Orchestrating the Data-plane of Virtual LTE Core Networks

Rajesh Mahindra*, Arijit Banerjee†, Karthik Sundaresan‡,
Sneha Kasera§, Jacobus Van Der Merwe § and Sampath Rangarajan‡
*Uber Inc.
†Federated Wireless Inc.
‡NEC Laboratories America, Inc.
§University of Utah

ABSTRACT

Growing demand for data and increasing number of devices are drastically changing the scale of operation in mobile networks. Future services and business models require efficient provisioning with enhanced traffic management. It is hard to meet these requirements on today's mobile networks that are deployed over specialized hardware. While operators are keen to adopt NFV (Network Function Virtualization) to virtualize their networks, virtualized mobile network deployments face a few technical barriers. To address these challenges, we design `SCOPE` *that effectively applies concepts from SDN and distributed systems to realize NFV-based LTE core networks. Using centralized allocation,* `SCOPE` *effectively manages the resources across multiple telecom data-centers in a way to meet the traffic requirements. To enforce the computed allocations,* `SCOPE` *includes flexible and efficient mechanisms to configure the data-plane. With full compliance to 3GPP-based protocols,* `SCOPE` *ensures faster and cost-effective deployments. The efficacy of* `SCOPE` *is shown using a prototype implementation and large-scale simulations.*

I. INTRODUCTION

The rise of cloud computing will cause data traffic per smartphone to grow to 5GB in 2020 [1]. 25 billion connected devices are predicted by 2020 [2]. The coupled effect of the growth of devices and data will require mobile networks to operate at scales well beyond the capabilities of the current architectures. Additionally, a saturated voice market and limited long-term growth from data access requires mobile operators to expand to newer network-as-a-service business models. Enabling such models requires additional capabilities for rapid provisioning of network resources of both the radio access and the core network. In the context of LTE core networks, current specialized hardware based deployments will fail to cost-effectively meet these requirements [3].

Motivated by IT Clouds, such as Amazon Web Services that can provide high reliability at lower costs, operators are considering NFV [4] as the first step towards evolving their networks. The network functions would be deployed over a platform based on general-purpose hardware, enabling fast and agile provisioning. While operators do realize the benefits of NFV, one of the primary reasons for its slow adoption in LTE core networks has been due to the the lack of a comprehensive management framework. Recent works have proposed scalable NFV-based design for LTE control plane management [5], [6]. In this paper, we focus on the data plane management as data-plane is responsible for carrying all user traffic, and its efficiency directly impact user experience and operator revenue.

To bring NFV-based LTE deployments closer to reality, we design and implement `SCOPE`: A system that efficiently manages the compute and network resources of LTE networks across services; while providing primitives to perform policy-driven traffic management. `SCOPE` strives to meet the traffic requirements for both, *delay-sensitive* services and *elastic* services. Based on current traffic demand and resource availability, `SCOPE` allocates the network resources to each service acoss DCs to reduce provisioning costs. `SCOPE`'s design also includes flexible and efficient mechanisms to configure the data-plane, including both the standardized LTE-gateways and middleboxes.

Challenges: To support resource allocation at scale, `SCOPE` needs to devise algorithms that are light-weight and practical. Although the requirements of `SCOPE` seem similar to WAN resource management systems [7], [8], there are multiple challenges specific to LTE networks. First, such systems primarily consider the constraints of inter-DC bandwidth. However, with LTE networks consisting of several middleboxes, `SCOPE`'s resource allocation algorithm is jointly constrainted by inter-DC bandwidth as well as compute resources. Second, the presence of service-chaining in LTE, further complicates the problem. `SCOPE` employs overlay routing on the data-plane to ensure that the traffic of each service is routed across the appropriate middleboxes at the DCs according to the allocation computed for that service. However, to ensure a practical design for the data-plane, `SCOPE` need to solve a harder problem, where the compute resources to all the flows belonging to a service must be assigned along the same path.Finally, unlike regular middleboxes that employ IP-routing, the standardized LTE-gateways employ 3GPP-defined standard IP-tunnels to route user traffic across the gateways. The devices are statically

This work was done when Rajesh was an employee at NEC and Arijit was an intern at NEC.

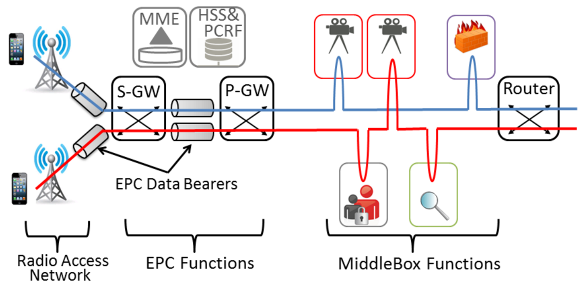


Figure 1: LTE Core Network Architecture.

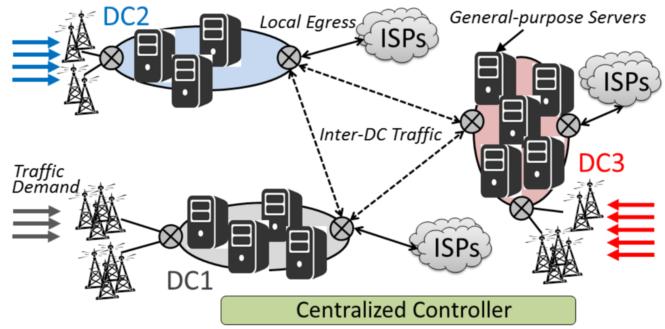


Figure 2: Future Core Network deployments with SCOPE.

assigned to the gateways, making it challenging for SCOPE to dynamically enforce the allocation for services across the data-plane. To ensure that SCOPE is readily deployable in today's LTE networks, it is critical to design SCOPE to deal with the well defined *standard interfaces* and protocols, and the *persistent sessions* of the LTE-gateways.

