OpenADN: Mobile Apps on Global Clouds Using OpenFlow and Software Defined Networking

Subharthi Paul
Washington University in Saint Louis
pauls@cse.wustl.edu

Raj Jain
Washington University in Saint Louis
jain@cse.wustl.edu

ABSTRACT

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 based on the user context (user device type, mobility, link conditions, location, energy constraints, etc.) and the variations in the user access demographics. Cloud computing provides unique new opportunities for application service providers (ASPs) to implement such deployment strategies by making it possible to dynamically allocate geographically distributed computing resources to the application. However, managing such a dynamic and distributed Internet-scale application deployment environment is hard; requiring ASPs to be able to intelligently route application traffic based on high-level application deployment policies over a dynamically changing deployment environment. To this end, we propose the design of an open and standard data plane abstraction called Open Application Delivery Networking (OpenADN) that will allow ASPs to express and enforce application traffic management policies and application delivery constraints at the required level of granularity. The key motivation to designing the OpenADN abstraction is to be able to extract and standardize a set of common application delivery requirements across a wide class of applications deployed over the Internet. OpenADN is designed within the Software Defined Networking (SDN) framework allowing each ASP to implement a separate control plane application to manage the data plane entities over OpenADN to suit the specific requirements of the application. The data plane entities may belong to the ASP itself or may be delegated to third party providers such as Internet Service Providers (ISPs) or Cloud Service Providers (CSPs). We also make a case for augmenting the flow abstraction layer of SDN (OpenFlow) to add adequate support for OpenADN. OpenADN uses IP for forwarding packets to endpoint locators, making it easy to deploy over the current Internet with only a few OpenADN aware entities.

Keywords

Network Architecture, Software Defined Networks, Application Delivery, Application traffic Flows, Application Flow-label, Label Stacking, Label Switching, ID/Locator Split, Cross-layer communication.

1. INTRODUCTION

Cloud computing allows globally distributed services and enterprises, e.g., Facebook, YouTube, and Bank of America, to quickly deploy, manage and optimize their computing infrastructure dynamically. Partitioning or replicating a service across multiple globally distributed instances allow these services to move closer to the users thus providing richer user experiences, avoid infrastructure bottlenecks, and implement fault tolerance. With cloud computing, application service providers (ASPs) can provide better services to mobile end users by dynamically changing their deployment topologies based on user access patterns, user mobility, infrastructure load characteristics, infrastructure failures and many such situations that may cause service degradation, disruption or churn.

Another key trend is the explosion of mobile applications (apps). Every business, every newspaper, every bank, and every game has its own app. The world has become flat and most of these apps service a world-wide audience and therefore may use cloud computing services to replicate their services and optimize user experience.

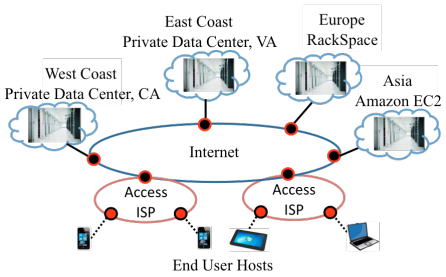

Fig. 1 Distributed service over private and public clouds

Fig.1 presents an example of a typical deployment scenario of an application service provider (ASP). This ASP owns and operates multiple data centers across the US. However, in Europe or Asia it may need to use third party cloud computing facilities provided by RackSpace or Amazon EC2. Also, as some particular application becomes popular among users it would need to instantiate more globally distributed resources to this application and release these resources as the popularity dies out.

Similarly, replicated service instances might need to be moved/instantiated/released to mask infrastructure failures, load conditions, or optimize the deployment based on access patterns and social interaction graphs. Such dynamic service deployment scenarios need intelligent infrastructure support.

Google also has this problem and has installed a WAN-like infrastructure [1, 2] that intercepts most of the traffic for Google-owned services at edge-network POPs and sends them over its private WAN infrastructure. At these POPs, Google (probably) operates application layer (layer 4-7) proxies to intelligently route a service request to one of its geographically distributed data centers. While it is possible for large ASPs to operate such infrastructures, it is prohibitively expensive for smaller ASPs.

Middleboxes are an essential component of modern application delivery [3]. Within private datacenters or within enterprise network environments, ASPs generally operate a middle layer of variety of network appliances to implement load balancing, fault tolerance and other intelligent infrastructure support.

This work was supported in part by an award from Cisco University Research Program and NSF CISE grant #1019119.

