7

EPC in the Cloud

James Kempf and Kumar Balachandran
Ericsson Research, San Jose, CA, USA

7.1 Introduction

The architecture of the Internet is being fundamentally restructured by cloud computing and software defined networking (SDN) [1–3]. Cloud computing and SDN conceptually insert computation into the center of the network and separate control and execution in a way that decentralizes execution.[1] Control can then be flexibly exercised, often from a central entity running in a cloud data center. SDN and cloud find applicability in next-generation operator networks through network function virtualization (NFV) [4–6]. SDN for operator networks and NFV are currently under much discussion in the literature and in fora specifically dedicated to standardizing their interfaces [7, 8].

SDN and NFV have relevance for the mobile network operator too, specifically with respect to the Evolved Packet Core (EPC) [9]. Additionally, SDN and NFV can realize true convergence of various different kinds of core networks, transport backbones, and services; businesses can now flexibly deploy common services across different operator domains such as broadband Ethernet, mobile networks, and broadcast media networks. Such convergence will ideally enable services to transcend the complications of working across different security, identity, and mobility mechanisms, while the lower-level implementations of such functionality are themselves altered over time to a common framework. SDN and NFV have the potential to substantially change how operators deploy and manage mobile and fixed networks and, in addition, how operators develop and offer services to their users.

[1] This is meant to be normative rather than stricture.

Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture, First Edition.
Edited by Madhusanka Liyanage, Andrei Gurtov, and Mika Ylianttila.

7.1.1 Origins and Evolution of SDN

SDN started with the introduction of OpenFlow 1.0 in 2009 [10]. OpenFlow initially featured a simple flow switch model where flows are a time sequence of packets identified by having a packet header that matches a pattern. Only unicast packets having Ethernet headers tagged with simple 802.1Q VLAN [11], IPv4, and TCP/UDP were supported. The latest version of OpenFlow 1.4, published in 2013, supports broadcast/multicast packets with headers having carrier Ethernet, Multiprotocol Label Switching (MPLS), IPv6, and a variety of transport protocols [12]. An extension mechanism supports experimental or vendor-specific extensions.

The OpenFlow architecture has not changed much over the course of its development. The control plane is centralized in a controller where flow forwarding on the user plane is programmed, unlike a standard IP-routed network. The controller sets next-hop forwarding on the switches using the OpenFlow protocol over a secure channel. The user plane could be implemented as a softswitch rather than in hardware. In data center applications, softswitches are the predominant way in which OpenFlow is deployed, as in, for example, the open virtual switch (OVS) softswitch [13]. Softswitches are not as common in networking applications outside the data center.

In an OpenFlow switch, packets enter through hardware switch ports into a pipeline consisting of a series of forwarding tables. Each table has conceptually six columns:

- A rules column, containing a pattern that matches fields in the packet header, the input port, and the metadata from previous tables. The header field patterns support exactly matching the pattern or a matching to a wildcard expression for which a part of the field can be anything.
- A priority column, which specifies the priority order of this pattern with respect to other patterns that match a header.
- A counters column, which indicates the OpenFlow counters that should be updated if the packet matches the pattern. The OpenFlow protocol supports messages for querying these counters to obtain statistics on flows running through the switch.
- An action column, which specifies the actions that must be executed should the packet header match the pattern. Examples of actions include rewriting the packet header in some fashion, forwarding the packet to an output port, or sending the packet to the next table in the pipeline for further processing. The packet can also be dropped, which is the default action if no pattern matches, or forwarded to the controller.[2]

The headers of incoming packets are matched against the match column, and if any patterns match, the actions associated with the top priority rule are collected. If the actions include sending the packet to an output port, then the actions are executed; otherwise, the packet is sent to the next table and the actions are executed when the packet exits the packet processing pipeline. An action can collect part of the header for metadata and pass that along through the pipeline to use in matching during the next phase. The switch design also includes group tables to support programming multicast and broadcast flows and meter tables to support quality of service (QoS) on flows.

[2] OpenFlow 1.0 by default sent the packet to the controller if no pattern matched. This led to some confusion initially about whether OpenFlow could handle large volume traffic flows, but the "first packet to the controller" meme was never really a fundamental part of the design and has since been deprecated. The ability to forward packets to the controller has been kept as an option however since it is extremely useful for routing control plane packets to the controller that are incoming from networks outside the OpenFlow domain.

While SDN started with OpenFlow, the concept has since broadened beyond a single protocol and switch design to include a variety of different systems and protocols. For example, the Contrail data center network virtualization system [14] uses XMPP[3] [15], a protocol developed for instant messaging, to program the user plane elements. In addition, centralization of the control plane has led to a merging of the control and management planes into a single controller at the architectural level [16], with control plane decisions having a characteristic time constant for decision making that is shorter than the management plane.

All SDN systems have the following two key principles in common:

- Separation of Control Plane from User Plane: The control plane is handled by a centralized controller, which then programs routes into the user plane with a protocol connecting the controller to the packet switching and routing elements. The controller performs forwarding and routing calculations for the network. The interface between the controller and the user plane elements is arbitrarily called the *southbound interface*, merely because it usually appears on the bottom of diagrams showing an SDN controller. Newer controllers, such as OpenDaylight [16], feature support for installing and deploying multiple control and management protocols on the southbound side.
- Abstraction of the User Plane Control into a Collection of Application Programming Interfaces (APIs) and Objects Exposed to Programmers Constructing Applications that Control Forwarding and Routing: An API can be in Java, Python, or a Representational State Transfer (REST) format [17] for remote procedure calls (RPCs). These APIs allow a programmer to control and manage the user plane while limiting the possibility of misconfigurations and errors by encapsulating correct behavior in abstractions. Some SDN protocols, for example, OpenFlow, also feature extensive support via an API for fine-grained measurement beyond that available in legacy network management protocols such as SNMP. The interface between the programmer and the SDN controller is called the *northbound interface* for obvious reasons.

These principles distinguish an SDN network from a traditional distributed IP routing control plane. In a traditional IP routing network, forwarding and routing calculations are done by the individual forwarding elements, and network management is controlled by vendor-specific command line interface scripts that don't hide the network complexity. In contrast, the abstractions presented by the SDN controller to the programmer are typically less complex and more consistent. This simplifies the job of constructing correct and understandable network management and control programs. Network management thus becomes a matter of program development against a standardized API.

The application to OpenFlow/SDN to mobile networks, and, in particular the EPC, is the subject of Section 7.3.

7.1.2 NFV and Its Application

NFV emerged out of a rethinking of how to deploy communication system applications involved in running an operator network, specifically on how to apply the advances in enterprise and Web services deployment technology over the past ten years to communication

[3] Extensible Messaging and Presence Protocol.

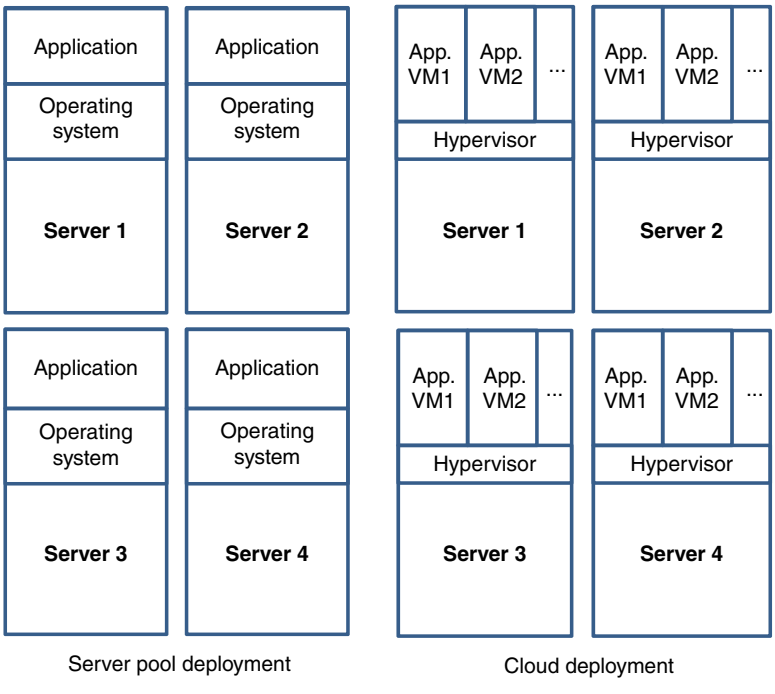

Figure 7.1 Comparison of the server pool versus cloud deployment patterns.

system applications. Many core network subsystems, for example, the Internet Multimedia Subsystem (IMS) [18] and the EPC, are implemented in software. The applications in these subsystems are deployed directly on servers, with only the operating system between the communication system application and the server (see the left-hand side of Figure 7.1). Server pools dedicated to particular functions can be configured to handle extra load, but this kind of overprovisioning has been found to be costly and wasteful of power in enterprise and Web applications. Server pools also scale poorly in enterprises, requiring an additional piece of hardware to be installed if more application capacity is necessary.[4] In addition, since the clock rate of processors reached an upper limit of around 3 GHz in the early 2000s, chip manufacturers have been increasing the number of processor cores available on a chip as a way to continue densifying the amount of processing power. Most operating systems are written to manage hardware threads, but they typically don't handle separate processing cores efficiently.