Contributions: SCOPE systematically overcomes the afore-mentioned challenges with the following contributions: First, SCOPE includes efficient algorithms to ensure a scalable central controller, that allocates resources of DCs across multiple services. While being computationally light-weight, the algorithms meet the constraints of compute and bandwidth resources. SCOPE is efficient in minimizing the end-end delays for delay-sensitive services, while providing max-min throughput fairness across the elastic services. Second, SCOPE employs a two-step resource allocation solution. The delay-sensitive services are handled first, where given the hardness of the problem, SCOPE compromises slighlty on the efficiency of VM usage to yield latencies that are very close to a genie lower bound. Then, it distributes the elastic services to efficiently utilize the unallocated bandwidth and compute resources. Third, SCOPE re-architects the LTE-gateways by splitting the control and forwarding planes to enable efficient offloading of LTE-gateway functions of selected services across DCs. We implement SCOPE on an end-to-end LTE testbed using OpenEPC [9] and the Click modular router [10]. Our prototype implementation on an LTE standards-compliant testbed shows the feasibility of SCOPE to work with existing protocols, making it readily deployable in today's networks. Our implementation is also supplemented with a large-scale systems simulator. Our SCOPE prototype illustrates how a practical data plane implementation involving standard-compliant LTE gateways and Click router improves peformance of interactive services with varying demands by intelligent resource allocation and flow routing. In a particular instantiation, SCOPE reduces the 99th percentile delay for interactive HTTP flows from 250msecs to 80msecs. We also show that SCOPE implementation results in better resource (VM) utilization, and improves throughput of elastic services.

II. Background: LTE Core Networks

The LTE network consists of the Radio Access Network (RAN) and the Evolved Packet Core (EPC) as shown in Figure 1. The RAN includes eNodeBs (or basestations) that serve the user devices (UEs), and the EPC consists of both the control-plane entities that *manage* the devices and data-plane entities that *route* the data traffic. While the main control-plane element is the MME (Mobility Management Entity), the data-plane functions comprise of both standardized LTE-gateways and generic middleboxes:

LTE-Gateways: The packets from the eNodeB are routed through two gateways, namely the Serving Gateway (SGW) and the Packet Data Network Gateway (PGW). The SGW's primary function is to maintain the data path as the UE moves across a set of eNodeBs. The PGW is the Internet gateway, it both assigns an IP address to the UE and routes its data traffic to and from the external networks. The data traffic of each UE is carried over a *bearer*, a logical transport channel between the UE and the PGW. In the core network, the user traffic belonging to a data bearer is carried over a tunnel using the GTP-U [11] headers.

Middlebox Functions: The LTE core network also consists of several middleboxes to perform various functions, e.g., Firewalls, Proxies, etc. Unlike the SGW/PGW,operators are free to deploy custom protocols to route IP traffic flows across the middleboxes. Since not all flows will require service from the same set of middleboxes, routing has to ensure that the flows are sent to the appropriate middleboxes in the correct order or a *Service-Chain*. Figure 1 depicts 2 service-chains, the blue flow is composed of a video transcoder and a firewall, while the red flow is composed of a parental controller, a video transcoder, and an intrusion detection service.

Key Differences: Middleboxes [12] only hold *state* for active flows or connections. However, the SGW and PGW persistently hold *state* for a UE. Once a UE registers with the network, the LTE mobility protocols ensure that the UE is managed by the same SGW and PGW as the UE cycles between *Idle* and *Active* modes. Hence, the traffic flows of a UE are always routed through the same SGW and PGW.

III. SCOPE: Overview & Architecture

SCOPE **Deployment:** The goal of SCOPE is to scale the LTE core network deployments to enable future services by leveraging concepts from NFV, SDN and distributed systems. SCOPE carefully instruments these concepts within the constraints posed by LTE networks through the design of efficient data-plane management algorithms. SCOPE is applicable to future LTE deployments with multiple DCs as shown in Figure 2; such that each eNodeB is directly connected to the closest

DC in physical proximity, referred to as the local DC for that eNodeB. To achieve the goals of improved network utilization and traffic QoS, SCOPE employs a centralized controller, with the global-view of the traffic demands and the network topology. The controller periodically assigns both the compute (VM) resources and the inter-DC bandwidth resources across traffic flows. The basic unit of compute resource allocated by SCOPE is a VM. Each DC is assumed to have a fixed number of physical servers, provisioned based on the long-term average traffic statistics. A DC can host a given number of VMs; such that each VM is allocated the same amount of CPU and memory. Each DC is inter-connected with other neighboring DCs using either leased or dedicated links. There may be multiple links between a given pair of DCs, but SCOPE treats them as a single logical link with capacity equal to the aggregate capacity of the individual links.

Traffic Types: To ensure scalability, the traffic is represented in the units of a *service*: group of flows belonging to an entity with the same: (i) QoS traffic type and (ii) Service chain. Driven by typical mobile application traffic characteristics, SCOPE supports 2 traffic types: Interactive and Elastic. Since interactive flows are delay-sensitive, SCOPE strives to minimize their end-to-end delays, while meeting their traffic demand. In the case of elastic flows, SCOPE strives to maximize the aggregate throughput while striving to be fair. SCOPE keeps track of a moving average of the traffic demand for each *service*, originating from eNodeBs at every DC. In addition, at every DC, SCOPE periodically measures the average delay from the DC to the end-server for flows belonging to interactive *services*. The aggregate delay measurements per *service* are maintained by SCOPE. Using these measurements, SCOPE can effectively reduce the end-end delays for interactive *services*, rather than just the delay within the core networks, which may not be representative of the end-end delays.

Inter-DC Offloading: While trying to meet the requirements of the *services*, the traffic from a subset of *services* that originate from the eNodeBs connected to their local DC, may be egressed through a neighboring or remote DC, as shown in Figure 3. These *services* may be either partially or fully processed at the remote DC. For instance, *service*#1 in Figure 3, which has a service-chain consisting of the functions S,P (SGW,PGW) and the network functions V,F and T, is routed locally through DC1 to its end server(s). However, *service*#2 is partially processed and egressed through the remote DC3, while *service*#3 is completely processed and egressed through the remote DC2. Intuitively, it may seem best to allocate all the VM resources for a particular *service* at the local DC. However, SCOPE selectively *offloads* processing and egresses certain *services* through remote DCs for the following reasons: (i) By striving to offload selective flows, either for partial or complete processing of functions in their service-chains, to remote DCs, SCOPE can perform fine-grained multiplexing of compute resources across DCs. (ii) There are a few reasons why routing certain flows from the egress point of their local DC may not be optimal since: (a) Well-known triangle inequality violations in interdomain

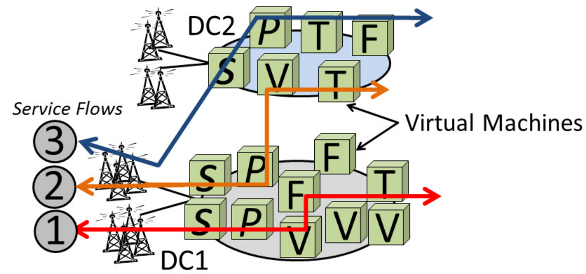


Figure 3: Service Flow allocation in SCOPE.

routing, (b) In the context of certain IoT and MEC(Mobile Edge Computing) applications, the operator hosts the servers (e.g., transcoding, analytics etc.) for the services within the network. Such servers may not be deployed in all the DCs for all the *services*, and (c) Several content-providers and enterprises have a relatively centralized presence. The servers or cloud resources for the traffic of such entities may have better peering with a subset of the operator's DCs.