First International workshop on Management and Security technologies for Cloud Computing (ManSec-CC) 2012, Decenber 7, 2012, in conjunction with IEEE Global Communications Conference (Globecom) 2012, Anaheim, CA, December 3-7, 2012

However, such solutions are not available for wide area dynamic multi-cloud environments that can be shared by multiple ASPs to implement their specific distributed service delivery contexts.

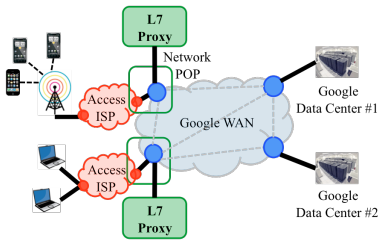

Fig. 2 Google's Application Delivery Network

Our vision is to design an open and standard data plane abstraction called Open Application Delivery Networking (OpenADN) that will allow ASPs to express and enforce application traffic management policies and application delivery constraints at the required level of granularity. As shown in Fig. 3, using OpenADN aware data plane entities, ISPs can offer services similar to Google WAN to smaller ASPs. Any new ASP can quickly setup its service by using ADN services provided by OpenADN carriers. The ASP's control plane may program the ISP provided OpenADN data plane entities such as OpenADN-aware switches and middelboxes to manage its application deployment distributed across several cloud-infrastructure sites.

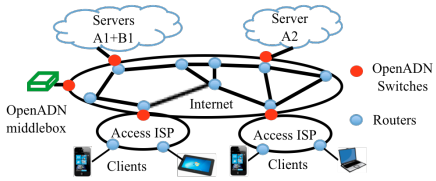

Fig. 3 Open Application Delivery Network (OpenADN)

To achieve this we combine the following six innovations:

1. OpenFlow
2. Software Defined Networking
3. ID/Locator Split
4. Layer 4-7 Middleboxes
5. Cross-Layer Communication
6. MPLS-like application flow labels

The organization of the rest of the paper is as follows. Section 2 briefly explains the features of OpenFlow and SDN that are helpful in our goal. Section 3 discusses the extensions of the six concepts that make OpenADN possible. Related works are presented in Section 4 followed by a summary in Section 5.

2. SDN and OpenFlow

Software Defined Networks (SDN) is an approach towards taming the configuration and management complexities of large-scale network infrastructures through the design of proper abstractions. This complexity is inherent to the design of distributed control algorithms and ensuring the consistency of distributed state.

On the computing side, cloud computing brings new opportunities for ASPs to considerably reduce their capital and operational expenditures in terms of provisioning computing infrastructures for peak loads. Also, ASPs can leverage geographically distributed computing infrastructure provided by multiple cloud providers (multi-cloud environments) for dynamically distributing their services through replication and partitioning. Managing such a distributed and dynamic deployment environment can be extremely challenging. More so, without support from the underlying network infrastructure. ISPs implementing an SDN domain can provide support to such environments as shown later in this paper.

Most critical to the design of the proper abstractions for a system is the design of the *primitive abstraction layer*. The primitive abstraction layer is the layer that interposes between the aspects of flexibility and expressability requirements of the higher layer policy expressions and the performance requirements of the lower layer system components. The key design issue of this layer is to be able to arrive at the right granularity that properly addresses the flexibility-performance tradeoff. The generic concept of a *packet flow* seems to be the right granularity for the design of this layer for SDN. We refer to it as the *flow abstraction layer.*

The flow abstraction layer will be responsible for mapping *flow-level* policies specified by control applications to *packet-level* enforcement of these policies by the data plane components. The SDN may build additional layers of abstraction between the flow abstraction layer and control applications to address two general goals. The first goal is to provide a logically centralized platform for implementing control plane applications, abstracting away the details of state distribution and distributed state management from these applications. The second goal is to support the deployment of multiple control applications over the SDN platform by providing resource and policy isolation.

In order to specify *flow-level* policies, a control application needs to provide two types of rules: 1) *flow classification rules* to identify packets as belonging to a flow of interest to the control application, and 2*) flow-level enforcement rules* to specify the set of actions over each packet belonging to the flow.

The current effort to standardize the *flow abstraction layer* is based on the evolving OpenFlow [4] standard. OpenFlow allows flows to be specified over a combination of layer 2 (including layer 2.5), layer 3 and layer 4 header fields. Thus, OpenFlow provides enough context for designing control applications for the network-level flow processing functions including destination-based routing and forwarding and various traffic engineering applications optimizing different parameters such as energy efficiency, congestion, latency, etc.

3. OpenADN Vision

As mentioned earlier, 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.