Enterprise and Web applications have begun to deploy on top of a virtualized platform in large data centers. The operating system and application are packaged as a software image called a *virtual machine* (VM).[5] The VM is elevated from being deployed directly on dedicated hardware to being deployed on a *hypervisor*, a software virtualization layer that manages the interaction with the hardware. The hypervisor has been written to efficiently handle processors

[4] In telecommunication networks, it is difficult to say that this difference is very big, but a network operator can create services that naturally balance the load over time.

[5] They may also be deployed in special operating system processes in VMs called containers that have extra protection to ensure isolation between applications.

with multiple hardware cores. The result is a clean separation of concerns between the physical hardware and the VM. This kind of deployment paradigm is generically called *cloud computing* and is outlined on the right side of Figure 7.1. Virtualization has been applied to networking and storage resources in addition to computation. Here as well, a software virtualization layer is interposed between applications and the actual physical resources, allowing sharing of the physical resources between multiple users.

In enterprise cloud deployments, increased hardware utilization efficiency is achieved by *oversubscription*, scheduling more VMs to a server than cores available to execute them. This ensures that the server is kept busy more than 80–90% of the time, rather than less than 30% as is typical of nonvirtualized server deployments. In public cloud (utility computing) deployments, such as Amazon Web Services (AWS) [19], oversubscription is less common because it would put the public cloud operator at risk of violating the service-level agreement toward customers. Public data centers also support *multitenancy*, where multiple individuals and organizations share computational resources with isolation enforced between the different tenants. Isolation ensures that each tenant sees a slice of the compute, storage, and networking resource that they have contracted for and that their slice incurs no interference from other tenants. Cloud operating systems such as OpenStack [20] manage the deployment of multitenant virtualized compute/storage/network infrastructure at a high level, enforcing isolation between the tenants.

Deployment of the application is managed by an *orchestration* system that monitors the load on the running application VMs. If the orchestration system detects excessive loading by an application, it can *scale out* the application by starting up new application VMs to handle the additional load.[6] Idle VMs can similarly be deactivated. A server that is not running any VMs can operate in low power mode or even be powered down, saving operating cost and reducing carbon footprint.

Additionally, the orchestration system can arrange for a pool of application VMs to handle traffic forwarded through a front-end *load balancer*. The load balancer sprays user traffic packets to all the active VMs, reducing the load on any one. Load balancing is facilitated by constructing applications so that they are *stateless*, with all the user state held in a back-end database whose data consistency is protected by transactions. The only state held in the application is state that can easily be reconstructed by a short user interaction, such as shopping cart contents. These kinds of applications are often called *three-tier applications*. The client interaction software, typically in a browser, is the first tier, the stateless application containing the business logic is the second tier, and the database forms the third tier. The load balancer is seen as part of the routing infrastructure.

The NFV manifesto [4, 5] advocates the cloud computing paradigm for communication system applications. The following technical and business benefits were identified:

- A reduction in capital expenditure for specialized hardware. Subsequent study showed the benefits to be minimal since many communication system applications were already running on standardized IT components.

[6] In contrast, traditional server-based scale-up applications require the enterprise or Web operator to purchase and install a larger server when more capacity is required, a costly and time consuming proposition. Telecommunications applications deployed using the traditional server pool architecture use a scale-up paradigm to handle load and for redundancy.

- A reduction in operating expenditure and carbon footprint by more efficiently utilizing hardware to reduce power consumption and by enabling use of the underlying hardware and software platform for multiple applications.
- Faster innovation in service development and deployment and the ability to target services more narrowly at particular geographic areas and customer demographics without incurring multi-year-long development cycles.
- Ability for multiple tenants to utilize the software and hardware platform, so a variety of applications could be deployed on the same infrastructure.
- Opening up the infrastructure procurement process in operators to new entrants, both commercial and nonprofit (such as academics), since the barrier of entry becomes lower due to the use of software rather than hardware.

In Sections 7.2 and 7.3, we discuss how an NFV deployment of the EPC might evolve.

7.1.3 SDN and Cross-Domain Service Development

Most network operators generate the bulk of their revenue from services purchased by enterprise and individual customers, for example, enterprise VPNs or wireless plans. Routing and forwarding are simply a means toward providing the connectivity needed to enable those services. To achieve the level of innovation sought by the NFV manifesto authors, service development and deployment need to be simplified and integrated much more tightly across the operator's network than is currently the case. Functions not specifically involved in the immediate delivery of packets such as identity management, policy management, and charging and billing need to become as available as services like routing and forwarding are for transport SDN. Access to functionality from different domains (cloud, fixed WAN, and mobile core) needs to be simplified. While routing and forwarding have received almost all of the attention in the SDN community, the topic of how to achieve a simplified platform for service development and deployment has been mostly ignored.

This aspect of managing an operator's network is the domain of OSS/BSS systems.[7] Today, such systems typically require extensive human intervention. A customer service representative takes a call for a service and starts an order flow that may require a technician to drive to the customer site and install a piece of equipment or change a switch setting. Once the technician has accomplished the task, the customer service representative must notify the customer that their service is ready. Troubleshooting may be required to resolve issues. Provisioning a service such as an enterprise VPN[8] can often take weeks or months. Accessing the business logic involved in charging and billing systems is often complex and differs depending on the particular domain of the network (WAN,[9] mobile core, and cloud), further complicating service development.

Ericsson's service provider SDN (SP-SDN) [21] is an approach to rapid and flexible cross-domain service creation that complements SDN and NFV. The SP-SDN features Web service APIs crafted with abstractions representing objects and operations involved in service creation,

[7] OSS, operations support system; BSS, business support system.
[8] Virtual private network.
[9] Wide area network.

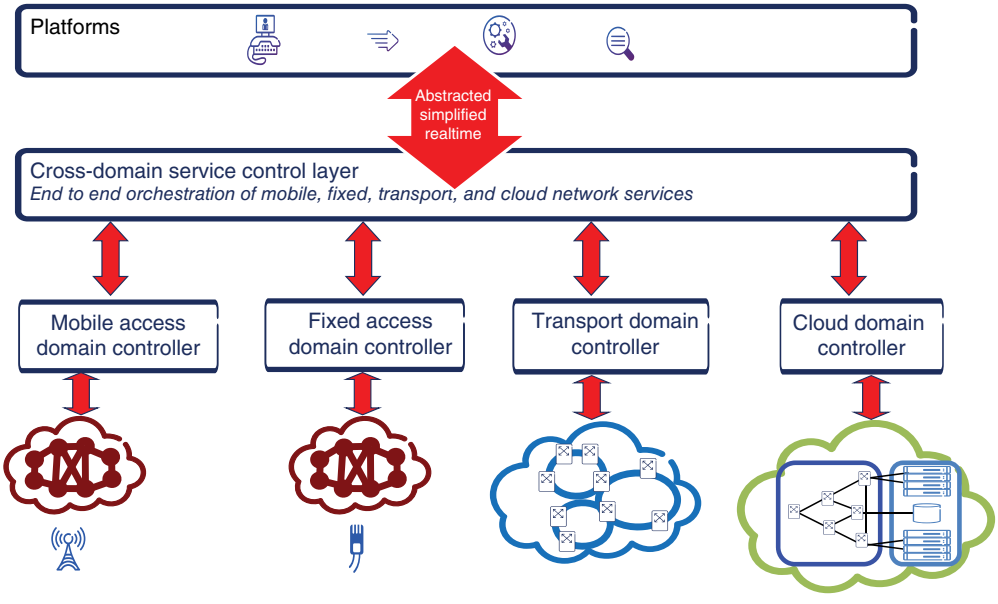

Figure 7.2 SP-SDN architectural concept.

deployment, and management, just like the SDN controller provides for routing and forwarding. These APIs expose network functionality at the service layer rather than the transport layer and in many cases can be based on functionality provided by SDN or NFV if it is available. But given the existence of a large installed base and legacy software controlling it, SP-SDN also offers the potential to simplify network service creation based on existing legacy equipment and software.