A1. Architecting Flexible LTE-Gateways: To solve the challenges pertaining to enforcing the resource allocations across the data-plane, specifically for the LTE-gateways, SCOPE re-architects the LTE-gateway implementations. It is important for SCOPE to have the ability to re-assign flows of *services* across these gateways. There are 2 key challenges to enable such flexibility: Firstly, the re-assignment of flows across LTE-gateways has to happen at the level of UEs since UEs are managed by a specific SGW and PGW. Moreover, the assignment of UEs is performed prior to the initiation of the application traffic. Hence, it is not possible to know the *service* type for the flow(s) that will be initiated by a UE. Secondly, it is necessary to ensure that the protocols and interfaces of the SGW-PGW with other entities like eNodeBs, MME, PCRF etc. are supported. Such a design choice ensures incremental deployment alongside existing EPC platforms and avoids the need to design, deploy and test novel protocols. Finally, to support the expected growth of IoT devices, the re-assignment of UEs across the LTE-gateways should be scalable and light-weight.

To overcome the above challenges, SCOPE makes 2 key design choices. Firstly, SCOPE classifies devices (or UEs) into 2 types depending on their *service* usage: *(i) Managed UEs*: In this scenario, there is one-to-one mapping between the UE and the *service*, i.e., the devices are specifically designed to access or provide a particular *service*. For instance, several IoT-based *services*, certain vertical-MVNOs and enterprise *services* fall in this category. *(ii) Unmanaged UEs*: This category is representative of smartphones, tablets that are not tied to a particular *service*. These devices can access several different OTT-based *services*. By classifying UEs, SCOPE can dynamically offload the processing of SGW and PGW functions for the traffic from certain *services*, that have managed UEs, to remote DCs.

Secondly, to enable dynamic (re)-assignment of UEs across the gateway resources, SCOPE decouples standard interfaces and MME based gateway selection. SCOPE achieves such decoupling by architecting the SGW/PGW cluster as two separate logical functions as shown in Figure 4. The figure shows

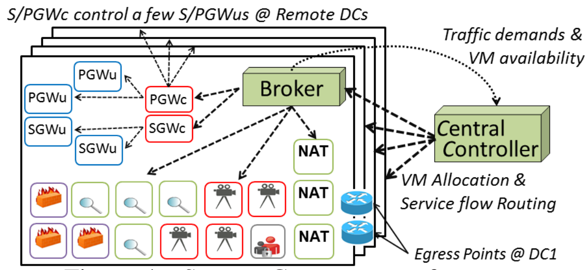


Figure 4: System Components of SCOPE.

a single instance of an SGW/PGW pool realized by SCOPE that would be deployed at a particular DC. *(i)* SGWc/PGWc: The control function that maintains the standard interfaces with other entities of the EPC. Using the SDN paradigm, the SGWc/PGWc pool manages several SGWu/PGWu VMs and dynamically (re)-assigns UEs across the SGWu/PGWu VMs. *(ii)* SGWu/PGWu: The data-plane function that performs GTP-based packet forwarding in addition to other functions, such as QoS enforcement, charging.

A2. Multiplexing across DCs: Since the assignment of SGWu/PGWu VMs has to be done before the UE generates traffic, SCOPE does not offload the processing of SGWu/PGWu functions to the remote DCs for the flows belonging to unmanaged UEs. The processing for the flows of unmanaged UEs is offloaded only after the SGWu/PGWu functions in their service-chains. This is because, for such UEs, their flows can only be classified into the appropriate *service*, once they generate traffic, that happens after the assignment of SGWu/PGWu VMs. Since, there is one-to-one mapping between a managed UE and its corresponding *service*, SCOPE has the flexibility to offload the flows of such UEs. To enable offloading for the flows for managed UEs, the SGWc and PGWc pool of each DC are connected to a subset of SGWu and PGWu VMs in remote DCs. In addition to assigning 2 SGWu and PGWu VMs in the local DC, SCOPE assigns an additional SGWu and PGWu VM in the appropriate DC for managed UEs. For simplifying configuration, all the UEs belonging to the same *service* are assigned a SGWu and PGWu VM in the same remote DC. The selection of the remote DC for a *service* depends upon its QoS class: *(i)* In the case of an interactive *service*, the remote DC is selected such that the end-to-end delay for the traffic of the *service* will be the lowest. *(ii)* In the case of an elastic *service*, the remote DC is selected at random. This capability gives additional flexibility to the SCOPE controller when assigning SGWu/PGWu resources to selective *services*.

IV. SCOPE: Resource Ochestration

SCOPE's resource orchestration component aims to cater to the requirements of both interactive and elastic traffic by addressing the challenges specific to resource allocation and data-path routing. Figure 4 shows the system components of SCOPE, with a *controller* that coordinates the resource allocation across a pool of DCs. At each epoch (several mins), the *broker* at every DC j updates the controller with the following: (i) total number of VMs (V_j), (ii) the estimated current demand (t^i_j) of each *service* (i) obtained from the SGWc/PGWc. The SGWc/PGWc aggregate the demand statistics obtained from the SGWu/PGWu VMs, (iii) available bandwidth (B_{jk}) between every pair of DCs (j, k), (iv) in the case of a *service* with managed UEs, the location (remote DC) of the SGWu/PGWu VMs that are pre-assigned with the UEs of the *service*, and (v) in the case of interactive *services*, the average end-end delays (d^i_{jk}) that are measured when the flows of the *service* (i) are routed through the local DC (j) and each of the remote DC (k) in the DC pool.

S 1 : SCOPE_**Allocation()**

Inputs:

t^i_j: flow demands for *service* i originating at DC j

$\mathcal{S}$: # of *services*, $\mathcal{D}$: # of DCs, C_m: Capacity of MB m

V_j: # of VMs available at DC j, B_{jk}: Max. BW available between DCs j and k

d^i_{jk}: End-end delay perceived by the interactive *service* i originating at DC j and egressed through DC k

I^i_{ml} is an indicator variable that is equal to 1 if the middlebox function m lies before the function l in the Service Chain for *service* i.

