# 7

# Mobile Applications on Global Clouds Using OpenFlow and Software-Defined Networking

SUBHARTHI PAUL, RAJ JAIN, JAY IYER, AND DAVE ORAN

**Contents**

## Introduction

In recent years, there has been an explosive growth in mobile applications (apps), most of which need to serve global audiences. This increasing trend of service access from mobile computing devices necessitates more dynamic application deployment strategies. Cloud computing provides unique opportunities for these application service providers (ASPs) to manage and optimize their distributed computing resources. For this trend to be successful, similar facilities need to be developed for the on-demand optimization of connectivity resources to enhance user experience.

Application delivery, on the surface, is simply connecting a user to a server. The original host-centric Internet architecture was well designed for this end-to-end two-host communication using source and destination IP addresses or names. However, today's Internet is mostly service centric, where users are interested in connecting to a service instead of a particular host. Delivering services over a host-centric design has led to complex application deployment environments, as discussed below, for the case of a single private data center (Figure 7.1), a single cloud environment, and a multicloud environment.

### *Private Data Center*

*Service Replication* To scale a service, the service needs to be replicated over multiple servers. Network-layer load balancers were introduced into the network datapath to dynamically map the requests to

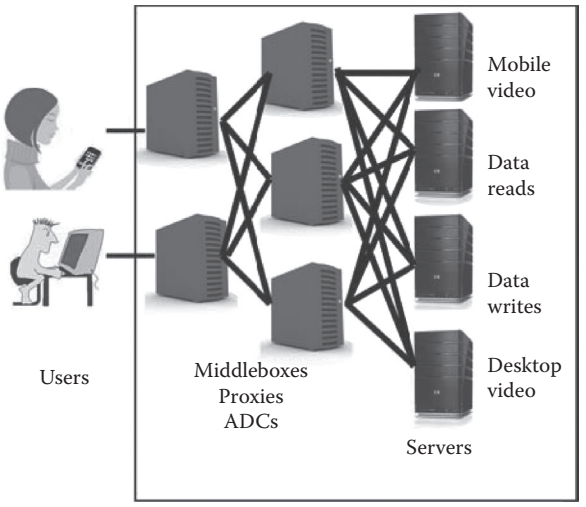


**Figure 7.1** Application delivery in a private data center.

the least loaded server. Pure control plane mechanisms, such as rotating domain name system (DNS), provide static load balancing.

*Service Partitioning* To improve performance, services often need to be partitioned, and each partition is hosted on a separate server. Each partition may be further replicated over multiple servers. A service may be partitioned based on content and/or context.

*Content-Based Partitioning* Even for the same service (e.g., xyz.com), accounting messages, recommendation requests, and video requests are all sent to different server groups. This can be done either by giving a different DNS name to each content type server or, as is common, by putting a proxy that classifies messages based on content.

*Context-Based Partitioning* User context, network context, or service context may require the application messages to be routed differently. An example of user context is a mobile smart phone user versus a desktop user. An example of network context is the geographical location of the user and the state of network links. An example of service context is that database reads and writes may be sent to different servers. Context-based partitioning is supported by proxies that classify messages based on the context.

*Service Composition* A service may represent a composed context where accessing the service actually requires going through a sequence of devices providing security (e.g., firewalls, intrusion detection systems [IDSs]), transformation/translation (e.g., transcoders, data compression), and performance enhancement (e.g., secure socket layer [SSL] offloaders, wide area network [WAN] optimizers) functions to the service deployment. These services, either provided by separate devices, generically called middleboxes, or by monolithic, integrated platforms, generically called application delivery controllers (ADCs), are now very common. In fact, the number of middleboxes in a data center is comparable to the number of routers [1].

*Multisegments* In many of the aforementioned examples, the proxy/middlebox/ADC terminates the transmission control protocol (TCP) and starts a new TCP connection. In general, a user-to-server connection is no longer end to end; it consists of many segments. Each of these segments can be served by multiple destinations (based on the replication of middleboxes). The ASPs implement complex application policy routing (APR) mechanisms inside the data centers.

*TCP Multiplexing* A special case of multisegments is TCP multiplexing. The goal is to improve the latency over long TCP connections by minimizing the effect of TCP's self-regulating window adjustment algorithm. ASP may place several proxy servers in different geographical locations so that the TCP connection between the user and the proxy has a short round trip and the window increases fast. Although the connection between the proxy and the server is long, it is permanently connected and, therefore, operates at optimally large windows. This technique is commonly used for short data transfers, such as Web services.

