H-0315 (HAI1206-006) June 17, 2011
Computer Science

IBM Research Report

DOVE: Distributed Overlay Virtual nEtwork Architecture

Rami Cohen, Katherine Barabash, Benny Rochwerger
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Vinit Jain, Renato Recio
IBM STG
Austin, TX
USA

DOVE: Distributed Overlay Virtual nEtwork Architecture

Rami Cohen
IBM Research - Haifa
ramic@il.ibm.com

Katherine Barabash
IBM Research - Haifa
kathy@il.ibm.com

Benny Rochwerger
IBM Research - Haifa
rochwer@il.ibm.com

Vinit Jain
IBM
vjain@us.ibm.com

Renato Recio
IBM
recio@us.ibm.com

ABSTRACT

Networking is currently a field of innovation and massive changes driven by several factors, the most prominent of which is the shift to cloud computing. It is already widely understood that networking requirements of cloud computing infrastructures differ from anything that the established networking technologies can offer.

We present Distributed Overlay Virtual nEtwork (DOVE), an architecture based on a novel network abstraction allowing to define provider-tenant contracts at application level. DOVE network abstraction is decoupled from the particulars of the physical infrastructure design. Still, it captures the network functionality that matters: connectivity, security, performance, etc. DOVE architecture allows cloud providers to consolidate multiple abstractly defined networks on large scale commodity physical infrastructures, utilizing the advantages of specialized hardware appliances, and delegating full control to their tenants that now can manage and administer their virtual networks.

1. INTRODUCTION

Why do we build roads? Probably the most obvious, if not the only, answer is to efficiently transport people and goods from point A to point B. All the technical details of the road do not really matter to the "road customers", i.e. the people in need of transportation. Unfortunately, roads alone are not enough to achieve the efficient ground transportation, so an entire system of "transportation aids" have been designed and added to the initial simplistic model, e.g., synchronized traffic lights, tolls, priority lanes, etc. Even with all this additional complexity, the main goal stays the same, and that is what the vast majority of the users care about.

Similarly, we build networks to provide computer communication services, and just as in the case of roads, the technical details of how this goal is achieved are really not that important to the users and applications transferring data over the networks. And just as roads-system has become more complex over time even if its main goal hasn't changed, networks have become very complex systems and require a lot of expertise to build and manage while their main role hasn't really changed.

What has changed in recent years is the way we deal with servers. Virtualization technologies have been the main enablers of a significant paradigm shift in IT: server provisioning has become a simple task that users of the server can usually perform on their own without any help from the IT department. Moreover, virtualizing workloads has enabled the consolidation of many functional servers (virtualized) into fewer physical servers, a process that leads to increased utilization of physical resources, significant space and energy saving, and simplifies many management aspects (e.g. backups, maintenance window, etc). As a result, modern consolidated data center must support very large scale, highly dynamic workloads, and multitenancy.

Network virtualization lags behind and has achieved neither the ease of provisioning, nor the management and savings advantages that server virtualization has brought to the data center. On the contrary, in some cases the technologies developed to address networking in a virtualized data center increase both complexity and cost. For example, the attempts to extend the physical network all the way to the virtual machines, such as VNTag [2], 802.1Qbg [3], while solve real problems and add some flexibility, miss a crucial point: less physical servers (due to server consolidation) mean less ports are needed, while exposing the VMs to the physical bridges means that the internal tables stay large, hence the big expensive networking equipment is still required. Very large scale and highly dynamic nature of virtualized environments are not adequately supported by the existing, legacy based, network virtualization attempts.

What is really needed is a clean abstraction that, just as in the case of server virtualization, fully separates the logical function from the underlying physical infrastructure. This abstraction needs to capture only the high level functionality needed leaving outside the low level physical mechanisms used to achieve such functionality. In case of network virtualization, the main function is clearly providing connectivity between endpoints. However, in today's rich IT environments, connectivity alone is not enough and much more is expected from the networking services. In particular, for security reasons users may want to inspect all the data flowing in and out of some endpoint, or for efficiency users may want to compress/uncompress data from/to a backend networked storage device. In the physical network, these advanced functions are usually achieved by placing dedicated middleboxes[1] along the physical path travelled by packets in and out of servers. In a virtualized environment, users still want these services, e.g. security, compression, acceleration. There are two important conflicting issues with moving such services to a virtual level. On the one hand, the required function has to be described as part of abstract description of virtual network and in a way independent of physical location and topology. On the other hand, it is crucial that the actual function is provided by real hardware devices, highly optimized and efficient, and not by software substitutes as, for example, is the case with VMware vShield product family. Clients are not willing to sacrifice the performance and the high level of service they are used to with hardware appliances and wish to be able to leverage investment they made into their existing costly equipment.

In addition to a well defined abstraction, we need the ability to easily create and maintain multiple isolated virtual networks without requiring to reconfigure the physical network. Moreover, the move to cloud computing means that these isolated virtual networks may belong to different tenants that will demand not only isolation but also the ability to manage their virtual networks (including their address spaces, topology, etc) independently of other tenants and without having to turn to the infrastructure provider for every change.

From the cloud computing provider point of view it is crucial to meet the following requirements: *Scalability*, i.e. large number of highly dynamic virtual networks and virtual network endpoints; Full *isolation* between the different virtual networks, including performance isolation, management control, etc; Ability to use commodity infrastructure components, including simple switches and specialized hardware appliances, to fully realize *advantages of scale* related to consolidation. In addition, good network *abstraction* is required to allow network specifications to be, on the one hand, agnostic of particulars of infrastructure design, while, on the other hand, to capture all the network considerations that matter: security, QoS, etc. Meeting these requirements based solely on commonly used networking technologies is cumbersome and in many cases prohibitively expensive. There is a need for a network virtualization layer that can support multitenancy in a highly scalable, very dynamic, and cost efficient system.