Interactive: $r^i_j, V^{\text{rem}}_j, B^{\text{rem}}_{jk}$ = **AllocInteractive(**$t^i_j, d^i_{jk}, V_j, B_{jk}$**)**

Outputs:

r^i_j: The DC where the VM resources are allocated for *service* i originating at DC j

$V^{\text{rem}}_j, B^{\text{rem}}_{jk}$: The unallocated VM and inter-DC bandwidth resources at each DC j, given as input for elastic allocation.

Elastic: $z^i_{jk}(l)$ = **AllocElastic(**t^i_j, V^{rem}_j, B^{rem}_{jk}**)**

Outputs:

$z^i_{jk}(l)$: Fraction of t^i_j assigned to flow i, with VM resources allocated at DC j until MB l and MB l onwards in the service chain are allocated at DC k

The controller uses the above information of traffic demands and topology to compute the allocation for each *service* (Section B1). In addition to compute resources, the allocation specifies the data-plane path for each *service*: (i) whether the flows of the *service* should be egressed at the local DC or a specific remote DC, depending upon the location of the VMs allocated to that flow. (ii) if selected for offload, the point in the service chain at which the *service* should be routed to the remote DC. In case of a *service* with managed UEs, the offload point could include any function, including the SGWu or the PGWu. However, if a *service* is offloaded at the SGWu or the PGWu VM, the specific remote DC is pre-determined based on the location of the SGWu-PGWu VMs that contain the state of the UEs of that *service*. In the case of *services* with unmanaged UEs, the offload point is a function that lies after the SGWu/PGWu in the service chain. The allocations computed by the controller are sent to and enforced by the individual brokers. To ensure that the routes of the *services* are configured appropriately by the broker, SCOPE employs a programmable overlay-routing framework (Section B2). In the case of LTE-gateways, the broker enforces the allocation through the SGWc-PGWc VMs.

B1. Computing Resource Allocations: SCOPE employs Algorithm S1 in the central controller to allocate both the VM and the inter-DC bandwidth resources among the *services*. It adopts a two-step process, where it first assigns resources across the network of DCs to the interactive service flows to

S 2 : Original Formulation for Interactive Service Flows

MINIMIZE $\sum_i \sum_j \sum_k \sum_l y^{il}_{jk} \cdot (d^i_{jk}/\min_k\{d^i_{jk}\})$; Subject to:

1: $\sum_i \sum_k \sum_l (y^{il}_{jk} \cdot \sum_m \frac{t^i_j \cdot I^i_{ml}}{C_m} + y^{il}_{kj} \cdot \sum_m \frac{t^i_k \cdot I^i_{lm}}{C_m}) \le V_j; \quad \forall j$

2: $\sum_i \sum_l (y^{il}_{jk} \cdot t^i_j + y^{il}_{kj} \cdot t^i_k)] \le B_{jk}; \quad \forall j,k$

3: $\sum_k \sum_l y^{il}_{jk} = 1 \;\; \forall i,j;\; y^{il}_{jk} \in \{0,1\}$

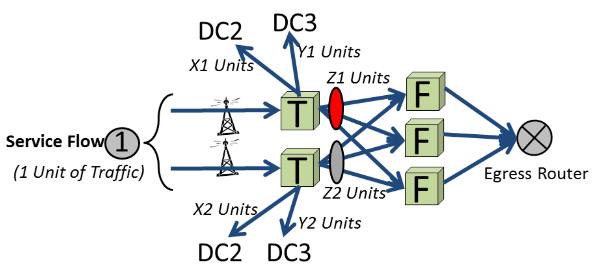


Figure 5: Design Challenge for SCOPE's resource allocation.

minimize their end-end delays; then it allocates the remaining resources to the elastic service flows to maximize their throughput. We now elaborate on these two steps.

B1.1 Interactive Services (Problem Formulation): The formulation for allocating resources to the interactive *services* is given in S2. Note that the delay d^i_{jk} for a *service* i originating at DC j and egressed at DC k is computed as the sum of the avg. delay between the DCs j & k and the avg. delay from DC k to the end-server for the flow i. The objective is to minimize the aggregate of the (normalized) flow delays, where the latter normalizes each flow's delay to its minimum delay possible $(d^i_{jk}/\min_k\{d^i_{jk}\})$. Constraints 1 & 2 define the VM resource constraint at each DC and the inter-DC bandwidth constraints B_{jk} respectively. The output decision variable, y^{il}_{jk} is 1 if t^i_j rate is allocated to the *service* i originating at DC j, such that the VM resources until the middlebox function l will be allocated at DC j and the functions beyond l in the service chain of i will be allocated at DC k. If all the VM resources are allocated locally, $y^{il}_{jk} = 1$ for $j = k$. The formulation for the elastic *services* would be similar to S2 with a different objective. Since the output variable y^{il}_{jk} is binary, the formulation becomes an Integer LP, making it computationally intractable. Due to the unique nature of the problem and the dual constraints of VM and bandwidth resources, we cannot apply known techniques like knapsack and bin-packing. While multi-dimensional knapsack may leave out certain flows from allocation, multi-dimensional bin-packing algorithms cannot work with a fixed number of VMs.

It is possible to relax the problem to translate it to a LP formulation by allowing the values of y^{il}_{jk} to be fractional. However, this can lead to $y^{il}_{jk} > 0$ for more than two values of l or k for the same *service* i, where the allocation of the *service* is either split at multiple functions or across multiple DCs or both. However, such an allocation becomes hard to realize in a practical system. This is because, although all the flows of the same *service* have to be processed by the same network functions, different flows may get mapped to different VMs of the same function to ensure efficient load balancing. Hence, to ensure proper fractional splits, the VMs of the same function will need to co-ordinate among each other as depicted in Figure 5. In this scenario, 1 unit of traffic from *service*#1 has to be split at function T with X units to DC2, Y units to DC3 and remaining locally at DC1. Assuming there are 2 VMs of the function T at DC1, they will need to co-ordinate to ensure that the fractions $X1$ and $X2$ add up to X units and fractions $Y1$ and $Y2$ add up to Y units. To ensure a light-weight, practical routing-plane (Section B2), SCOPE assigns resources such that (i) all the traffic of a *service* follows the same routing-path and (ii) once offloaded to a remote DC at any point in its service chain, the *service* is not re-routed to another DC and is egressed at that DC.

B1.2 Interactive Services (Algorithm): While we use the LP relaxation of S2 to obtain a genie lower bound on delays, SCOPE devises a simple but efficient greedy algorithm that operates in two steps as follows. In the outer loop (Steps 5-23), the resources are allocated at the DC level. Here, the DC (whose services are not allocated yet) that yields the minimum utility is selected and resources are allocated to all the *services* that originate from that DC (Step 18). The utility for a DC U_j is the aggregate of the utilities of the individual *services* that originate at that DC and is computed in the inner loop. The utility for a *service* i originating in DC j and offloaded to DC k is given by its corresponding normalized end-end delay, namely $U^i_{jk} = \frac{d^i_{jk}}{\min_k\{d^i_{jk}\}}$. The inner loop (Steps 7-16) determines the utility delivered by an un-allocated DC in the set π as follows. It follows an iterative procedure, where at each step the *service* that yields the smallest utility is selected for resource allocation, subject to the bandwidth and VM constraints of the allocation (Step 12); and the procedure continues till no further *service* can be selected. Note that at this stage, the *services* are not actually assigned resources; the steps are simply used for computing the utility delivered by the DC if it were chosen for allocation (Step 13) After every DC in the set π is visited, the DC with the smallest utility is selected (Step 18) and the allocation for all the *services* of that DC is committed by updating their their VM and inter-DC bandwidth usage in the appropriate DCs (Step 22). The output is stored as r^i_j that specifies the index of the DC where the *service* i originating at DC j would be processed and egressed.

Remarks: Two points worth noting are: (i) SCOPE allocates resources for an interactive *service* either completely at the local DC or a remote DC but not at both (i.e splitting the allocation across DCs at arbitrary points in the service chain). While this leads to a small efficiency loss in the usage of VMs across DCs, it greatly simplifies the allocation complexity. Further, this efficiency loss is negligible and is made up the elastic *services* (as we detail next), whose allocations are allowed to be split across DCs at arbitrary points in the service chain to better utilize the VM resources. (ii) The inter-DC bandwidth constraints coupled with the VM constraints can result in scenarios, where a solution is infeasible, i.e. the traffic demand of all interactive *service* cannot be satisfied. In our

S 3 : Function AllocInteractive(t_j^i, d_{jk}^i, V_j, B_{jk})

$v_j^i = \sum_l \frac{t_j^i}{C_l}, \quad \forall(i,j)$
$\pi \leftarrow \{\mathcal{D}\}$
% Outer Loop
for $d \in [1 : |\mathcal{D}|]$ **do**
  $V_d^T = V_d; \quad B_{dk}^T = B_{dk} \quad \forall k$
  % Inner Loop: Within every DC that is unassigned
  for $j \in \pi$ **do**
    $\mathcal{A}_j \leftarrow \emptyset, U_j = MAX$
    for $i \in \mathcal{S}$ **do**
      for $k \in [1 : |\mathcal{D}|]$ **do**
        $U_{jk}^i = d_{jk}^i / \min_k\{d_{jk}^i\}$
      end for
      %Select only among the flows that satsfy the VM and BW constaints
      $i^*, k^* = \arg\min_{(i,k) s.t.\ i \notin \mathcal{A}_j}\{U_{jk}^i\}$
      $U_j = U_j + U_{jk^*}^{i^*}; \quad \mathcal{A}_j \leftarrow \mathcal{A}_j \cup i^*$
      $V_{k^*}^T = V_{k^*}^T - v_j^{i^*}; \quad B_{jk^*}^T = B_{jk^*}^T - t_j^{i^*}$
      $r_j^i = k^*$
    end for
  end for
  $j^* \leftarrow \arg\min_j U_j$
  $\pi \leftarrow \pi - j^*$
  % Update the resources used by the *services* of DC j
  for $i \in \mathcal{S}$ **do**
    $k = r_{j^*}^i$
    $V_k = V_k - v_{j^*}^i; \quad B_{j^*k} = B_{j^*k} - t_{j^*}^i$
  end for
end for

S 4 : Formulation AllocElastic(t_j^i, V_j^{rem}, B_{jk}^{rem})

MAXIMIZE λ; Subject to:
1: $\sum_k \sum_l z_{jk}^{il} \geq \lambda; \quad \forall(i,j)$
2: $\sum_i \sum_k \sum_l (z_{jk}^{il} \cdot \sum_m \frac{t_j^i \cdot I_{ml}^i}{C_m} + z_{kj}^{il} \cdot \sum_m \frac{t_k^i \cdot I_{lm}^i}{C_m}) \leq V_j^{\text{rem}}; \quad \forall j$
3: $\sum_i \sum_l (z_{jk}^{il} \cdot t_j^i + z_{kj}^{il} \cdot t_k^i)] \leq B_{jk}^{\text{rem}}; \quad \forall j,k$
4: $\sum_k \sum_l z_{jk}^{il} \leq 1 \quad \forall(i,j)$

evaluations, we find that this happens rarely, and when it does, it happens in only a very small number of *services*İn such cases, SCOPE allocates VM resources to those *services* at their local DC. This may incur more VMs than what is allocated at the DC. However, we find this additional VM allocation to be less than 10%, which can potentially be borrowed from the slack compute resources available at a DC.

B1.3 Elastic *services*: Once the interactive *services* are satisfied, the allocation for elastic *service* is performed using the LP formulation in S4. The idea is to compute the maximum fraction (λ) of demand (t_j^i) for each *service* such that ($\lambda \cdot t_j^i$) will be allocated to the flow, thereby resulting in a max-min formulation. The t_j^i for an elastic flow can either be defined as the estimated traffic demand or the maximum data rate allocation according to SLAs with the service provider. The output of the LP: z_{jk}^{il} gives the fraction of t_j^i that will be allocated to the *service* i originating at DC j, such that the VM resources until the middlebox function l will be allocated at DC j and the functions beyond l in the service-chain of i will be allocated at DC k. The sum of the fractional allocations z_{jk}^{il} should be atleast λ for a particular flow (Constraint 1). To conform with SCOPE's routing plane implementation, in

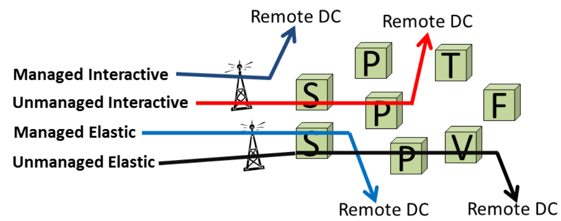


Figure 6: Offloading options for the various traffic types.

case the allocations for a *service* are split across multiple DCs or at multiple points in the service-chain, the algorithm simply selects the largest fractional allocation for the *service*. If needed, SCOPE re-assigns the resources un-used by the (un-allocated) smaller fractions of that *service* to other *services* assigned to the same DC. In most of our evaluations, we see at most 2-3 out of 500 *services* receiving such split allocations. Note that by employing a fraction allocation SCOPE allows resource allocation to elastic *services* to be split across DCs at any arbitrary middlebox function in their service chain.

