MME-FaaS
Cloud-Native Control for Mobile Networks

Sonika Jindal
sonijindal@gmail.com
University of Utah

Robert Ricci
ricci@cs.utah.edu
University of Utah

Abstract

The control plane for mobile wireless (eg. cellular) networks faces challenges with respect to scaling, robustness, and handling of bursty traffic. In this paper, we take a cloud-native approach to building a mobile control plane, employing a design that maps transitions of device state to serverless functions. Using a prototype of the LTE/EPC Mobility Management Entity (MME), we demonstrate how to architect a mobile control plane using serverless computing primitives. We demonstrate the practicality of this approach, which differs significantly from designs based on traditional telecom infrastructure.

CCS Concepts

• **Networks → Mobile networks**; *Control path algorithms*; • **Computer systems organization → Cloud computing**.

Keywords

Serverless, FaaS, mobile networks, MME, cloud-native

ACM Reference Format:
Sonika Jindal and Robert Ricci. 2019. MME-FaaS Cloud-Native Control for Mobile Networks. In *ACM Symposium on Cloud Computing (SoCC '19), November 20–23, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3357223.3362722

1 Introduction

Serverless computing provides an execution model in which a cloud provider manages the provisioning of resources dynamically, abstracting management and low-level infrastructure away from developers and operators. Services managed by the cloud provider include Function as a Service (FaaS), databases, networking, and security. For example, AWS offers Lambda (FaaS), S3 (storage), DynamoDB (database), and IAM (Identity and Access Management). FaaS is the "core" of serverless computing, providing a platform for customers to develop, run, and manage application functionality at the granularity of individual functions. The application developer leverages this model to deploy independent actions or business logic in the form of stateless functions. Functions are run in a sandboxed environment like containers or lightweight VMs, and any state that they may need can be stored in other service offerings such as databases. With the promise of inherent scalability, availability, pay-per-invocation, and higher developer velocity offered by serverless computing, it is an attractive choice for implementing some network functions.

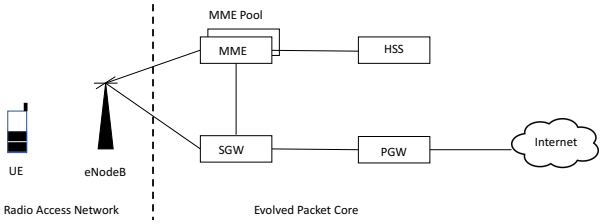

Figure 1: EPC Architecture

In this work, we examine how the control plane of LTE/EPC mobile network and future 5G networks can be implemented using serverless functions. The overall LTE network architecture (shown in Figure 1) consists of the Radio Access Network (RAN) and Evolved Packet Core (EPC). The RAN consists of eNodeB (eNB) "base stations" and User Equipment (UE) such as phones, IoT devices, etc. Functionality in the EPC is divided into a *control* plane and a *data* plane. The control plane handles authentication of UEs (through the Home Subscriber Service—HSS), attaching them to the network and managing their mobility (Mobility Management Entity—MME), and policy and billing (Policy and Charging Rules Function—PCRF). The main task of the control plane is to configure the data plane, which forwards the actual traffic between UEs and other networks such as the Internet.

We focus on the MME, as it is the key element of the control plane for managing the state of UEs. With the huge growth in IoT and mobile devices, the signaling traffic in the control plane has increased considerably and it is projected to increase further[9]. If an MME becomes overloaded, new devices cannot attach, existing devices may not be able to migrate to new base stations as they move, and the network degrades even if there is plenty of data-plane capacity. For this reason, commercial networks place a great emphasis on making these devices reliable and scalable, and vastly over-provision them (60-100% [28]) to respond to bursts in traffic. Traditionally, MMEs have been large, extremely expensive, dedicated hardware devices designed as monolithic services. A provider might typically have $O(10)$ MMEs to serve a country the size of the United States and must put in significant resources into ensuring that those few devices do not fail. Recently, providers have moved towards hosting these in NFV environments for cost and management, but the basic monolithic architecture has remained

SoCC '19, November 20–23, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6973-2/19/11…$15.00
https://doi.org/10.1145/3357223.3362722

the same, making it difficult (and expensive) to scale and replicate these services.