We have architectured the Distributed Overlay Virtual nEtwork (DOVE), a novel networking architecture that allows building systems answering the above set of requirements. DOVE architecture is based on a clean and simple network abstraction allowing to define provider-tenant contracts at application level. DOVE network abstraction is decoupled from the particulars of the physical infrastructure design. Still, it captures the network functionality that matters: connectivity, security, performance, etc. DOVE architecture allows cloud providers to consolidate multiple abstractly defined networks on large scale commodity physical infrastructures, utilizing the advantages of specialized hardware appliances, and delegating full control to their tenants that now can manage and administer their virtual networks.

The rest of this papers is structured as followed: Section 2 describes the most influential related works. In Section 3 we present the DOVE network abstraction and in Section 4 the DOVE architecture is described in details. We present our evaluation of architectural ability of DOVE to scale in Section 5, and summarize the work and discuss future directions in Section 6.

2. RELATED WORK

The most recent attempts to address the need for creating and managing multiple isolated virtual networks in large scale consolidated environments at low deployment and operational costs are VXLAN [21] and NetLord [23]. Both these, as well as many other works (see [35, 13, 17]), acknowledge the limitations of existing network virtualization technologies. For example, both recognise that VLAN [1] technology is too rigid and too tied to the physical infrastructure so it can not allow for proper network virtualization where dynamics, flexibility and scale are required both in operation and in configuration. NetLord also rightfully criticizes solutions requiring the networking state of every attached (virtual) endpoint to be exposed to the physical switches (e.g. SPAIN [22]) for their inability to scale without heavily increasing switch resources, as well as architectures depending on complex feature support in physical switching hardware (e.g., SEATTLE [17] requires programmable MAC-in-MAC encapsulation support).

Both VXLAN and NetLord provide multiple virtual networks by creating overlays to achieve benefits of scale, isolation of tenants one from another and from the physical infrastructure, and ease of operation and configuration. However, neither provides proper abstraction for describing the virtual network services, both have non-trivial requirements from the underlying infrastructure and both are limited in their ability to scale. Not

[1]We use term "middlebox" interchangeably with the term "network appliance", to signify a piece of network equipment installed on a data forwarding paths [7]. Examples of middleboxes are firewalls, intrusion detection systems, encryption and compression engines, SSL accelerators, etc.

VXLAN and not NetLord address supporting network services besides the simple connectivity. In what follows we briefly summarize how these works come short of answering the complete set of requirements.

In order to stretch multiple L2 networks over a shared L3 data center infrastructure, Virtual eXtensible Local Area Network (VXLAN [21]) emulates the learning based control protocols used by L2 including data and control flooding. For this purposes, in order to provide unicast service, VXLAN relies on a multicast infrastructure where each multicast group emulates a virtual L2 broadcast domain. Thus, while avoiding the limitations associated with VLAN maintenance and configuration, it inherits the complexity and inflexibility of physical L2 network by reusing the fully distributed, learning based method of dealing with unknown information, and thus can not adequately scale in highly dynamic environments.

NetLord is a novel multitenant network architecture that, like VXLAN, encapsulates tenant's L2 packets. Encapsulation is different and involves a custom mix of L2 and L3 headers to ensure both tenant isolation and efficient packet delivery in the underlying multipath L2 network. Although NetLord provides isolation between tenants and takes care of efficient packet forwarding, it equates network virtualization with address space virtualization allowing tenants to control their L2 and L3 addresses. We claim that address virtualization alone is not enough and there is a need to fully abstract the application level network services from the underlying resources. In addition, in NetLord architecture, a fully consistent network state is maintained and synchronized between all the NetLord Agents (NLAs). Thus, every change (VM creation, migration, or termination, virtual network creation, etc) requires the information to propagate to all the overlay endpoints which clearly can not scale in highly dynamic environments. Another, lesser, NetLord limitation is requiring L2 based forwarding fabric underneath while it is starting to become apparent that simple L3 based data center designs are cheaper and more robust at large scales.

Most of other recent research works advocating the creation of novel network architectures focus on specific network properties or services: security [30], access control [9, 8], policy based routing and switching [16], multitenancy [5], and performance isolation [33, 34], while some, like Portland [24], target specific physical network environments. Most solutions expose to the network clients constructs and interfaces familiar from the physical networks: L2 broadcast domains, switch ports, per-port access control properties and even VLANs [17, 24, 14, 23]. The reason is that most of the community is caught in the conceptual trap of trying to provide, in one way or another, the same type of controls and interfaces as everyone is used to receive from physical networks. Some works that do try to create a better abstraction of virtual network services [5, 4, 10], still end up with abstractions that closely follow the existing networking constructs. For example, Oktopus [4], when aiming at abstraction that can be used to create cloud-tenant contracts with network performance guarantees, ends up providing concepts emulating the existing data center network design constructs like "cluster" emulating switch and "oversubscribed cluster" emulating a two tier fat tree switching network.

OpenFlow [20], initially developed to add programmability into the physical data forwarding devices, has been taken very far since its initial conception and have given raise to the Software Defined Networking (SDN), a network architecture in which the network control plane is decoupled from the physical topology. While most often the SDN concept is applied to programmatic control of the physical network [27, 15, 18], there are initial attempts to adopt it to the virtualization use case [28, 10]. To the best of our knowledge, there are no SDN based solutions answering the full set of requirements presented in this paper. We believe that the DOVE architecture is aligned with the vision and the direction of SDN.

The policy-aware switching layer, proposed by Joseph at al [16] acknowledged the importance of network appliances to provide essential enhanced networking services, and showed how traffic engineering techniques can be used to enforce forwarding of different types of traffic through different sequences of middleboxes in a flexible, efficient and guaranteed way considering a dynamic environment such as data center. The way we've chosen to support hardware appliances in DOVE architecture (see Section 4) is greatly inspired by this work.