Figure 7.2 illustrates the SP-SDN architectural concept. The different operational domains in the operator network are located at the bottom: data center, wide area networking (IP and transport), and mobile and fixed access. Each of these has a set of transport control functions involving networking that are reflected up through their respective transport controller as a collection of APIs for configuring and managing transport. Ideally, the transport control and management functions will be virtualized, and the APIs will reflect a collection of useful abstractions, but the Service Control layer can work with legacy networks as well. In addition, the data center includes APIs for controlling VM execution and placement, for example, using a cloud operating system such as OpenStack.

The cross-domain Service Control layer spans across the transport domains. Each transport domain is likely to have its own transport controller, reflecting the specific technical and administrative aspects of controlling that domain. For example, the transport controller for the mobile domain will reflect the need for mobility. Similarly, the data center controller will need to coordinate allocation and deployment of networking and computing resources. Some domains might share a controller if the demands of control are similar enough. In addition, the Service Control layer encompasses other control aspects of network services that do not involve transport: access control, for example, the radio scheduler, the analytics related to location and routes, etc.; policy control, for example, QoS settings based on the time of the

day, location, service type, etc.; cloud compute and storage, such business support functions as dynamic charging; and many others. The cross-domain Service Control layer thus includes the high-level intelligence of the EPC and additionally enables rapid provisioning and deployment of services.

The Service Control layer exposes an interface that has the following characteristics:

- Abstracted—The API features a set of abstractions carefully chosen to represent the objects important for operators defining services. An example is a user identity. At the network transport layer, the user identity isn't important since transport is concerned with flows, routes, and circuits. Once a user has been authenticated and the user's authorization is verified—functions of the Service Control layer—the network control layer can authorize the user to operate on the network.
- Simplified—Many network functions today feature a lot of parameters that a service designer must specify before a service can be instantiated. In most cases, the majority of these parameters may be duplicates or derivable from a service-level parameter. For example, when establishing a VPN between the data center domain through the IP and optical transport domain, the VLAN identifier for a single customer is often the same.
- Real time—An attractive feature of cloud platforms is that resources are allocated elastically in real time; that is, the amount of resource expands and contracts within predefined limits to meet demand. The Service Control layer allows services to be defined, provisioned, and deployed through a customer portal in a few minutes, rather than taking hours, days, or even months of time.

The APIs that the Service Control layer exposes to clients are not the traditional protocol APIs from previous generation network architectures nor are they the command line interface APIs exposed by network equipment, but rather Web APIs, for example, REST APIs. In addition, Secure Sockets Layer (SSL) security and Web-based authentication can easily be added to an interface if security is necessary. Many tools exist for conveniently programming Web APIs in commonly used languages such as Java and Python.

The actual content of the Service Control layer APIs—the abstractions, objects, and operations that are exposed—will evolve from specific use cases. As commonalities between use cases become better understood, abstractions will emerge in the same manner as in software engineering, where, for example, function calls developed out of a need to abstract common operations into parameterized chunks of code. Abstraction has considerably simplified reasoning about and developing and deploying software.

SP-SDN is an application of the service-oriented architecture (SOA) [22] concept from the enterprise and Web worlds to communication services. In a system designed according to SOA, discrete software components provide functionality to other components as services. These components are distinguished by having well-defined interfaces so that the services can be deployed on any platform. The interfaces do not allow programmers access to the internal implementation, allowing the implementation to be changed for optimization purposes. The interfaces are typically implemented as RPCs using the REST or SOAP[10] [23] HTTP format. Architecting a system according to SOA principles allows system

[10] Simple object access protocol.

components to be combined in arbitrary ways, exactly the kind of flexibility that is needed for the speedy innovation that NFV advocates.

Section 7.4 describes application of the SOA principle to the EPC.

7.2 EPC in the Cloud Version 1.0

The initial version of a virtualized EPC will follow the same pattern as with enterprise cloud software deployments: the existing applications will be lifted up, packaged in VMs, and deployed on a virtualized platform with their functionally exposed through HTTP APIs. Both mobile control plane applications such as the Mobility Management Entity (MME), the Home Subscriber Server (HSS), and the Policy and Charging Rules Function (PCRF) as well as the Serving/Packet Gateways (S/P-GWs), which have both a control plane and a user plane routing function, will deploy applications in VMs. Figure 7.3a contains a high-level schematic of how such a deployment pattern looks.

In this type of deployment pattern, both the control and user plane flows run through a data center.

Given the existing state of cloud operating system and orchestration software, represented by OpenStack, there are a variety of technical problems involved in deploying the EPC on a cloud platform [24]. Two in particular stand out:

- The P-GW manages session state in the Packet Data Protocol (PDP) context [25]. Unlike enterprise 3-tier applications, the PDP context can't realistically be managed through a simple database since it must be constantly consulted when managing packet data flows. It also can't be quickly reconstructed from a user session if the P-GW fails. Thus, the auto-scaling and reliability pattern for managing 3-tier applications won't carry over to managing MME, S-GW, and other EPC entities like the PCRF and the HSS[11] where substantial system state must be quickly available.
- The virtual networking capabilities of cloud operating systems are rudimentary and not capable of supporting the demands of challenging EPC applications. The enhanced bearer capability of the EPC requires the network to handle QoS properly. Cloud operating system network virtualization usually only handles best effort traffic. Connectivity options into cloud data centers are also relatively limited, confined primarily to best effort Internet service or occasionally to enterprise VPN service, and setup has long lead times. In the worst case, the EPC might need to manage multiple BGP autonomous systems (ASes) in the cloud if the operator supports mobile virtual network operators (MVNOs).

These problems are already receiving attention in the OpenStack consortium as NFV deployments begin to roll out. The existing telecommunications style mechanisms for handling EPC redundancy and scalability such as OpenSAF [26] can probably address the first point in the short run, but more sophisticated distributed system techniques for managing state will be required to support the more flexible redundancy and scalability potential of cloud deployments. The second point is already being addressed by some

[11] Home Subscriber Server.

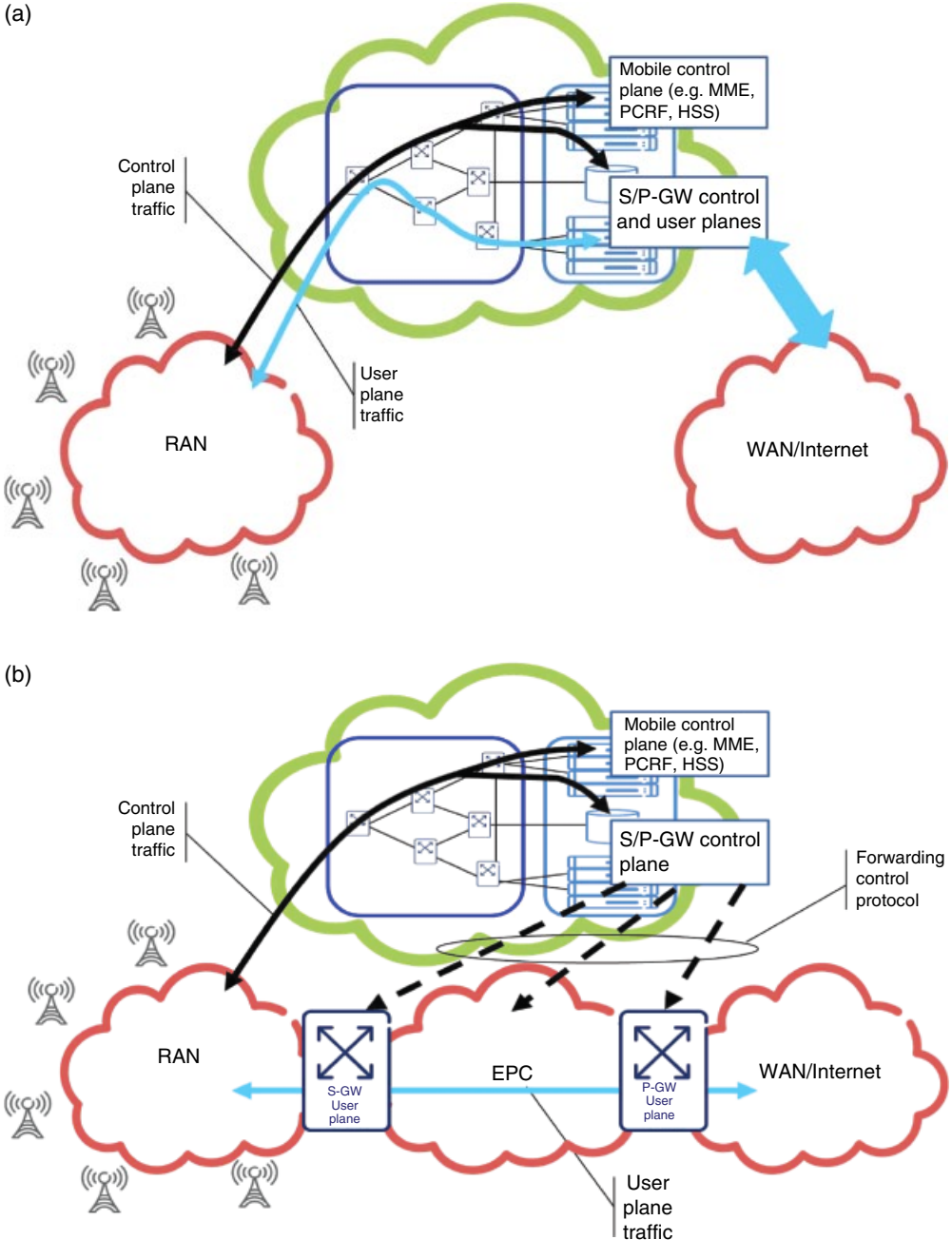

Figure 7.3 Different EPC cloud deployment patterns. (a) Control and user plane in cloud. (b) Control plane only in cloud.

ongoing work in OpenStack on QoS for virtual networks. However, the more complex challenges of managing a multi-AS virtual cloud have yet to be addressed.

An issue that stands out with respect to the virtualized EPC is the performance of software forwarding on standard server hardware.[12] While the control plane entities logically don't require high-performance forwarding, the user plane entities such as the S/P-GWs do. The canonical software forwarding entity in most open source cloud deployments is the OVS [27]. OVS provides OpenFlow support as well as support for the ability to configure IP tunnels, one of several ways that the cloud operating system virtualizes the physical network, using the OVSDB protocol [28], a database management protocol specifically designed for configuring OVS. Performance of OVS even when optimized is quite variable with packet size. OVS can almost achieve 10G line rate performance for 1024-byte packets; however, the performance for 64-byte packets struggles to reach 1G [29]. Optimizations can improve forwarding above OVS performance [30]. Recent work indicates that an optimized version of Intel's Data Plane Development Kit (DPDK) [31], a collection of specialized libraries designed specifically for accelerating user plane applications and recent work, can easily achieve line speed on a 10G network interface card (NIC) [32]. However, mere achievement of high line speeds is not sufficient. Switching issues must still be addressed by OVS [33].

7.3 EPC in the Cloud Version 2.0?

The next step would go beyond simply moving the existing EPC network functions to a cloud by moving the EPC onto an SDN substrate, where control and user plane are completely separated as in Figure 7.3b. In this case, the control plane flows run into the control plane entities in the cloud, but the user plane flows run through dedicated switching hardware controlled by a protocol from the control plane entities. Certain use cases become easier to support if the control plane is SDN based. And as discussed in the previous section, the performance of forwarding on standardized Intel hardware may ultimately become enough of an issue to again recommend hardware specialized for forwarding. User plane devices located remotely may be easier to deploy than a data center. One possible implementation strategy could involve extending OpenFlow to handle routing of GTP[13] tunneled flows [34], which we will use as an example for purposes of discussion. Whether or not the EPC in the Cloud Version 2.0 is deployed depends on how quickly the performance of software-based switching becomes unacceptable and how important the use cases become that are difficult to implement using the centralized scheme. Here, we discuss a single use case, UE multihoming. Further use cases can be found in Ref. [34].

7.3.1 UE Multihoming

Support for multihomed UEs can add complexity, especially in mobile networks [35]. The problem is that the IP address acts as both a routing locator and an endpoint identifier.

[12] Indeed, the first routers deployed in the 1980s and early 1990s were implemented in software on minicomputers such as the VAX, and later on Sun Microsystems workstations. It was only later as the amount of traffic increased beyond what could be handled by software at line speed that routers were built with dedicated hardware.

[13] GPRS Tunneling Protocol.

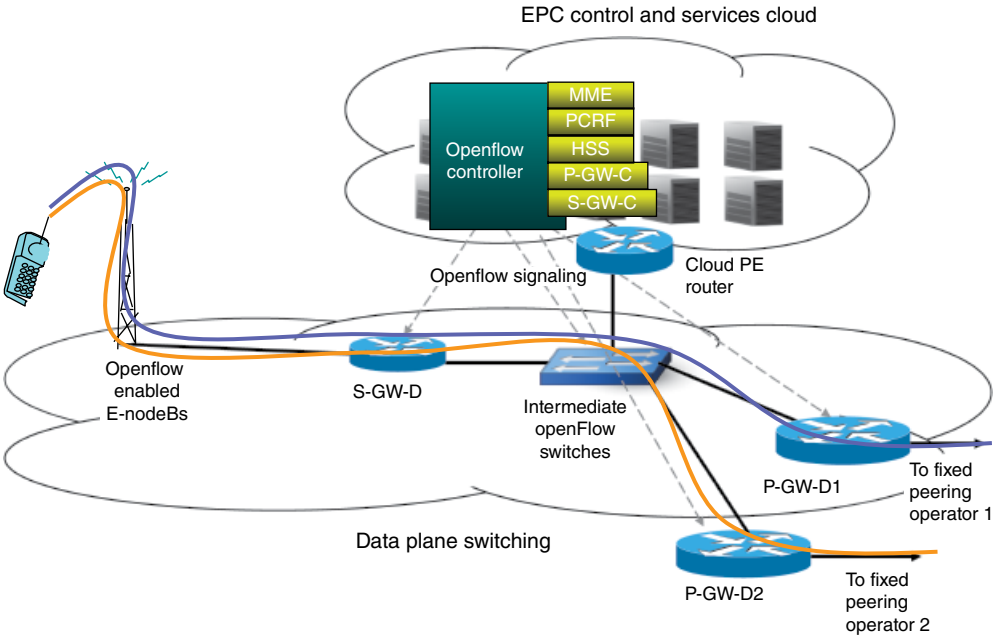

Figure 7.4 Upstream multihoming with OpenFlow. Kempf et al. (2012) [34]. Reproduced with permission from IEEE. © IEEE.

Unless the UE has two addresses, it is difficult in a standard distributed control plane IP-routed network to have a UE be multihomed. Aside from the issue of IPv4 address scarcity, the EPC uses the IP address as an endpoint identifier in GTP. With OpenFlow, however, multihoming is considerably simpler. OpenFlow treats IP addresses as pure endpoint identifiers, discarding the routing topology internally. Forwarding is done according to the programmed OpenFlow rules and not based on the IP network longest prefix matching. As a consequence, the EPC can advertise different sets of subnet prefixes externally and have different kinds of traffic directed to different gateways. Alternative schemes for supporting upstream multihoming sometimes don't provide adequate provision for operator accounting and charging. The mobile network PCRF needs to be involved in all decision making about where to place flows. OpenFlow even provides a collection of statistics that can be utilized for accounting and charging purposes.

As Figure 7.4 illustrates, the upper flow goes through upstream provider 1, while the lower flow goes through upstream provider 2. The OpenFlow controller and PCRF set up GTP tunnels to the P-GWs user plane with different IP addresses for the mobile UE externally, and these are rewritten at the P-GW to point to the same IP address on the mobile UE. The mapping is handled based on the application. The same technique could be used for handling multiple wireless interfaces. Techniques that do not use SDN for supporting multiple wireless interfaces in the network are quite complex [36, 37], while techniques requiring changes on the end nodes run up against the diversity of end nodes in today's mobile networks.

7.3.2 The EPC on SDN: OpenFlow Example

Using OpenFlow as an example control/user plane separation protocol, we can see how the EPC could be redesigned on an SDN platform. The following subsections describe the modifications needed in the OpenFlow switch architecture to support GTP TEID routing and the changes in the EPC architecture that an OpenFlow substrate would enable.

7.3.2.1 Switch Architectural Modifications

OpenFlow 1.4 supports an n-tuple of header match fields in the flow table, with one additional field matching metadata for communication of data collected by matches between tables. Unlike earlier versions of OpenFlow, the header field tuple size is not fixed because multiple MPLS labels can be pushed onto the header. The user plane protocols that are supported are Ethernet including carrier Ethernet (802.1aq) [38], MPLS, IPv4 and IPv6, and the IP L4 protocols (TCP, UDP, SCTP, DCCP, ARP, and ICMP). GTP TEID routing extends the tuple with two additional fields: the 2-byte GTP header flag field and the 4-byte GTP TEID field. The header flag fields are used to distinguish between GTP-U packets that are subject to fast path OpenFlow GTP TEID routing and other types of GTP packets including some GTP-U packets that need to be handled by the slow path on the switch.

In addition to the flow table extension, GTP TEID routing requires the addition of virtual ports to support encapsulation and decapsulation [39]. A *virtual port* is an abstraction that handles complex header manipulation specific to particular protocols. Virtual ports are particularly useful for tunneling protocols because they hide the complexities of the tunnel header manipulations from the forwarding pipeline implementation. On input, a virtual port accepts packets from a physical port or another virtual port; processes them to add, remove, or modify a tunnel header; and then passes the packets along to the next virtual port or inserts them into the flow table classifier pipeline. On output, virtual ports become the target for forwarding rules exactly as for physical ports, and they similarly add, remove, or modify a tunnel header, and then pass the packet along to another virtual port or to a physical port for output.

Virtual ports for processing GTP-U tunnel packets are needed on the Serving Gateway, the PDN Gateway, and on the wired network interfaces of the eNodeB.[14] GTP virtual ports are configured from the OpenFlow controller using a configuration protocol. The details of the configuration protocol are switch dependent; for an example of a standardized configuration protocol, see Ref. [40]. The configuration protocol must support messages that perform the following functions:

- Allow the controller to query for and return an indication whether the switch supports GTP fast path virtual ports and what virtual port numbers are used for fast path and slow path GTP-U processing.
- Allow the controller to instantiate a GTP-U fast path virtual port within a switch data path for use in the OpenFlow table *Set-Output-Port* action, and bind a GTP-U virtual port to a physical port.

[14] Scheduling constraints for the wireless interface could also be incorporated into the scheme in some fashion.

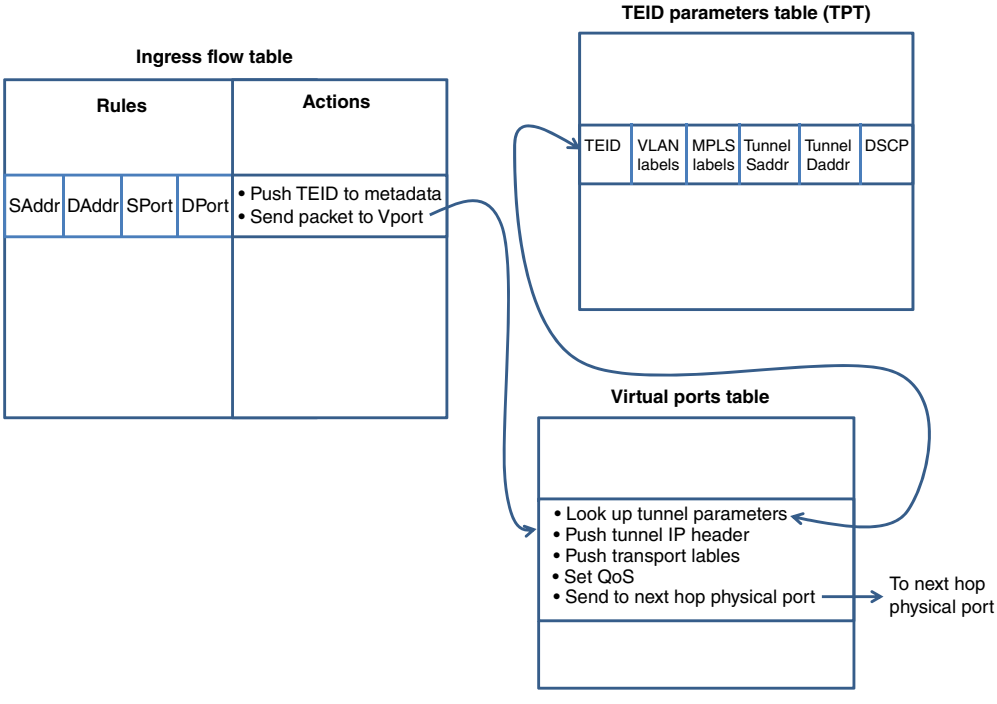

Figure 7.5 Tunnel ingress GTP OpenFlow gateway architecture.

The controller instantiates an encapsulation virtual port for each physical port on the eNodeB wired interface and on the interfaces of the S/P-GWs that may forward packets received from outside the EPC (i.e., from the UE or the Internet) to inside, and a decapsulation port that may forward packets from inside the EPC to outside.

Figure 7.5 contains a diagram of the switch architecture for the tunnel ingress side of the GTP OpenFlow gateway. An OpenFlow 1.4 GTP encapsulation gateway maintains a hash table mapping GTP TEIDs into the tunnel header fields for their bearers, called the TEID parameters table (TPT). The hash table stores the TEID, VLAN tags for the tunnel (if any) and MPLS labels for the tunnel (if any), the tunnel source and destination IP addresses, and any DSCP markings for QoS. The TEID hash keys are calculated using a suitable hash algorithm with low collision frequency.

The table maintains one such row for each GTP TEID/bearer originating on the gateway. The TEID field contains the GTP TEID for the tunnel. The VLAN tags and MPLS labels, if used by the EPC transport network, are ordered within the corresponding label fields and define transport network tunnels into which the packet can be routed. The labels also include the VLAN priority bits and MPLS traffic class bits. The tunnel origin source IP address contains the address on the encapsulating gateway to which any control traffic involving the tunnel should be directed (e.g., error indications). The tunnel end destination IP address field contains the IP address of the gateway to which the tunneled packet should be routed, where the packet will be decapsulated and removed from the GTP tunnel. The QoS DSCP field contains the DiffServ code point, if any, for the bearer. This field may be empty if the bearer

is a default bearer with best effort QoS but will contain nonzero values if the bearer QoS is more than best effort. An OpenFlow GTP gateway also supports three slow path software ports for GTP traffic that is not handled by GTP fast path routing. Slow path forwarding is handled by the switch control plane software.

Tunnels are managed by the OpenFlow controller in the following way. In response to a GTP-C control packet requesting that a tunnel be set up, the OpenFlow controller programs a gateway switch on the tunnel ingress side to install rules and actions in the flow table and TPT entries for routing packets into GTP tunnels via a fast path GTP encapsulation virtual port. The rules match the packet filter for the input side of GTP tunnel's bearer. Typically, this will be a 4-tuple of IP source address, IP destination address, UDP/TCP/SCTP source port, and UDP/TCP/SCTP destination port. The IP source address and destination address are typically the addresses for user plane traffic, that is, a UE or Internet service with which a UE is transacting, and similarly with the port numbers. An action is installed in the flow table to forward the packet to a virtual port bound to the next-hop physical port and to write the tunnel's GTP TEID directly into the metadata.

When a packet header matches the packet filter fields in a GTP TEID routing, the GTP TEID is written into the lower 32 bits of the metadata and the packet is directed to the virtual port. The virtual port calculates the hash of the TEID and looks up the tunnel header information in the TPT. The virtual port then constructs a GTP tunnel header and encapsulates the packet. Any VLAN tags or MPLS labels are pushed onto the packet to ensure proper transport routing, and any DSCP bits or VLAN priority bits are set in the IP or MAC tunnel headers to ensure proper QoS. The encapsulated packet is then forwarded out the bound physical port.

On the egress side of the GTP tunnel, the OpenFlow controller installs rules and actions for routing GTP encapsulated packets out of GTP tunnels. The rules match the GTP header flags and the GTP tunnel endpoint IP address for the packet as follows:

- The IP destination address is the IP address of GTP tunnel termination that is on the switch.
- The IP protocol type is UDP (17).
- The UDP destination port is the GTP-U destination port (2152).
- The GTP header fields match a GTP-U packet without any extension headers.

If the rules match, the action is to forward to the virtual port. The virtual port simply removes the GTP tunnel header and any transport labels and forwards the user plane payload out the bound physical port.

Figure 7.6 contains a diagram of the switch architecture for the tunnel egress GTP OpenFlow gateway. Both gateway and nongateway switches in the EPC can also utilize GTP TEID routing to route packets in individual GTP tunnels to a specific destination. A flow table rule matches on the TEID for the tunnel in question, and the action forwards the tunneled packet out the next-hop port.

7.3.2.2 Architectural Modifications to the EPC on an OpenFlow SDN Substrate

GTP TEID routing allows the GTP control plane part of the S/P-GWs to be, in effect, separated from the gateway and situated in the OpenFlow controller as an OpenFlow application. Note that in almost all cases, the user plane packets do not need to be redirected to the controller. Flow routes are computed with policy applied by the PCRF, which itself may be

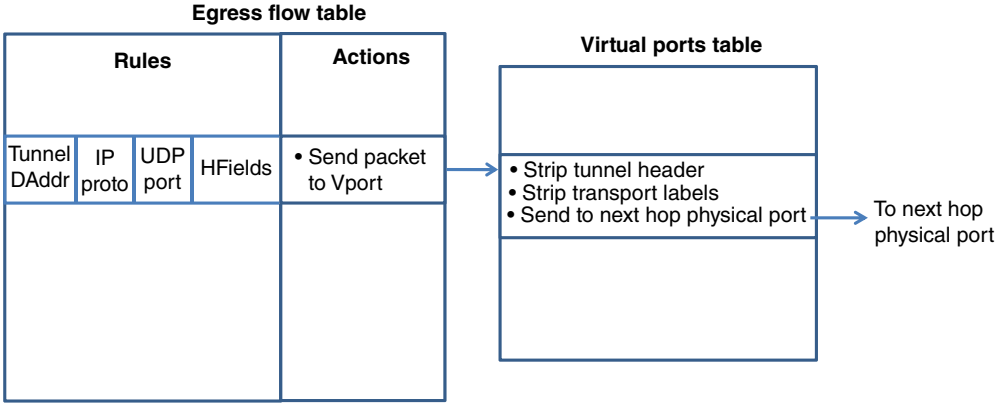

Figure 7.6 Tunnel egress GTP OpenFlow gateway architecture.

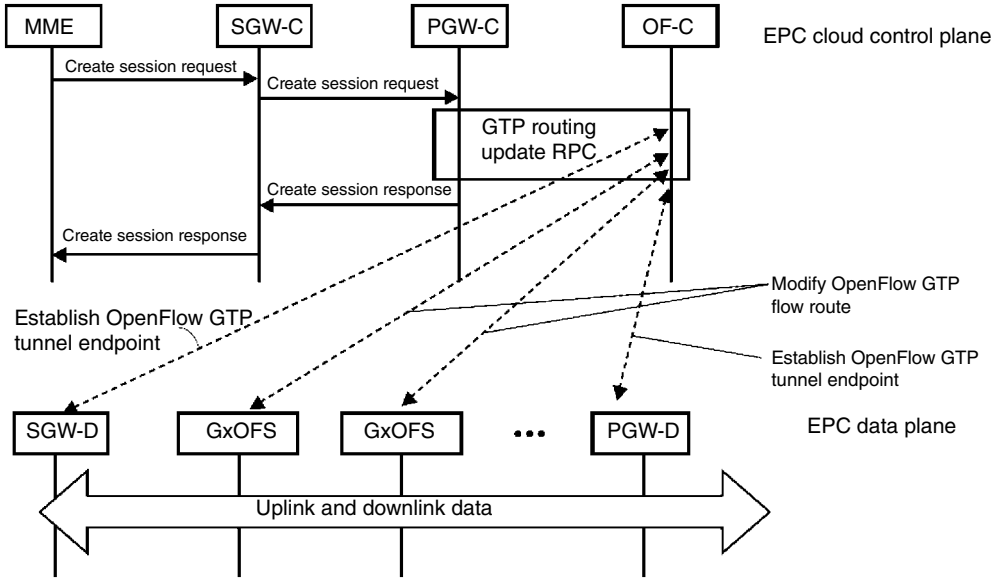

Figure 7.7 GTP *Create_Session*_Request on OpenFlow. Kempf et al. (2012) [34]. Reproduced with permission from IEEE. © IEEE.

structured on top of an SDN platform [41]. We present an example of how EPC control operations can be modified by this capability. In the example, we assume that communication between the gateway GTP control plane entities and the OpenFlow controller, both running in the cloud, happen via RPC.

The example is the creation of a new bearer and associated GTP tunnels. Bearers and GTP tunnels are set up using the GTP-C *Create_Session_Request* message. This procedure is used in a variety of message sequences, for example, in the E-UTRAN *Initial_Attach* procedure described in Section 5.3.2.1 of Ref. [9]. In Figure 7.7, the OpenFlow message flows for the *Create_Session_Request* message are shown. In the figure and following discussion, OF-C is

the OpenFlow controller, SGW-C is the Serving Gateway control plane entity, PGW-C is the PDN Gateway control plane entity, SGW-D is the Serving Gateway GTP enhanced OpenFlow switch, PGW-D is the PDN Gateway GTP enhanced OpenFlow switch, and GxOFS is a nongateway GTP enhanced OpenFlow switch.

The MME sends a *Create_Session_Request* to the SGW-C, and the SGW-C sends the request to the PGW-C. The PGW-C calls into the OF-C through a *GTP_Routing_Update* RPC, requesting the OF-C to establish a new GTP tunnel endpoint at the SGW-D and PGW-D and to install routes for the new GTP bearer/tunnel on intermediate switches, if necessary.

The OF-C issues a sequence of OpenFlow messages to the appropriate GTP enhanced OpenFlow switches on the user plane. The sequence begins with an *OFP_BARRIER_REQUEST* to ensure that there are no pending messages that might influence processing of the following messages. Then, an *OFPT_FLOW_MOD* message is issued, with a GTP extension to the match field. The message specifies actions and instructions to establish a flow route for the GTP tunnel that encapsulates and decapsulates the packets through the appropriate virtual port. In addition, immediately following the *OFPT_FLOW_MOD* message, the OF-C issues an OpenFlow vendor extension message to the gateways containing the TPT entries for the encapsulation virtual port. The two OpenFlow messages are followed by an *OFPT_BARRIER_REQUEST* message to force the gateways to process the flow route and TEID hash table update before proceeding.

Prior to returning from the *GTP_Routing_Update* RPC, the OF-C also issues GTP flow routing updates to any GTP extended OpenFlow switches (GxOFSs) that need to be involved in customized GTP flow routing. The messages in these updates consist of an *OFP_BARRIER_REQUEST* followed by an *OFPT_FLOW_MOD* message containing the GTP match extension for the new GTP flow the actions and instructions described in Section 2.2.1 for customized GTP flow routing. A final *OFP_BARRIER_REQUEST* forces the switch to process the change before responding. The flow routes on any GxOFSs are installed after installing the GTP tunnel endpoint route on the SGW-D and prior to installing the GTP tunnel endpoint route on the PGW-D, as illustrated in the figure. The OF-C does not respond to the PGW-C RPC until all flow routing updates have been accomplished.

Once the RPCs have returned, the PGW-C and SGW-C return *Create_Session_Response* messages. When the MME receives such a response, it can signal the eNodeB with the *Initial_Context_Setup_Request/Attach_Accept* message indicating that the terminal is free to start using the bearer/tunnel.

Similar results can be achieved for other GTP-C operations.

7.4 Incorporating Mobile Services into Cross-Domain Orchestration with SP-SDN

In order to support integrated, cross-domain service development, the EPC services need to be encapsulated by APIs that represent usable abstractions to allow service programmers access. Here, we discuss an example of how an API on the PCRF can enable flexible creation and control of cross-domain services. The specific example is enabling on-demand, policy-driven QoS for mobile cloud services [21].

3G and 4G mobile networks provide the ability to control mobile service QoS through the PCRF. The PCRF communicates with the Subscriber Policy Repository (SPR), which contains the subscription records for users, to authenticate and authorize services, and the S/P-GWs to implement routing policy, such as marking packets with DiffServ code points for enhanced

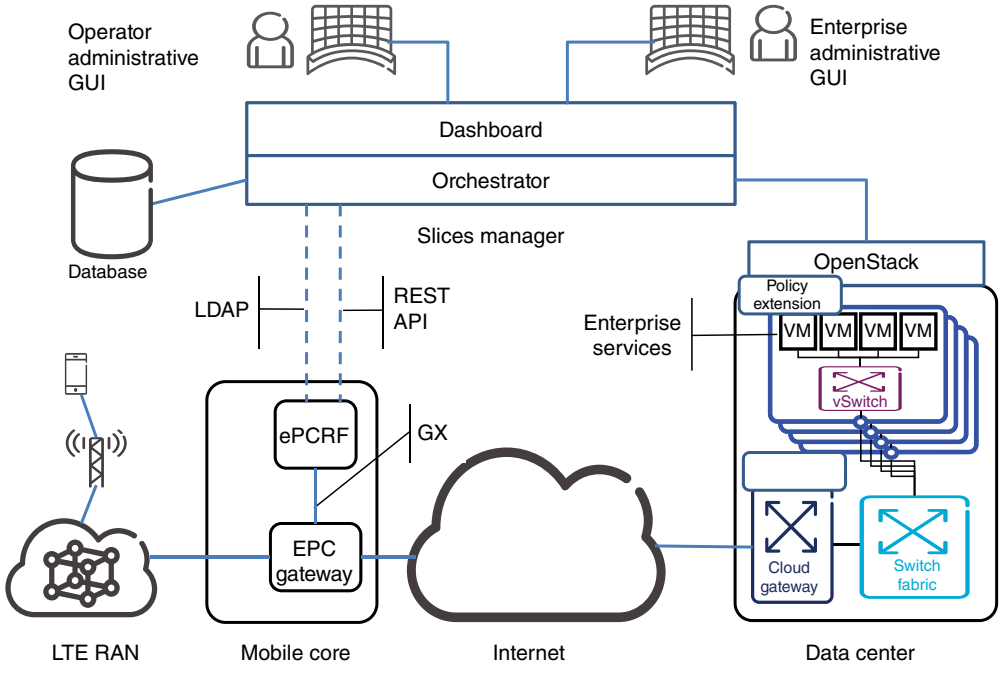

Figure 7.8 On-demand policy-driven QoS architecture for end-to-end network slices.

QoS [42]. It is possible to implement the PCRF according to SOC design principles in an enhanced function that we denote as the enhanced PCRF (ePCRF). The ePCRF would be structured as a service with a REST API to provide control access for QoS policy to a cross-domain controller. One desirable capability is the ability to relate policy control to charging. Such a capability would realize a flexible approach to policy and charging for mobile networks, and it can be combined with other services. Initial investigations in this direction have been demonstrated [43].

In the cloud domain, an OpenStack cloud gateway allows the cross-domain controller to configure notification of mobile service access when mobile devices connect to cloud services. Figure 7.8 illustrates the on-demand policy-driven QoS architecture. An orchestration layer implements the cross-domain Service Control function and provides user interface (UI) access to the network operator and enterprise customers with enterprise accounts in the cloud and the mobile network. The cloud data center supplies network, compute, and storage resources for enterprise services, including mobile cloud services such as streaming video, which may need enhanced QoS in the mobile network. The cloud data center is connected to the wide area network through a cloud gateway, essentially a software router in a VM, which includes the mobile service access notification enhancement.

In what follows, we provide some ideas on the possible enhancements to the PCRF; these directions are however by no means agreed on. The ePCRF REST API would expose abstractions for the following objects in the mobile network:

- The capabilities available for a particular subscriber: Depending on their subscription plan, a subscriber and service may have different rights to QoS.

- A mobile session identifier to determine which session is being controlled: Sessions are associated with subscribers and their rights to enhanced QoS treatment are determined by the subscriber profile.
- A packet flow within a session that may be entitled to enhanced QoS treatment: Packet flows are identified by a maximum requested bandwidth, a priority, a resource reservation, and the GTP bearer identification 5-tuple—the source IP address, the source port, the destination IP address, the destination port, and the protocol.

The REST API can further support operations to get the capabilities associated with a subscriber, to create and delete a session, and to obtain session status. It also supports a full set of operations on flows: add and remove a flow, update a flow, and get the status of a flow. As an example of mobile service orchestration, consider a use case where an enterprise has an account with a mobile operator and an OpenStack cloud service provider (the cloud and mobile network could be under the control of the same organization or different organizations, e.g., a public cloud). The mobile operator sets up an account for the enterprise through the mobile operator GUI, and the Orchestrator utilizes the OpenStack Keystone service to obtain tokens from OpenStack for configuring cloud resources. These are stored in the Orchestration database. The enterprise administrator adds mobile services entitled to enhanced QoS through the enterprise administrator's GUI. The Orchestrator obtains the global IP address of these services from the OpenStack Nova service. The service IP address along with the name and other information are stored in the Orchestration database. The enterprise administrator also adds subscribers entitled to obtain enhanced QoS for particular services. The subscribers' International Mobile Subscriber Identity (IMSI) and policy profile are obtained from the ePCRF and recorded. When a subscriber starts a device, the ePCRF reports the IP address of the device, identified by the IMSI, to the Orchestrator. When a device entitled to enhanced QoS service accesses a mobile cloud service, the mobile cloud service access notifier in the OpenStack cloud gateway triggers and reports the source and destination IP addresses to the Orchestrator. The Orchestrator issues an *updateFlow* message to the ePCRF to update the QoS privileges on the flow, and the flow is moved to an enhanced 3GPP bearer.

7.5 Summary and Conclusions

The next 10 years are likely to see more changes in the mobile packet core than in past ten years. The LTE packet core is in many ways an evolution of the original GPRS service that was provided with GSM and was introduced in 2000 [44]. The pressures of the expected media traffic volume are likely to require higher volume forwarding performance. The operator requirements for simplified cross-domain service construction to support rapid innovation on the same scale as Internet service providers are likely to drive a restructuring of the EPC software into services. Yet some aspects of the EPC are unlikely to change much if at all. Although using SDN may remove the need for having a physical mobility anchor where the IP address has both locator and identifier function, GTP provides a virtualization solution for mobile networks similar to VXLAN[15] [45] and GRE[16] [46] in fixed networks and therefore is likely to remain if in somewhat modified form. Mobile operators will continue to want the

[15] Virtual extensible LAN.
[16] Generic routing encapsulation.

ability to flexibly provide different QoS to different traffic types and to charge users for the service in a flexible manner, so some version of the PCRF and HSS is likely to remain. And of course the unparalleled ability of the EPC to efficiently support multiple radio types is required. These aspects of the EPC have been very successful and are worth preserving. In this chapter, we've discussed a few options about how the EPC might evolve, but the real contours of its shape are as yet unclear.

References

[1] J. Kempf, P. Nikander, and H. Green, "Innovation and the Next Generation Internet," Infocom IEEE Workshops on Computer Communications Workshops, March 2010.

[2] N. McKeown, "Software Defined Networking," Keynote talk, INFOCOM, April 2009. Available at: http://www.cs.rutgers.edu/~badri/552dir/papers/intro/nick09.pdf (accessed on February 18, 2015).

[3] P. Hui and T. Koponen, "Report on the 2012 Dagstuhl Seminar on Software Defined Networking," September 2012. Available at: http://vesta.informatik.rwth-aachen.de/opus/volltexte/2013/3789/pdf/dagrep_v002_i009_p095_s12363.pdf (accessed on February 18, 2015).

[4] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng, J. Benitez, U. Michel, H. Damker, K. Ogaki, T. Matsuzaki, M. Fukui, K. Shimano, D. Delisle, Q. Loudier, C. Kolias, I. Guardini, E. Demaria, R. Minerva, A. Manzalini, D. López, F. Salguero, F. Ruhl, P. Sen, "Network Functions Virtualization," Position paper presented at SDN and OpenFlow World Congress, 2012, 16 pp. Available at: http://portal.etsi.org/NFV/NFV_White_Paper.pdf (accessed on February 18, 2015).

[5] M. Chiosi, S. Wright, D. Clarke, P. Willis, C. Donley, L. Johnson, M. Bugenhagen, J. Feger, W. Khan, C. Cui, H. Deng, C. Chen, L. Baohua, S. Zhenqiang, X. Zhou, C. Jia, J. Benitez, U. Michel, K. Martiny, T. Nakamura, A. Khan, J. Marques, K. Ogaki, T. Matsuzaki, K. Ok, E. Paik, K. Shimano, K. Obana, B. Chatras, C. Kolias, J. Carapinha, DK Lee, K. Kim, S. Matsushima, F. Feisullin, M. Brunner, E. Demaria, A. Pinnola, D. López, F. Salguero, P. Waldemar, P. Grønsund, G. Millstein, F. Ruhl, P. Sen, A. Malis, S. Sabater, A. Neal, "Network Functions Virtualization 2," Position paper presented at SDN and OpenFlow World Congress, 2013, 16pp. Available at: http://portal.etsi.org/nfv/nfv_white_paper2.pdf (accessed on February 18, 2015).

[6] ETSI, "NFV," 2013. Available at: http://portal.etsi.org/portal/server.pt/community/NFV/367 (accessed on February 18, 2015).

[7] Open Network Foundation, "Software Defined Networking Definition," September 2013. Available at: https://www.opennetworking.org/sdn-resources/sdn-definition (accessed on February 18, 2015).

[8] ETSI, "Network Functions Virtualization," September 2013. Available at: http://www.etsi.org/technologies-clusters/technologies/689-network-functions-virtualisation (accessed on February 18, 2015).

[9] "LTE: General Packet Radio Service (GPRS) Enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) Access," Release 10, 3GPP, Version 10.5.0, TS 123.401, 2011.

[10] "OpenFlow Switch Specification: Version 1.0 (Wire Protocol 0x01)," Open Network Foundation, December 2009. Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf (accessed on February 18, 2015).

[11] IEEE Std. 802.1Q-2011, "Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks," Institute of Electrical and Electronics Engineers, 2011.

[12] "OpenFlow Switch Specification: Version 1.4 (Wire Protocol 0x05)," Open Network Foundation, October 2013. Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf (accessed on February 18, 2015).

[13] Open Virtual Switch. Available at: http://www.openvswitch.org (accessed on January 20, 2015).

[14] A. Singla and B. Rijsman, "Contrail Architecture," Juniper Networks, 2013. Available at: http://www.juniper.net/us/en/local/pdf/whitepapers/2000535-en.pdf (accessed on February 18, 2015).

[15] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120, Internet Engineering Task Force, March 2011.

[16] OpenDaylight Consortium, "OpenDaylight," 2013. Available at: http://www.opendaylight.org/ (accessed on February 18, 2015).

[17] Wikipedia, "Representational State Transfer," 2013. Available at: http://en.wikipedia.org/wiki/Representational_state_transfer (accessed on February 18, 2015).

[18] "IP Multimedia Subsystem (IMS); Stage 2," 3GPP, TS 23.228, Release 9, 2010.
[19] Amazon Web Services (AWS), "Cloud Computing Services," 2014. Available at: http://aws.amazon.com/ (accessed on February 18, 2015).
[20] OpenStack Foundation, "OpenStack Open Source Cloud Computing," 2013. Available at: http://www.openstack.org/ (accessed on February 18, 2015).
[21] J. Kempf, M. Körling, S. Baucke, S. Touati, V. McClelland, I. Más, and O. Bäckman, "Fostering Rapid, Cross-domain Service Innovation in Operator Networks through Service Provider SDN," Proceedings of the ICC, June 2014.
[22] Wikipedia, "Service Oriented Architecture," 2014. Available at: http://en.wikipedia.org/wiki/Service-oriented_architecture (accessed on February 18, 2015).
[23] Wikipedia, "SOAP," 2013. Available at: http://en.wikipedia.org/wiki/SOAP (accessed on February 18, 2015).
[24] G. Karagiannis, A. Jamakovicy, A. Edmondsz, C. Paradax, T. Metsch, D. Pichonk, M. Corici, S. Ruffinoyy, A. Gomesy, P. S. Crostazz, T. M. Bohnertz, "Mobile Cloud Networking: Virtualisation of Cellular Networks". Available at: http://www.iam.unibe.ch/~jamakovic/MCN_ICT2014_IEEE.pdf (accessed on January 20, 2015).
[25] "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); General Packet Radio Service (GPRS); GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface", 3GPP, TS 129 060 version 10.1.0, 2011.
[26] OpenSAF: The Open Service Availability Framework. Available at: http://www.opensaf.org/ (accessed January 20, 2015).
[27] Open vSwitch. Available at: http://www.openvswitch.org/ (accessed January 20, 2015).
[28] B. Pfaff and B. Davie, "The Open vSwitch Database Management Protocol," RFC 7047, Internet Engineering Task Force, December 2013.
[29] M. Honda, F. Huici, G. Lettieri, L. Rizzo, and S. Niccolini, "Accelerating Software Switches with Netmap," Proceedings of the European Workshop on SDN, 2013. Available at: http://www.ewsdn.eu/previous/presentations/Presentations_2013/mswitch-ewsdn.pdf (accessed on February 18, 2015).
[30] L. Rizzo, "Netmap: A Novel Framework for Fast Packet I/O," Proceedings of the USENIX ATC Conference, 2012.
[31] Intel Corporation, "Intel® Data Plane Development Kit (Intel® DPDK) Overview Packet Processing on Intel® Architecture," December 2012. Available at: http://www.intel.com/content/dam/www/public/us/en/documents/presentation/dpdk-packet-processing-ia-overview-presentation.pdf(accessed on February 18, 2015).
[32] 6WIND, "6WIND Continues 195 Gbps Accelerated Virtual Switch Demo at SDN and NFV Summit in Paris." Available at: http://www.6wind.com/blog/6wind-continues-195-gbps-accelerated-virtual-switch-demo-at-nfv-and-sdn-summit-in-paris-march-18-21/(accessed on February 18, 2015).
[33] G. Pongrácz, L. Molnár, Z. L. Kis, and Z. Turányi, "Cheap Silicon: A Myth or Reality? Picking the Right Data Plane Hardware for Software Defined Networking," Proceedings of HotSDN, 2013.
[34] J. Kempf, B. Johansson, S. Pettersson, H. Lüning, and T. Nilsson, "Moving the Mobile Evolved Packet Core to the Cloud," Proceedings of the IEEE Wireless and Mobility Conference, November 2012.
[35] A. Mihailovic, G. Leijonhufvud, and T. Suihko, "Providing Multi-Homing Support in IP Access Networks," The 13th International Symposium on Personal, Indoor, and Mobile Radio Communications, 2002.
[36] S. Ahson and M. Ilyas, *Fixed Mobile Convergence Handbook*. Boca Raton: CRC Press, 2011.
[37] S. Radosavac, J. Kempf, and U. Kozat, "Security Challenges for the Current Internet Architecture: Can Network Virtualization Help?," NetEcon '08: Workshop on the Economics of Network Systems and Computation, 2008.
[38] Metro Ethernet Forum, "Metro Ethernet Network Architecture Framework—Part 1: Generic Framework," March 2004. Available at: http://www.metroethernetforum.org/Assets/Technical_Specifications/PDF/MEF4.pdf (accessed on February 18, 2015).
[39] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, and H. Green, "OpenFlow MPLS and the Open Source Label Switched Router," Proceedings of the International Teletraffic Conference, IEEE, San Francisco, September 2011.
[40] "OF-CONFIG 1.2: OpenFlow Management and Configuration Protocol," Open Network Foundation, ONF TS-016, 2014.
[41] M. Amani, T. Mahmoodi, M. Tatipamula, and H. Aghvami, "Programmable Policies for Data Offloading in LTE Network," Proceedings of ICC, June 2014.
[42] "Universal Mobile Telecommunications System (UMTS): Policy and Charging Control over Rx Reference Point," 3GPP, TS 129 214 version 7.4.0, 2008.

[43] F. Castro, I. M. Forster, A. Mar, A. S. Merino, J. J. Pastor, and G. S. Robinson, "SAPC: Ericsson's Convergent Policy Controller," Ericsson Review, (January), 2010.

[44] 3GPP, "GPRS and EDGE." Available at: http://www.3gpp.org/technologies/keywords-acronyms/102-gprs-edge (accessed on January 20, 2015).

[45] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks," Internet Draft, work in progress. Available at: https://datatracker.ietf.org/doc/draft-mahalingam-dutt-dcops-vxlan/?include_text=1 (accessed January 20, 2015).

[46] S. Hanks, T. Li, D. Farinacci, and P. Traina, "Generic Routing Encapsulation (GRE)," RFC 1701, Internet Engineering Task Force, October 1994. Available at: http://www.rfc-editor.org/rfc/rfc1701.txt (accessed January 20, 2015).