### *Single Cloud Environment*

In a private data center, as discussed above, high-capacity monolithic proxies are used for APR. When the applications move to a public cloud environment, the situation becomes more complex because private monolithic hardware boxes cannot be deployed. Moreover, the

ASP may not allow the cloud service provider (CSP) to look at its data for APR, and so software-based virtual machines are used for such services, and the amount of replication and dynamics (caused by Virtual Machine [VM] mobility and hardware failures) is much more common than a private data center.

*Multicloud Environment*

Mobile apps that need to serve global audiences can easily get computing and storage facilities using cloud services from multiple cloud providers distributed throughout the world, for example, Amazon, Google, Rackspace, Microsoft, and so on. However, the problem of routing using ASP's policies in a very dynamic multicloud environment is not possible because Internet service providers (ISPs) offer no service to dynamically route messages to a different server using an ASP's policies.

Enterprises that operate multiple data centers already have this problem. For example, Google operates multiple data centers across different geographical locations and has installed a WAN-like infrastructure [2] that intercepts most of the traffic for Google-owned services at edge-network points of presence (POPs) and sends them over its private high-speed WAN infrastructure. At these POPs, Google (probably) operates application layer (layers 5–7) proxies to intelligently route service requests to appropriate data centers. However, for smaller ASPs, it is prohibitively expensive to operate such global networking infrastructures. Moreover, Google proxies have a complete view of the application data, which may not be desirable if such forwarding decisions were to be made by ISPs.

Our vision is to design a new session-layer abstraction called open application delivery networking (OpenADN) that will allow ISPs to offer services similar with Google WAN to smaller ASPs. ASPs can express and enforce application-traffic management policies and application delivery constraints to ISPs. It allows ASPs to achieve all the application delivery services that they use today in private data centers (mentioned above) in the global multicloud environment. As shown in Figure 7.2, using OpenADN aware data plane entities, any new ASP can quickly set up its service by using ADN services provided by ISPs.

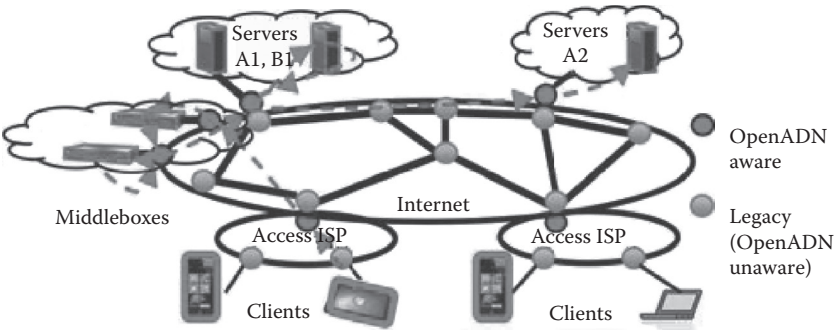


**Figure 7.2** Open application delivery network.

To achieve this, we combine the following innovations:

1. OpenFlow
2. Software-defined networking (SDN)
3. Session splicing
4. Cross-layer communication
5. Multistage late binding
6. Identifiers (IDs)/locator split
7. MPLS-like application flow labels
8. Rule-based delegation

The rest of this chapter is organized as follows: "SDN and Open Flow" briefly explains the features of OpenFlow and SDN, which are helpful in our goal. "OpenADN Concepts" explains the several new concepts that we need to explain OpenADN. "OpenADN Extensions" discusses the aforementioned extensions that make OpenADN possible. "Related Work" provides a brief survey of the background literature, followed by the "Summary."