A. Application Level Policies

Application level policies are features that ASPs need to manage their distributed and dynamic application deployment environemnts. For example, ASPs may want network to help in the following:

1. Replication and Partitioning: Route the traffic to the right application instance from a partitioned application space where each partition may be replicated across a group of geographically distributed instances.
2. Load Balancing: Load balance among a group of geographically replicated application instances.
3. Fault Tolerance: Divert traffic to a live application instance within the failover group of the failed application instance.
4. User Context: Route traffic to an application instance based on the user context.
5. Service Mobility: Allow application instances to move among cloud data centers through virtual machine migration.
6. Client Mobility: Allow clients to move.
7. Service Composition: Compose a service from multiple individual service components.
The ASPs may provide policies that when implemented will provide optimal user experience. These are all examples of application level policies. Application level policies must be contrasted with network level policies that are the features that ISPs need and enforce. These include routing, traffic engineering, congestion control, etc. These policies are applied to all packets that belong to a network flow class. For example, all packets that have the same destination may belong to one network flow class and may be routed to the same port or on the same path. Alternately, a network flow class may consist of all packets that have the same MPLS tag and therefore will be routed on a particular label switched path (LSP).

B. Application Flow Class

Application level policies described above require classifying packets into application flow classes. For example, an ASP may want all its voice and video messages to be sent to Server group 1 (any one of the servers in the group), while all accounting messages to be sent to Server 2 (not replicated). Such policies are currently enforced in private data centers using middleboxes (network appliances). Several vendors specialize in such appliances that provide load balancing, intrusion detection, firewalls, etc. Most of these middleboxes operate at the application layer and need to reassemble application messages from network packets.

Since OpenFlow works at the packet level, it 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. We solve this problem by using a cross-layer communication technique described next.

C. Cross-Layer Communication

OpenADN uses a cross-layer design that allows application-traffic flow information to be placed in the form of a label between the network layer (layer 3) and transport layer (layer 4) packet headers (See Fig. 4). In other words, a "APplication Label Switching (APLS)" layer forms layer 3.5 in the protocol stack. Legacy routers forward packets based on layer 3 (destination prefix match) or layer 2.5 (MPLS labels). Layer 3.5 is handled only by OpenADN aware devices, such as clients, servers, OpenADN switches and middleboxes. The protocol type field in the layer 3 header indicates the presence of APLS header. Another protocol type field in APLS header indicates the layer 4 protocol, e.g., TCP, UDP, SCTP, etc. Thus, OpenADN works with all L4 protocols and both IP and MPLS routing.

The APLS header augments the *flow abstraction layer* allowing us to design control applications for application-traffic flow processing in OpenADN aware OpenFlow switches. This way, OpenADN provides a *constrained* and *standardized* interface for delegating application-traffic flow processing to the ISP.

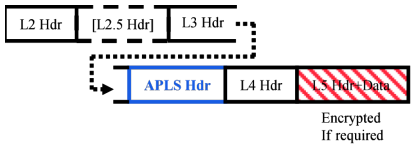

Fig. 4 APLS header provides sufficient information for enhanced OpenFlow switches to enforce application level policies.

In particular, APLS header allows OpenADN aware OpenFlow switches to offer the following services:

1. Message Affinity: All packets that are part of an application-layer message need to be classified into the same application flow class. The semantics of a message is application specific.

2. Session Affinity: All packets of a particular application session are bound to the same session endpoint during the session. The definition of a session is application specific.

3. Receiver Policies: Load balancing among multiple servers is an example of a receiver policy. APLS header enables OpenFlow switches to implement such policies. In OpenADN, we don't distinguish servers and clients. Either one can be a receiver or a sender and, therefore, may have its policies. In this sense, the application of OpenADN is not limited to mobile hosts, even the traffic between two datacenters can use these features.

4. Sender Policies: Application-level flows are subject to both, *sender* and *receiver* policies. Thus, a single end-to-end application-level flow may be processed by two separate control applications. These control applications may reside over the same or different SDN domains. The OpenADN design provides mechanisms to ensure that only one control application processes the flow at any given time. After the *sender* policies have been enforced on the flow, it needs to be handed-off to the *receiver* control application for *receiver* specific policy enforcement.

5. Network Policies: APLS header enables the packets to receive the QoS (e.g., drop policy, priority) for application flows as specified by the ASP from the network service provider (e.g. ISP).

6. Middle-box services: The packets will be forwarded through a chain of middle boxes as specified by the ASP. APLS layer results in a forwarding plane abstraction that allows the control application to steer an application traffic flow through multiple intermediaries (called *waypoints*) between two end-points. A middle-box is an example of a waypoint. Also, it allows the control application to dynamically steer flows to accommodate application-level churn as a result of server/user mobility or server failures.

In summary, OpenADN-aware OpenFlow (Fig.6) switches enable application traffic handling at the packet layer.