B2. Configuring the Data-path: Once the controller computes the allocations for an epoch, the broker at each DC configures the appropriate number of VMs per middlebox function and the routing-plane for each *service*. In the case of the LTE-gateways, the broker configures the SGWc/PGWc VMs. First, SGWc/PGWc VMs provision the allocated number of SGWu/PGWu VMs. Second, if the flows of a *service* have to be offloaded to a remote DC for processing of the SGW & PGW functions, the SGWc/PGWc configures itself to select the SGWu/PGWu VMs at the appropriate remote DC for the UEs belonging to that *service* (Section III:A2). The next time a UE belonging to that *service* becomes active, the SGWc/PGWc selects a SGWu/PGWu VM at the remote DC. For UEs that belong to *services* that do not need to be offloaded to remote DCs in an epoch, the SGWc/PGWc selects the master or replica SGWu/PGWu VM based on their instantaneous load in the epoch (Section III:A1). Note that offloading of the SGW & PGW functions to remote DC is limited to *services* with managed UEs (Section III). The supported offload options for SCOPE for the different types of *service* are shown in Figure 6. In the case of interactive and elastic *services* with unmanaged UEs, the SGWu & the PGWu VMs have to be allocated at the local DC as shown in the figure. For elastic flows with managed UEs, either the SGWu or the PGWu VM or both could be offloaded to a remote DC. However, in the case of interactive flows with managed UEs that are offloaded to a remote DC, both the SGWu & the PGWu VMs have to be offloaded since SCOPE does not split the allocation at arbitrary functions for interactive *services*. Note that these constraints would be accounted for in the resource allocation.

B2.1 Overlay-Routing: To enable routing after the PGWu VMs, SCOPE uses an overlay-routing framework to configure the network-paths for each *service* as shown in Figure 7. Doing so ensures (i) deployments independent of the network hardware, potentially even in public clouds, (ii) scaling the network functionality is easier when physical servers are added

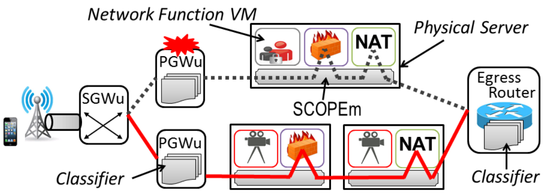
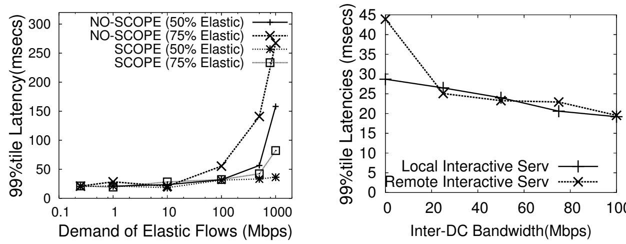


Figure 7: Service-Chain routing in SCOPE.

since the framework is well-distributed. Overlay routing is achieved using labels, so that the complex step of *service* classification is performed only once at the edge nodes, while the intermediate nodes perform label-switching. As shown in the figure, the VMs of the PGWu and the egress-router perform classification of the uplink and downlink traffic respectively. In the intermediate nodes, the routing is agnostic to the middlebox functions. Each physical server is equipped with a shim-layer: SCOPEm that performs both the VM selection and next-hop routing. When a PGWu or an egress router receives a packet, the 5-tuple is used to classify the flow into the appropriate *service* based on information stored in its database. The next step is to select a VM for the next function in the service-chain of the *service*. While there are multiple ways to select a VM among the active VMs for a function, including hashing, the current prototype of SCOPE randomly selects a VM for a flow for simplicity. Once the VM is selected for a flow, an entry is made to ensure that subsequent packets of the flow can be routed appropriately. The packet is then encapsulated with an outer header containing the IP-address of the physical server that hosts the selected VM. Before forwarding the packet, the PGWu or the egress router adds a label with the following information: (i) `serviceId`: Id of the *service*, (ii) `currVMId`: Id of the VM that will process the packet, (iii) `lastfnId`: Id of the function in the service-chain, after which the packet should be forwarded to a remote DC for further processing, (iv) `remoteDCId`: Id of the remote DC where the packet should be forwarded after local processing. When the SCOPEm module receives a packet from an interface, it strips the outer header and forwards the packet to the VM based on the `currVMId`. When the SCOPEm module receives the packet back from the VM, it performs either of the following: (i) if the packet needs further processing in the same DC, it is forwarded to the appropriate physical server. The header is re-attached to the packet after `currVMId` is updated with the Id of the selected VM that will process the packet. Similar to the edge nodes, SCOPEm selects a VM randomly among the active VMs for the next function in the service chain of the flow. Once a VM is selected, an entry is stored in SCOPEm to ensure that subsequent packets of the flow are routed to the same VM. (ii) if the packet needs further processing in a remote DC (i.e., if the VM with Id:`currVMId` belongs to the function with Id: `lastfnId`), it is forwarded to the default gateway for that DC (`remoteDCId`). The headers are re-attached to the packet after the field:`currVMId` is updated to represent the index of the function rather than a particular VM. The specific VM for the next function is chosen by the ingress router at the remote DC. This ensures that the VM-level information does not have to be shared across DCs.