In this paper, we approach the problem of building a mobile control framework from the cloud-native direction, using serverless computing at the base. We build a simple model of some MME functionality, and make the following contributions:

(1) We evaluate the serverless platform for the features it promises as part of background research and motivation.
(2) We redesign MME as stateless functions (using remote datastore) running on the serverless platform.
(3) Then we evaluate the MME FaaS solution on OpenFaaS[7] and AWS Lambda[2] with different datastores.

2 Motivation and Related Work

Why Serverless? Most work on scaling the EPC takes the current monolithic design and changes it in some way to make it more scalable. These designs co-locate state and computation, that is, they tightly couple the data used to track elements such as UEs with the computational resources used to manage that state. This makes scaling challenging: when new resources are "spun up" for computation, complex synchronization is required to hand off responsibilities from one server to another. Load balancing is similarly difficult. Our approach starts from a different direction: in the style of many "cloud native" services, we decouple state from computation: state is stored in a scalable database, and computation is performed in serverless functions. Because most control logic updates the state of one UE, independently, this requires no coordination between functions, keeps database access simple, and allows us to scale and load balance without complex synchronization. It also has the benefit of making cost directly proportional to traffic, enabling easy scale-up for uncommon events and reducing the need for costly over-provisioning.

In PEPC [26] a state-driven packet core is designed such that all the state associated with a user is consolidated into a slice. In C3PO [8], the bottlenecks in mobile core packet performance are addressed by separating and independently scaling the data plane and control plane. In both cases, the UE state is local to the VNF. Some control plane specific solutions are DMME [11] and SCALE [15]; DMME splits the control plane processing task among multiple servers which maintain the UE mobility independently. The states are transferred between DMME replicas. In SCALE [15], the control plane functionality is divided into frontend load balancer which maintains the standard interfaces and a backend elastic MME processing cluster of VMs. While SCALE stores the UE information in the VM, DMME stores it to an external datastore. Keeping the state locally at VM can give fast access to the data, but using a cloud-based datastore is highly reliable and scalable. There have been attempts to build MMEs that use cloud services even more extensively, such as CNS-MME [10]. CNS-MME uses a stateless MME and runs as microservices or on a VNF pool. Since it uses these cloud services, CNS-MME provides auto-scaling and availability. These stateless microservices are instantiated behind an L7 load balancer which classifies packets and forwards them to respective VNF.

Our approach starts from a different direction: rather than starting from the telecom-inspired monolithic design of today's MME and *adapting* it to cloud technologies, we start from a *purely cloud-native design*. The work we present here represents an early proof-of-concept of this design; while it is not a complete EPC or MME implementation, by using a different fundamental design, further development of this idea will result in a control plane with substantially different scaling properties.

3 Background: EPC and Cloud Features

Telecom networks contain purpose-built infrastructure which is comprised of different devices coming from different third-party vendors. This results in a lot of integration effort which reduces the agility of deployments. Increasing capacity is costly and slow, as it involves the physical installation and configuration of new devices, and many services are designed in a monolithic fashion that makes it difficult to flexibly balance the load. While network operators are moving towards softwarization (VNF and SDN) and Infrastructure-as-a-Service to bring agility, IaaS requires overprovisioning to meet bursty traffic demands. Thus, using managed services and serverless platform presents an effective way to reap the benefits of the cloud, mainly scalability and cost-effectiveness. Next, we discuss relevant features of serverless computing, matching them up with control plane requirements.

Auto-scalability: We examined the scaling provided by AWS Lambda using the method proposed by L. Wang et al. [29]. Figure 2 shows the effect of increased concurrency on the total number of containers created for processing 10k requests. We observe that the system scales up to meet the increased requirement, which in turn increases the throughput. The latency of each request remains almost flat. In another experiment, the response time of 10k AWS Lambda requests (Figure 3) at a concurrency level of 200, show a few spikes which indicate the spawning of new containers to handle the increased load. Once the scale-up is performed, the response times stabilize. The serverless model makes scale-up/-down dynamic without pre-provisioning resources. Thus, it can be used for network functions if the scaling is seamless.

3GPP Timers: The 3GPP standards that govern LTE and EPC networks define timers which set the maximum limits for different procedures to execute. For example, a UE will wait up to 15s for an Attach request to succeed, as defined by timer T3410. Thus, we need to build an MME which can perform the processing within these time limits.

Independent and event-driven: Services in FaaS are written as event-driven functions, which aligns with the signaling model of control plane entities in 3GPP. Each instance works independently on the state of a UE; instead of building a monolithic application, FaaS allows the control plane to be built in the form of independently scalable functions. Rather than scaling the entire app, scaling can be done at a much finer-grained level in terms of the resources used and in time. Such scaling has the potential for significant cost savings.

5G: As per the specs for 5G core [1], a Service Based Architecture (SBA) is being followed for connecting the control plane entities using Service Based Interfaces (SBIs). In this design, the control plane is broken down into services communicating via HTTP [5, 6]. Thus, we expect that a serverless approach will be an even better fit for 5G than 4G.

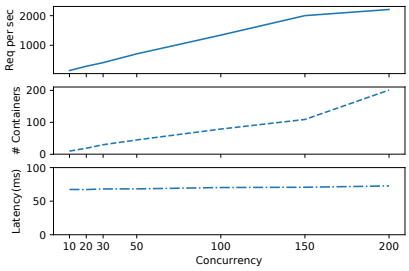

Figure 2: AWS RPS, Containers, and Latency with concurrency

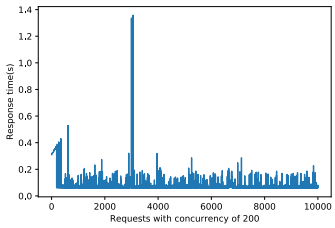

Figure 3: Response times of 10k reqs on AWS Lambda

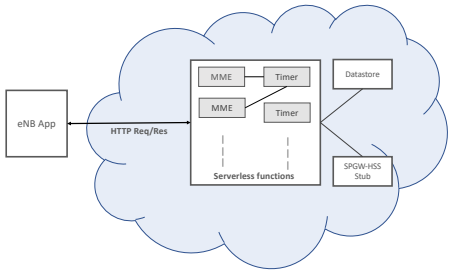

Figure 4: MME FaaS architecture and different components

4 Design

Having seen that a serverless platform can provide scalability with acceptable latency, we present the basic design features of a mobile control plane based on a serverless architecture:

Asynchronous response from MME: In a mobile network control plane, functions are required to process the incoming request and generate a new response/request to the caller or other network components. FaaS functions are not intended to be long-lived; it is in large part their short duration that makes it easy to flexibly schedule them. Thus, in our design, we use an asynchronous style: for example, when the MME needs to communicate with a data plane component, the function terminates after sending the message. When the reply from the data plane arrives, it is treated as a new event, causing a new FaaS function to be invoked, perform its processing work, send a reply to the UE, and then terminate.

Async mode also helps avoid API Gateway timeouts. Even though the function timeout is 15 minutes in AWS Lambda, the AWS API Gateway times out after 29 seconds. As a result, the calling entity might get a failure message even though the MME function is still waiting for some response from other entities. OpenFaaS supports async responses using Callback-URLs while AWS supports it using dead letter queues[21]. For simplicity, our prototype uses sync responses from AWS Lambda and async responses from OpenFaaS functions.

Separate processing and state: FaaS functions do not, themselves, hold state between invocations; this is another key factor in their scalability. Since the purpose of the mobile control plane is to manage connection states for UEs and the data plane, we must (like many serverless applications) access external state storage. We find that NoSQL [12, 17] datastores are ideally suited to this application: because each UE's state is separate, it fits this data model well, and NoSQL stores can achieve high scalability.

State passing: Using asynchronous function invocation with an external state store raises two challenges to state management: failures could leave stale state in the store, and function invocations are likely to repeatedly re-access the same state, increasing latency and cost. We address these problems by passing state along with our messages: For example, in Figure 5, when MME-4 sends a Location Update Request to HSS, it passes the state to HSS. The HSS sends Location Update Answer along with the state needed by MME-5. As a result, we don't have to generate another access to the datastore to retrieve MME-5's state and can delay writing state into the datastore until an entire operation has successfully completed. This is analogous to the "Continuation Passing Style"[30] used in many functional programming languages.

Optimistic concurrency control: While most state in the control plane is specific to a particular UE, there are a few cases where globally unique identifiers must be generated. We use "Optimistic Concurrency Control" [31] when generating these identifiers: we generate them randomly, then insert them using a method that succeeds only if the identifier does not already exist in the state store. If it does, we generate a new one and try again. The size of these identifiers is sufficiently large to make such collisions extremely rare.

Using these design principles, we built the prototype MME FaaS shown in Figure 4. Our early proof-of-concept implements only the Attach procedure (the most complex of all MME functions), and while it models the major functions of the messages, it is not standards-compliant.

5 Evaluation

We used both proprietary and open source platforms for our evaluations. For the serverless platform, we used OpenFaaS and AWS Lambda. For the datastore, we used CassandraDB [12] deployed on CloudLab [4] and AWS DynamoDB [17].

5.1 OpenFaaS Functions

We used CloudLab [16] servers to deploy CassandraDB and OpenFaaS for our first set of tests. We capture the time taken using both synchronous and asynchronous interactions with the function (Figure 6). In the synchronous mode, the eNB App sends requests in HTTP POST and receive responses in the body of the 200 OK replies. In the asynchronous mode, when the eNB App sends a request, it also sends a callback URL to OpenFaaS in the HTTP POST header. In both the cases, the latency is similar but we suggest Async mode for the reasons discussed in Section 4.

5.2 AWS Lambda

Our next set of tests uses AWS Lambda with AWS's managed DyanmoDB service and a CassandraDB instance that we managed ourselves. Figure 7 shows the spread of time taken by 1,000 requests. DynamoDB was deployed in the same region as the MME

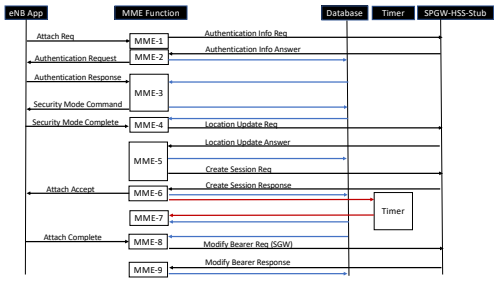

Figure 5: Message flow during Attach procedure

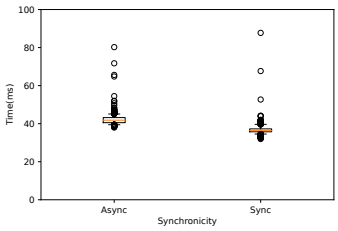

Figure 6: OpenFaaS Sync vs. Async

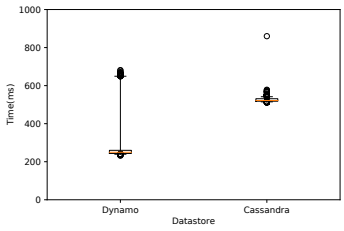

Figure 7: AWS Lambda with different datastores

FaaS Lambda (US-East-2). This resulted in faster query responses compared with CassandraDB, which was running at the CloudLab Utah cluster in the western US. With DynamoDB, the 95% latency extends up to about 650ms while the median remained at 250ms. This shows that DynamoDB can take a variable amount of time for different queries. With CassandraDB, although the median lies at about 500ms, all requests finish between 500ms to 600ms except one. Since all of these latency ranges fall well within the limit of the timer (15s), both solutions are viable options.

5.3 Concurrent execution

Next, we use different concurrency levels for requests. Figures 8 and 9 show latency for both CassandraDB and DynamoDB with MME FaaS running in AWS Lambda. In both tests, as the concurrency increases, the time taken to complete the procedure remains stable until a concurrency level of 100. This shows that at high concurrency, the databases don't scale well with our configuration: this could be due to our single-node CassandraDB and capacity mode for DynamoDB. Another observation is that CassandraDB has a higher latency for low concurrency, but as concurrency increases, its latency curve is flatter and less variable than DynamoDB.

The tables in DynamoDB (in Provisioned mode) are configured with RCUs and WCUs which limits the number of read/write access per second to the DB. A working configuration for our tests was 200 RCUs and 400 WCUs. From our sensitivity analysis (Figure 10) the latency of Attach procedure remains constant with RCUs 100 or more and WCUs 200 or more. Since we observed a few spikes in some tests with this config, we performed all our tests with 200 RCUs and 400 WCUs. With lower capacity units we observed many failures due to throttled requests at the datastore. For 10k requests, we saw 1523, 1010 and 502 failures with 25rcu/50wcu, 25rcu/50wcu with auto-scale and 50rcu/100wcu respectively.

5.4 Comparison with other implementations

To compare our latency results with other EPC solutions we performed UE Attach with OpenEPC [20] and OpenAirInterface [24], using the setup suggested in [19] and [18] to attach a simulated UE by running experiments on the PhantomNet [25] testbed. All devices ran in the same datacenter. OpenEPC takes 450ms to attach and OpenAirInterface takes 1800ms. In comparison, our single-datacenter configuration (OpenFaaS and CassandraDB) completes the Attach procedure in 42 ms, as seen in Figure 6. Because OpenEPC and OpenAirInterface are much more mature than MME-FaaS, and have more complete implementations of the 3GPP protocols, we don't claim that this shows that MME-FaaS is "faster"; what it does demonstrate, however, is that it can be competitive with existing implementations.

5.5 Cost

In order to show that this system can be deployed at a reasonable cost, we use data from our experiments and AWS's current pricing to estimate the costs of a full deployment. From Figure 5, an Attach involves 10 function calls (9 MME FaaS function + 1 timer function), 5 write calls to the database, and 4 read calls to the database. Call duration is 100ms (CloudWatch logs shows that AWS billed each function call for 100ms) and memory is 128MB, giving 0.0125 GB-sec per request. Thus, the cost for an Attach can be calculated as follows:

Resource	Cost per	Qty.	Cost (¢)
Requests	$0.20 per 1M [3]	10	0.0002
Duration	$0.06 per GB-hr [3]	3.5e-5	0.0002
API Gateway	$3.50 per 1M [13]	10	0.0035
DynamoDB R	$0.25 per 1M [14]	4	0.000100
DynamoDB W	$1.25 per 1M [14]	5	0.000625

The total cost *0.00465 cents* per Attach procedure, or ≈ $46.5 for 1M attaches. Since UE attach is infrequent[22]—we assume once per day per device—this is a very reasonable cost. The API Gateway is the biggest contributor (75%). Since UE context is small (about 400bytes), we can maintain 2.7M UE states without extra cost at DynamoDB (first 25GB free). We extrapolate costs using rough estimates for 1M control events based upon example signaling distribution from [23]. Using the AWS prices and the number of control messages per NAS procedure (from Table 1 of [27]), we get:

Operation	Fraction	Cost (¢)
Mobility and Handover	30%	0.00091
Paging	30%	0.00091
Service Request	20%	0.00144
Service Release	10%	0.00144
Attach	4%	0.00465
Detach	1%	0.00091
Bearer Act./Deact./Mod.	5%	0.00182

The total cost for 1M events for the above distribution is about $12.50. As per [23], 1M subscribers can generate 31,000 transactions per second during busy hours resulting in 111.6M transactions per busy hour for 1M subscribers: the cost for handling *busy hour* control traffic for 1M subscribers is about $1,400—again, reasonable given the national scale of these networks. Cost at off-peak hours

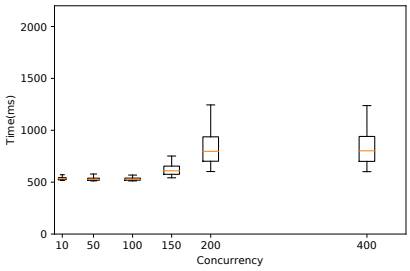

Figure 8: AWS Lambda and CassandraDB

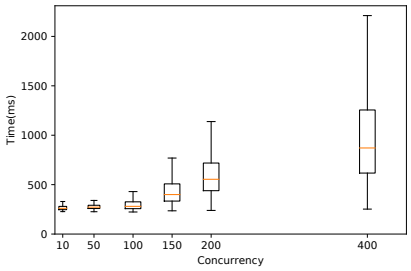

Figure 9: AWS Lambda and DynamoDB

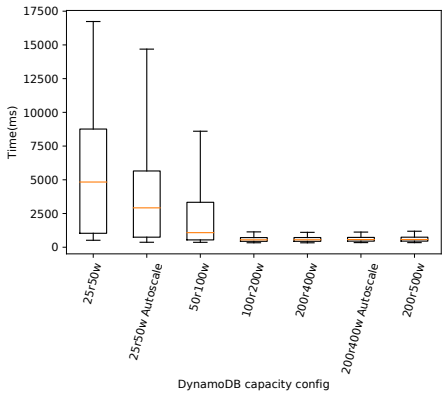

Figure 10: Sensitivity Analysis with DynamoDB

can be assumed to be lower, though no reliable utilization numbers are publicly available.

6 Conclusion & Future work

We presented the design of a cloud-native serverless control plane for mobile core networks, focusing mainly on the dependencies and architectural challenges related to serverless platforms. Given the expanding use of cloud technologies, there is clear incentive for telecom companies to use cloud-native architectures for supporting rapid traffic growth. MME-FaaS provides a base for further research and development of mobile network elements as cloud-native applications.

With our initial analysis of MME FaaS for cost, latency and scalability, there is a need to explore the serverless platforms further to support fully-functional data and control planes. More work needs to be done to break down the components to reduce the number of interactions, since there is a cost associated with each interaction in the serverless model. Strict latency and bandwidth requirements for the data plane need further investigation to fit the serverless model. Reducing cold start latency also needs more work at the platform side, and is an active research topic. There are areas for saving cost like instead of using an API gateway, and other event triggers can be examined. Another important area for research is availability: telecom grade-services need high availability of "five nines" which is much higher than offered by an IT-cloud (about 99.95% availability). Overall, our current work establishes a direction for further research on fulfilling the requirements of the telecom world using IT-grade cloud services.

All code used for this paper is available at:
`https://github.com/sonijindal/lte-mme-faas`

Acknowledgements

We thank Binh Nguyen, Junguk Cho, and Bozidar Radunovic for their discussions regarding early versions of this work. We also thank our shepherd, Lalith Suresh, and Ms. Jindal's thesis committee members, Kobus Van der Merwe and Ryan Stutsman, for their feedback. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1827940.

