved Packet Core (EPC). A user equipment (UE) and the base station it connects to (eNodeB) comprise the RAN, while the MME, HSS, SGW, PGW and (an optional) PCRF make up the EPC. Following are the main procedures implemented by the various EPC components.

UE Attach: A UE that wishes to connect to an LTE network sends an attach request to the MME via its eNodeB. Prior to sending the attach request, the air interface radio connection is established between the UE and eNodeB. The UE sends its IMSI (International Mobile Subscriber Identity) as its identifier along with the attach request. After receiving the request, the MME proceeds to implement the various authentication and security setup procedures as follows: (i) *UE authentication:* MME authenticates the UE with the help of the HSS, and a mutual authentication between UE and network takes place. (ii) *Security setup:* Next, various security setup procedures take place, during which encryption and integrity keys are exchanged among the UE, eNodeB, and MME, to ensure secure communication between them. (iii) *Data tunnel setup:* After successful security setup, a default "bearer" is created for the UE through the packet core. During this process, the UE is assigned an IP address from the PGW, and tunnel endpoint identifier (TEID) values for this bearer are exchanged amongst the eNodeB, MME, SGW and PGW. At the end of this procedure, a user plane tunnel gets established for the UE from eNodeB to the PGW through the SGW.

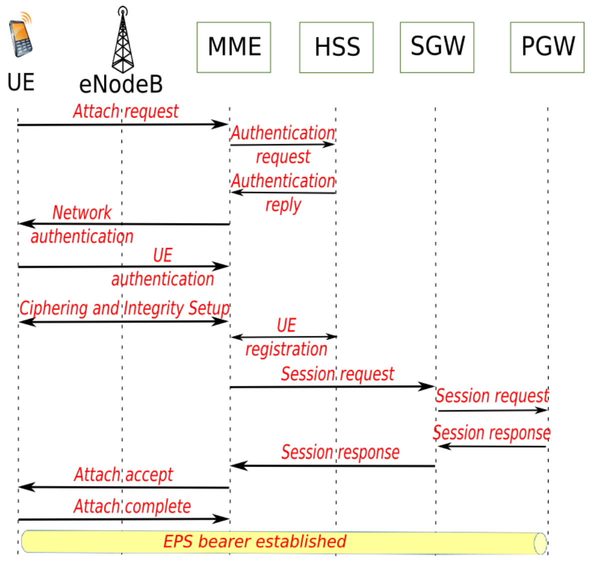

Fig. 4: An overview of LTE Attach Procedure.

Uplink/downlink UE data transfer: Once a UE attaches and establishes a tunnel through the EPC, it can send/receive data to/from the PDN via eNodeB and EPC. UE sends data packets through the air interface to eNodeB. The eNodeB encapsulates the received packet in a GTP (GPRS tunneling protocol) header along with UDP/IP headers, and then forwards the packet to the SGW. The SGW strips off the outer header of the received encapsulated packet, adds GTP/UDP/IP headers of its own, and then forwards to the PGW. The PGW decapsulates the received packet by stripping off the outer headers (IP/UDP/GTP), and then forwards to the PDN. Similarly, the downlink data transfer takes place from the PDN to UE.

UE initiated detach: When a UE wishes to disconnect from the mobile cellular network, it initiates a detach procedure. All state associated with the UE, including its bearer, is cleared from all the EPC components. Finally, the MME sends a detach accept response to the UE via eNodeB.

In addition to the procedures described above, the EPC also implements procedures to page idle UEs, handle UE mobility via handover procedures, among other things. The EPC implementations evaluated in this paper handle the basic attach, detach, and data transfer procedures, as this minimal set of procedures was found to be sufficient to cover the control and data functionalities at all nodes. We are in the process of implementing handover and other features. We would like to emphasize that standards compliance is a non-goal, as we do not envision our code being used in production settings. On the other hand, we carefully capture all aspects of the procedures that impact performance, so that meaningful performance comparisons can be made across designs.

IV. SDN-BASED EPC

We now describe our SDN-based LTE EPC implementation. A typical SDN-based EPC (Figure 2) entails a separation of control and data planes for the EPC Gateways—SGW and PGW. The MME, of course, is only a control plane entity. So the MME, SGW-C and PGW-C run as applications on top of an SDN controller. The SGW-D and PGW-D are realized as SDN (OpenFlow) switches which forward packets based on the forwarding rules installed by the controller. Also, the eNodeB has an SDN switch apart from the radio components, which contains the forwarding rules installed by the controller to divert traffic to the EPC switches. The HSS is implemented as a separate application used by MME, whereas the PCRF (Policy and Charging Rules Function) is a separate application used by PGW.

Figure 5 shows the architecture of our actual implementation. We use the Floodlight SDN controller [18]. We use OpenvSwitch (OVS) in lieu of OpenFlow-enabled hardware switches for the data plane of the gateways, due to high cost and difficulty in procuring hardware SDN switches. HSS uses a MySQL database. We also built a RAN (Radio Access Network) simulator that combines the functionalities of the UE and the eNodeB, in order to generate control and data traffic to our EPC. We do not implement the radio procedures that take place between UE and eNodeB in the RAN simulator, and focus only on the traffic that is seen by the EPC. A sink node to represent the PDN (Packet Data Network) has also been implemented, and it connects to the PGW. The PCRF has not been implemented currently.

We have made a few practical modifications to the vanilla SDN-based EPC design for feasibility of implementation. An additional OpenFlow switch, which we refer to as the *default switch*, is required in our setup to connect the RAN simulator with the EPC. In real life SDN EPC, the role of the default switch would be served by an SDN switch located in the eNodeB. Also, since the OVS has no support for matching on GTP headers, we use the VLAN ID field of the Ethernet header (IEEE 802.1Q) to store the TEID (Tunnel Endpoint Identifier). So, we do not encapsulate/decapsulate data packets with GTP headers at the eNodeB/PGW, and instead edit the VLAN ID field. Finally, the control traffic between RAN and MME (via the default switch) uses UDP unlike in the case of LTE which uses SCTP.

V. NFV-BASED EPC

We now provide an overview of our NFV-based EPC implementation, as shown in Figure 6. The core network functions of the EPC (MME, HSS, SGW and PGW) are built as software modules, and are run on VMs hosted on a private cloud. Much like in the case of the SDN-based EPC, we also built a RAN-simulator module that combines the functionalities of the

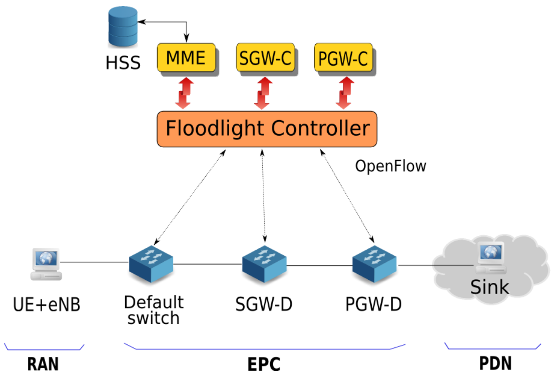

Fig. 5: SDN-based LTE EPC implementation.

UE and eNodeB into a single logical entity for the purpose of generating traffic to our EPC. A separate sink module is used to receive the generated traffic from the RAN and send back downlink traffic. We use `tun` devices to perform GTP encapsulations and decapsulations of TCP traffic at the source and sink. MySQL is used for database operations by the HSS entity. Standard communication protocol stacks prescribed by 3GPP are used across the implemented system, as shown in Figure 6. We have modified the formats of the S1AP and diameter protocols slightly for ease of implementation; the rest of our implementation faithfully follows the standards.

Each of the EPC components (MME, SGW, PGW, HSS) is implemented as a multi-threaded server that services requests (e.g., UE attach request) from the downstream nodes and sends responses back. Note that every component acts as a server to the downstream node, and as a client to the upstream node in the chain of VNF components traversed by a request. The MME application module uses a multi-threaded SCTP server to handle requests from the downstream eNodeB. For ease of implementation, we did not use the multi-streaming feature of SCTP. The EPC gateways are built around a multi-threaded UDP server, because the protocol stack of the gateways uses UDP to communicate with the other components. Below, we elaborate on our multi-threaded server architectures.

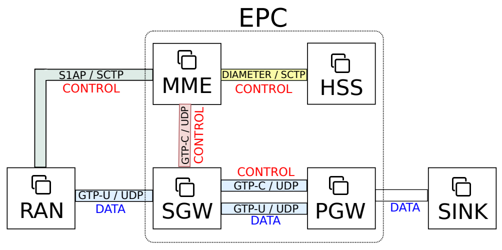

Fig. 6: NFV-based LTE EPC implementation.

Multi-threaded SCTP Server Architecture: The SCTP server architecture consists of a single master thread and several worker threads. The master thread communicates with each of the worker threads via a logical Unix pipe. The master thread listens for incoming client connections on a stream socket, and creates an additional socket file descriptor for every new SCTP client that is accepted. It then sends the new connection socket file descriptor to the worker threads in a round-robin fashion, to balance load across all worker threads. The worker thread is responsible for servicing the request on the new connection, through all the steps required to handle the request. For example, the worker thread in the MME must implement the various steps required to process a UE attach request, blocking for responses at each stage from the other nodes. The workers threads use the event-driven *select* mechanism to multiplex the multiple requests being handled by each of them concurrently. The number of worker threads in this architecture can be sized to fully utilize all the processing cores of the VM.

Multi-threaded UDP Server Architecture: The UDP server at the gateways consists of a single datagram socket being serviced by multiple server threads. All server threads try to read a UDP packet from the common shared socket. The thread that succeeds in reading the packet is responsible for processing the request and sending a response back to the downstream node. By fixing a suitable number of server threads in proportion to the number of system cores, this architecture provides good performance and scalability.

In both the server architectures, a global data structure is used by the threads to store/retrieve necessary connection data and state. This data structure is protected by locks to prevent concurrent inconsistent access by the multiple threads. As part of future work, we plan to redesign our system using lock-free data structures to eliminate contention between the threads for shared data structures. We also plan to explore alternatives to the kernel-based socket communication model, e.g., a userspace network stack built on top of kernel bypass techniques like Intel DPDK [19].

VI. Evaluation

We now compare our two EPC implementations on various performance metrics under different traffic conditions, to assess the pros and cons of the designs. We begin with a description of our testbed setup.

Setup. We test our EPC implementations using two types of traffic: control traffic, consisting mainly of UE attach (registration) requests generated by simulated users in the RAN simulator, and data traffic, consisting of TCP traffic generated from the RAN simulator to the sink. We also ran experiments with varying mixes of control and data traffic; we do not report those results here due to lack of space. For experiments with control traffic, we simulate a number of concurrent UEs in the RAN simulator, and make the UEs continuously attach and detach from the LTE network, to create a continuous stream of control traffic. We vary load on the EPC by varying the number of concurrent UE threads. We then measure the throughput, or the number of registration requests successfully completed by the EPC per second.

For experiments with data traffic, we attach a specified number of UEs to the EPC from the simulator, and pump traffic from the RAN to the sink using `iperf3` [20]. We vary the input data traffic into the EPC by varying both the number of concurrent UEs and the rate at which each UE generates traffic. We measure two performance metrics: (i) throughput, or the amount of data traffic successfully forwarded by the EPC gateways to the sink per second, and (ii) round trip latency, or the amount of processing overhead added by the EPC gateways in the data plane, as measured by a `ping` command during a load test. We ran all experiments for a duration of 300 seconds.

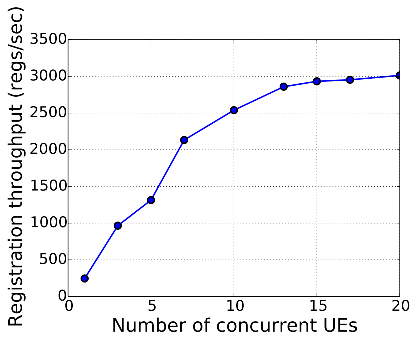

Fig. 7: SDN EPC: Registration throughput vs. no. UEs.

For the NFV-based EPC setup, all VMs were installed on a private OpenStack-based cloud, and were provisioned resources as shown in Table I. We use two different setups when testing our SDN-based EPC. For control traffic experiments, where the bottleneck is likely to be the controller CPU, we installed each of the components shown in Figure 5 on separate physical machines. All machines were connected by a 1Gbps LAN. However, for data traffic experiments, the network between the machines became a bottleneck. To avoid the network bottleneck, we installed the various components within Linux containers (LXC) on the same physical machine, and connected them up using the Linux bridge. In all experiments with the SDN EPC, all machines/containers of all components were provisioned with 2-core Intel i5-4440 CPU @ 3.10 GHz, and 4GB RAM. Care was taken to provision similar resources in both the EPC implementations, in order to be able to fairly compare the performance results.

TABLE I: Configuration of NFV-based EPC testbed.

ENTITY	CPU	#CORES	RAM	OS
RAN	Intel Xeon E312xx @ 2.6 GHz	8	4GB	Ubuntu 14.04 Server
MME, SGW, PGW, HSS		2	2GB	
Sink		4	2GB	

A. Control plane performance

We now generate continuous control traffic to our EPC setups, and compare their performance along the metrics of throughput and latency. Figures 7 and 8 show the registration throughput of the EPC implementations, as we increase the number of concurrent UEs in the RAN simulator. We find that the NFV-based setup is able to support a much higher control plane throughput (approx. 9000 reqs/s) than the SDN-based

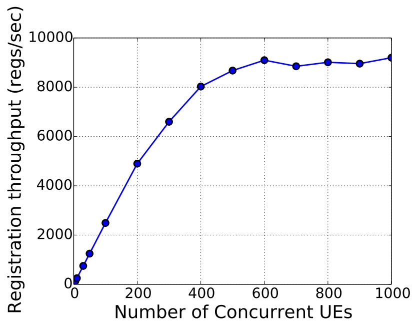

Fig. 8: NFV EPC: Registration throughput vs. no. UEs.

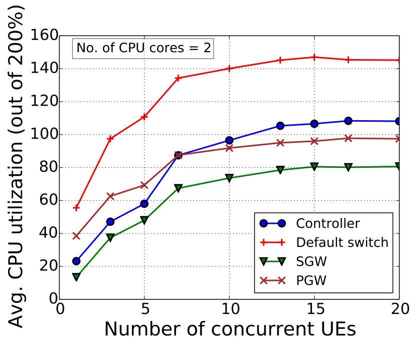

Fig. 9: SDN EPC: CPU utilisations of various components.

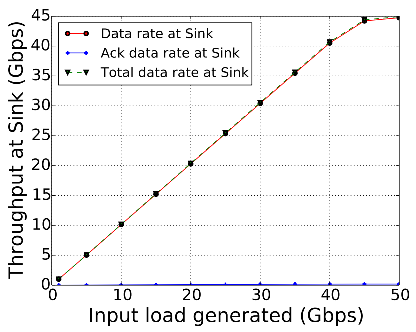

Fig. 10: SDN EPC: Data plane throughput vs. input load.

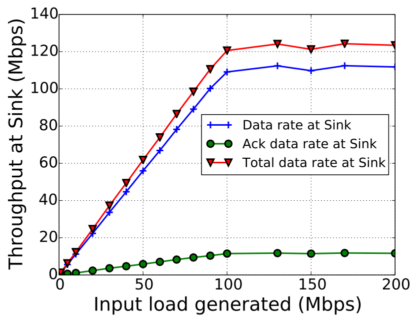

Fig. 11: NFV EPC: Data plane throughput vs. input load.

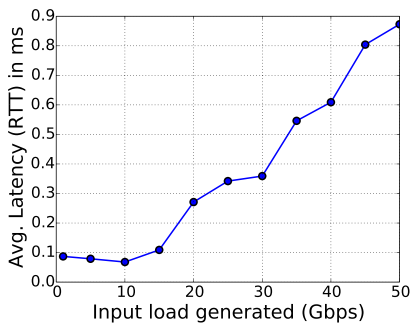

Fig. 12: SDN EPC: Data plane latency.

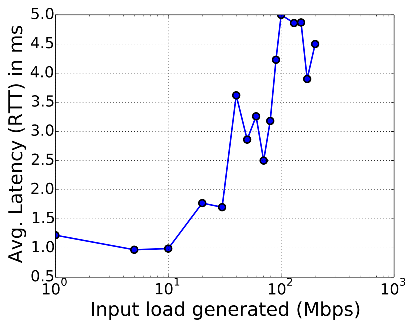

Fig. 13: NFV EPC: Data plane latency.

setup (approx. 3000 reqs/s) on comparable CPU provisioning. While the NFV-based EPC is limited by the CPU of the MME, the SDN-based EPC was limited by the CPU of the default SDN switch. We verified that the CPU alone, and not the network, was the bottleneck in all cases. Figure 9 shows that the default switch that forwards the control traffic to the SDN controller was overloaded even before the controller, because the switch had the additional job of installing OpenFlow rules and communicating via OpenFlow with the controller. This finding was a surprising conclusion from our experimental study, and has implications for SDN-based EPC designs. SDN-based EPC designs must carefully consider the overhead of installing rules and communicating with the controller at the switches also, in addition to processing overheads at the controller, when processing control plane signaling messages. Therefore, for networks with heavy signaling traffic, an NFV-based EPC design might provide better performance than an SDN-based design, where switches must pass all signaling messages to the controller and install flow rules.

B. Data plane performance

We now load our EPC implementations with only data traffic and measure their performance. Figures 10 and 11 show the data plane throughput forwarded by the EPC gateways to the sink in the SDN and NFV setups respectively, as we increase the offered data plane traffic to the EPC. We find that the SDN EPC gateways that forward traffic directly from the OpenvSwitch in kernel space are able to handle higher amounts of traffic (approx. 45 Gbps) than the NFV EPC gateways (approx. 110 Mbps) which must send the traffic to user space in the VM to make a forwarding decision. In fact, while our NFV-based EPC gateways hit a CPU bottleneck in our experiments, we were never able to fully saturate our

SDN-based gateways, as we were limited by the maximum network bandwidth for load generation (even in our container-based setup), and not the forwarding capacity of the gateways. Therefore, we believe their forwarding capacity is much higher than what our results show.

We believe that our SDN EPC will give much higher performance if run on hardware SDN switches instead of software switches. Similarly, we expect the performance of the NFV EPC gateways to also improve considerably if techniques to bypass the kernel and improve network I/O for VMs, such as Intel DPDK [19], were to be employed in the development of our gateways. We are working on these improvements currently. That said, with our current implementations, our results show that an SDN-based EPC can give a much better data plane performance than an NFV-based EPC, simply because data plane forwarding can be done more effectively in a hardware or kernel switch than in a user-space VM. Figures 12 and 13 also show the data plane processing round-trip latencies of the SDN and NFV EPC setups. As expected, the SDN EPC has a lower processing overhead than the NFV EPC for the same reasons discussed above.

VII. Conclusion and Future Work

In this paper, we presented two implementations of the LTE EPC, one based on the principles of SDN, and the other based on the idea of NFV. The SDN-based architecture separates out the control and data planes in the EPC components. The control functions run as applications on top of an SDN controller, while the data plane runs on general purpose SDN switches. The NFV-based architecture builds the EPC components as software appliances running on virtual machines in datacenters. Both the architectures eliminate the need for custom hardware equipment in deploying network functions, and provide great flexibility in managing resources. While these architectures have been proposed before, we have provided a complete implementation and performance comparison of both architectures across different types of traffic. Our results show that an SDN-based EPC is a better design choice when handling large amounts of data traffic, since it incurs lesser overhead when forwarding packets from the kernel or the switching hardware, as compared to an NFV setup that makes forwarding decisions in user space. On the other hand, an NFV-based EPC setup is better at handling large signaling load, because communicating every signaling message with the controller and processing the resulting rule updates quickly overwhelms the SDN switches. Therefore, the optimal EPC architecture for a given network would depend on, among other things, the mix of control and data traffic expected by the operator. We have released our entire code on Github [5], [6] as open-source, in order to spur further research in this direction.

While we have drawn some conclusions on the pros and cons of various design options using our initial prototypes, we envision this work to be the beginning, and not the end, of the discussion on EPC design choices. We invite other researchers to extend the code to build and compare several other EPC architectures. We believe that such experimentation will help us make informed choices in architecting the mobile core in future standards. As part of our future work, we plan to enhance our code by adding more features to make our implementations more standards-compliant (though full compliance and use in production settings are non-goals). We are upgrading our code to use the latest high-performance network I/O frameworks such as Intel DPDK. We are also working on distributed versions of the various NFV-based EPC software components, to explore the design choices in building scalable virtual network functions.

References

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2015-2020." [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[2] "Connectem," http://www.brocade.com/en/products-services/mobile-networking.html.

[3] "OpenEPC," http://www.openepc.com.

[4] "Virtual Mobile Network," http://www.affirmednetworks.com/products-solutions/vepc/.

[5] "SDN_LTE_EPC," https://github.com/networkedsystemsIITB/SDN_LTE_EPC.

[6] "NFV_LTE_EPC," https://github.com/networkedsystemsIITB/NFV_LTE_EPC.

[7] A. Basta, W. Kellerer, M. Hoffmann, K. Hoffmann, and E.-D. Schmidt, "A Virtual SDN-Enabled LTE EPC Architecture: A Case Study for S-/P-Gateways Functions," in *Proc. of IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013.

[8] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, and K. Hoffmann, "Applying NFV and SDN to LTE Mobile Core Gateways, the Functions Placement Problem," in *Proc. ACM SIGCOMM Workshop on All Things Cellular: Operations, Applications, Challenges*, 2014.

[9] S. Ben Hadj Said, M. Sama, K. Guillouard, L. Suciu, G. Simon, X. Lagrange, and J.-M. Bonnin, "New control plane in 3GPP LTE/EPC architecture for on-demand connectivity service," in *Proc. IEEE International Conference on Cloud Networking (CloudNet)*, 2013.

[10] L. E. Li, Z. M. Mao, and J. Rexford, "Toward Software-Defined Cellular Networks," in *Proc. European Workshop on Software Defined Networking*, 2012.

[11] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and Flexible Cellular Core Network Architecture," in *Proc. ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013.

[12] J. Cho, B. Nguyen, A. Banerjee, R. Ricci, J. Van der Merwe, and K. Webb, "SMORE: Software-defined Networking Mobile Offloading Architecture," in *Proc. ACM SIGCOMM Workshop on All Things Cellular: Operations, Applications, and Challenges*, 2014.

[13] Brent Hirschman, Pranav Mehta, Kannan Babu Ramia, Ashok Sunder Rajan, Edwin Dylag, Ajaypal Singh, and Martin McDonald, "High-performance evolved packet core signaling and bearer processing on general-purpose processors," *IEEE Network*, vol. 29, no. 3, 2015.

[14] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. Van der Merwe, and S. Rangarajan, "Scaling the LTE Control-Plane for Future Mobile Access," 2015.

[15] X. An, F. Pianese, I. Widjaja, and U. Günay Acer, "DMME: A Distributed LTE Mobility Management Entity, journal=Bell Labs Technical Journal, year=2012, volume=17, number=2,."

[16] Y. Takano, A. Khan, M. Tamura, S. Iwashina, and T. Shimizu, "Virtualization-Based Scaling Methods for Stateful Cellular Network Nodes Using Elastic Core Architecture," in *Proc. IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014.

[17] "Open Air Interface," http://www.openairinterface.org/.

[18] "Floodlight," http://www.projectfloodlight.org/floodlight/, 2015.

[19] "Intel Data Plane Development Kit for Linux*: Guide," http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/intel-dpdk-getting-started-guide.html.

[20] "Iperf3," https://iperf.fr/, 2014.