(a) Traffic Isolation (b) Interactive Routing

Figure 8: Efficacy of SCOPE on a real LTE Prototype

V. Implementation and Evaluation

Our prototype implementation of SCOPE consists of the systems components shown in Figures 4, 7. The implementation of the LTE-gateways was done using OpenEPC, which is a LTE Release 9 compatible EPC network consisting of standard EPC entities and an eNodeB emulator. We modify the SGW and PGW implementations of the openEPC to implement the SGWc,PGWc,SGWu and PGWu. In our prototype, a few bytes in the IMSI are reserved for the `serviceId` for SGWc/PGWc to map service flows to UEs. We implement the routing framework: SCOPEm using the Click [10] modular router. The GRE tunnel module in Click was modified to build the functionalities specific to SCOPEm. To implement the classifiers at the edge nodes, a click module is included in the PGWu and egress routers. The classifier encapsulates each data packet with a GRE header and inserts the label to each packet with the relevant information in the *key* field of the GRE header. We use several user-level middleboxes, such as iprelay, Squid HTTP-proxy as generic functions. And finally, the central controller and broker are implemented as user-space applications. The algorithm for the allocation of interactive *services* was written in python, while we use GLPK linear programming solver [13] to solve the LP to allocate resources for the elastic flows. To perform the necessary rounding for the allocation of elastic *services*, we implemented the post-processing module in python. The controller encodes the allocation information as JSON format and sends it to the brokers. The broker at a DC configures both (i) SGWc/PGWc VMs and (ii) appropriate routing rules, at the granularity of each *service*, in the SCOPEm shim layer.

6.1 Prototype Evaluation: It would be ideal to verify the efficacy of SCOPE with real LTE data using large-scale testbed. However, operators are too sensitive about their network operations. Additionally, arranging an end-end LTE testbed is challenging and costly. Our license agreements with OpenEPC prevent us from deploying the source code over public clouds to experiment at scale. Although we use a small-scale prototype, it provided certain key design insights and verifies the feasiblity of SCOPE within the protocol constriants of the LTE data-plane. We use web-traffic to emulate traffic for interactive *services* and iperf to generate traffic for elastic *services*. We employ the Vaurien TCP proxy to introduce random variability in the network, such as packet loss, delay.

E1. Traffic Isolation: To show the efficacy of traffic management with SCOPE, we set up an experiment with 3 DCs, such that the number of VMs is 10 in each DC and the interDC-bandwidth fixed to around 200Mbps between each DC pair. In one of the DCs, we setup around 25 HTTP flows (interactive *service*), that access a given website from the internet. We measure the end-end delay perceived by the HTTP flows for 2 scenarios: 25 elastic flows (50% of the total flows) and 35 elastic flows (75% of the total flows). We repeat the experiment with increasing demand from the elastic traffic and plot the 99%tile end-end delays for the interactive flows for the case with SCOPE and with no-SCOPE, that basically routes and processes all the flows at the local DC. Refering to Figure 8(a), as the traffic for the elastic flows is increased beyond 100Mbps, the delays for the interactive flows drastically increases to about 250 miliseconds with no-SCOPE. While, SCOPE is effective in allocating the resources for the elastic traffic at the other 2 remote DCs, since it gives priority of allocation to the interactive *services*. With SCOPE, the delay for interactive flows is below 100 milisec for both the scenarios. In the figure, inter-DC bandwidth of zero corresponds to the case where all services are processed locally even with SCOPE.

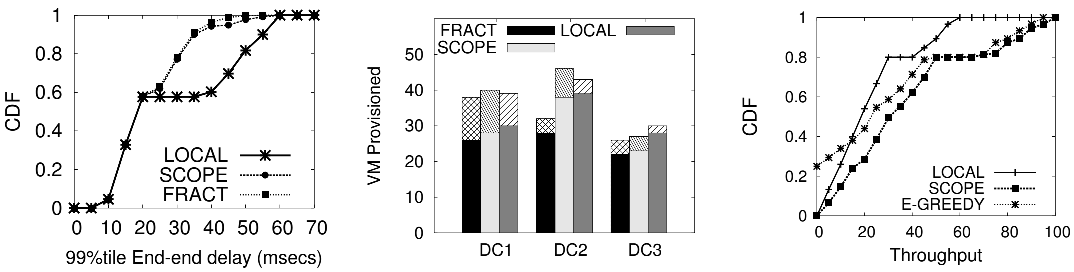
E2. Effective routing for interactive: In addition to providing isolation from elastic flows, SCOPE strives to allocate resources to the interactive *services* along a network-path that minimizes the end-end delays. To show the efficacy of SCOPE in minimizing delays, we conduct an experiment with a similar setup to E1. At each DC, we run a mix of 2 interactive *services*, such that each interactive *service* has 20 HTTP flows at each DC. The interactive *service*, named *Local* service, is setup such that the end-end delays for the flows of the *service* are lowest when routed through the local DC. Such a *service* is representative of entities, such as Google that have distributed content networks expected to have good peering with the operators DCs. On the other hand, the *Remote* service, is such that the end-end delays for the flows of the *service* are lower when egressed at a subset of the DCs. Such a *service* is representative of most enterprise networks that are expected to be relatively more centralized. To show the efficacy of SCOPE's offload mechanism for interactive *services*, we measure the end-end delays for the flows of the *Remote* service with different values of the inter-DC bandwidth. As shown in Figure 8(b), the delays for such flows reduces from about 45 milisecs to around 20 milisecs as the inter-DC bandwidth is increased. As the bandwidth increases, it facilitates SCOPE to allocate resources for higher number of interactive *services* at remote DCs that would minimize their delays. Furthermore, we notice that the offload of *Remote* service also reduces the end-end delays for the *Local* service. The reason for this effect is that the DCs were relieved of VM resources when offloading the *Remote* service, ensuring more resources for the *Local* services.

6.2 System Simulations: To show the efficacy of SCOPE in larger setups with higher number of VMs and *services*, we built a custom event-driven simulator in Python.

S1. End-End Experiment: We setup the experiment with 500 *services*, including both interactive and elastic *services* in a network of 3 DCs; such that each DC has a compute capacity of 40, 44 and 30 VMs and the inter-DC bandwidths are 1Gbps. To show the design choices behind the algorithms employed by SCOPE, we compare SCOPE's allocation with the following schemes: (i) LOCAL that allocates the resources for all *services* in the local DC, (ii) FRACT that allocates resources for the interactive *services* according to the original formulation S2 in Sec III, but relaxed to LP. FRACT may result in splitting the allocation for a *service* across DCs, but is highly efficient in packing the VM allocations across *services* and (iii) eGREEDY that allocates resources to elastic *services* using a greedy algorithm. During allocation, eGREEDY iterates through all the elastic *services* and selects the *service* which gives the best utility. The utility is given as log(throughput), resulting in a proportional allocation across flows. We compare the performance of the above algorithms with SCOPE based on the end-end delays of interactive *services*, the number of VMs provisioned at every DC and the throughput received by the elastic *services*. As shown in Figure 9(a), the CDF of the end-end delays obtained with SCOPE is very close to the delays obtained with FRACT. This result shows that SCOPE is as effective in minimizing delays for the interactive *services* as the relaxed LP formulation S2 in Sec III, without splitting the allocation of a *service* across multiple DCs. The delays with SCOPE are higher than those obtained with LOCAL, since LOCAL does not have the ability to route interactive *services* across DCs to minimize delays.

We now plot the number of VMs allocated by the algorithms: LOCAL, FRACT and SCOPE for both interactive (as denoted by figure legends) and elastic *services* (corresponds to the upper shaded bar) in each DC. Refering to Sec III, to ensure a practical design of the routing plane, SCOPE employs a greedy algorithm to allocate interactive *services* to ensure integral allocation along the same network-path for each *service*. Hence, as compared to FRACT that splits allocations for the interactive *services*, SCOPE compromises on efficient VM packing for interactive *services*. However, as seen in Figure 9(b), the VM usage with SCOPE is only about $10-15\%$ higher than FRACT. Moreover, the number of VMs allocated with SCOPE is marginally higher compared to the VM usage with LOCAL. The key difference is that SCOPE is able to better assign and fit the resources across the *services* on a network-wide level as opposed to LOCAL.

Finally, in the same experiment, we also plot the CDF of the throughput received by the elastic *services* with LOCAL, SCOPE and eGREEDY. Clearly, from Figure 9(c), SCOPE out-performs LOCAL, since it is more efficient in multiplexing the resources for both interactive and elastic *services* as opposed to LOCAL, that can only use the unused resources at the local DCs. Since, SCOPE adopts an LP formulation (S4 in Sec III-B1.4) for elastic *services*, the fractional allocations for elastic *services* are rounded by selecting the largest fraction. However, despite this approximation, SCOPE outperforms the allocation performed by eGREEDY. We observed that the eGREEDY algorithm was



(a) Interactive Delays (b) VM Provisioning (c) Elastic Throughput

Figure 9: Simulation Results

ineffective in packing the elastic *services* across the residual VMs and bandwidth unallocated after the allocation for the interactive *services*.

VI. Related Work

WAN Research: At a high level, the data plane management aspects of SCOPE is similar to the traffic engineering systems for inter-DC wide area networks (WAN) [7], [8]. They have shown that centralized resource management coupled with a flexible software-defined data-plane leads to efficient utilization of network resources in large scale network operations. While SCOPE is motivated by the concepts and key findings of such systems, the resource allocation algorithms and the configurable data plane design are instrumented to the requirements of mobile networks, resulting in a standards-compliant, ready-deployable efficient data-plane.

Core Network Research: A couple of studies [3], [14] have proposed devising new architectures and protocols to improve efficiency of mobile networks. Although effective, such approaches are costly to deploy as they require a complete overhaul of the existing network architecture, requiring modifications to the eNodeBs as well. SCOPE supports current standards, ensuring cost-effective, incremental deployments. On the industrial front, a few companies, such as Alcatel Lucent and NEC have launched software based EPC components [15], [16], including the LTE-gateways. However, they have migrated the implementations from hardware-based systems to VMs, with focus on packet-processing optimizations. SCOPE is complementary, since it proposes an enhanced data-plane management layer to allocate the virtual resources efficiently.

Middlebox Management: Several prior research works have focussed on the problem of efficiently routing flows of middlebox functions in the context of data-center networks [17], [18], [19]. These works focus on different aspects of middlebox deployments, such as executing middleboxes in the cloud [17], software-defined routing for legacy middleboxes [18] and virtualized middleboxes [19]. However, such works are limited in scale within the context of data center networking. More importantly, they do not have to consider the constraints that are unique to LTE networks.

VII. Conclusion

To ensure faster, incremental deployments of NFV-based mobile core networks, we present the design and implementation of SCOPE. Firstly, SCOPE employs central resource allocation to effectively allocate the compute and bandwidth resources of core networks across multiple DCs. Secondly, SCOPE includes primitives to perform policy-based prioritized traffic management across services with different requirements. Finally, SCOPE re-architects the LTE-gateway implementations to ensure flexibility in enforcing the allocations across different services. Our implementation on a end-end LTE Core network testbed and large-scale simulations demonstrate both, the efficacy of SCOPE and its feasibility within the context of today's mobile networks.

References

[1] Ericsson Traffic Report. http://goo.gl/iMzj9M.
[2] Forecast: The Internet of Things. http://www.gartner.com/newsroom/id/2636073.
[3] Mehrdad Moradi, Wenfei Wu, Li Erran Li, and Zhuoqing Morley Mao. SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture. In *ACM CONEXT*, 2014.
[4] AT&T Domain 2.0 Vision White Paper, 2013. http://tinyurl.com/p4uv3s3.
[5] Banerjee et. al. Scaling the lte control plane for future mobile access. ACM CONEXT, 2015.
[6] Xueli et. al. An. DMME: A Distributed LTE Mobility Management Entity. *Bell Labs TR*, 2012.
[7] Jain et. al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM CCR*, 2013.
[8] Hong et. al. Achieving high utilization with software-driven wan. In *ACM SIGCOMM CCR*, 2013.
[9] OpenEPC. http://www.openepc.com/.
[10] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The click modular router. *SIGOPS Oper. Syst. Rev.*, 33(5):217–231, 1999.
[11] 3GPP Spec for GTP-U v1. http://www.3gpp.org/DynaReport/29281.htm.
[12] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *NSDI*, 2013.
[13] GLPK (GNU Linear Programming Kit). https://www.gnu.org/software/glpk/.
[14] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. Softcell: Scalable and Flexible Cellular Core Network Architecture. In *ACM CONEXT*, 2013.
[15] NEC Virtualized Evolved Packet Core. http://tinyurl.com/nuefq28.
[16] The Journey to Packet Core Virtualization. http://resources.alcatel-lucent.com/asset/174234.
[17] Sherry et. al. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM CCR*, 2012.
[18] Qazi et. al. Simple-fying middlebox policy enforcement using sdn. In *ACM SIGCOMM CCR*, 2013.
[19] Sekar et. al. Design and implementation of a consolidated middlebox architecture. In *NSDI*, 2012.