References

[1] 5G System; Principles and Guidelines for Services Definition. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3341.
[2] AWS Lambda. https://en.wikipedia.org/wiki/AWS_Lambda.
[3] AWS Lambda Pricing. https://aws.amazon.com/lambda/pricing/.
[4] CloudLab. https://www.cloudlab.us.
[5] Evolution of HTTP and DNS for 5G. https://www.netscout.com/news/article/http-and-dns-5g-world.
[6] HTTP in the 5G Core. https://apistraining.com/5g-core/.
[7] OpenFaaS - Serverless Functions Made Simple. https://github.com/openfaas/faas.
[8] Sprint Launches C3PO. https://newsroom.sprint.com/sprint-launches-c3po-open-source-nfvsdnbased-mobile-core-reference-solution.htm.
[9] State of the IoT 2018. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.
[10] P. C. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma, and A. A. Franklin. A cloud native solution for dynamic auto scaling of mme in lte. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7, Oct 2017.
[11] X. An, F. Pianese, I. Widjaja, and U. G. Acer. Dmme: A distributed lte mobility management entity. *Bell Labs Technical Journal*, 17(2):97–120, Sept 2012.
[12] Apache. Apache Cassandra. http://cassandra.apache.org.
[13] AWS. Amazon API Gateway Pricing. https://aws.amazon.com/api-gateway/pricing/.
[14] AWS. Pricing for On-Demand Capacity. https://aws.amazon.com/dynamodb/pricing/on-demand/.
[15] Arijit Banerjee, Rajesh Mahindra, Karthik Sundaresan, Sneha Kasera, Kobus Van der Merwe, and Sampath Rangarajan. Scaling the lte control-plane for future mobile access. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 19:1–19:13, New York, NY, USA, 2015. ACM.
[16] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, July 2019.
[17] DynamoDB. Amazon dynamodb. https://aws.amazon.com/dynamodb/.
[18] Emulab. End-to-end lte/epc network with openairinterface (oai) simulated enb/ue and oai's epc. https://wiki.emulab.net/wiki/phantomnet/oepc-protected/oai-sim-epc.
[19] Emulab. Openepc tutorial - using the profile driven phantomnet portal. https://wiki.phantomnet.org/wiki/phantomnet/oepc-protected/openepc-tutorial-profile.
[20] Core Network Dynamics GmbH. Openepc. https://www.openepc.com.
[21] AWS Lambda. Dead letter queues. https://docs.aws.amazon.com/lambda/latest/dg/dlq.html.
[22] Nokia. Managing lte core network signaling traffic. https://www.nokia.com/blog/managing-lte-core-network-signaling-traffic/.
[23] NSN. Signaling is growing 50% faster than data traffic. https://docplayer.net/6278117-Signaling-is-growing-50-faster-than-data-traffic.html.

[24] OpenAirInterface. Openairinterface. https://www.openairinterface.org.
[25] Phantomnet. The phantomnet wireless testbed. https://www.phantomnet.org/.
[26] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. A high performance packet core for next generation cellular networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 348–361, New York, NY, USA, 2017. ACM.
[27] A. S. Rajan, S. Gobriel, C. Maciocco, K. B. Ramia, S. Kapury, A. Singhy, J. Ermanz, V. Gopalakrishnanz, and R. Janaz. Understanding the bottlenecks in virtualizing cellular core network functions. In *The 21st IEEE International Workshop on Local and Metropolitan Area Networks*, pages 1–6, April 2015.
[28] searchnetworking. In mobile networks, sdn and nfv mean service orchestration. https://searchnetworking.techtarget.com/tip/In-mobile-networks-SDN-and-NFV-mean-service-orchestration.
[29] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 133–146, Boston, MA, 2018. USENIX Association.
[30] Wikipedia. Continuation-passing style. https://en.wikipedia.org/wiki/Continuation-passing_style.
[31] Wikipedia. Optimistic concurrency control. https://en.wikipedia.org/wiki/Optimistic_concurrency_control.