D. ID/Locator Split

There are two types of application-level entities in OpenADN, *endpoint entities* and *waypoint* entities. Each entity is assigned *a fixed identifier (*ID), which is separate from its *locator*. The ID/locator split [8] is necessary to uniquely identify and address an application level entity. This is required for 1) enforcing sender/receiver policies on a flow, 2) specifying session affinity of a flow over a set of intermediaries and end-points, and 3) correctly routing a flow to mobile servers/users.

E. SDN Control Application

OpenADN leverages the *flow-level abstraction* provided by an SDN-like platform for application-traffic flow processing. SDN allows ASPs to write their own control applications. Leveraging the power of abstraction, network infrastructure domains implementing SDN can now easily accommodate such third party provided control applications without relinquishing control of their infrastructure.

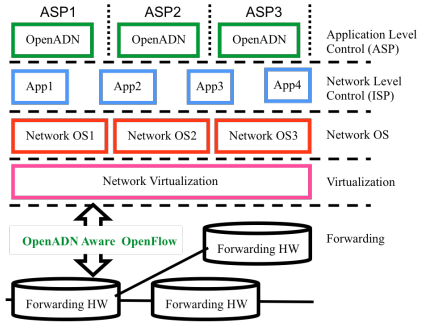

Fig. 5 OpenADN control applications and SDN

As shown in Fig. 5, SDN consists of 3 abstraction layers consisting of virtualization, network operating systems, and network control applications. ASPs can implement OpenADN based control applications to specify application-level flow identification and policy enforcement rules. *Note, now ASPs can also invoke network level services provided by the ISPs (as proposed by the Application-Layer Traffic Optimization (ALTO) [7] framework).*

F. OpenADN Aware OpenFlow Switches

As indicated in Fig. 6 (OpenADN aware OpenFlow switches), in the augmented *flow abstraction layer* with support for application-traffic flow processing, OpenADN specific processing precedes OpenFlow specific processing. This follows naturally from the layered abstraction in the control plane. Application-traffic flows processed by the OpenADN control applications need to be mapped to network flows processed by the SDN control modules for accessing network level services. Hence, it is required that the OpenADN flow abstraction can directly interface with OpenFlow in the data plane.

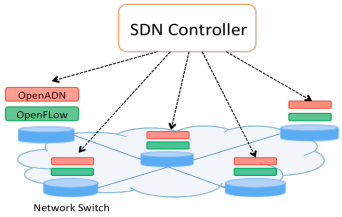

Fig. 6 OpenADN and OpenFlow

As shown in Fig. 7, explicitly chained virtual tables specified in the OpenFlow data plane specification 1.1 [12] 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 more flow-context specific processing. 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 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. In Fig. 7, we only show a packet being explicitly handed-off at level 1 in the hierarchy, from application-traffic flow processing to network level flow processing. However, it is also possible to allow packet handoffs at level 2 and level 3 in the hierarchy, to accommodate layered abstraction in the control plane.

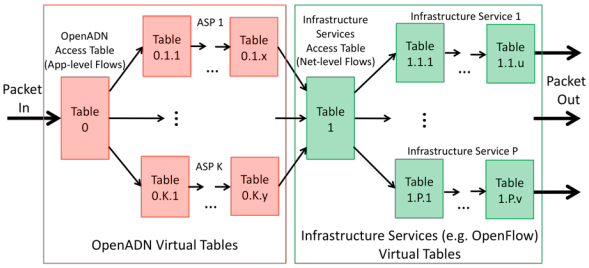

Fig. 7 Data Plane *design providing access to both* OpenADN and Infrastructure Services

G. OpenADN Label Mechanisms

The requirements of application-traffic flow processing are supported through two key mechanisms.

The APLS header carries application-traffic flow context in a *meta tag*. It is placed into the flow label by the communicating end-points. The *meta tag* may be interpreted as the result of an application-traffic flow classification. Unlike network level flow classification, application level flow classification needs to be done at the end-hosts. There are several reasons for this:

1. Semantic Gap: Application-traffic flow classification rules may be specified over application content, application headers, application session information, etc. Most of this information is not available to a network node at the packet level granularity. Moreover, even the whole application level header may not be part of an IP datagram as a result of TCP segmentation. Thus, the semantic gap between the scope of the classification rules and the context available to network nodes to interpret those rules makes it more viable to do the classification at end hosts.

2. Diversity: The diversity of application-level protocols makes it difficult to standardize a node that could classify all application level headers.

3. Performance: Application layer classification is a compute intensive task mostly because of their lack of standardization and the machine unfriendly encoding.

