OpenADN: A Case for Open Application Delivery Networking

Subharthi Paul[a] pauls@cse.wustl.edu
Raj Jain[a] jain@cse.wustl.edu
Jianli Pan[a] jp10@cse.wustl.edu
Jay Iyer[b] jiyer@cisco.com
Dave Oran[b] oran@cisco.com

***ABSTRACT* -There are two key issues that prevent Application Service Providers (ASPs) from fully leveraging the cloud "advantage." First, in modern enterprise and Internet-based application environments, a separate middlebox infrastructure for providing *application delivery* services such as security (e.g., firewalls, intrusion detection), performance (e.g., SSL off loaders), and scaling (e.g., load balancers) is deployed. In a cloud datacenter, the ASP does not have any control over the network infrastructure, thus making it hard for them to deploy middleboxes for their cloud-based application deployments.**

Second, modern services virtualize the application endpoint. A service can no longer be statically mapped to a single end host. Instead, the service is *partitioned* and *replicated* across multiple end hosts for better performance and scaling. In enterprise datacenters, service requests are intercepted by an application-level routing service (APR) in the data plane and dynamically mapped to the correct service partition and the best (e.g. least loaded) instance of that partition. However, although multi-cloud (or Inter-cloud) environments allow ASPs to globally distributed their applications over multiple cloud datacenters leased from multiple cloud providers, ASPs need support of a globally distributed APR infrastructure to intelligently route application traffic to the right service instance. But, since such an infrastructure would be extremely hard to own and mange, it is best to design a shared solution where APR could be provided as a service by a third party provider having a globally distributed presence, such as an ISP.

Although these requirements seem separate, they can be converged into a single abstraction for supporting application delivery in the cloud context. A sample design of this abstraction is OpenADN, presented here. [1]

Keywords - Application Delivery, Cloud Computing, Software Defined Networks, Middlebox Deployments, Internet Architecture.

1. INTRODUCTION

As the Internet gained popularity, a separate infrastructure for *application delivery* evolved out of the practical necessity to separate *application logic* from the security, performance and scaling issues of *application delivery*. While the application logic is implemented over **endpoint** *application servers*, application delivery is implemented over **intermediary** *middleboxes* – together comprising a modern *application deployment environment*.

[1] Supported by NSF Grant 1249681 and Cisco URP.

[a] Washington University in Saint Louis, Saint Louis, MO 63130

[b] Cisco Systems Inc.

Middleboxes serve as the front end to application servers and implement several key application delivery functions including security (firewalls, intrusion detection), offloading specific compute intensive application-level functions to specialized hardware (SSL offloaders, transcoders), and application delivery optimization (WAN optimizers, content caches). Another set of middleboxes provide application-level policy routing (APR) services that allow the application to be *partitioned* and *replicated* across multiple end-hosts for better performance and scaling. The application may be partitioned based on content (video vs. data vs. accounting, distributed databases, etc.) user context (mobile vs. desktop, wireless vs. wired access, location, etc), and application contexts (database read vs. writes, different functions of a SOA service deployment, etc.). An APR service (implemented as a content/context based router with load balancing), intercepts application requests, and routes them to the right service partition and the best instance of that partition (such as least loaded server) suitable to serve the request.

Table 1. Middlebox deployment in a large enterprise environment [13]

Appliance Type	Number
Firewalls	166
NIDS	127
Conferencing/Media Gateways	110
Load Balancers	67
Proxy Caches	66
VPN devices	45
WAN optimizers	44
Voice Gateways	11
Middleboxes total	636
Routers	~ 900

The number of middleboxes deployed in modern enterprise application environments is comparable to the number of routers, as shown in Table 1. Also, market reports project that the market size of application delivery controllers or network appliances (generally used terms in the industry for middleboxes) will grow from 1.5 billion dollars in 2009 to 2.25 billion by 2013 [17]. This projection is only for acceleration and optimization middleboxes and does not include security appliances, which is projected to grow to a 10 billion dollar market itself by 2016 from 6 billion in 2010 [13]. These numbers clearly suggest that the *application delivery infrastructure* is an essential component of modern Internet-based application environments. However, in the context of future cloud-based application deployments; most discussions (and research) on how to migrate enterprise applications to the cloud pertain to moving only the *application servers*. But, as we have already pointed out, in modern application deployments, *application servers* cannot function without *application delivery* support. It is therefore necessary to re-think *application delivery* in the context of cloud computing.

1.1 Motivation: Architectural Requirements

The motivation to design OpenADN is derived from re-thinking application delivery in the context of cloud-based application deployment environments. Here, we discuss some example deployment scenarios that motivate the high-level architectural requirements.

1.1.1 Single-Cloud Datacenter Environments

One of the barriers to migrating enterprise applications, from enterprise datacenters to cloud environments, is the lack of application delivery support in the clouds. In this context, we consider two types of cloud-based deployments:

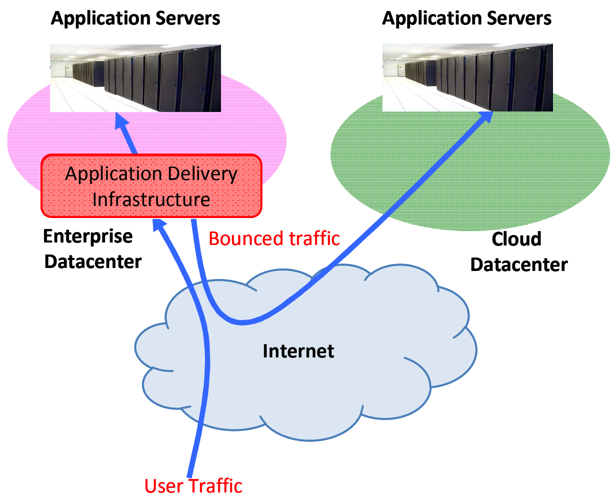

Fig. 1 Failover Deployment

1. Failover Deployment: This represents the most common use of cloud computing at present. In this environment, the cloud acts like a failover infrastructure to the enterprise data center. Under conditions such as extreme or unexpected usage spikes, failures and planned maintenance, the enterprise datacenter may migrate or replicate some of its application servers in the cloud. Since this is a temporary arrangement for short duration, during this time application traffic may be *indirected* (or bounced) through the middlebox infrastructure in the enterprise datacenter (Fig 1). The scope of this solution is limited to intermittent (and rare) failure and scaling events that can be solved by simply throwing in more hardware to the problem. Also, the solution assumes that even under such extreme circumstances (except for planned maintenance), all the middleboxes as well as the network infrastructure of the enterprise datacenter is still available.

2. Independent Cloud Deployment: In independent cloud deployments (Fig 2) where the cloud datacenter directly serves application requests, the *application delivery* infrastructure is provided by virtual appliances. Monolithic application delivery controllers, which are common in private data centers, cannot be used in a cloud environment. Virtual appliances are the software (virtualized) counterparts of hardware-based middleboxes that may be deployed over general-purpose hardware. Naturally, the performance of these virtual appliances is severely degraded owing to the lack of special hardware support. Also, they exhibit variable performance, depending on the specifications of the physical machine over which they are instantiated.

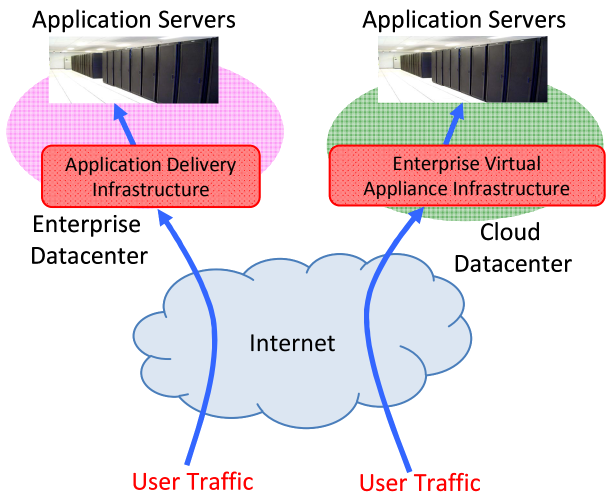

Fig. 2 Independent Cloud Deployment

Deploying middleboxes, even within enterprise datacenters, is hard. It has been a longstanding problem addressed before [7, 16, 19]. The root of this problem is that there is no explicit architectural support in the Internet Protocol Stack that allows middleboxes to be explicitly inserted into the applications data path. Hence, network administrators use non-standard techniques to deploy middleboxes, either by physically interposing them in front of the application server or through ad-hoc configuration tricks such as externally changing the link weights in routing protocols, to steer traffic through a middlebox (or a sequence of middleboxes). These non-standard techniques are hard to manage and are prone to errors. Also, the correctness of such deployments cannot be assured under routing dynamics, such as IP path changes. However, in cloud environments, even these ad-hoc techniques are not available since the application service provider (ASP) has no control over the underlying network infrastructure in a cloud datacenter. The only solution is to add **explicit**

support for middlebox-switching, either in the application level (layer 4 -7) and thus implemented by the virtual appliances themselves, or in the network level, as a configurable switching service to the ASP provided by the cloud service provider (CSP).

1.1.2 Multi-Cloud Deployment Environments

Eventually, enterprise and cloud datacenters are expected to converge on standardized access APIs and collaborative business models to provide a platform for global application deployment. This convergence to a ubiquitous computing infrastructure, termed as multi-cloud, is similar in spirit to the convergence of the networking infrastructures (across different networking technologies and ownerships) that resulted in the Internet.

In such multi-cloud environments, it will be possible for applications to dynamically deploy themselves across geographically distributed sites. ASPs can leverage this opportunity to optimize their deployment based on several parameters including load, cost, failures, access patterns, and network events.

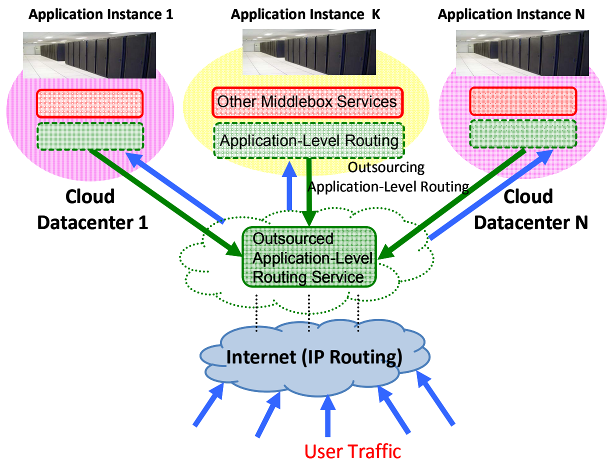

Fig. 3 Multi-Cloud Deployment

As applications move to such globally distributed deployment environments, a globally distributed front-end service to steer application traffic to the right application instance based on application specified policies is required. This front-end service is generically called ***application-level routing*** (Fig. 3) and may be implemented by composing several different *application delivery* functions including content-based partitioning, context-based partitioning, load balancing, and fault-aware adaptive routing. An example of application-level routing is Facebook's *page-routing* [14].

For enterprises operating multiple geographically distributed data-centers, the effort to design applications for such globally distributed deployment has already started [3, 9]. Large ASPs like Google have solved this problem by installing application-level (Layer 4-7) proxies at all major network point-of-presence (POP) locations [4, 5]. Access to Google services is intercepted at the nearest POP and intelligently routed to the correct Google datacenter. However, although large ASPs can afford to deploy and maintain such global networking infrastructures, it is prohibitively expensive for smaller ASPs.

Our vision is to design an architecture so that smaller ASPs can get this as a service from ISPs, who would route application messages through an appropriate set of application delivery controllers, proxies, and middle boxes located in various clouds throughout the global Internet as shown in Fig. 4.

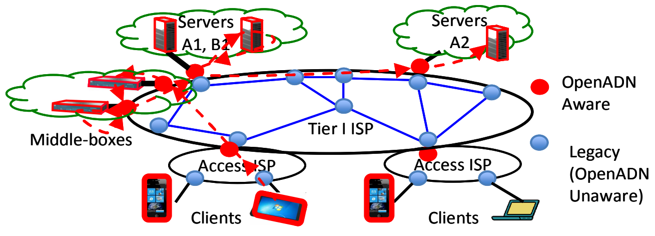

Fig. 4: OpenADN Vision

The dotted line in Fig. 4 shows the path taken by a particular set of application messages through middle boxes and servers located in various cloud data centers. Other messages for the same ASP may take another route through various middle boxes. Note that only a few edge devices are OpenADN aware and work with the legacy Internet devices that are not OpenADN aware.

2. DESIGN ISSUES

The architectural requirements presented in § 1.1 need to be qualified further to arrive at the specific **design issues**.

2.1 Explicit Support for Application Delivery

The following issues need to be addressed:

1. Who will implement?: Middle boxes are called so, since they *sort of* exhibit an *application layer (layer 4-7)-network layer duality* – depending on the perspective. From the network layer's perspective, middleboxes are application layer entities that process application data and even (sometimes) terminate end-to-end connections. Whereas, from the application layer's perspective, middleboxes are *intermediaries* providing some service to the application endpoints just like network layer entities.

Given this *sort of dual* behavior, middlebox-switching lies at the critical junction between the application layer and the network layer, i.e., between the ASP and the ISP.

This junction is critical for several reasons. In terms of *functionality*, any function designed at this junction is specific to the application and therefore very diverse while it has to have *performance* similar to the network layer. In terms of *semantic-gap,* the network layer has no information about application messages and so cannot route them differently.

In terms of *administration*, this junction separates the ASP from the infrastructure provider (ISP or CSP). The ASP wants to control the message routing policies but ISP does not want to relinquish the control of its router. Hence, there are two options to implement application policy routing (APR). The first is to implement it as an extension of the application layer (and hence by the ASP), similar to HTTP's extensions to explicitly support forward and reverse proxies [4]. The other option is to implement it as an infrastructure (ISP or CSP) service through a generic and standardized abstraction. Such an abstraction will need to expose certain degree of programmability such that an ASP may be able to configure the service based on its requirements and the state of its middlebox deployments.

2. Application Segments: Middle boxes are often deployed in a chain to compose an application delivery service. For example, an encrypted user message might first need to be sent to a *SSH offloader* to decrypt it, followed by an *IDS* and *web application firewall* for security related checks and finally routed through a *content-based router* and *load balancer* to be sent to an *application server*. The connection between the user and the server is no longer end-to-end. It actually consists of several TCP connections in a sequence. Each of these connections spans, what we call, a **segment**.

3. Generic representation of application endpoints: Each segment ends in a middle box, server, or proxy that may be replicated for performance. We use the generic term "**waypoint**" to denote middleboxes, application delivery controllers, proxies, or servers. Thus, each segment spans between two groups of waypoints. Each segment may have many possible instances, which we call **stream**. Each stream connects the source waypoint to one member of the destination waypoint group.

4. Granularity of the switching context: As discussed earlier, applications are partitioned such that different messages follow a different path based on content, user context, application context or network context. We, therefore, need to put meta-tags in the packet headers that help classify packets in to application level flows and then waypoints can route packets based on these tags. Each way point only receives traffic that it needs to operate on and does not receive other traffic that it does not operate on.

The right granularity of specifying the switching context is especially important for virtual appliance deployments. This is because virtual appliances are already performance constrained (as compared to their hardware counterparts) and are therefore more sensitive to *deployment efficiency* [8], meaning that the virtual appliance should not receive any unnecessary application traffic that it is not meant (or required) to process.

5. Support for sender and receiver policies: Both sender and receiver may have policies about routing messages that they send or receive. For example, the user may itself be a service that uses other services as is the case in service composition. In a communication context involving two application end-points, the sender policies –switching through the sender's application delivery infrastructure; needs to be followed by the receiver policies –switching through the receiver's application delivery infrastructure.

2.2 Outsourcing Application-level Routing

Some of the issues in outsourcing the application delivery service to ISPs include:

1. Application data privacy: Application-level routing requires access to the application-level data, application headers, application session information, etc., especially for content-based routing. However, although ASPs would benefit from outsourcing this application-level function to third party ISPs, most ASPs cannot allow ISPs access to application data owing to the application's privacy requirements. It is for the same reason that we believe that Akamai' edge-side include platform [1] for serving dynamic content by allowing ASPs to outsource *page rendering* to Akamai, is not a successful model. Most modern applications including banking, healthcare, shopping and social networking are emerging as personalized services where privacy of the data is of paramount importance. Hence, a mechanism for outsourcing application level routing without access to the application-level data needs to be designed.

2. Access to the dynamic application deployment state: To make intelligent application-level routing decisions, the ISP's service needs access to the application's dynamic deployment state. For example, ISP may need to know where and how many waypoints are up at a particular time. This dynamic deployment state may be the result of the application's policies to optimize delivery, such as partitioning or replicating the application space, spawning new instances, shutting-down existing instances, moving instances to new locations, etc. Also, they may be due to churn in the deployment environment caused due to instance load, instance failure, network congestion, etc.

3. DESIGN APPROACH

In this section, we derive the design of OpenADN based on the requirements discussed in § 2.1 and § 2.2.

1. Convergence of the architectural requirements: In our discussion so far, we have presented the need for *explicit support for middlebox-switching* and *outsourcing application-level routing;* as two separate issues related to application delivery in the multi-cloud context. OpenADN handles both these issues by implementing a generic *application delivery networking layer*. As shown in Fig 5, middlebox-switching is implemented at network POPs to indirect traffic through the ISPs application-level routing service. The same can be done in a cloud datacenter, where a common middlebox-switching layer switches between the ASPs virtual appliances.

As already noted (§ 2.1, Point 1), middleboxes exhibit a *sort of dual* behavior and hence *middlebox-switching* can be implemented either as an extension to the application layer, or as a programmable service provided by the network layer. Owing to the constraint of application data privacy (§ 2.2, Point 2), *outsourcing* the application-level routing service needs the implementation to separate the *classification step* from the *routing decision step.* In OpenADN, the two steps are connected through an *application meta-tag* (or simply referred to as *meta-tag*) that encodes the result of the application-level classification. The *classification step* needs to be implemented on an ASP trusted entity that has full access to the application data and the communication context, such as the end-hosts. The *routing decision* is taken at the outsourced middlebox entity that can be granted *access to the dynamic application deployment state* (§2.2, Point 3). The routing decision is a function of the *meta-tag*, ASP policies, and the dynamic application deployment state.

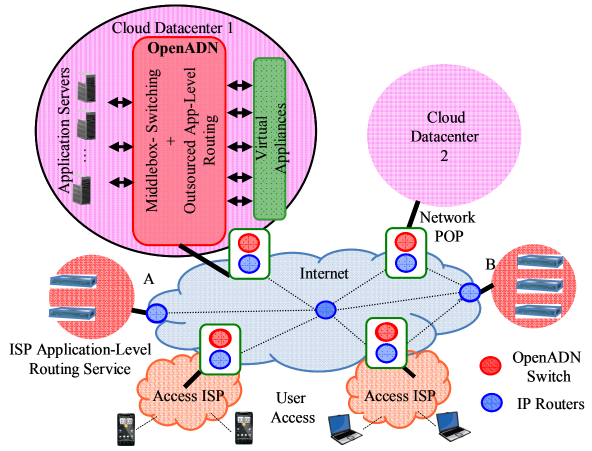

Fig. 5 OpenADN: Overall View

Apart from this, the *meta-tag* has performance benefits as well. It allows for high-speed implementation by removing the requirement of general purpose processing needed to do application-level classification. Also, the *meta-tag* may be made available at packet level granularities, allowing better resource utilization and not requiring to assemble the whole message before being able to make a decision.

The OpenADN data plane spans across two layers as shown in Fig. 6, (i) Layer 3.5 called the APplication Label Switching (APLS) layer, and (ii) Layer 4.5 called the Segment Switching Layer (SSL). Layer 4.5 implements an *application segment* (§4.1, Point 2). Layer 3.5 provides a hop-by-hop transport over OpenADN waypoints within a layer 4.5 application segment. Layer 3.5 is implemented by all OpenADN aware waypoints. The waypoints are deployed as an overlay over IP for backward compatibility. Entities are assigned unique IDs in the OpenADN layer, thus using IP only for routing on network locators.

2. Accessing the ISP's application-level routing service: How do the packets get from the user to the first OpenADN aware ISP waypoint? The answer is that the ISP may reserve a special anycast IP prefix for the application-level routing service and each ASP using the service is allocated an address relative to this prefix. When users try to access the ASPs service at this anycast address, it is indirected through the ISPs service.

3. Label Switching Mechanism: Fig 6 shows a proposed OpenADN label. The OpenADN label unlike a MPLS label is a composite label consisting of several sub-labels. Each sub-label may be individually switched by different entities to support different requirements. We will discuss each sub-label in the context of the design requirements.

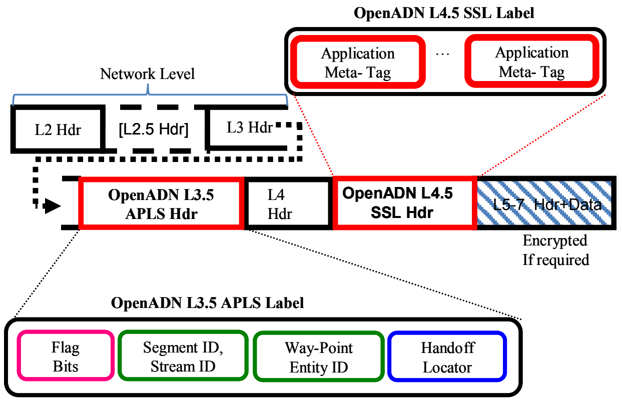

Fig. 6 OpenADN Application Label

(i) Layer 4.5 SSL Label: The layer 4.5 APLS label carries a stack of *application meta-tags* representing a sequence of application segments. The *meta-tags* is inserted by the application end-points, at the egress of an application segment, an APR pops the next *meta-tag* and maps it to a <segment ID, stream ID> *two-tuple* of the next application segment. The *two-tuple* is placed into the layer 3.5 APLS header, discussed next.

(ii) Layer 3.5 APLS Label: A layer 3.5 APLS label represents a single layer 4.5 application segment. It serves as a hop-by-hop transport header through the sequence of waypoints implementing the layer 4.5 application segment. It consists of the following fields.

(a) *Two-tuple* <Segment ID, Stream ID>: The <*segment id, stream ID*> two-tuple represents a specific instance of an application segment.

(b) Waypoint ID: The *waypoint entities* represent the middleboxes in an application segment. Together with the *flag bits,* the *way-point entity ID* may be switched to implement the middlebox-sequence. Since, OpenADN is implemented as an overlay; the middlebox may not be directly connected to an OpenADN switch, but indirectly connected over an IP network. Hence, it is important to determine at any point of time, whether the *way-point ID* represents the next middlebox in the sequence (that needs to be traversed) or the middlebox that has already been traversed. The *flags* are used to indicate this. The OpenADN layer in the middlebox just flips one of the flag bits after the message has traversed through it, to indicate this.

(c) Handoff-Locator: The handoff locator is used to deploy OpenADN as an overlay over IP. It needs to be discussed in the context of two specific requirements:

OpenADN as an overlay over IP (Middlebox sequence): To switch through a middlebox-sequence distributed over an IP network, an OpenADN switch might put its own IP address (or an anycast group address) in the *handoff-locator* before forwarding it to the middlebox. The middlebox just copies this to the *destination IP address* field in the IP header in return packets. The packets are sent back to the originating OpenADN switch (or any switch in the anycast group) that then switches it to the next middlebox in the sequence.

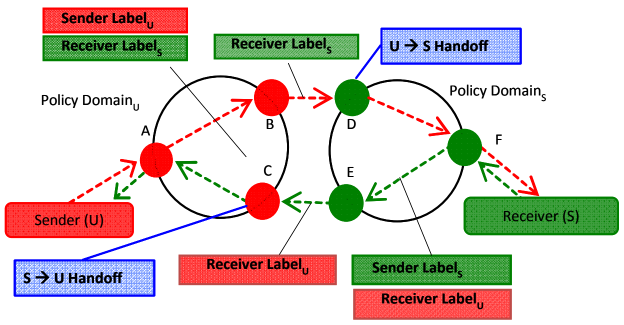

Fig. 7 APLS Label Stacking: Sender and Receiver Policies

Sender and receiver policies: To implement *sender and receiver* policies, the packet has two labels - a sender label ***stacked*** over a receiver label (Fig. 7). The *handoff locator* of the sender label is first copied to the IP destination address field. This is the address (or *anycast address*) of the ingress OpenADN switch of the sender's policy domain. When the sender policies have been enforced (i.e., all sender specified middleboxes traversed), the sender label is popped at the sender's egress OpenADN switch and the *handoff locator* of the next label (receiver label) is copied to the IP destination address. This address sends the packet to the ingress OpenADN switch of the receiver domain.

4. RELATED WORK

Several research efforts have focused on the problem of adding explicit support for middleboxes into the Internet architecture [7, 16, 19]. However, the ad-hoc network configuration techniques and explicit middlebox support in HTTP somehow managed the show and thus the motivation for adopting an architectural change never became too strong. We believe that in the context of multi-cloud environments the problem is more urgent. Also, the problem is very different since the environment imposes a new requirement that the network infrastructure ownership is separate from the application provider. None of the previous proposals try to abstract out application-level semantics into standard representation for a generic and high-performance implementation while still preserving *some richness* of application diversity; allowing application level policies to be enforced on third-party infrastructures.

Research efforts on in-network processing may be considered to lie in the broader periphery of the OpenADN research scope. The in-network processing proposed in ***active networks*** *research* [18] allowed ASPs to write programs in the packet headers. This violated the administrative boundary between ASP and ISP by giving direct control of ISP's router to ASPs. OpenADN avoids this by providing a very restricted and standardized switching abstraction making some allowance for representing the application context. OpenADN may be considered similar to **Rule-based Forwarding architecture (RBF)** [12]. RBF proposes a flexible forwarding plane design by forwarding packets to a *rule* instead to a *destination address.* The key difference is that RBF provides more *flexibility* at the cost of *performance* whereas OpenADN is designed to tradeoff *flexibility* for *higher performance.*

Serval [10] is another recent approach that addresses the problem of accessing geographically distributed service deployments. Serval provides a point solution for accessing distributed services through a service router infrastructure. The service router infrastructure is similar in spirit to the requirement of *outsourcing application-level routing* in OpenADN. However, OpenADN provides a richer interface than just service IDs to generic middlebox-switching solution which allows flexible implementation of any service access mechanism by composing specific mechanisms such as CBR, load balancer, cluster fault-manager, etc. Also, OpenADN allows application context to be included into the switching abstraction. Other distinguishing features include support for middlebox-switching both in the data and control planes and support for both sender and receiver policies.

CloudNaaS [2] proposed an OpenFlow-based data plane to allow ASPs deploy off-path middleboxes in Cloud datacenters. 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 the user and ASP policies, unlike CloudNaaS that allows only ASP policies. Also, CloudNaaS does not allow application-level flow processing like OpenADN, thus unable to provide support for dynamic *service partitioning* and *replication*, routing on user context and *context-based service composition*.

Some other works, such as **CoMB** [13] and **APLOMB** [15] 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 be able to have access to the ASPs application-level data for privacy reasons. Therefore, OpenADN provides a platform, where, instead of third parties, ASPs themselves can manage their middleboxes distributed across different cloud sites more easily.

5. DISCUSSION

We tried to make a case for the need to define a standard common abstraction for supporting application delivery in the context of multi-cloud application deployments. Our specific design of this abstraction layer, OpenADN, is based on the idea that apart from explicit support for ASP owned middlebox deployments, *application-level routing* should be a generic service (like IP) that may be delegated to third party providers by ASPs. The OpenADN implementation is based on an APplication Label Switching (APLS) mechanism. OpenADN's *application label* is a composite label in which one or more than one subfield may be switched by an OpenADN entity (end-hosts, middleboxes, and OpenADN switches) to implement a specific application delivery support function.

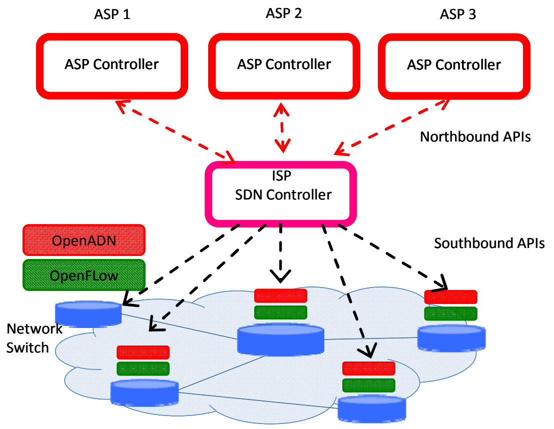

Fig. 8 OpenADN Deployment Plan

OpenADN is an ongoing project and this paper is positioned to make a case for the need of a generic application delivery networking layer to aid the migration of enterprise applications from their private datacenters to cloud-based deployments. OpenADN extends the OpenFlow standard and incorporates OpenADN into the Software Defined Network (SDN) abstraction (Fig. 8). OpenADN introduces new *northbound* (between the ASP controller and the ISP/CSP SDN controller) and *southbound* APIs (to make OpenFlow switches OpenADN-aware). Most importantly, this can be done now while the OpenFlow and SDN research is still evolving. OpenADN is also designed to be backward compatible, in the sense that existing applications over IP can keep working the way they are and new applications can use OpenADN to manage and deliver their dynamic and distributed cloud based deployments more efficiently.

6. REFERENCES

[1] Akamai, "Edge Side Includes," http://www.akamai.com/html/support/esi.html

[2] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "CloudNaaS: a cloud networking platform for enterprise applications," *Symposium on Cloud Computing*, SOCC, 2011.

[3] G. DeCandia, D. Hastorun, M. Jampani, et al., "Dynamo: Amazon's Highly Available Key-value Store," September 2007

[4] R. Fielding, J. Gettys, J. Mogul, et al., "HTTP/1.1," RFC 2616, June 1999.

[5] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," 7th ACM SIGCOMM conference on Internet measurement, pp. 15-28, 2007.

[6] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "The flattening Internet topology: natural evolution, unsightly barnacles or contrived collapse," PAM, pp. 1-10, 2008.

[7] S. Guha, P. Francis, "An End-Middle-End Approach to Connection Establishment," ACM SIGCOMM, 2007.

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

[9] A. Lakshman, P. Malik, "Cassandra – A Decentralized Structured Storage System," SIGOPS Operating Systems Review 44, April, 2010

[10] E. Nordström, D. Shue, P. Gopalan, et al., "Serval: An End-Host Stack for Service-Centric Networking," NSDI, April, 2012.

[11] OpenFlow Switch Specification, Version 1.3.1, September 6, 2012, https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.1.pdf

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

[13] V. Sekar, S. Ratnasamy, N. Egi, et al., "The Middlebox Manifesto: Enabling Innovation in Middlebox Deployments," ACM HotNets 2011.

[14] Jason Sobel, "Scaling Out," Facebook Engineerings Notes, 2008, http://www.facebook.com/note.php?note_id=23844338919

[15] J. Sherry, S. Hasan, C. Scott, et al., "Making Middleboxes Someone else's Problem, Network Processing as a Cloud Service," SIGCOMM, 2012.

[16] I. Stoica, D. Adkins, S. Zhuang, et al., "Internet Indirection Infrastructure," ACM SIGCOMM, 2002

[17] Technavio, "Global Application Delivery Controllers Market in Datacenters 2009-2013," March 2010, http://www.technavio.com/content/global-application-delivery-controllers-market-datacenters-2009-2013.

[18] D. Tennenhouse, J. Smith, W. Sincoskie, et al., "A Survey of Active Network Research," IEEE Comm. Mag., Jan 1997.

[19] M. Walfish, J. Stribling, M. Krohn, et al., "Middleboxes no longer considered harmful," OSDI, 2004.