### SDN and OpenFlow

SDN is an approach toward taming the configuration and management complexities of large-scale network infrastructures through the design of proper abstractions. It proposes a separation between the network control and data planes. This would allow the control plane to be logically centralized, making it easier to manage and configure the distributed data plane components (switches and routers)

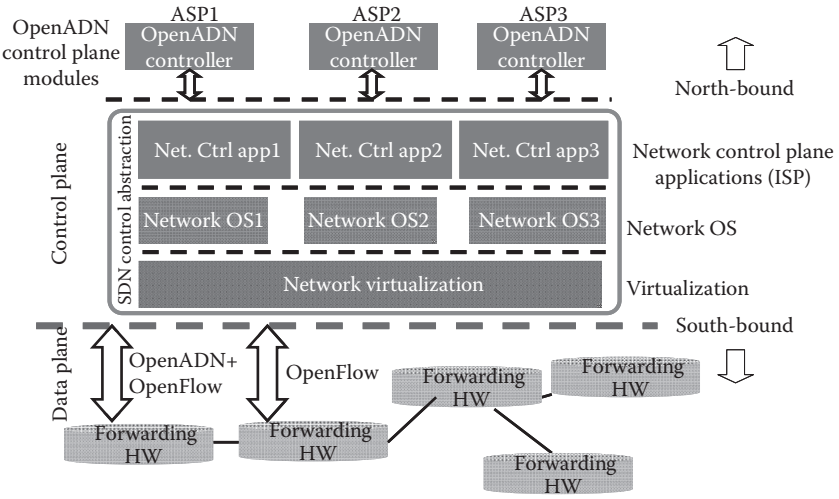


**Figure 7.3** OpenADN uses meta-tags in the headers, to be used by forwarding elements as an extension of OpenFlow, and uses a north-bound interface from the controller for policy communication. HW, hardware; OS, operating system; Net. Ctrl, network controller.

involved in the actual packet forwarding. To implement this separation, SDN needs to design a basic abstraction layer interposed between the control plane and the data plane. This is explained further in Figure 7.3.

The evolving OpenFlow [3] standard is the protocol used between the controller and data plane forwarding entities (see Figure 7.3). The data plane entities are significantly simplified compared with today's switches in the sense that they do not do the usual control plane activities of preparing forwarding tables. Instead, they simply classify the packets based on their headers and use the forwarding table prepared by the central controller. This simplification of the data plane is expected to result in a significant cost savings in data centers where a large number of switches are used. This is one of the reasons why the industry is having an interest in this technology.

A flow in OpenFlow refers to a set of packets that are classified into the same policy class based on a combination of layer 2, layer 2.5 (MPLS), layer 3, and layer 4 header fields. We call such flows network-level flows. In OpenFlow, all packets are classified into separate network flow classes, and all packets belonging to the same network flow class are applied with the same control plane policies. Examples of control plane policies include different forwarding mechanisms (e.g., multicast,

unicast) and different traffic engineering policies optimizing different parameters, such as energy efficiency, congestion, latency, etc.

### OpenADN Concepts

In this section, we explain some of the concepts required to understand the OpenADN extensions that are described in the next section.

#### *Waypoints*

As previously discussed, the path between the user and the server may consist of multiple segments connecting intermediate middleboxes, proxies, or ADCs. Some of the middleboxes terminate TCP, and some do not. We use the term waypoint to indicate all such intermediate nodes. If a service is composed of multiple services, the packets may be forwarded to multiple servers, with intermediate nodes between those servers. The waypoints include these intermediate servers as well.

#### *Streams*

Each waypoint may be replicated for fault tolerance or performance. We call a single stage connection between two specific nodes a stream. Thus, each segment may have multiple available streams. Once a session is assigned to a particular stream (a particular destination waypoint), all packets of that session will follow that stream. This is called session affinity.

#### *Application-Level Policies*

Application-level policies specify the rules for forwarding the application traffic to help ASPs manage their distributed and dynamic application deployment environments.

The ASPs may design policies for setting up various session segments based on replication, content-based partitioning, and context-based partitioning as previously explained. These policies, when implemented in the network, will provide optimal user experience.

*Application Flows*

Enforcing the application-level policies in the Internet requires ISP to classify application traffic into separate application-specific application flow classes. This is currently not possible because this information is not available in the headers.

Note that application flows are different from network flows because the packets with the same L1 to L4 headers used to determine the network flow may contain different data types that need to be routed differently (a.k.a. content-based routing).

OpenFlow has a very limited context for expressing application-level policies through the transport layer port number and transport protocol ID header fields. This is inadequate for designing control applications for managing application-traffic flows.

OpenADN solves this problem by putting meta-tags in the message header, which help classify the packets appropriately. Thus, the data can be kept private from the switches and the routers. Of course, the waypoints that need to operate on such data (virtual machines or virtual appliances) will be under the control of the ASP.

In OpenADN, after the application-level flow classification is done at the network edge, the application flow class is included as a meta-tag in the OpenADN header, as discussed later in this section.

*Affinity*

OpenADN is designed for a very dynamic global environment in which each intermediate point is replicated and may move inside or among clouds while the virtual machine on which it is running moves. We need some rules on how often to change forwarding decisions. This is called affinity. OpenADN offers both message affinity and session affinity.

*Message Affinity* All packets that are part of an application-layer message need to be classified into the same application flow class and applied the same application-level policy.

*Session Affinity* All segments of a session are initially bound to appropriate physical segments. Each segment is bound to a particular stream in that segment. This binding remains until the end of the session, where the definition of the end of the session is application specific (explicit or implicit). Setting up such multisegment sessions is called session splicing.

#### *Sender and Receiver Policies*

OpenADN does not distinguish between the sender and the receiver, and allows both the sender and the receiver to express their policies in an end-to-end communication. Note that the user may also be a service (instead of a human), as is the case with Web 2.0 mash-up applications and service oriented architecture (SOA).

### OpenADN Extensions

As previously mentioned, OpenADN extends OpenFlow and SDN concepts and combines them with several recent networking paradigms to provide application delivery. These extensions are discussed in this section.

#### *Cross-Layer Communication*

OpenADN provides a session-layer abstraction to applications. It is a cross-layer design where the application layer (layer 7) places the meta-tag (the result of the application-level classification that indicates the application flow class) in the OpenADN header. The header is split across layers 4.5 and 3.5 of the TCP/IP protocol stack. Layer 4.5 implements the OpenADN data plane slow path (dynamic policy-based binding), whereas layer 3.5 implements the data plane fast path (static switching transport). The L4.5 meta-tag is used for setting up various session segments. It has the information required to enforce the session-forwarding policies. The L3.5 header is used by OpenADN aware switches to forward packets in the data plane.

*Application Label Switching*

Layer 3.5 meta-tag processing mechanism uses techniques similar to MPLS label processing, with semantic differences. We, therefore, call this layer application label switching (APLS). APLS uses a mechanism similar to label stacking (label pushing and popping) for enforcing sender and receiver policies on an application-traffic flow. Moreover, APLS uses a mechanism similar to label switching for switching a packet through multiple application-level waypoints. Space constraints do not permit us to include all the details of the label processing.

*Multistage Late Binding*

OpenADN is designed to support dynamic application deployment and access. It uses indirection as the key primitive to support dynamicity. The first indirection mechanism that OpenADN uses is multistage late binding. This mechanism is the basis of the session splicing primitive. Each session consists of many session segments. The endpoint of each segment is determined at the time that the segment is set up based on the application state and the meta-tags in the L4.5 header.

*ID/Locator Split*

This is the second indirection mechanism in OpenADN. All client, waypoints, and servers in OpenADN are assigned with fixed IDs, which are separate from their locators (IP addresses). The indirection layer mapping the ID to a locator adds intrinsic support for mobility to the architecture. It also provides other benefits to construct policy and security frameworks [4].

*SDN Control Application (Control Plane)*

As shown in Figure 7.3, SDN consists of three abstraction layers: virtualization, network operating systems, and network control applications. ASPs can implement OpenADN-based control applications and place them at the top of the SDN stack.

The ASP's control application computes session-forwarding tables implementing the ASP's deployment/delivery/management policies. The ISP's SDN abstraction provides the ASP controller with a virtual view of the APLS data plane as if it was implemented over a single, centralized APLS switch. The ASP controller passes the session-forwarding tables to the ISP's SDN controller that is then responsible for deploying it (preferably distributedly) over the OpenADN-enabled OpenFlow switches in the data plane. Note that, now, ASPs can also invoke the network-level services provided by the ISPs (as proposed by the application-layer traffic optimization [ALTO] [5] framework).

*OpenADN Aware OpenFlow Switches (Data Plane)*

OpenFlow switches classify packets based on L2, L3, and L4 headers. OpenADN aware OpenFlow switches also use the L3.5 and L4.5 headers for packet classification and forwarding. As shown in Figure 7.4, explicitly chained virtual tables specified in the OpenFlow data plane specification 1.1 [3] can be used for this. Incoming packets are first passed through a generic flow-identification table, which then redirects the packet through a virtual table pipeline for a more specific flow processing context. Using this virtual table support, the OpenADN data plane may interpose application-traffic flow processing before handing off the flow for network-level flow processing.

We propose a three-level naming hierarchy for virtual tables. The first level identifies whether it is performing application-level or

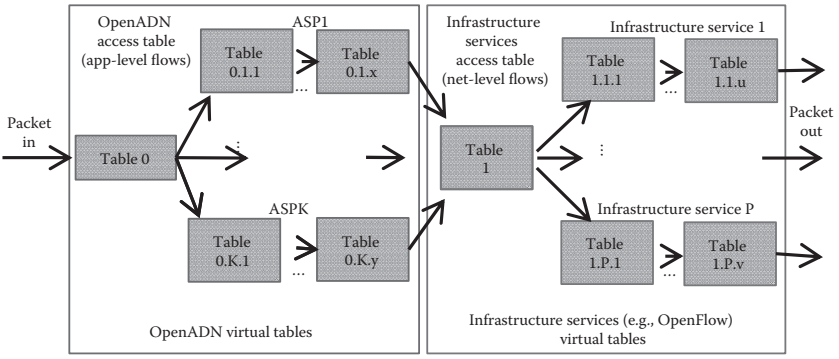


**Figure 7.4** OpenADN and OpenFlow processing.

network-level flow processing. The second level identifies the SDN control module that configures the virtual table (e.g., ASP IDs for OpenADN, infrastructure service IDs for OpenFlow). The third level identifies the specific flow-processing context within an SDN control module.

Different types of data, e.g., Voice over IP (VoIP), video, Web, etc., require different quality of service at Internet switches and routers. The meta-tags in OpenADN headers easily allow this to be accomplished.

*Rule-Based Delegation*

The rule-based delegation mechanism is one of the key innovations of OpenADN. It allows ASPs to create and optimize their specific networking environment over the common infrastructure. Unlike a private WAN (e.g., that of Google), in OpenADN, the ASPs and ISPs are different organizations, and so they do not completely trust each other and do not want to give full control to each other. Rule-based delegation solves this by allowing ASPs to securely communicate with the control plane of the ISP, which then arranges the data plane to satisfy the ASP's requirements, as shown in Figure 7.5. How the ISP distributes these rules to the data plane APLS entities is completely

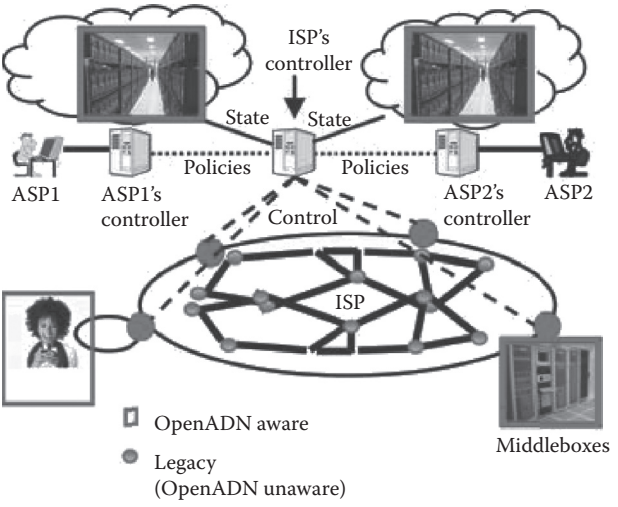


**Figure 7.5** In OpenADN, ASP conveys its policies to ISP in the control plane.

up to the ISP. Moreover, the ISP does not need to look into the application data fields in the packets as required by Content Distribution Networks (CDNs). Furthermore, rule-based delegation creates a network-wide distributed intelligence that dynamically adapts to the dynamic changes of the applications and network conditions.

### Prototype Implementation

We have implemented a proof-of-concept prototype of an OpenADN switch using the click modular router [6]. To validate the functionality, we simulated a simple use-case scenario (Figure 7.6) consisting of three application servers (AppServer1$_A$, AppServer1$_B$, and AppServer2), two waypoints (IDS A and IDS B), a user (simulating multiple traffic sources), an OpenADN controller, and an OpenADN switch.

The use-case scenario is derived from the example of real-time services from smart cell towers. In this simulated scenario, the OpenADN controller belongs to the ISP that programs its OpenADN switch. Moreover, for this prototype, we assume a scenario where all traffic belongs to a single ASP. The application in this example deploys two different types of session segments: (1) SS1 (IDS, AppServer1) and

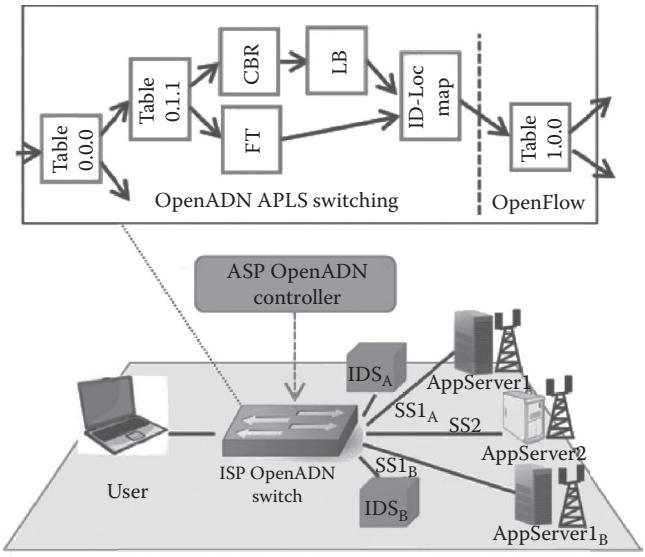


**Figure 7.6** Prototype implementation.

(2) SS2 (AppServer2). SS1 has two streams (<$IDS_A$, $AppServer1_A$> and <$IDS_B$, $AppServer1_B$>).

This use-case scenario shows a centralized OpenADN switch–based implementation where all the entities are connected to the switch. The switch is responsible for indirecting the traffic for SS1 through an IDS to the application server. The IDS also implements an OpenADN stub that is statically configured to return the traffic to the interface over which it received it. When our final implementation of OpenADN (as a general session layer for the networking stack) is completed, it will be possible for the OpenADN controller to configure the OpenADN layer in the IDS to directly forward the traffic to the application server in the session segment. Moreover, in this example, we specifically named the tables FT, CBR, ID-Loc Map, LB, etc., to explain their functions. In fact, no such naming semantics is used, and tables are only numbered using the three-level numbering system previously discussed.

### Related Work

Application-specific packet processing has eluded network researchers for long. However, the full generality of in-network application-specific packet processing proposed by active networks research [7] failed to motivate real deployments. The active networks approach required applications to be allowed to run custom application processing codes on network nodes creating policy and security concerns for the network infrastructure providers.

Delegation-oriented architecture (DOA) [8] was proposed for off-path middlebox deployment to avoid the need to interpose middleboxes directly in the datapath. However, DOA was not designed for dynamic application delivery environments made available through cloud computing today. OpenADN borrows the principles of delegation from DOA and applies it to modern application delivery contexts. More recently, a flexible forwarding plane design has been proposed by the rule-based forwarding (RBF) architecture [9]. RBF proposes that packets must be forwarded to a rule rather than to a destination address. The rule would encode the specific processing required by a packet at a network node. However, rules bind early a packet to a set of processing nodes. Moreover, rules only allow

enforcing receiver-centric policies. In OpenADN, packets carry application context, and it is bound late to a rule in the network. Moreover, OpenADN provides a standardized data plane abstraction for application-traffic flow processing and is thus more suitable for being deployed on high-performance network switches as compared with the (more) general purpose rule processing required by RBF.

Serval [10] is another recent approach for service-centric networking. However, Serval implements a separate control plane mechanism to support service replication across multicloud environments. OpenADN, on the other hand, is a data plane mechanism that allows dynamic service partitioning and service composition, in addition to service replication, for multicloud environments.

CloudNaaS [11] proposed an OpenFlow-based data plane to allow ASPs to deploy off-path middleboxes in cloud data centers. OpenADN is different from CloudNaaS in that it provides a session-layer abstraction to applications preserving session-affinity properties in an environment where the application servers and virtual appliances may need to scale dynamically. OpenADN allows both user and ASP policies, unlike CloudNaaS that only allows ASP policies. Moreover, CloudNaaS does not allow application-level flow processing such as OpenADN and, thus, does not provide support for dynamic service partitioning and replication, routing on user context, and context-based service composition. Some other works, such as CoMB [1] and APLOMB [12], have proposed delegating middlebox services to third-party providers. However, they directly conflict with the design principle of OpenADN that third-party providers should not have access to the ASPs' application-level data for reasons of privacy. Therefore, OpenADN provides a platform, where, instead of third parties, ASPs themselves can manage their middleboxes distributed across different cloud sites more easily.

## Summary

A recent explosion of mobile apps serving a global audience requires smart networking facilities that can help ASPs to replicate, partition, and compose their services on cloud computing facilities around the world on demand to dynamically optimize routing of their traffic.

The key features of OpenADN that may be of interest to the industry are as follows:

1. OpenADN is an open networking platform that allows ISPs to offer routing services for application delivery.
2. OpenADN uses OpenFlow concepts of data and control plane separation, extends headers to move application flow classification to the edge, and uses SDN concepts of a virtual centralized view of a control plane.
3. OpenADN offers ISPs and ASPs benefits of easy and consistent management and cost efficiencies that result from SDN and OpenFlow.
4. OpenADN takes network virtualization to the extremes of making the global Internet look like a virtual single data center.
5. Proxies can be located anywhere on the global Internet. Of course, they should be located in proximity to users and servers for optimal performance.
6. Proxies can be owned and operated by ASPs, ISPs, or CSPs.
7. Backward compatibility: Legacy traffic can pass through OpenADN boxes, and OpenADN traffic can pass through legacy boxes.
8. No changes to the core Internet.
9. Only some edge devices need to be OpenADN/SDN/OpenFlow aware. The rest of the devices and routers can remain legacy.
10. Incremental deployment: It can start with just a few OpenADN aware OpenFlow switches.
11. Economic incentives for first adopters: Those ISPs that deploy few of these switches and those ASPs that use OpenADN will benefit immediately from the technology.
12. Full resource control: ISPs keep complete control over their network resources, whereas ASPs keep complete control over their application data that may be confidential and encrypted.
13. All this can be done now while the SDN technology is still evolving. This will also help in the development of northbound APIs for SDN.
14. CSPs, such as Google, Amazon, Rackspace, etc., can also add these features to their offerings for networks inside their clouds and also for networks connecting their multiple cloud sites.

## Acknowledgment

This work was sponsored in part by a grant from Cisco University Research Program and NSF grant number 1249681.

## References

1. Sekar, V., N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. 2012. Design and implementation of a consolidated middlebox architecture. *Proceedings of the 9th USENIX Conference on System Design and Implementation (NSDI'12)*, pp. 24–37.
2. Gill, P., M. Arlitt, Z. Li, and A. Mahanti. 2008. The flattening Internet topology: Natural evolution, unsightly barnacles, or contrived collapse. *9th International Conference on Passive and Active Network Measurement*, pp. 1–10.
3. OpenFlow Switch Specification 1.3.1. 2012. http://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.1.pdf.
4. Paul, S., R. Jain, J. Pan, and M. Bowman. 2008. A vision of the next-generation Internet: A policy-oriented perspective. *Proceedings of the British Computer Society (BCS) International Conference on Visions of Computer Science*. pp. 1–14.
5. Seedorf, J. and E. Burger. 2009. Application-layer traffic optimization (ALTO) problem statement. *RFC 5963*.
6. Kohler, E., R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. 2000. The Click modular router. *ACM Transactions on Computer Systems* 18(3), pp. 263–297.
7. Tennenhouse, D. L., J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. 1997. A survey of active network research. *IEEE Comm.* (35)1, pp. 80–86.
8. Walfish, M., J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. 2004. Middleboxes no longer considered harmful. *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation (OSDI'04) - Volume 6*.
9. Popa, L., N. Egi, S. Ratnasamy, and I. Stoica. 2010. Building extensible networks with rule-based forwarding (RBF). *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*.
10. Nordstrom, E., D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman. 2012. Serval: An end-host stack for service-centric networking. *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*.
11. Benson, T., A. Akella, A. Shaikh, and S. Sahu. 2011. CloudNaaS: A cloud networking platform for enterprise applications. *2011 Symposium on Cloud Computing (SOCC)*.

12. Sherry, J., S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. 2012. Making middleboxes someone else's problem: Network processing as a cloud service. *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'12)*, pp. 13–24.