4. *Security:* Allowing an intermediate network node to do application traffic flow classification would require access to application level data, thus interfering with end-to-end security.

APLS label processing mechanisms uses techniques similar to MPLS label processing, with semantic differences. APLS uses a mechanism similar to *label stacking* (label pushing and popping) for enforcing *sender* and *receiver* policies on an application traffic flow. Also, 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 details of the label processing.

4. 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 [9, 11] has failed to motivate real deployments. The active networks approach required applications to be allowed to run custom application processing code on network nodes creating policy and security concerns for the network infrastructure providers.

Application delivery intelligence in modern application deployments is implemented through specific purpose middleboxes interposed in the application delivery path. However, due to the lack of support for middleboxes in the original Internet architecture, it poses considerable challenge for network administrators to configure policy-routing of application-traffic flows through a specific set of middleboxes. To alleviate this difficulty, delegation–oriented architecture (**DOA**) [10] was proposed. 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 architecture (**RBF**) [6]. RBF proposes that packets be forwarded to a "rule" instead of a destination address. The rule would encode the specific processing required by a packet at a network node. However, rules early bind a packet to a set of processing nodes. Also, rules only allow enforcing receiver-centric policies. In OpenADN, packets carry application context and it is late bound 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 to the (more) general purpose rule processing required by RBF.

Serval [5] is another recent approach for service centric networking. Serval treats all packets of a service identically and therefore cannot distinguish differing requirements for various application level messages from the same service. In addition to message level granularity, OpenADN allows ASPs to specify a sequence of middleboxes and end entities that specific messages will travel. In addition, OpenADN allows both senders and receivers policies. OpenADN is also more general in the sense that both packet level and message level middleboxes are allowed.

5. CONCLUSION

Recent explosion of mobile apps serving a global audience requires smart networking facilities that can help ASPs to replicate their servers on cloud computing facilities around the world on demand to dynamically optimize for user access patterns. OpenADN is an open networking platform that allows ISPs to offer such services. It uses the flow abstraction layer of SDN and adds ID/locator split and cross-layer communication in the form of an application label-switching (APLS) header in layer 3.5 that allows OpenFlow switches to be enhanced to offer application level services without the need to reassemble application level messages.

A key feature of OpenADN is that it can be incrementally deployed with just a few OpenADN aware OpenFlow switches and is fully compatible with current Internet. Those ISPs that deploy these switches and those ASPs that connect to these switches will be able to benefit immediately from the technology. Also, ISPs keep complete control over their network resources while ASPs keep complete control over their application data that may be confidential and encrypted. ISPs can also deploy OpenADN aware middle-boxes and offer middle-box services to ASPs. Best of all, this can be done now while the SDN technology is still evolving. Cloud service providers like Amazon, Google, Microsoft, Rackspace, etc. can also add these features to their offerings. Some components of OpenADN, such as, APLS header can be used for virtualizing application classes and offering application specific services in other contexts. In this paper we have presented a work-in-progress space-constrained report of a large project.

6. REFERENCES

[1] P. Gill, et al., "Youtube traffic characterization: a view from the edge," 7th ACM SIGCOMM conference on Internet measurement, pp. 15-28, 2007.

[2] P. Gill, et al., "The flattening internet topology: natural evolution, unsightly barnacles or contrived collapse," 9th international conference on Passive and Active Network Measurement, pp. 1-10, 2008.

[3] D.A. Joseph, A. Tavakoli, I. Stoica, "A Policy-aware Switching Layer for Data Centers," SIGCOMM, 2008.

[4] N. McKeown, et.al., "OpenFlow: Enabling Innovation in Campus Networks," Whitepaper, OpenFlow Switch Consortium, May 2008.

[5] E. Nordström, et.al., "Serval: An End-Host Stack for Service-Centric Networking," NSDI, April, 2012.

[6] L. Popa, N. Egi, S. Ratnasamy, I. Stoica, "Bulding Extensible Networks with Rule-based Forwarding (RBF)," USENIX OSDI 2010.

[7] J. Seedorf, E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement," RFC 5963, 2009.

[8] I. Stoica, et.al., "Internet Indirection Infrastructure," ACM SIGCOMM, August, 2002.

[9] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, G. J. Minden, "A Survey of Active Network Research," IEEE Comm., 1997.

[10] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, S.Shenker, "Middleboxes no longer considered harmful," OSDI, 2004.

[11] D. Wetherall, "Active network vision and reality: Lessons from a Capsule based system," ACM SOSP, 1999.

[12] OpenFlow Switch Specification, Version 1.1.0 Implemented, February 28, 2011, http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf