PROTEUS: A Network Service Control Platform for Service Evolution in a Mobile Software Defined Infrastructure*

Aisha Syed
School of Computing, University of Utah
Salt Lake City, Utah
aisha.syed@utah.edu

Jacobus Van der Merwe
School of Computing, University of Utah
Salt Lake City, Utah
kobus@cs.utah.edu

ABSTRACT

We present PROTEUS, a mobile network service control platform to enable safe and rapid evolution of services in a mobile software defined infrastructure (SDI). PROTEUS allows for network service and network component functionality to be specified in templates. These templates are used by the PROTEUS orchestrator to realize and modify service instances based on the specifics of a service creation request and the availability of resources in the mobile SDI and allows for service specific policies to be implemented. We evaluate our PROTEUS prototype in a realistic mobile networking testbed illustrating its ability to support service evolution.

CCS Concepts

•**Networks** → **Mobile networks; Network management; Network manageability;** *Programmable networks;*

Keywords Software Defined Infrastructure; Service Evolution; Orchestration; Data Centric; Templates; Mobile Networks

1. INTRODUCTION

Cloud computing platforms have revolutionized the manner and rate at which web-based services and applications are designed, deployed and evolved. "As-a-service" cloud offerings continue to proliferate and large scale web based services have famously developed systems and mechanisms that allow them to introduce new services or service features on a daily basis [44, 51].

In contrast, in the networking domain, service deployment and evolution move at a glacial pace. Historically, mobile networking companies and standards bodies produce a new generation of services (i.e., a new "G," 2G, 3G, etc.) approximately every ten years [48]. We argue that future mobile networks will not be able to meet the varied application demands without having more *varied service offerings* and being fundamentally more *evolvable*. In the context of this paper we define a network service as the collection of protocols, mechanisms and network elements that realize well defined functionality that is being offered to and/or used by customers, end-users and applications. Future mobile networks will almost certainly still provide the kinds of services we use today, i.e., broadband access, voice and data services, etc. In addition, however, demands from the continued growth in machine-to-machine communication [30], the emergence of Internet of things (IoT) [29], and anticipated low latency/high reliability tactile services [42, 4], e.g., vehicle safety, augmented reality, industrial automation, and the proliferation of wearable devices, will unlikely be satisfied by a single monolithic service abstraction. Compared to current mobile network services, future services might have very different network requirements and dynamically changing and varied coverage needs. For example, future mobile networks will need to support a large number of IoT devices [9]: around 26 billion by 2020, according to some estimates [26]. The massive number of devices and the fact that IoT communication patterns are expected to be quite different from today's human-to-human and human-to-machine type communication, suggests that fundamental changes in the service abstractions supported by future mobile networks might be needed. Conversely, augmented reality applications might have very different needs. For example, an augmented reality application in use by a national retail store [32] might require high bandwidth, low latency access to compute resources at the network edge and would require the enabling network service to be available at store locations across the country (i.e., broad coverage). However, if the augmented reality application is only enabled inside the retail stores, the actual service coverage needed might be quite sparse. Similarly, a geo-aware inter-vehicle safety application [4], even if it is deployed to have broad coverage, will require very low latency vehicle-to-vehicle and vehicle-to-cloud communication. However, the resulting communication patterns would be "localized" and changing very rapidly as determined by the set of vehicles in a small geographic area.

We argue that the only way in which these diverse and evolving needs of future mobile network can be realized is through the adoption of various "soft" technologies—e.g., software defined networking, cloud computing, and network function virtualization so that the mobile infrastructure can become a *software defined infrastructure* (SDI). The success of a mobile SDI will critically depend on the ability of the *con-*

*Instructions for accessing the PROTEUS code and using it in the PhantomNet testbed are available here: https://wiki.phantomnet.org/wiki/phantomnet/proteus

MobiCom'16, October 03-07, 2016, New York City, NY, USA

DOI: http://dx.doi.org/10.1145/2973750.2973757

trol platform of the infrastructure to deal with the complex, mobile-specific needs of this environment. Toward this end, in this paper we present our work on the PROTEUS platform. PROTEUS is a mobile network service control platform to enable safe and rapid deployment and evolution of services in a mobile network. The role of the PROTEUS platform is to allow the creation and evolution of services within a mobile SDI. With PROTEUS, service designers specify the details of network services in *service templates*. A service template captures the network components involved in realizing the service, the relationships between such components, the way these components need to be configured and managed, and service specific policies associated with the service. In PROTEUS, network components are realized as *virtualized network functions*, the capabilities, dependencies and functionality of which are similarly defined in *component templates*. Given this base functionality, network service instances can be dynamically instantiated. Specifically, the PROTEUS *orchestrator* uses service and component templates and combines that with resource availability in the infrastructure and a specification of the desired characteristics of a service instance, e.g., in terms of scale and coverage, to dynamically realize service instances.

The dynamic nature of this environment has a number of important consequences. First, much like in a cloud platform, PROTEUS allows for different instances of a service to be instantiated in parallel [13]. Second, because different service instances can be executing in parallel, PROTEUS makes it possible to instantiate variants of the same service, or indeed radically different service instances. For example, a network service capable of supporting real time vehicle safety applications [4] might require specialized handover and group communication mechanisms, which might warrant its realization in a network service instance (or slice) separate from (regular) mobile broadband access services. Third, PROTEUS allows the current single mobile provider environment to be "opened up." Specifically, PROTEUS enables third-party service and application providers to readily deploy services and applications on the mobile SDI, i.e., a hosted mobile service environment. Fourth, PROTEUS inherently supports the evolution of network services and features. A new service or service feature might involve modifying a service template to change the configuration of a component, or providing a new network component or replacement implementation of a component and updating the service template to make use of it. For example, a basic mobile broadband access service might be enhanced to provide offloading functionality to a cloud platform for selective low-latency applications [14, 23]. Another example might involve replacing a network component, e.g., a mobile network control plane element like the mobility management entity, with a refactored implementation that is inherently more amenable to a dynamic SDI environment [16].

Our goal with PROTEUS is to *enable* the realization of diverse mobile network services as well as the service evolution and changes required by a future mobile network environment. As such, in the first instance, PROTEUS is analogous to the cloud control software, e.g., OpenStack [6], that turns a datacenter into a cloud platform. That is, having a cloud-like control stack being applied to a mobile SDI would enable multiple network service instances, as well as specialized service instances, to be deployed and would enable the multi-player environment described above. While cloud-like, the PROTEUS control platform is different from existing cloud platforms. First, PROTEUS deals with mobile network services, providing the means to specify and instantiate network services, with the implied protocol and component dependencies and providing mobile specific primitives and mechanisms. Second, PROTEUS operates across a mobile SDI that is geographically highly distributed [3] and involves dealing with different underlying technologies. (Some of these technologies are not readily virtualizable, e.g., base stations, implying that placement decisions in PROTEUS needs to be service, location and network element aware.) More importantly, however, as described above, PROTEUS needs to accommodate the evolution of mobile network service instances. This is particularly challenging. In PROTEUS we deal with this by requiring virtualized network components to expose *management primitives* that the PROTEUS orchestrator can invoke to facilitate change. Specifically, PROTEUS network components are equipped with lifecycle management functions to allow the components to be dynamically deployed, decommissioned or migrated, or scaled up or down according to service needs.

We make the following contributions:

- We present an architecture and prototype implementation of the PROTEUS platform. PROTEUS allows for network service and network component functionality to be specified in templates and follows a data-centric approach whereby services can dynamically adapt based on network or service conditions. Specifically, the generic PROTEUS orchestrator uses service and component templates and information from the data store to instantiate service instances based on the specifics of a service creation request and the availability of resources in the mobile SDI.

- We illustrate the feasibility of our approach to dynamically deploying network services by realizing three different services with PROTEUS, namely a broadband mobile network service based on the standards-based evolved packet core (EPC) [40], a selective mobile offloading service [23] and a personalized mobile-aware cloud service [52]. PROTEUS allows for the automated instantiation, configuration and modification of these services. PROTEUS can instantiate multiple standalone instances of these services (with instance specific scale and coverage), and can safely evolve an existing service (e.g., provide selective offloading of traffic in an existing broadband service) allowing for the sharing of network elements between compatible services.

- We evaluate our PROTEUS implementation in a realistic mobile networking testbed [15]. We perform both end-to-end and component level performance evaluations of the PROTEUS prototype and perform a functional evaluation of PROTEUS' ability to support hosted service instantiation, management and evolution.

2. CONTEXT AND RELATED WORK

2.1 PROTEUS Overview

Mobile software-defined infrastructure. With PROTEUS we assume a future mobile infrastructure that will be fully software-defined. Figure 1 depicts such a mobile software-defined infrastructure (SDI). Specifically, we assume a *core* network infrastructure that will consist of sets of distributed cloud computing platforms interconnected with SDN-enabled networks. Such a software-defined core infrastructure can be

combined with a "conventional" RAN consisting of mobile devices and base stations. However, the increasing popularity of cloud-RAN approaches [21], where remote radio heads (RRHs) feed baseband signals to a pool of cloud-based baseband processing units (BBUs), would be synergistic with a mobile SDI approach, allowing shared use of cloud resources for core or radio access network functions. Ultimately we expect the software-defined nature of such an infrastructure to extend into the radio access network (RAN), e.g., as software defined radio (SDR) functionality, allowing great flexibility in both mobile devices and base stations. For the purposes of the work reported here, however, we will assume a conventional RAN and concentrate on a mobile core network realized on SDI.

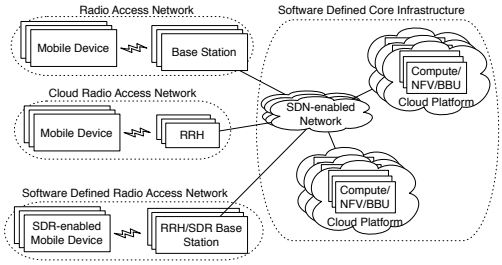


Figure 1: Mobile software-defined infrastructure (SDI)

Services in a mobile SDI. Given a mobile SDI, the most basic purpose of Proteus is to realize mobile network services on the infrastructure. Figure 2 illustrates this with a number of example services. (We use nomenclature associated with current long term evolution (LTE)/evolved packet core (EPC) mobile networks.)

First, Figure 2(a) shows the base case of realizing a vanilla LTE/EPC mobile broadband service [40]. With Proteus, a service template will capture the mobile network-specific components involved (i.e., eNodeB, MME, SGW and PGW), the fact that some of these components (the eNodeBs or base stations) are physical devices (associated with cell towers in specific physical locations), while other components can be realized via NFV instances, the fact that the mobile specific components are to be connected via an IP network, etc. Figure 2(a) also depicts the fact that in a mobile SDI, multiple separate instances of a mobile network service might be operational, each in its own logical "slice."

Figure 2(b) shows an example where the same basic mobile broadband service is realized. However, in this case, one of the mobile network components—the MME—is being replaced by an alternative realization, MME'. For example, previous work described the refactoring of an MME into a more elastic NFV realization involving an MME load balancer (MLB), which maintained the existing protocol interaction with the other mobile network components and used a pool of MME processing entities (MMP) to achieve elasticity [16]. Such a refactored component might expose new primitives (e.g., associated with its load balancing functionality) that would be captured in its component template.

In Figure 2(c) we show a configuration where the IP substrate component is replaced with an SDN-enabled network fabric to enable offloading to an in-mobile-network cloud platform. Such a mobile edge computing infrastructure is attracting significant industry attention and can enable new mobile network services [1, 43]. Realizing such services in Proteus would again involve the introduction of new component implementations, i.e., the SDN fabric and the cloud platform, and the need to create new service and component templates that describe the new service and its relationship with existing mobile network components. Because offloading services will need to work with and share components (and resources) associated with existing services, service templates for such services will need to describe policies and mechanisms that will allow new services to be introduced safely.

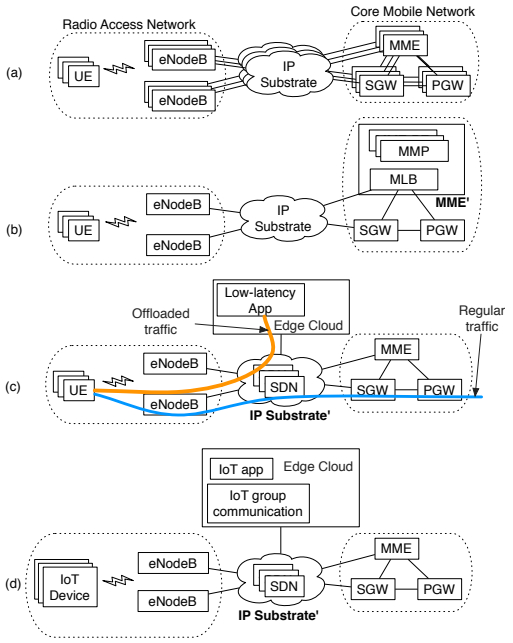


Figure 2: Example mobile services

Finally, Figure 2(d) shows a hypothetical example network service specialized for IoT devices. For example, automotive safety applications might require ultra low-latency [38], sophisticated group communication [24, 36] and increased reliability [19]. Overlaying such a service abstraction on top of the existing human-to-machine mobile network service might not be feasible and expected IoT traffic patterns have been shown to be detrimental to current mobile network control and data plane functions [17]. As suggested by Figure 2(d), such a specialized IoT network service would be best realized in a mobile SDI as a separate network service "slice," existing in a parallel with existing network services.

As mentioned earlier, for our work we consider a network service to be the collection of protocols, mechanisms and network elements that realize functionality that is used by customers, end-users and applications. Network services are therefore different from applications (or application services) that might be implemented on a cloud platform. For cloud application services, network functionality (e.g., best-effort connectivity) is assumed, and the application service builds its logic on this base. In a Proteus mobile SDI, the SDI is used to *create this basic network functionality*, and further provide *novel network service abstractions* (beyond best-effort connectivity) to better suit the needs of future mobile applications and application services. (The needs associated with mobile networking and especially the needs of anticipated future mobile network applications also differentiate our work from related efforts in the wired network domain. We highlight the mobile specific challenges in § 2.2 and consider related work in § 2.3.)

PROTEUS *role players.* Services like those described above could be instantiated by a single service provider to mimic today's single provider environments. However, we reason that the real power of a mobile SDI and the PROTEUS service control platform comes from the ability to allow different service providers to offer services on the same infrastructure, i.e., a hosted mobile service environment. As such we identify different role players in our environment. First, the *infrastructure provider* owns and operates the mobile SDI and the PROTEUS service control platform to manage and control the deployment of services on it. *Service providers* offer services on the infrastructure based on service templates registered with the platform. *Application service providers* make use of the services instantiated on the platform. For example, our hypothetical IoT-specific service might be provided by a third-party service provider, using the mobile SDI operated by an infrastructure provider and application service providers might deploy different IoT applications on the edge cloud using the IoT service.

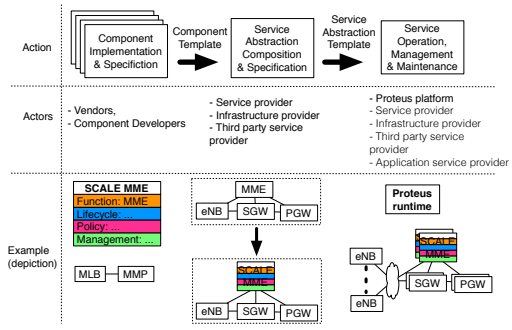


Figure 3: PROTEUS workflow

PROTEUS *workflow.* Deploying a service with PROTEUS involves a three step workflow. As shown in Figure 3, the first step involves developing the technology to implement the (new) network components to be introduced. The resulting component template would include a specification of the function of the component and how it relates to other components, lifecycle and management primitives as well as any component specific policies. With a component template in hand, the next step involves developing a service template that specifies how the new component can be used, together with other components, to realize the new service. These first two steps are performed offline and the resulting service (and component) templates are provided to the PROTEUS runtime system, which can then be used to realize services based on the template.

Example services in PROTEUS. To illustrate the feasibility of our approach, we extend three existing network services to work within PROTEUS. Specifically, we use:

•*Standard LTE/EPC:* An NFV-based realization of a standard LTE/EPC broadband access service [40] forms an interesting base case for our purposes.

•*SMORE:* SMORE is an edge cloud offloading architecture for mobile networks [23]. SMORE makes use of an SDN substrate at an aggregation point between the EPC core and the RAN to realize offloading. This service example raises challenges for PROTEUS related to overall management, resource placement optimization and scaling up or down in the presence of multiple clouds.

•*MobiScud:* MobiScud is an offloading architecture that realizes a personalized cloudlet-like service [52, 49]. In support of augmented reality applications, e.g., cognitive assistance applications [22], a personalized virtual server [50] is associated with users of the application. To maintain the low-latency requirements of this environment, in MobiScud, the personal virtual server is migrated from one edge-cloud to another, in concert with a user's mobility in the physical world. This coordination between the mobile network (dealing with user mobility) and the service (migrating the virtual server to "follow" the user) represents challenging interactions between the role-players in a PROTEUS environment.

2.2 Design Principles, Challenges, Solutions

Design principles. The fundamental design principle we employ in PROTEUS is *strong separation and clean abstractions* between the service-agnostic functionality provided by PROTEUS and the mobile SDI, and the service-specific functionality that can be realized on that. As we outline below, variants of this basic design principle are manifested in addressing the PROTEUS challenges by carefully combining *service specific* and *service agnostic mechanisms and primitives*.

Hosted service multiplicity and diversity: The most fundamental challenge for PROTEUS is how to allow the instantiation of multiple, potentially radically different services by one or more service providers, and to do that in a rapid and safe manner.

PROTEUS addresses this challenge by providing *service agnostic automated orchestration* in the platform itself, while capturing *service specific* details in *service and component templates*, which are used by the orchestrator to realize the services. This partitioning of functionality strikes a delicate balance between designing a template specification that can capture the service and components completely, and an orchestrator, which, although it is service agnostic, still needs to manage resources in the infrastructure on behalf of services.

Diversity in implementation and requirements: A related set of challenges involves the diversity of possible implementations of network components and the diverse needs of service instances. In a mobile SDI network, elements can be realized as conventional dedicated hardware components, as virtualized network functions, as a collection of virtualized components, or as a combination of these. Similarly, different service instances, even of the same base service, might have very different requirements in terms of coverage, delay and network capacity. For example, the placement of an offloading server for a low latency application would naturally want to minimize the latency between users and the server. In a mobile SDI, these problems are exacerbated by the fact that the environment is distributed and not homogeneous.

We address these challenges through a number PROTEUS solutions. First, we employ *polymorphic templates* for both component and service specification. Templates are taken as *types* and allow inheritance and specialization, such that, for example, both a physical and virtual eNodeB inherit from a generic eNodeB template type. Second, we enable sophisticated resource placement by having the PROTEUS platform be inherently *data centric* in that it employs a knowledge centric datastore that captures the current and past status of the infrastructure. Data from the datastore is exposed to services to enable *dynamic data centric service specification*. Specifically, the datastore exposes a query interface that can

be used by service templates to define resource placement constraints, that can also take into consideration variables that represent current infrastructure status contained in the infrastructure datastore. We maintain PROTEUS' service agnostic orchestration in the face of service-specific constraints through a generic constraint solving layer as part of the orchestration that provides *service agnostic resource placement functionality*. For example, this functionality allows a service template to specify that the compute server it gets allocated should be located close (e.g., in terms of round-trip-time (RTT) measurements retrieved from the datastore) to the eNodeB that has had the largest number of its customers for the past two days (based on traffic measurement analyses retrieved from the datastore).

Mobility specific requirements: Dealing with mobile specific requirements represent another set of challenges. Compared to data center and other wired networks, mobile networks are inherently more complex, dealing with both wireless (in the RAN) and wired (in the core) networks. First, mobile networks have a greater variety of network elements with specialized functionality and with a myriad of protocols (in both control and data plane) tying the components together. Second, current mobile network architectures maintain significantly more state, including device and/or user specific state. Third, mobile network infrastructures represent a superset of wired network deployments: central offices (similar to data center networks), wide area core mobile network (often built as an "overlay" on regular packet wide area networks), highly distributed "edge" networks in the form of the radio access network.

We employ a variant of our basic design principle to address these challenges, by carefully combining *service specific* and *infrastructure specific mobility aware mechanisms*. Specifically, PROTEUS' service templates allows *logical service specific topologies* to be specified, capturing dependencies between mobile network components and their constituent protocol interactions. PROTEUS' orchestrator takes these logical service-specific (but generic) specifications and maps them to realizable infrastructure specific *actual service instances* in the SDI. (This might take into account data-centric placement directives described earlier.) Once mapped, the orchestrator employs a *service specific realization plan* from the service/component template to instantiate and configure the service.

Safe service evolution: A key requirement (and challenge) for PROTEUS is the need to enable service evolution with minimal disruption. Of particular concern in a mobile networking environment is that some services might enhance and depend on other services, e.g., a traffic offloading service that offloads some of the traffic associated with a standard mobile access service.

We address these challenges through PROTEUS' *data centric service management*. Specifically, we employ a combination of *generic redirection primitives* in the infrastructure, *service specific* migration and other life-cycle primitives in the mobile network components and a *data-centric runtime system*, which, while it is service agnostic, nonetheless is service-aware, allowing for resource sharing between service instances. For example, PROTEUS allows for a service template to specify sharing of resources with some existing service instance. The data centric nature of PROTEUS service templates enables this sharing by allowing the new templates to be dynamically parameterized with references to existing instances of components that the new template wants to reuse.

2.3 Related Work

The possibility of using software defined technologies to realize future networks has received broad attention from both industry [11, 12, 41, 7] and academia [54, 27, 45, 35, 33]. Various industry players are collaborating on "re-architecting central offices as data centers" [41]. Also, efforts are underway to realize an open, well engineered NFV platform [7]. These efforts are primarily aimed at the engineering aspects of software defined platforms and further focus on wired (as opposed to mobile) networks. Our work specifically aligns with AT&T's proposed ECOMP architecture [12]. Unlike PROTEUS, however, this work has not addressed service evolution or mobile networking concerns. More closely related to our own work are efforts to bring software defined approaches to the mobile networking environment [45, 35, 33]. Of these, the KLEIN system [45] proposes an orchestration framework for a software defined mobile core network and is most closely related to our own work. The KLEIN work is, however, mostly focused on resource management in a "regular" standards compliant mobile core network. In contrast with these earlier efforts, our focus is on using SDI as an agent for *service evolution*.

PROTEUS' traffic redirection and migration primitives are related to earlier efforts involving packet processing with virtualized network functions [27, 46, 47]. Like the PROTEUS primitives, these earlier efforts deal with state both in network components and in the network itself. These efforts dealt with scaling of middlebox functions in a network, whereas PROTEUS focuses on using these primitives to enable the evolvability of mobile network services through the composition and configuration of virtualized network functions.

PROTEUS' template based approach is related to earlier work on template based configuration management [25], resource specification in networking testbeds [53, 2] and more recent cloud orchestration efforts [8, 20]. Presto composed "configlets" to realize network services and service options [25]. Emulab ns files [53] and GENI RSpecs [2] have been used to specify the set of network resources associated with dynamic network experiments. TOSCA follows a template based approach to allow for the orchestration of cloud applications. Our work incorporates aspects of these earlier efforts and extends them to enable mobile network service evolution.

Our work also relates to earlier efforts associated with virtualizing the radio access network, i.e., cloud RAN efforts [21] as well as more recent proposals to establish generalized cloud platforms close to the mobile edge [1, 43]. Cloud RAN efforts are limited to the RAN and would naturally form part of a comprehensive mobile SDI. Similarly, efforts related to mobile edge cloud would enable the mobile SDI required by PROTEUS. These efforts are complementary to PROTEUS.

The need for a rich set of new services is particularly evident in efforts related to 5G, the next generation mobile architecture [48, 10, 9, 38]. Our vision of a future mobile network being realized on a software defined infrastructure is shared by these works [10]. To our knowledge, however, our work on PROTEUS is the first practical mobile service control platform that enables mobile service evolution.

The notion of allowing third party service providers on a common platform is well entrenched in cloud computing. Opening up the network in the same manner has been proposed, although not in a mobile networking context [31, 18, 28]. These efforts have met with limited success. We argue that this is due, at least in part, to a lack in these earlier

efforts of the clean abstractions and automation offered by PROTEUS.

3. PROTEUS ARCHITECTURE

PROTEUS is a service control platform, i.e., a software stack run by an infrastructure provider on a mobile software-defined infrastructure, to enable service providers to instantiate and operate services on the infrastructure. A high-level depiction of the PROTEUS architecture is shown in Figure 4.

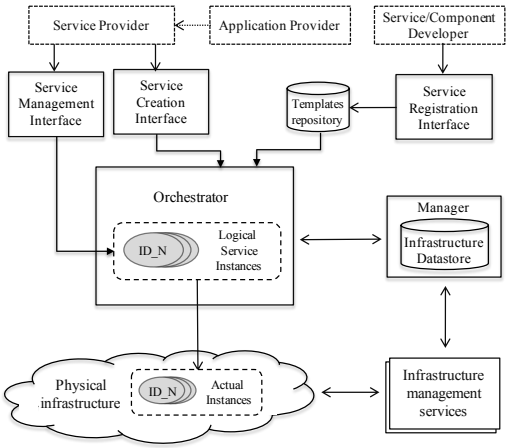


Figure 4: High-level architecture

PROTEUS provides a *service registration interface* that can be used by component and service developers to register new or modified templates with the framework by placing them within the *templates repository*. The *service creation interface* can then be used by service providers (often on behalf of their application provider customers) to instantiate instances of the registered service templates, where each service in turn may be composed of instances of registered component templates.

Once a service creation request has been placed, the *orchestrator* is responsible for catering to the request by orchestrating a *logical service instance* based on the relevant templates imported from the templates repository and template-specific parameters input to the orchestrator as part of the service creation request. A logical service instance does not consume any real resources in the infrastructure. Rather, during its creation the orchestrator sorts out any dependencies and determines whether the service could be instantiated if resources are available. To enable the creation of actual service instances, the orchestrator contains resource placement logic which finds appropriate physical resource mappings for the logical components in the service. The orchestrator has access to physical resource availability and connectivity information contained within the *infrastructure datastore*. Each service template contains a *service realization plan*, i.e., a set of commands used by the orchestrator to create the actual service and component instances by configuring physical resources in the SDI. Generic *infrastructure management services* are responsible for populating the infrastructure datastore with information about the mobile SDI. This includes information about the topology and health and performance metrics of the infrastructure. The orchestrator also has a networking controller that provides implementations for primitives supported by PROTEUS (such as migration and traffic redirection primitives for continuous location-aware resource placement and seamless service evolution).

Once the actual service instance is realized, the *service management interface* can be used to invoke lifecycle management primitives on the service. Such service management operations might result in a modification of existing components or addition of new components in the service instance, in which case some components may need to be re-orchestrated or new components may need to be added to the instance.

The following subsections describe the important pieces of the architecture in more detail.

3.1 Infrastructure Datastore

The PROTEUS data-centric approach is enabled by a logically centralized *infrastructure datastore*. The datastore contains network infrastructure information and status, network management data (e.g., monitoring and performance measurements) and data produced by network analysis tools. The datastore plays a key role in supporting various tasks during orchestration and management of services and the infrastructure components: e.g., finding physical resource mappings that satisfy the constraints for services being orchestrated, or retrieving current network and resource status for data centric management of services and the infrastructure.

PROTEUS follows a knowledge centric approach by realizing the infrastructure datastore as a *knowledge graph*. A knowledge graph captures data using basic, low-level relations. It simplifies complex reasoning over data connected by relationships, and it also facilitates the inference of undiscovered relationships by walking over the paths formed by relationships between data elements.

Data in a knowledge graph is represented as "facts," which are triplets of the form [entity1]-[relation]-[entity2]. To allow PROTEUS components, including service and component templates, to make use of information in the datastore, it exports an interface with *insert*, *delete*, *query* and *subscribe* primitives. The latter two are of particular interest to enable PROTEUS functionality:

Query takes as input a subgraph made up of facts that may contain variables or constants. The subgraph is matched against the datastore graph and all matching subgraphs (that is, those that have the same pattern) are returned. For example, consider a graph datastore with the following contents.

```
pc1 isA Compute
pc1 hasLocation SF
pc2 isA Compute
pc2 hasLocation SF
pc3 isA Compute
pc3 hasLocation NY
```

Then the following is an example of a query to retrieve all the Compute node names that are located in SF. "X_" terms represent variables while others are constant.

```
X_PC isA Compute
X_PC hasLocation SF
```

The query primitive will essentially run a subgraph matching algorithm over the graph datastore and return matches for this query subgraph. The resulting matches for our current datastore graph will be as follows.

```
results = [
            [pc1 isA Compute; pc1 hasLocation SF],
            [pc2 isA Compute; pc2 hasLocation SF]
          ]
```

Subscribe takes as input a query subgraph similar to the *query* primitive but instead of a one-time *query*, *subscribe* also allows the application or service to get new results of the query as they become available by providing one more input, a callback function to be notified of new results.

3.2 Service and Component Templates

A PROTEUS template acts as a model of the component or service it represents and contains all the pieces of information needed to realize an instance of it, customized based on user-supplied parameters. Our component and service modeling and template design is inspired by object-oriented programming principles, abstraction, inheritance, and polymorphism and was further influenced by existing cloud orchestration platforms [8].

Inheritance and polymorphism are conceptually natural choices for template specification and allow the orchestrator great flexibility in realizing a service. For example, if a service requires compute capabilities, this can be provided either through a physical host (or PC) or a virtual machine (VM). In PROTEUS we capture this with both PC and VM components inheriting from a generic Compute component. The approach generalizes to more sophisticated components. For example, in Section 2.1 we described an elastic NFV-based refactoring of an MME [16], which would similarly inherit from a generic EPC MME component in our model.

A template package for a component or service contains the following.

Properties. These are attributes or state variables associated with the component or service. For instance, for a PGW component, *location*, *isRunning*, and *migrationEnabled* could be properties. For an EPC service that may consist of multiple S/PGWs, *numPGWs* and *numSGWs* could be possible properties.

Input parameters. These are service or component specific parameters used to customize the template: for example, the number of constituent components of each type in an EPC service and desired deployment location information.

Lifecycle management functions. These make up the management interface for the component or service. Since a service is composed of components, the service implementation can abstract away the underlying component interfaces as needed and expose only a service interface that is based on a combination of the component interfaces. The service interface is then made accessible to the orchestrator, and it is also made accessible to the relevant service providers through the *service management interface*. *Start*, *stop*, *scale-up*, *scale-down*, and *migration* are example lifecycle management primitives.

Constraints. These are specified as one or more functions that each check constraints on the values of variables(s) that could be a template *property* or dynamically calculated based on user-input parameters and current service and component status. If a management or configuration primitive executed on a component or service will result in a state that violates any of the constraints then it is not executed. For example, component- or service-level constraints could be defined on the user-input parameters used to instantiate the template. An example constraint for an EPC service could be that the *numPGWs* cannot be specified to be less than one (since it is a core component).

Policy functions. These provide a way to explicitly export a service- or component-level policy. This involves subscribing to the values of variables, providing thresholds and a corresponding operator function for comparing the values of variables with the thresholds, and actions to undertake if the thresholds are violated. For example, an EPC service template has a sub-component PGW, so a simple elastic scaling policy could subscribe to the value of *PGW.load* (variable) in the monitoring logs datastore. When it is seen to be *greater than* (operator) *80%*, *continuously for 5 minutes* (threshold), then the *PGW.scale-up* function tries to remedy it (action).

Service topology specification. This describes the service's constituent compute and network resources and associated requirements or constraints. During orchestration, the specification is dynamically converted into a query that is executed by the orchestrator on the datastore to find possible physical resource mappings for the service topology. Being able to generate the query at runtime is important since the requested topology in the query will typically depend on user-supplied parameters and resource availability at the time. We provide more details on this process in Section 3.3.

Implementation artifacts. These are any OS images, code, configuration files, or similar items that might be needed during orchestration. These are either added to the template package or their publicly accessible location is specified.

Service/component plans. A plan is a workflow that takes management and lifecycle primitives defined in service and component templates as building blocks. A plan composes them in appropriate (execution) sequences that can be used to (dictate) instantiate and manage the service. This offers flexibility by allowing composition in different ways rather than having one rigid workflow. Service behavior can be changed only by modifying the workflow (and attaching new policies to it). The plans can be specified in a standard such as BPEL [39] or could be implemented as a script in languages such as Python or Java.

(i) Service realization plan. This is a service or component-specific sequence of actions that need to be taken in order to realize a new instance or re-orchestrate an existing instance based on parameters input to the template. For example, a plan may be the sequence of functions to invoke on a component in order to configure and set up an instance, such as *installPackages* → *setupDatabases* → *config* → *start*. The sequence description can include control statements such as if-else or loops to make the orchestration more flexible and dynamic (for example, based on values of user-supplied parameters or the current states of physical resources).

(ii) Component migration plan. The specific sequence and timing of actions for migration is component specific and may also be altered based on service policies. The migration plan is the generic workflow specification that ties all the component migration actions together, including reconfiguration of other dependent components in the service. The plan allows specification of conditional if-else or loop statements for greater flexibility, e.g., for specifying conditional retry or rollback loops when failure is observed during a migration step. The plan is used by PROTEUS for orchestrating migration and is parameterized at runtime with the component instance being migrated.

Import/export state functions. These are invoked to perform state transfer during component migration.

Flow specifier. During migration, if traffic needs to be redirected to the target, then a flow specifier is needed to redirect appropriate flows. For example, when an SGW is being migrated, all of its traffic will be defined using the flow specifier and will be redirected. It may also be that for some components, only certain sessions are being migrated. In that case, the flow specifier would be used to specify traffic belonging to only those sessions.

3.3 Orchestration and Re-orchestration

The PROTEUS orchestrator receives *orchestration* requests from the Service Creation Interface via its `ORCH_IN` queue (Figure 5). The orchestrator dequeues requests from `ORCH_IN` and execute the following steps for each request.

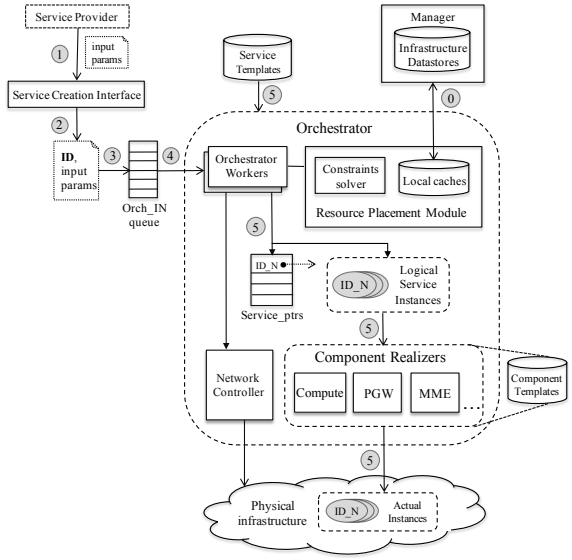


Figure 5: Service orchestration

Import templates. The orchestrator starts by importing the relevant template from the framework's template repository in order to orchestrate the type of service or service features requested by the user.

Parameterize. The imported template is then parameterized with the user input received as part of the request. The ability to parameterize generic templates allows the orchestration of instances that are customized to the user's requirements and also enables data-driven parameterization, where relevant data is retrieved from the datastore.

As an example, consider the case of a request for instantiating an EPC service. Input parameters that can be used to customize the service can include the number of various components such as the S/PGWs, the scaling policy thresholds for the service, and the locations where coverage needs to be provided. The parameters can of course be simple constant values, e.g., *location=[SF]*. However, they can also be dynamically calculated based on values of various variables that depict the current network status, e.g., *location=[X where X is a location where service provider SP has the greatest number of customers]*. In this way, the initial service resource placement can include functions to dynamically calculate the value of the variable *num_of_customers_of_SP* in a given location, based on the current monitored network status, and pick a location that has the highest value for this variable.

Resource placement. Next, the orchestrator retrieves the *logical service topology* from the parameterized service template. During orchestration, this topology specification is converted into a *service topology query* by the resource placement logic in the orchestrator. The resource placement logic then uses a constraint solver and physical topology and resource availability information from the datastore to find physical node assignments that satisfy the query and its constraints.

Generating a service topology query. The service topology is a specification that describes the number and type of components needed to create the service, their specific connectivity with each other and any constraints associated with the components. The exact service topology query is dynamically generated from the template at runtime.

As an example, a potential topology specification for a SMORE service instance with one instance of type eNodeB, one instance of type Compute for the SMORE server and one Compute instance each for the other types to run as VNFs (S/PGW, MME) might look like the one shown in Figure 6. Each node in the figure can have constraints attached to it, e.g., location or type of node or connections to other nodes. All boxes in the figure represent variables with associated constraints.

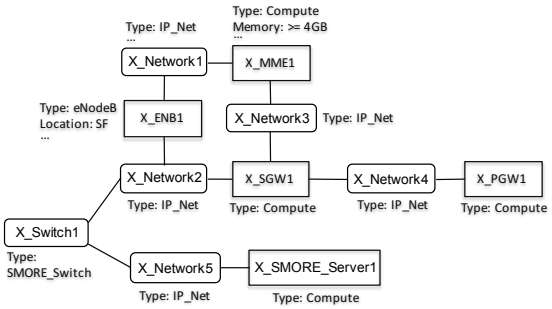


Figure 6: Example service topology specification generated from a SMORE service template

A partial query generated from this specification will look like the one shown below.

```
X_ENB1 isA eNodeB
X_ENB1 hasLocation SF
X_ENB1 isConnectedTo X_Network2
X_Network2 isA IP_Network
X_Switch1 isA SMORE_Switch
X_Switch1 isConnectedTo X_Network2
X_Switch1 isConnectedTo X_Network5
X_Network5 isA IP_Network
X_SMORE_Server1 isA Compute
X_SMORE_Server1 isConnectedTo X_Network5

X_ENB1 isConnectedTo X_Network1
X_Network1 isA IP_Network
X_SGW1 isA Compute
X_SGW1 hasMemoryGB X_SGW1_MS
X_SGW1_MS >= 4
X_SGW1 isConnectedTo X_Network2
...
```

The "`X_`" terms are taken as variables so that when the query finishes execution over a given datastore (e.g., a datastore with physical resource information), the variables will have been assigned zero or more values (i.e., physical resource assignments) that fit the specified constraints (such as location or specific network connectivity constraints).

Resource placement. The resource placement module (RPM) in the orchestrator uses the service topology query for resource placement decision-making and reservation of compatible physical resources by executing it over the infrastructure datastore.

The query primitive can itself result in the application of various constraints, but the query primitive supported by the infrastructure datastore cannot be used to specify constraints containing comparison operators (e.g., greater than, less than, etc.) and minimization or maximization goals (e.g., selecting resources in a way that minimizes the total cost). To remedy this, the RPM in the orchestrator has a generic constraint solver that can apply further constraints over the results received in response to the query executed over the infrastructure datastore. Specifically, the presence of the constraint solver in the orchestrator allows *comparison operators* and *minimize* and *maximize* objective functions to be used for the service topology specification in the service template.

For example, consider a small service template with a service topology that consists of only one Compute node such that it has 4 or more CPU units. Since there can be many nodes that fit this criterion, the service topology in the template further specifies that out of all the nodes that satisfy the CPU constraint, it would be preferable to select nodes with the lowest cost (assuming there is a cost attached to every Compute node). This objective can be specified using the *minimize* function. The entire service topology specified in terms of a query would then look like the following.

```
X_PC isA Compute
X_PC hasNumCPU X_PC_CPU
X_PC_CPU >= 4   // (extension)
X_PC hasCost X_PC_Cost
\minimize(X_PC_Cost) // (extension)
```

The RPM divides the service topology query into two parts, (1) the base query supported by the infrastructure datastore and (2) the extensions supported by the RPM. The RPM inputs part (1) to the query interface of the infrastructure datastore and then executes part (2) itself over the results returned from the datastore. The results obtained after executing both query parts will contain resources that fits all the constraints specified by the service topology. If such resources are successfully found, the RPM then reserves them for the service instance being orchestrated. The reservation information is also inserted in the infrastructure datastore.

Instance realization in the SDI. Once a result is found by the RPM logic in the orchestrator and physical resources have been reserved, the orchestrator retrieves the *service realization plan* from the parameterized template. The plan is essentially a sequence of steps with possible if-else and loop constructs that specifies exactly how to compose the components to orchestrate the service instance from them. Using this, the orchestrator can start to realize the physical service instance.

Finalize. Once the service instance creation is successfully done, the orchestrator adds the mappings of service component IDs for component instances created during orchestration to the infrastructure datastore in order to keep service and component instances persistent. Finally the orchestrator returns a service ID to the orchestration requester, which can be used to invoke management lifecycle functions on the service instance via the Proteus *service management interface*. In case of a failure, an error hierarchy is implemented in order to help debug the error or provide more information to the orchestration requesters. The orchestrator also has to undo the resource reservation step and do cleanup.

The Proteus orchestrator supports *re-orchestration* in two scenarios.

First, the topology of an active service instance might need to be modified over time, either explicitly initiated by the service provider through the *service management interface* (e.g., a decision to scale-up an EPC service to expand coverage to a new location) or by implicit triggers from within the framework. As an example of the latter, an EPC service might have a policy function that *subscribes* to monitored resource usage measurements for its PGW component. The service might then dynamically invoke the scale-up management primitive of the PGW component if the observed measurements cross a given threshold. In such cases, the service will need to have new components orchestrated. The orchestrator will essentially follow all the steps mentioned earlier for orchestration, except in this case resource placement and instance realization only happens for the new component, within the context of the existing service.

The second re-orchestration scenario occurs when a service template requires the "reuse" of component instances from an existing service instance—for example, when a service such as SMORE adds new components (SDN and some compute servers) to the EPC architecture. A SMORE service template can specify that the SMORE service should reuse all the component instances of an existing EPC instance and only request the orchestration of the SMORE-specific components. The orchestrator will follow the same orchestration steps as mentioned earlier, but during the parameterization phase the service template will be populated to allow the SMORE service instance to be associated with the existing EPC instance.

3.4 Redirection and Migration Primitives

We emphasize two primitives provided by the *network controller* in Proteus (Figure 5) that specifically help with our goals of minimally disruptive service evolution and data-centric resource placement.

Traffic redirection. With the availability of software-defined infrastructure serving as an enabler, Proteus provides a generic traffic redirection mechanism that helps with seamless datapath migration. This is useful in various scenarios, such as, for traffic redirection when migrating a live component from one location to another, or when interposing a new or evolved component or functionality within an existing service instance, e.g., during SMORE or MobiScud augmentation with an EPC instance.

Component migration. The generic migration primitive enables live migration of components with help from the traffic redirection mechanism, and support from the migration functionality implemented within component templates. It provides a generic way for state migration between components with varying implementations as long as they support the specification of their internal state in a standard way. The primitive thus enables live migration from source to target where the source and target can be different in terms of implementation (e.g., virtual or physical, different vendors, etc.). This facilitates evolution from one component instance to another, e.g., from a traditional MME implemen-

tation to an elastic MME [16]. It is also essential for enabling data-centric resource placement once the service is active. For example, a service may have requested its SGW nodes to be placed in a location that is closest to the eNodeBs that serve most of its customers. This variable "most of the services' customers" can dynamically change based on new measurements fed into Proteus by monitoring applications and would thus require Proteus to provide transparent migration of the SGW nodes to enforce the requested placement policy. Migrating the entire VM holding the SGW function is a heavyweight operation and will result in higher periods of unavailability. Also, a fine grained approach to migration is more attractive since it can be extended to allow migration of individual sessions (e.g., for load balancing purposes) which is generic enough to also be used with hardware resources in addition to virtualized functions.

The migration primitive depends on the components being *migratable*. Components are considered migratable if they implement the required API for migration and have underlying mechanisms behind the API that implement component specific migration functionality. Components such as S/PGWs, MME, and CDF can be made to support migration and plugged into the generic primitive. Alternatively, existing handover functionality provided by components such as MME or eNodeB can be utilized by the respective components to realize the underlying support for migration. This is an implementation choice for the components. The migration primitive itself is generic and does not care whether the underlying migration support is implemented by the components using already-existing handover mechanisms or some other way. It only requires the standard migration API be implemented.

That is, to realize service migration, service specific functionality supported by the network component combines with generic migration mechanisms supported by the Proteus platform.

Support API needed from templates for migration. Functions for exporting/importing component state in a compatible format; specification of a flow specifier that can be used as a filter to get this component's traffic that will need to be redirected from source to target during migration; lifecycle functions such as initialize, start, stop, various configuration functions (e.g., a MME template would have functions to configure SGW IPs that will be connecting to it), and functions related to migration operations such as exportState, importState; and a migration plan.

Support provided by Proteus as part of its migration primitive. Using the migration plans it initiates the orchestration and configuration of target component by copying over source configuration; network state migration (GTP tunnels, bearer information, IP layer, ARP and MAC setup, depending on component); export, transfer and import of component state; redirection for traffic that fits the flow specifier specified by the component for its traffic; final teardown orchestration for the source; and any needed reconfiguration of other components in the service.

4. IMPLEMENTATION

We implemented a Proteus prototype in Python. We used PuLP [34] to implement some constraint solving tasks in the resource placement module. We used neo4j [37], a graph database, for our datastore and built a simple subscription primitive on top of it. (Neo4j already supports other datastore primitive such as insert, query, and delete.)

We acquired access to implementations for an EPC service (specifically the OpenEPC stack [5]), as well as the SMORE and MobiScud services. We refactored these to fit our template model. Our prototype includes service templates for EPC, SMORE, and MobiScud, as well as the component templates needed to compose these services. These component abstractions include components such as Compute, SDN switch, PGW, SGW, MME, and other EPC enabler components such as HSS, CDF, and ANDSF. The templates are implemented as classes in Python and the additions or modifications to adapt the SMORE, MobiScud, and the OpenEPC components for Proteus make up over 10K lines of code.

We implemented a basic network controller in the orchestrator to input high level parameters for traffic redirection and generate Openflow commands to install appropriate flows in switches. The orchestrator dictates the timing of redirection operations based on the component migration plan.

As a migration use case, we used OpenEPC's PGW and created a migration plan as part of its template. The Proteus migration primitive using SDN-based redirection and support from the PGW template then allows seamless migration of the datapath by creating a duplicate PGW instance and then shutting down the first one. There is no downtime due to the PGW being migrated, since there is always at least one active in the network. The export and import of state and target PGW orchestration on a compute node and preparation for migration (copying over PGW configuration parameters, IP addresses, GTP tunnel and bearer information, and networking set up) do not affect traffic, since no traffic has been redirected yet during preparation. Once the target is ready to receive traffic, the flow setup by the Proteus redirection primitive starts to redirect the traffic identified by the flow specifier in the template. This step is independent of the number of UEs or flows. At this time, the new PGW receives traffic and processes it and the old PGW is shut down. Since EPC components use GTP tunnels over UDP, we did not implement a migrate mechanism for the transport protocol layer (i.e., TCP).

5. EVALUATION

We evaluated our Proteus prototype in the PhantomNet [15] mobility testbed. In this context, the testbed serves as the mobile SDI environment that Proteus can configure as needed to run services. Our base SDI topology in PhantomNet is shown in Figure 7. It consists of an emulated RAN and close to 300 compute resources that Proteus can configure to various roles as needed. To emulate a more realistic mobile SDI, we divided and tagged the resources into multiple "locations" with varying RTTs. We used variants of this setup for all evaluations.

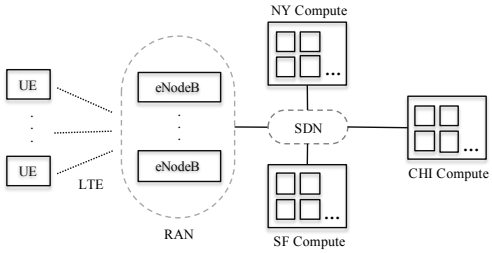


Figure 7: Mobile SDI topology in PhantomNet

We demonstrate Proteus' data-centric service management by dynamically adding service features to an existing service instance and enable data-driven policy-based actions. Next we evaluate Proteus by instantiating an EPC instance, then modifying it to add functioning SMORE and MobiScud instances. Finally, we consider Proteus' scalability in terms of end-to-end orchestration request completion time and show that the orchestrator running time increases linearly with the number of components that need to be orchestrated as part of the request. Note that in performing these evaluations we also illustrate Proteus' basic ability to instantiate multiple instances of the same type of service (with different constraints applied), as well as its ability to instantiate completely different service instances.

Data-centric service management. The following example use cases demonstrate the ability of Proteus to allow dynamically changing a service instance's behavior by triggering the invocation of its lifecycle functions.

Handling bursts of UE attachments. To illustrate data-driven performance management through templates, we periodically insert the current average number of UE attachments in an SGW into the Proteus datastore. A policy function can then be defined to make use of this information in service templates that have an SGW component. We define one such function in the EPC template that subscribes to the value of the number-of-UEs variable and requests orchestration of a new SGW in case an increasing pattern of UE attachments is observed. If a burst turns out to be temporary, then an opposite policy requests for the deletion of the unneeded SGW(s). Figure 8 shows the addition (T1) and subsequent deletion (T2) of SGW2 to share the burden of UE attachments when the existing SGW1 was seen to be overwhelmed. (The threshold was set to 15 attachments.) To illustrate these policies, we created a UE attachments burst and then made it disappear. For the first half of the experiment, we picked a random number in range 1 to 4 every minute and triggered that many UEs to attach. Then for the second half we picked a number and detached that many UEs every minute.

Migrating to a new location. We request orchestration for an EPC containing two SGWs and one PGW such that the placement policy is set to have the PGW close to both SGWs in terms of RTT measurements collected in the datastore. Later, we remove one SGW from this EPC and so Proteus has to migrate the PGW to be closer to the only existing SGW to satisfy the placement policy. Figure 10 shows RTT seen by parallel Ping flows (1 packet every 10ms) from 25 EPC subscribers to the Internet. At T0, Proteus starts transparently migrating the PGW. The Proteus migration primitive allows seamless migration of the datapath by creating a duplicate PGW instance and then shutting down the first one so no downtime has to happen. The export and import of PGW state, target PGW orchestration on a new compute node, and other preparation for migration finishes in close to 5s (the period between T0 and T1), but this does not affect traffic since it is still going through the old PGW. Once the target is ready to receive traffic, the flow setup for redirection starts and takes 0.41s (right before T1); it is independent of the number of UEs or flows. Finally at T1, migration ends and the old PGW is shut down.

Figure 9 shows TCP window scaling for a large file download by a UE through the same EPC instance, once before PGW migration and once after migration. T0 marks the migration event and a consequent faster increase in window size for the flow going through the migrated PGW because of a reduction in end-to-end RTT as a result of migrating the PGW and reducing the delay in the core mobile network.

Service evolution. We instantiate two instances of EPC service, E1 and E2, in parallel but with different placements. In Figure 11, the flows labeled E1F1 and E2F1 show the RTTs seen by parallel Ping flows (1 packet every 10ms and a total of 25 UEs creating background traffic) when two different UE subscribers, one from each EPC, ping to the same Internet server through their respective EPC networks. The difference in time is caused by the different locations of the compute nodes hosting their EPC core components. We demonstrate service evolution with this setup by augmenting E1 and E2, respectively, with a SMORE service instance and a MobiScud service instance.

We use the *service management interface* to instantiate and seamlessly augment E1 with a SMORE instance, which results in offloading functionality being orchestrated between the RAN and the E1 core. The orchestration finishes in less than a minute. Once instantiated, we use the management interface of the SMORE instance to add a customer UE to the SMORE subscribers database. This results in *selective offloading* of some of this UE's traffic (flow E1F2 starting at T1), such that the traffic destined for the IP address chosen by the SMORE service provider is offloaded to its compute instance located close to the E1 core. As expected, a lower RTT to the SMORE cloud server is observed by SMORE flow E1F2 (compared to RTT seen from pinging the Internet server seen by E1F1). Note that the other ongoing flows using the EPC (e.g., E1F1) are unaffected, since SMORE does selective offloading.

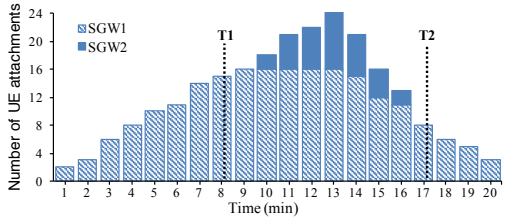


Figure 8: Dynamic scaling of SGW in an EPC instance

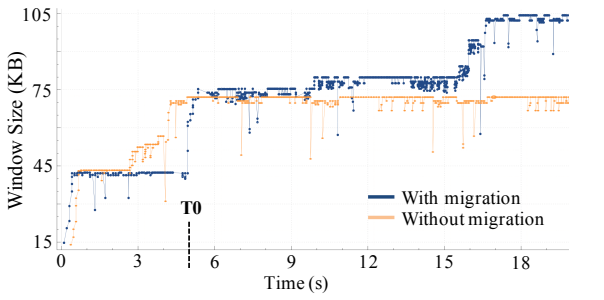


Figure 9: TCP window scaling with and w/o PGW migration.

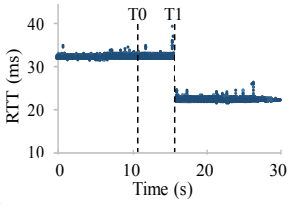


Figure 10: PGW migration

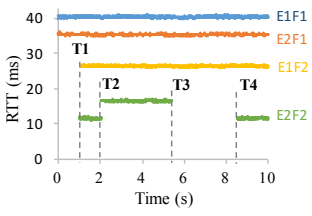
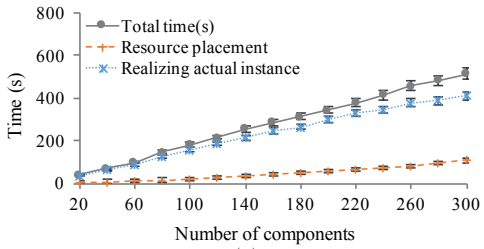
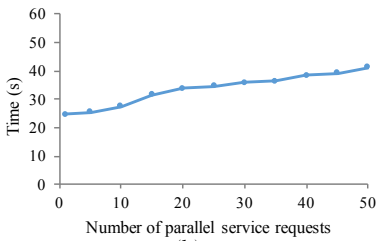


Figure 11: Evolution



(a)



(b)

Component	Avg. Time (s)
eNodeB	3.52
MME	2.12
SGW	3.40
PGW	4.18
ANDSF	7.51
CDF	2.12
HSS	3.35

(c)

Figure 12: Orchestration time when scaling (a) components in serial requests, (b) parallel requests. (c) Individual times.

Figure 11 shows a similar scenario where E2 is augmented with a MobiScud instance at T1. We request a UE to be added to the MobiScud database as a subscriber, and so it is assigned a personal compute instance (PVM) located such that it is close to the RAN in terms of RTT (the constrained resource placement step in the orchestration). MobiScud does selective offloading for its UE's traffic such that all its flows headed toward the service provider's chosen IP address are offloaded to the UE's PVM. As expected, a lower RTT is observed for E2F2 flow starting at T1, a Ping flow started by the MobiScud subscriber UE and destined toward the MobiScud IP. At T2, the UE's movement towards another eNodeB causes a handover event, and orchestration for this UE's PVM migration begins and continues until T4. First, the RTT goes up as the UE's traffic is forwarded from the target eNodeB (i.e., the eNodeB where the UE is, after handover) to the source eNodeB (i.e., the eNodeB from which UE has moved away, but its PVM is still within this eNodeB's edge cloud). At T3, Xen VM migration starts and a brief gap in RTT line shows packet loss since the VM cannot receive any packets. At T4, migration ends and RTT drops back, since the VM has now moved to the target eNodeB's edge cloud. Proteus' orchestrator does the compute allocation at the target edge cloud and flow setup to redirect traffic to the PVM, which does not cause any downtime. The only downtime is due to the unoptimized Xen VM migration implementation we use.

End-to-end orchestration request completion time. An orchestration request's completion time depends on the number of components requested (e.g., compute instances, or virtualized functions such as an SGW). Since most mobile network services are specialized architectures based on EPC, we picked the EPC service as the base to measure the effect of number of components in a service orchestration request on Proteus' runtime. Figure 12(a) shows the request completion time as a function of EPC request size (i.e. by requesting EPC instances composed of an increasing number of components such as S/PGWs). As seen from the graph, the runtime increases slowly and the confidence intervals are tight. The first major step in an orchestration request is the resource placement query's subgraph matching and constraint solving step. In each of our EPC requests, each component has at least 5 constraints associated with it: as the number of components is increased, the number of attached constraints in the EPC request also increases linearly and affects the runtime. The second major step in orchestration is the execution of the realization plan. This involves generating configurations and instantiating the actual instances in the infrastructure. Since physical layer commands are often time consuming, this also becomes a factor in the increase in runtime. While we parallelize the configuration of components that do not have dependencies, there is still some overhead due to service-specific dependencies. Figure 12(a) also shows the total request completion time divided into these two steps.

The resource placement module is the main user of the datastore when orchestrating new services or modifying existing ones (such as when scaling up, doing migration, or PVM allocation in MobiScud). So the resource placement scalability as the number of components in service request is increased (which in turn results in increase in query size and subsequent graph search time) is a realistic indicator of the datastore design's scalability. The resource placement time involves both creating and evaluating queries against the datastore to search for and reserve resources. Also, the datastore component in Proteus' architecture is logically centralized but can physically consist of multiple datastores at various levels: for example, infrastructure inventory, measurement logs, etc.

Orchestration time for individual components. Table 12(c) shows the average orchestration and realization times for individual components that make up the base for most mobile network services. These times involve both data-centric resource placement for the components using the datastore as well as configuration by the orchestrator in the infrastructure for actual realization of the component instance.

End-to-end parallel orchestration request completion time. Figure 12(b) shows request completion time as a function of number of parallel orchestration requests of a basic EPC service. As seen from the figure, the completion time increases slowly. The resource reservation and placement step, which is the main bottleneck, is able to respond quickly even when faced with multiple parallel resource placement queries.

6. CONCLUSION

We presented our work on a mobile network service control platform, called Proteus. Proteus is a software stack operating on a mobile software defined infrastructure, built to dynamically instantiate and manage mobile network services. In particular, Proteus allows for service evolution by enabling new services and network components to be introduced to the platform via templates. The Proteus orchestrator makes use of these templates to dynamically instantiate and modify mobile services. We developed a prototype implementation of Proteus and evaluated it on a mobile testbed.

7. ACKNOWLEDGMENTS

We would like to thank our anonymous reviewers and shepherd for their valuable comments. We are indebted to Vijay Gopalakrishnan for many insightful discussions on orchestration in a mobile network environment. This material is based upon work supported by the National Science Foundation under grant numbers 1305384 and 1302688.

8. REFERENCES

[1] Mobile-Edge Computing (MEC) Technical Requirements (Draft ETSI GS MEC 002 v0.4.2). https://docbox.etsi.org/ISG/MEC/Open/MECGS-002_Draft_techreqv042.pdf.

[2] Resource Specification (RSpec) Documents in GENI. http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs.

[3] Why distribution is important in NFV? http://tinyurl.com/kyewvqq.

[4] The tactile Internet - ITU-T technology watch report. http://www.itu.int/dms_pub/itu-t/oth/23/01/T23010000230001PDFE.pdf, Aug. 2014.

[5] OpenEPC. http://www.openepc.com/, 2015.

[6] OpenStack. https://www.openstack.org, 2015.

[7] OPNFV: An Open Platform to Accelerate NFV. https://www.opnfv.org/sites/opnfv/files/pages/files/opnfv_whitepaper_092914.pdf, 2015.

[8] Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf, 2015.

[9] 5GPPP. 5G Vision - The 5G Infrastructure Public Private Partnership: the next generation of communication networks and services. https://5g-ppp.eu/wp-content/uploads/2015/02/5G-Vision-Brochure-v1.pdf, 2015.

[10] J. Andrews, S. Buzzi, W. Choi, S. Hanly, A. Lozano, A. Soong, and J. Zhang. What will 5G be? *Selected Areas in Communications, IEEE Journal on*, 2014.

[11] AT&T. AT&T Domain 2.0 Vision White Paper. https://www.att.com/Common/about_us/pdf/AT&TDomain2.0VisionWhitePaper.pdf, 2013.

[12] AT&T Inc. ECOMP (Enhanced Control, Orchestration, Management & Policy) Architecture White Paper. http://about.att.com/content/dam/snrdocs/ecomp.pdf.

[13] A. Baliga, X. Chen, B. Coskun, G. de los Reyes, S. Lee, S. Mathur, and J. E. Van der Merwe. VPMN: Virtual private mobile network towards mobility-as-a-service. In *Proceedings of the Second International Workshop on Mobile Cloud Computing and Services*, MCS '11, 2011.

[14] A. Banerjee, X. Chen, J. Erman, V. Gopalakrishnan, S. Lee, and J. Van Der Merwe. Moca: A lightweight mobile cloud offloading architecture. In *Proceedings of the 8th ACM International Workshop on Mobility in the Evolving Internet Architecture*, MobiArch '13, 2013.

[15] A. Banerjee, J. Cho, E. Eide, J. Duerig, B. Nguyen, R. Ricci, J. Van der Merwe, K. Webb, and G. Wong. PhantomNet: Research infrastructure for mobile networking, cloud computing and software-defined networking. *GetMobile: Mobile Computing and Communications*, 19(2):28–33, 2015.

[16] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, and J. Van der Merwe and Sampath Rangarajan. Scaling the LTE control-plane for future mobile access. In *Proceedings of the 11th ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, Dec. 2015.

[17] A. Banerjee, B. Nguyen, V. Gopalakrishnan, S. K. Kasera, S. Lee, and J. Van der Merwe. Efficient, adaptive and scalable device activation for M2M communications. In *IEEE SECON*, 2015.

[18] R. Battiti, R. Lo Cigno, F. Orava, and B. Pehrson. Global growth of open access networks: from warchalking and connection sharing to sustainable business. In *Proceedings of the 1st ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 19–28. ACM, 2003.

[19] K. Benson. Enabling resilience in the Internet of Things. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*, pages 230–232, Mar. 2015.

[20] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner. OpenTOSCA – a runtime for TOSCA-based cloud applications. In *11th International Conference on Service-Oriented Computing*, LNCS. Springer, 2013.

[21] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann. Cloud RAN for mobile networks—a technology overview. *IEEE Communications Surveys Tutorials*, 17(1):405–426, Firstquarter 2015.

[22] Z. Chen, L. Jiang, W. Hu, K. Ha, B. Amos, P. Pillai, A. Hauptmann, and M. Satyanarayanan. Early implementation experience with wearable cognitive assistance applications. In *Proceedings of the 2015 Workshop on Wearable Systems and Applications*, WearSys '15, pages 33–38, New York, NY, USA, 2015. ACM.

[23] J. Cho, B. Nguyen, A. Banerjee, R. Ricci, J. Van der Merwe, and K. Webb. SMORE: software-defined networking mobile offloading architecture. In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges*, pages 21–26. ACM, 2014.

[24] G. Cugola and J. de Cote. On introducing location awareness in publish-subscribe middleware. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 377–382, June 2005.

[25] W. Enck, T. Moyer, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, Y. W. E. Sung, S. Rao, and W. Aiello. Configuration management at massive scale: system design and experience. *IEEE Journal on Selected Areas in Communications*, 27(3):323–335, Apr. 2009.

[26] Gartner. Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020. http://www.gartner.com/newsroom/id/2636073, 2013.

[27] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 163–174, New York, NY, USA, 2014. ACM.

[28] A. González, A. Vergara, A. Moral, and J. Pérez. Prospects on FTTH/EP2P open access models. In *Federation of Telecommunications Engineers of the European Union (FITCE) 49th Congress*, 2010.

[29] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[30] M. Hatton. The Global M2M Market in 2013. Machina Research White Paper, Jan. 2013. http://tinyurl.com/opovu8a.

[31] M. Hayashi, N. Matsumoto, K. Nishimura, and H. Tanaka. Design of network resource federation towards future open access networking. In *AICT 2011, The Seventh Advanced International Conference on Telecommunications*, pages 130–134, 2011.

[32] P. Jain, J. Manweiler, and R. Roy Choudhury. Overlay: Practical mobile augmented reality. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 331–344, New York, NY, USA, 2015. ACM.

[33] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies*, pages 163–174. ACM, 2013.

[34] S. Mitchell, M. O'Sullivan, and I. Dunning. PuLP: A linear programming toolkit for Python. *The University*

of Auckland, *Auckland*, *New Zealand*, 2011. http://www.optimization-online.org/DB_FILE/2011/09/3178.pdf.

[35] M. Moradi, L. E. Li, and Z. M. Mao. SoftMoW: A dynamic and scalable software defined architecture for cellular WANs. In *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, 2014.

[36] G. Muhl and A. Ulbrich. Disseminating information to mobile clients using publish-subscribe. *Internet Computing, IEEE*, 8(3):46–53, May 2004.

[37] Neo4j Developers. Neo4j. *Graph NoSQL Database [online]*, 2012.

[38] NGMN Alliance. 5G white paper-executive version. *White Paper, December*, 2014.

[39] OASIS. OASIS Web Services Business Process Execution Language (WSBPEL) TC. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[40] M. Olsson, S. Rommer, C. Mulligan, S. Sultana, and L. Frid. *SAE and the Evolved Packet Core: Driving the mobile broadband revolution*. Academic Press, 2009.

[41] On.Lab. CORD: The Central Office Re-architected as DataCenter. http://onosproject.org/wp-content/uploads/2015/06/PoC_CORD.pdf, 2015.

[42] M. Patel, Y. Hu, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, U. Rauschenbach, S. Abeta, L. Chen, K. Shimizu, A. Neal, P. Cosimini, A. Pollard, and G. Klas. Mobile-Edge Computing. https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf, Sept. 2014.

[43] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, et al. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[44] R. Paul. Exclusive: a behind-the-scenes look at Facebook release engineering. http://tinyurl.com/6lsfg26, 2012.

[45] Z. A. Qazi, P. K. Penumarthi, V. Sekar, V. Gopalakrishnan, K. Joshi, and S. R. Das. KLEIN: A minimally disruptive design for an elastic cellular core. In *ACM Symposium on SDN Research (SOSR)*, 2016.

[46] S. Rajagopalan, D. Williams, and H. Jamjoom. Pico replication: A high availability framework for middleboxes. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 1:1–1:15, New York, NY, USA, 2013. ACM.

[47] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 227–240, Lombard, IL, 2013. USENIX.

[48] J. Rodriguez, editor. *Fundamentals of 5G Mobile Networks*. John Wiley & Sons, 2015.

[49] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, Oct. 2009.

[50] A. Shakimov, H. Lim, R. Cáceres, L. P. Cox, K. Li, D. Liu, and A. Varshavsky. Vis-a-vis: Privacy-preserving online social networking via virtual individual servers. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pages 1–10, Jan. 2011.

[51] J. Thones. Microservices. *Software, IEEE*, 32(1):116–116, Jan. 2015.

[52] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, and K. Webb. MobiScud: A fast moving personal cloud in the mobile network. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, AllThingsCellular '15, 2015.

[53] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX.

[54] G. Xilouris, E. Trouva, F. Lobillo, J. Soares, J. Carapinha, M. McGrath, G. Gardikis, P. Paglierani, E. Pallis, L. Zuccaro, Y. Rebahi, and A. Kourtis. T-NOVA: A marketplace for virtualized network functions. In *Networks and Communications (EuCNC), 2014 European Conference on*, pages 1–5, June 2014.