3. DOVE NETWORK ABSTRACTION

Modern network applications are usually deployed as sets of interconnected components, and connectivity requirements between application components are integral part of deployment specification. Figure 1a presents an example three tier application consisting of a load balancer connected to the public Internet and to a set of application servers that in turn are connected to a set of database servers. In this example, security and performance constrains dictate that the following appliances must be properly installed and configured: firewall for checking all the traffic between the load balancer and the application servers, SSL accelerator to speed up HTTPS communications between the load balancer and the application servers, and compression engine on a way to/from the database servers.

Typically, deploying network applications as in Figure 1a requires the involvement of a network administrator to design and configure the network infrastructure supporting the above connectivity requirements.

This can be done, for example, by subnet configuration whereby a single L2 domain is divided into sub networks to enable, among other things, endpoint isolation and to enforce forwarding paths to pass through the required middleboxes. With these technologies, application deployment specifications include low level networking constructs such as VLAN tags, switch ports, and router configuration. Although these constructs are well known to and widely used by the network administrators [36], they are ill suited to specifying the essence of the networking requirements of the application. In addition to being difficult to grasp for not networking professionals, such low level specifications are inevitably tied to a particular network infrastructure, making it hard to correlate changes in application-level networking requirements (e.g. due to application dynamics) with physical network required to support them and vice versa. as a result, data center networks are usually rigid, hard to manage and thus non-scalable.

There is a need for a clear, convenient, high level, and infrastructure independent abstraction for specifying networking requirements of applications. Abstract network description must not contain any physical or control information but only high level functional information in order to provide a flexible and hardware independent blueprint that can be used for logical specification of an application network. An abstract network description should be technology and topology independent, and therefore can be instantiated into an actual deployment over any kind of physical infrastructure by means of infrastructure specific configuration actions.

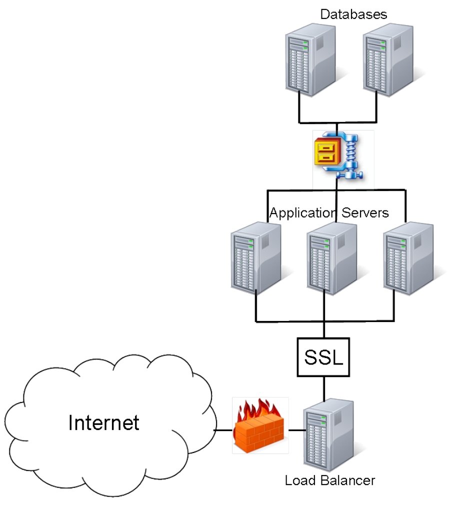
We create such an abstraction, observing that application level networking requirements are best described in terms of the **connectivity** between endpoints and the **policies** associated with the connectivity. In what follows we describe DOVE network abstraction that is based on the notions of *policy*, *policy action*, and *policy domain*. Later, in Section 4, we show how DOVE architecture resolves the abstract application network description into a set actions and instructions applied to a physical infrastructure in order instantiate and maintain the logical network blueprint.

In DOVE, *policy* is a set of criteria characterizing the connectivity between a pair of communicating entities, communication source and communication target. This characterization can specify, among other things, access control, QoS, security, and performance criteria. For this, policy characterization can include a sequence of *policy actions* that must be applied when forwarding data between the source and the destination endpoints. Policy actions define, among other things, security and performance related demands, which are typically enforced by network appliances such as network accelerators, intrusion detection and prevention systems, firewalls, etc. Note that although DOVE policies and actions associated with them are abstract and may include any type of criteria, their actual enforcement is carried out by the physical infrastructure. Thus, in practice, specified policy actions must be backed up by specific infrastructure capabilities. For instance, while a simple accept/deny authorization policy can be enforced by any infrastructure, enhanced firewall capabilities and accelerating SSL operations will usually depend upon existence and suitable configuration of corresponding network appliances.

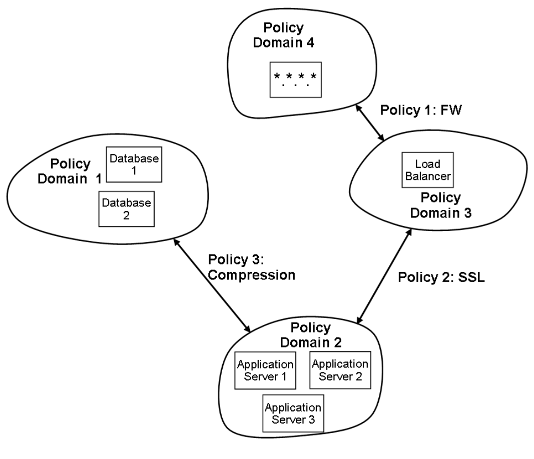
While, in general, policies characterizing network connectivity can be defined directly between communicating endpoints, in DOVE policies are defined between sets of endpoints sharing common policy criteria and called *policy domains*. Specifying policies at this level enables identical policies aggregation and reduces the total number of policies required to define network functionality. In addition, defining policies between policy domains and not between endpoints decouples policy management from endpoints management thus enabling better separation of duties between different management roles. For example, management entity responsible for application deployment can define empty policy domains and polices between them while a different management entity (or set of such) can subsequently add, remove and maintain endpoints within policy domains assigned to it. Such separation of duties allows for flexible and dynamic creation and maintenance of virtual networks.

With DOVE network abstraction, the network application presented in Figure 1a can be described using four policy domains as shown in Figure 1b: the first policy domain for the database servers, the second for the application servers, the third for the load balancer, and the forth policy domain for the external clients. Application blueprint can be deployed before actually instantiating any of the components or together with a minimal set of instantiated components. Later during the application lifetime, endpoints can be dynamically added to and removed from policy domains according to, for example, the application load, while the application network blueprint remains the same. Clearly, if different endpoints in a single policy domain require different network services, a policy domain may be further partitioned into several new policy domains so that all the endpoints within a single policy domain have the same policy criteria.

Note, that in DOVE, policies are unidirectional and typically two unidirectional policies should be defined for each pair of policy domains. Also, a policy from a policy domain to itself can be defined to specify the communication criteria associated with data sent between endpoints inside this policy domain. In addition, to simplify the network description, one may set a default policy that should be applied when there is no ex-



(a)



(b)

Figure 1: Network Application Example – (a) shows the application architecture as it is typically provided by the application architect, while (b) shows DOVE network description for the same application architecture. Note that both parts are completely logical and abstract and do not contain any concepts form the core networking domain.

plicit policy definition. Thus, the application network can be specified as a directed graph where the vertexes represent policy domains and the directed edges represent policies.

4. DOVE ARCHITECTURE

In this section we describe DOVE architecture allowing to create virtualization layer for deploying, controlling, and managing multiple independent and isolated network applications over a shared physical network infrastructure. As was stated in the introduction, our goal is to satisfy requirements of large scale consolidated data centers hosting multiple applications belonging to different tenants, while preserving the functionality and the flexibility of each application network and without reproducing the complexity and the overhead associated with typical network deployments. In DOVE, data traffic is handled by distributed data plane entities called DOVE Switches[2] and control is achieved through a control plane engine called DOVE Policy Service, as described below.

- *dSwitch*, described in Section 4.1, is responsible for connectivity and policy enforcement in DOVE environment. Each dSwitch serves a dynamic set of endpoints[3], and all the traffic sent and received by a DOVE hosted endpoint must traverse its hosting dSwitch. dSwitches obtain connectivity and policy information through interaction with the *DOVE Policy Service* (DPS), and cache it locally. In a typical virtualized data center deployment, a dSwitch is deployed on each physical server and serves the virtual machines currently hosted by this server.

- *DOVE Policy Service* (DPS), described in Section 4.2, is responsible for maintaining the abstract blueprint of virtual networks as described in Section 3, as well as for creation, modification, and deletion of virtual networks. In addition, DPS maintains the correlation between the logical description of the virtual networks and the physical infrastructure, including the physical location of virtual machines, implicit or explicit information regarding the location of network appliances, their configuration, etc. Based on this information, DPS resolves and validates policy requests received from dSwitches and maps these requests to packet sending instructions.

DOVE provides isolation between the virtual and the physical domains by creating an overlay network between dSwitches that is transparent to the endpoints and is used to send and receive data between them. Each packet sent from one endpoint to another, is encapsulated by the dSwitch serving the source endpoint using an IP based tunnelling protocol[4] and sent to the dSwitch serving the destination endpoint, subject to policy enforcement actions. Despite the overhead associated with tunnelling, such approach overcomes many limitations associated with direct connectivity in which

[2]In the rest of the paper we denote DOVE Switch by a short term *dSwitch*.

[3]Strictly speaking communication endpoint is an addressable network interface, either virtual or physical. In this paper, for simplicity, we refer to virtual or physical hosts as to endpoints as well.

[4]Here it is assumed the underlying physical infrastructure is IP based. Otherwise, a different tunnelling protocol should be used.

endpoints are clients of the physical network. First, physical switches do not longer need to deal with all the dynamic set of virtual servers, but only with much smaller set of static physical servers, so they need to support less addresses and less configuration and control protocols. Second, with overlay, virtual servers are isolated from the physical infrastructure, enabling to deploy DOVE over different network technologies (e.g. Ethernet, Infiniband, IPv4, IPv6) and topologies. Third, overlay achieves isolation between virtual networks, enabling each virtual network to define its own network characteristics independently, including its topology and address space. As a result, more than one virtual network can share the same address space in the virtual domain and different virtual networks (e.g. IPv4 and IPv6) can coexist on a shared physical infrastructure. For further reading regarding overlay advantages in the virtual environment see [23, 26].

DOVE provides isolation between the virtual and the physical domains, making it possible to allow each network to define its own network configuration including IP and MAC addresses so that address spaces used by virtual networks are decoupled from the physical network address space. We refer to addresses defined in virtual networks as to *virtual addresses* and to addresses used in the physical infrastructure as to *physical addresses*. Note that, architecturally, DOVE is agnostic to anything related to the core networking business and can be implemented in any physical network infrastructure with the only assumption of reachability (L2 or L3) between the dSwitches and between the dSwitches and the DPS. Data network connecting dSwitches can be either separate or shared with the control network connecting dSwitches to the DPS. Simple IP-based data center networks built with commodity components easily answer these requirements.

Although endpoints are connected to the DOVE environment by NICs (physical or virtual), DOVE does not emulate L2 protocol (e.g. Ethernet), making a point that L2 protocol is not a service but a means to carry data. DOVE enables endpoints to send and receive L2/L3 data without emulating and reproducing the complexity of the control protocols associated with L2 infrastructure. For instance, instead of emulating ARP in the virtual domain by sending ARP requests to all the members within virtual network or the policy domain, DOVE utilizes the DPS in order to deal with such resolutions, and instead of setting VLAN tags to enforce security requirements, DOVE controls the data forwarding by configuring the overlay network as will be described later in this section.

In general, policy may refer to many subjects, including security, QoS, etc., and its enforcement is typically subject to the physical infrastructure capabilities. The enforcement of QoS based policy, for example, may require some support from the forwarding entities, while a security policy, on the other hand, may require deployment of network appliances such as firewall, and the exact functionality is varied from one appliance to another, according to their specifications. Thus, the set of policies that can be defined in the logical network description is derived from the physical deployment and the implementation. In this work we focus on network appliances based policies (firewalls, intrusion detection and prevention systems, accelerators, etc.), that is, policies requiring the data to pass through a sequence of physical appliances. Inspired by Joseph at al [16], DOVE enforces such policies by controlling the data path. That is data is not sent directly from the source to the destination dSwitch, and a path control mechanism is required. The exact path control mechanism depends of the physical infrastructure and it may be based on loose source routing [31, 11] or MPLS [32] tagging. Other approaches are possible, for example one used by Spain [22] or by Joseph at al [16]. While DOVE is responsible to carry the data through the appliances, their configuration done independently. In particular, these appliances should be logically partitioned such that each tenant can manage and configure its own partition. In addition, such multitenant appliances should be able to distinguish between data sent from different virtual networks (see for example Juniper NetScreen Firewall).

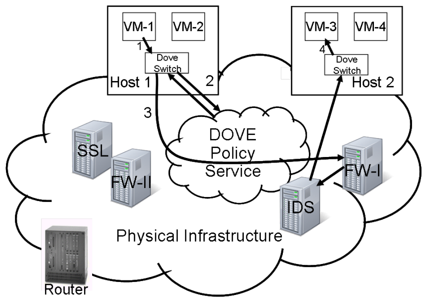


Figure 2: DOVE Data Flow between endpoints hosted by two different dSwitches: Step 1 – Packet interception by dSwitch hosting the sending endpoint; Step 2 – Local policy lookup and, if needed, acquiring it from the DPS; Step 3 – Packet encapsulation and sending through a set of appliances towards the destination dSwitch; Step 4 – Packet decapsulation and delivery to the destination endpoint.

Figure 2 illustrates DOVE data flow between endpoints served by two different dSwitches. In this example, the packet sent from VM-1 to VM-3 is intercepted by the dSwitch located on physical server 1 (Step 1 in Figure 2). If the policy associated with this packet is not found in its local cache, this dSwitch acquires it by sending a policy request to the DPS. The DPS parses and resolves the request, and sends a policy reply to the dSwitch. The policy reply contains the address of the dSwitch hosting VM-3, as well as the path control

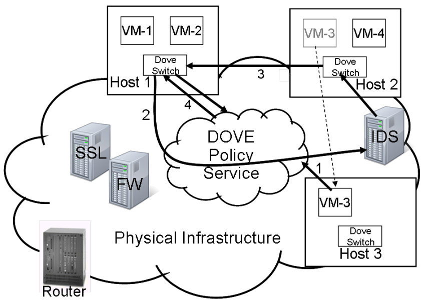


Figure 3: DOVE Data Flow of packets destined to a just migrated endpoint: Step 1 – Upon migration the DPS is updated by the dSwitch at the new location; Step 2 – Another dSwitch uses its stale cached policy and sends data to the dSwitch at the old location; Step 3 – dSwitch at the old location sends location invalidation message to the source dSwitch; Step 4 – Source dSwitch acquires the correct policy so that following packets are sent correctly.

information (a set of appliances the packet must pass through on its way to the destination) required to enforce the policy (Step 2 in Figure 2). Upon receiving the policy reply, dSwitch stores the policy information in its local cache so it can be reused for subsequent packets subject to the same policy. At this point, dSwitch encapsulates the packet so that the outer headers contain the physical addresses of the source and the destination dSwitches, as well as the path control information necessary for the packet to traverse the sequence of required appliances. In addition, packets sent over the wire must contain the virtual network identification to enable the destination dSwitch to dispatch the original packet to the destination endpoint (Step 3 in Figure 2). Once the packet is received by the destination dSwitch, it decapsulates the packet and delivers it to the destination endpoint according to the destination address and the virtual network identifier (Step 4 in Figure 2).

When some endpoint changes its location and its hosting dSwitch (for example, when virtual machine is migrated from one physical server to another), stale cached policy can be used to send data destined to the migrated endpoint to the old location. To achieve policy consistency in a scalable manner, we use cache invalidation messages as illustrated in Figure 3. In this example, VM-3 is migrated from physical server 2 to physical server 3. When the migration is completed, dSwitch at the new endpoint location updates the DPS about the new physical location of the endpoint (Step 1 in Figure 3). If, as shown in Step 2 of Figure 3, another dSwitch sends encapsulated packet to the old destination, the dSwitch located there sends a location invalidation control message to the source dSwitch (Step 3 in Figure 3). Upon receiving the location invalidation control message, the source dSwitch acquires the new policy containing the new location information and sends further packets according this new policy (Step 4 in Figure 3)[5]. Note, that if the old destination dSwitch is unavailable (e.g. due to physical server shutdown or in a server failover scenario), the policy consistency has to be achieved by other means. In DOVE, there are two additional mechanisms that are used in such cases: first, policy cache timeout will eventually force new policy acquisition, and second, DPS will push policy invalidation messages (as described in Section 4.2) to a computed set of dSwitches when required.

4.1 DOVE Switch (dSwitch)

dSwitches serve as middle men in all the communications between the hosted endpoints so there must be dSwitch adjacent to every endpoint in the environment. Deployement decisions can differ according to the environment, taking into account requirements of scale and resiliency as well as depending on a data center interconnect and the endpoint type. For example, dSwitch serving virtual machines can be deployed as part of the hypervisor virtual switch. Another possibility is to collocate dSwitch with the ToR switch, allowing it to serve all the physical servers in the rack. For endpoints external to DOVE environment (e. g. Internet clients, servers in the remote data center or computers in the home office), dSwitches can be introduced as part of or immediately adjunct to the perimeter routers (see Section4.3).

4.1.1 Protocols

dSwitches participate in several types of communications, each requiring its own protocol, as described below:

Delivering data packets. dSwitches deliver data packets between endpoints using the data encapsulation protocol. Particulars of the protocol are not enforced by the architecture but depend on the actual implementation. For example, to provide L2 type of service, data packets must contain meaningful endpoint-level L2 headers while to provide L3 type of service, L2 headers can be safely discarded by dSwitches thus reducing the complexity and processing overhead as well as to reduce the total packet size transported by the wire. Outer headers, on the other hand will depend on the physical infrastructure where DOVE is deployed and the same holds for fields carrying the path control information. DOVE specific fields of the encapsulation header absolutely must include the logical network identity but can be extended to include other fields, for example, finer grained identity of the destination endpoint or fragmentation related flags.

[5]Policy acquisition process can be eliminated if new endpoint location information can be included in the location invalidation control message

Sending inter dSwitch control messages. Control messages can be sent either over the data network or over the control network interconnecting dSwitches. One case where such message is needed, namely, invalidating stale cache entries relating to the migrated endpoints, is described above, with relation to Figure 3.

Acquiring policy information for packets generated by hosted endpoints. For this, dSwitches engage in policy request/reply protocol with the DPS. dSwitches store the resolved policy information received from the DPOS in the local cache with timed entries. When some information used to resolve the policy is changed in the DPS, it will push cache invalidation messages to relevant dSwitches as described in Section 4.2.

Reporting location and virtual address information for hosted endpoints. In most environments, the most efficient way to trace endpoint location and virtual address information is by being deployed immediately close to it in the physical network in a way enabling to oversee all its communications. This is exactly the way dSwitches are deployed. One way to actualize DOVE architecture is with dSwitches that collect and trace this information (either using hypervisor specific hints or by inspecting data packets) and send endpoint location updates upon every detected change.[6]

Providing network services for hosted endpoints. In some cases, dSwitch acts as a proxy between the hosted endpoint and other network component assumed by it to be immediately reachable in the network. For example, dSwitch intercepts address resolution (for example, ARP) requests and, instead of broadcasting them so they can reach the destination (endpoint with the requested IP address), consults the policy corresponding to the request and creates address resolution reply using data needed to create such a reply (in ARP example, the MAC address of the destination endpoint)[7]. For all such cases, dSwitch must engage in communication with endpoint (generate protocol messages for their consumption) and with the policy service which must be extended to handle all the supported protocols and provide enough information to implement the proxy functionality as part of policy resolution reply. To support ARP, for example, DPS is extended to keep the mapping between the virtual MAC and the virtual IP of the hosted endpoints, as described in Section 4.4.

4.1.2 Data Structures

To serve their hosted endpoints and to participate in the communications described above, dSwitches must maintain and manage simple state as follows:

Hosted endpoints table. For every currently active hosted endpoint, dSwitch keeps its NIC properties (e.g. virtual interface number), its DOVE related properties (virtual network identity, policy domain identity, etc), and its available virtual address information (e. g. virtual MAC, virtual IP).

Local policy cache. In its local cache dSwitch keeps information received from the DPS as part of replies for its own policy requests. Policy cash must be designed to allow efficient look ups in order not to delay data forwarding for packets that have their policy information in cache. Policy cache entries are timed according to the caching properties assigned to them by the DPS (see Section4.2). Policy cache entries can also be evicted by policy invalidation messages either from the DPS or form another dSwitch.

DPS access information and state. For DOVE to function, it is crucial that each dSwitch at any time is able to reach the DPS for resolving policy for a data packet it has to send. Depending on the DPS distribution and deployement models, dSwitches will have different ways to reach the DPS and to track the status of outstanding communication with it. This can include, for example, the heartbeat messages, retry state of the policy resolution protocol messages, etc.

4.2 DOVE Policy Service (DPS)

DOVE Policy Service is a critical component serving the entire DOVE environment, and as such it should be carefully designed, implemented and deployed so that it is *always* available to serve policy resolution requests by *any* dSwitch in the environment and provides policy resolution replies with a sufficiently *low latency.* For this, DPS must be highly available, resilient, and scalable. The exact quantitative requirements are subject to many parameters, including the number of virtual networks to be supported and their size, the service level to be provided, etc., and will vary from one deployment to another, resulting in different designs. In our design, DPS is a highly available servers cluster maintaining eventually consistent distributed state. For the sake of brevity, we omit the exact cluster design and present the architecturally significant functional properties of the DPS as if it was implemented by a single server.

4.2.1 Databases and Virtual Network Maintenance

DPS maintains all the information required to resolve policy requests issued by dSwitches, information regarding existing virtual networks, their policy domains, policies, endpoints, etc., and their correlation with the physical infrastructure (if available). Since policy resolution

[6]In some environments, global data center management tools can be used by DPS to acquire and maintain endpoint location and virtual address data.

[7]Another example would be DHCP proxying.

latency is critical, one should pay attention to the way these databases are implemented and maintained to allow for fast and efficient resolution. Such considerations are out of the scope of this work; we describe the databases as a set of tables focusing on their content and their correlation with each other.

DOVE Virtual Networks table. The first database is a global one that serves the entire DPS and contains all the currently existing virtual networks. Each virtual network is identified by a unique ID and is, in turn, associated with a set of databases maintaining information regarding the policy domains, the endpoints, the policies, the policy actions and other DOVE properties associated with this virtual network.

DOVE Policy Domain table. This table contains all the information regarding policy domains within a specific virtual network. Each policy domain has an ID, unique in the virtual network, as well as a list of the members in the policy domain, that is a set of endpoints, including virtual machines and external nodes.

DOVE Policy Action table. To define the abstract network policies as described in Section 3, each virtual network can define a set of logical actions. These actions, on the one hand, are describing what checks or filters must be enforced upon the traffic, while, on the other hand, must be backed up by actual appliances deployed and configured in a physical infrastructure. The DOVE Policy Action table maintains all the configured actions within a specific virtual network, identified by their IDs, unique in the virtual network. Policy action entry may also contain the actual configuration associated with each action (e.g. the set of firewall rules) and a mapping information used to correlate between the action and the physical infrastructure, based on the path control mechanism used to enforce the policy. For instance, if IP source routing is used to control the data path, then each action contains the IP address of the corresponding appliance.

DOVE Policy table. The DOVE Policy Table is used to maintain all the policies within a specific virtual network. Each entry in the table contains a unidirectional policy that is valid for data sent from endpoints in the source policy domain to endpoints in the destination policy domain. If both source and destination domains are identical, the entry describes policy internal to the domain, namely the policy associated with data sent between endpoints in the domain. For each policy, identified by an ordered pair of identities of involved policy domains, this table contains a sequence of logical actions identified by their IDs. In addition, each policy can be assigned its own caching property, determining the amount of time this policy should be cached from the moment is was acquired by the dSwitch. In addition, each policy has a tracking number, which is increased by one each time the policy is updated. dSwitches identify deprecated policy cache entry when they detect that tracing number associated with it is less when the one currently distributed by the DPS.

DOVE Endpoint table. The DOVE Endpoint table contains information regarding endpoints, both internal and external, accessible by a specific virtual network. Each endpoint entry contains an ID that is unique within the virtual network, and may vary from one type of endpoint to another. Virtual machine, for example, can be identified by a unique ID generated by the virtualization platform manager, while external servers can be identified by aunique ID or by their IP addresses, masks or domain names[8]. In addition, each endpoint entry contains the following information:

> *Policy domain:* Identity of the policy domain the endpoint belongs to.
>
> *Physical location:* Information regarding the current dSwitch serving the endpoint. This information is dynamic and can change when the endpoint migrates from one physical location to another. As described in Section 4.1, DPS tracks the location of virtual machines through interaction either with the virtualization manager or with the dSwitches.
>
> *Endpoint virtual addresses:* The set of L2 (e.g. MAC) and L3 (e.g. IPv4 or IPv6) addresses used by the endpoint to send and receive the data. While the virtualization platform management is usually responsible to the physical location upon provisioning, deletion, and migration of virtual machines, it may be not part of its role to handle the virtual machine address configuration which may be configured statically or automatically (e.g. using DHCP). As described in Section 4.1, dSwitches collect this information through data packets inspection and update the DPS on every detected change. In some cases, however, for example when the deployed endpoint has not sent/received any messages yet, this information is not available to its hosting dSwitch and thus to the DPS. For such (rare) cases, DPS implements the DOVE Address Resolution Protocol (described later Section 4.4) to discover the virtual addresses and fill in this field in the table.

Figure 4 shows one possible deployment of the logical network described in Section 3. In this example, the application components are deployed on three physical

[8]By using mask and domain, a set of servers can be aggregated, and presented as a single entry in the table.

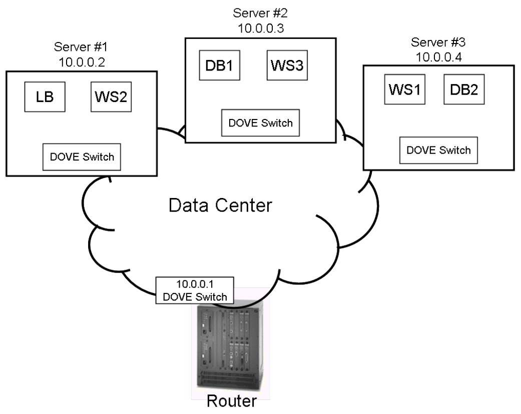


Figure 4: A possible deployment of the logical network presented in Figure 1

servers: physical server, with IP address of 10.0.0.2, hosts the load balancer and the second web server; second physical server, with the IP address of 10.0.0.3, hosts the first database and the third web servers; while the third physical server, with IP address of 10.0.0.4, hosts the second database and the second web servers. In every physical server, there is dSwitch serving hosted virtual machines, while an additional dSwitch, accessible through the 10.0.0.1, serves the edge router connecting the data center to the public Internet[9]. The DPS databases derived from this network description and deployment may consist of the tables 1, 2, 3, and 4. Note that in the network state shown in the tables, not all the virtual IP addresses are known to the DPS. When these addresses will be required to resolve a policy request, DPS will initiate the DOVE Address Resolution Protocol described in Section 4.4.

Unique ID	Name	Member List
D-1	External	E-1
D-2	Load Balancer	E-2
D-3	Application	E-3, E-4, E-5
D-4	Database	E-6

Table 1: Policy Domain Table for network application presented in Figure 1 and its deployment presented in Figure 4

Unique ID	Source Policy Domain	Destination Policy Domain	Tracking Number	Cache (Sec.)	Action
P-1	D-1	D-2	1	300	A-1
P-3	D-2	D-3	137	200	A-2
⋮					
P-6	D-4	D-3	1	300	A-3
P-9	Default		1	300	A-4

Table 2: Policy Table for network application presented in Figure 1 and its deployment presented in Figure 4

[9]This dSwitch can be collocated with the router, or it can be deployed on a different virtual or physical server. In both cases the router should be configured such that all the incoming external traffic is directed to this dSwitch, and vice versa.

Unique ID	Name	Policy Domain	Physical IP	Virtual IP
E-1	Ext.	D-1	10.0.0.1	255.255.255.255/32
E-2	LB	D-2	10.0.0.2	9.2.10.34
E-3	WS1	D-3	10.0.0.3	9.2.10.26
⋮				
E-7	DB2	D4-2	10.0.0.4	N/A

Table 3: Endpoint Table for network application presented in Figure 1 and its deployment presented in Figure 4

Unique ID	Name	Physical Location	Config.
A-1	FW-1	9.2.5.1	...
A-2	SSL	9.2.5.4	...
A-3	Compession	9.2.4.3	...
A-4	Deny	N/A	N/A
A-5	Accept	N/A	N/A

Table 4: Action Table for network application presented in Figure 1 and its deployment presented in Figure 4

4.2.2 *Policy Resolution*

Upon receiving a policy resolution request from a dSwitch, the DPS maps the request into a set of instructions required for the policy enforcement by means of a sequence of database lookups. First, the DPS uses the Endpoint table to resolve the destination endpoint location by mapping the destination virtual address to the destination physical address (i.e. the address of the hosting dSwitch)[10]. Using the same entry in the Endpoint table, the DPS obtains the policy domain of the destination endpoint. Then, using the source policy domain extracted from the request and the destination policy domain, the DPS locates the policy entry in the policy table. The path control information is determined according to the Action field and the Policy Action table lookup. For example, in source routing based path control, the sequence of logical actions in the policy entry is mapped to a sequence of physical addresses (one per action). When the resolution process is completed, the DPS creates policy reply containing the destination physical address, the path control information, the policy tracking number and the cache expiration time, and sends the reply to the requesting dSwitch.

4.3 External Connectivity and Addressing

Inside a DOVE environment, most endpoints can be configured using private address spaces. VPN techniques can be used to provide connectivity between DOVE virtual networks and remote servers. In many cases, however, DOVE endpoint should be addressable by external machines in the public domain. In this case, similarly to established networking practices, each virtual network must own at least one global public IP address

[10]In a rare case the table does not contain the virtual address of an endpoint, DPS initiates the DOVE Address Resolution protocol as described in Section 4.4.

that is used for external connectivity. These IP addresses are part of the virtual network configuration assigned by the data center operator, either manually or automatically, and must be part of DPS state maintained for this virtual network. The exact number of public IP addresses may vary, depending, for example, on a service level agreement.

External connectivity between virtual networks and public servers can be achieved using NAPT (Network Address Port Translation) and port forwarding. In this case, dSwitches connected to edge routers and responsible to outgoing and incoming traffic, perform address translation in addition to policy acquisition and packet encapsulation/decapsulation. Thus, upon receiving external packet, the edge router sends it to such dSwitch that performs address translation (replacing the destination public IP address with the private IP address), followed by policy acquisition using the public IP address as an identifier to the specific virtual network (recall that each public IP address is assigned to a specific virtual network). Following the address translation, dSwitch encapsulates the packet according to the policy instructions and sends it through a sequence of network appliances to the destination dSwitch. Outgoing packets, on the other hand, are sent by the dSwitch hosting the source endpoint to the dSwitch serving external endpoint. This switch decapsulates the packet, performs address translation and sends the packet to the edge router.

Note that such translating dSwitches can be either collocated with the edge router or deployed on a separate host. In addition, the address translation module can be separated from the dSwitch. In this case, it must be logically located between the edge router and the dSwitch and its configuration should enable the dSwitch to distinguish between packets sent to different virtual networks[11]. In addition, to enable scalability, resiliency, and high availability, a set of x dSwitches may serve a set of y edge routers by directing data from the router to the dSwitch based on the virtual network identity, session based load balancing, etc.

4.4 Dealing With Unknown Information

The process of sending data between network endpoints typically involves dealing with unknown information. This includes mapping a domain name to a network address, mapping a network address to a MAC address, setting routing information in the forwarding entities (switches, and routers), etc. The actual method for resolving unknowns depends on the type of information and its scale, but it is also derived from legacy architectures. In particular, data generated by endpoints is typically followed by a broadcast ARP (Address Resolution Protocol) [29] or multicast NDP (Neighbour Discovery Protocol) [25] request used to map the network address to a MAC address. These resolution protocols have a significant overhead over the network infrastructure, requiring to limit broadcast domains by dividing the network into smaller subnets for reducing the frequency and the span of such messages.

As all data to and from DOVE endpoints is intercepted by their hosting dSwitches, DOVE can deal with the problem of resolving unknown information in a unique, efficient and transparent to the endpoints way. The solution is to leverage DPS instead of emulating existing protocols and propagating their control messages to the entire virtual network.

Taking address resolution as an example, ARP request sent by an endpoint is intercepted by the hosting dSwitch that resolves it by sending a policy request to the DPS (if such information cannot be resolved by the local policy cache). Thus, while the endpoints keep using ARP, no ARP messages are propagated to the entire virtual network. Another example of the DPS based resolution mechanism is DHCP [12], where a broadcast message is used by the initiator to discover the set of DHCP servers. In DOVE, such broadcast requests (that can be identified according to the port numbers, or to the packet format) trigger a special policy request in which the DPS replies with a set of locations corresponded to the DHCP servers serving the policy domain (such information can be configured in the DPS in advance). In this case, the dSwitch may send multiple copies of the request, one for each DHCP server.

4.4.1 DOVE Address Resolution Protocol

As explained above, in order to resolve policy requests, the DPS needs to correlate between the virtual address and the physical location. Since host configuration is decoupled from the virtual network configuration and from the DOVE environment, such information may not be available to the DPS. However, dSwitches position on a data path allows them to intercept data packets, track virtual addresses of the hosted endpoints and send this information to the DPS. Nevertheless, there are cases, such as the presence of a silent host, in which this method will not work and the information may not be available to the DPS. In such cases, when the DPS needs to resolve the policy and the physical location based on unknown virtual address it performs DOVE Address Resolution sequence, in which a query containing the virtual address and the virtual network is sent to subset of the dSwitches (e.g. all the dSwitches serving endpoints from this virtual network with yet unknown virtual addresses). Each dSwitch that receives such request sends a set of ARP requests to all its endpoints that belong to the virtual network at hand, querying the virtual IP address. When the endpoint,

[11]For the sake of brevity, we omit details here and will provide them in the extended version of this paper.

owning this address, responds with an ARP reply, the dSwitch sends this information to the DPS. Note that once the DPS obtains endpoint's virtual IP address, it can retain it through subsequent migrations of the endpoint, so only endpoint creation or reconfiguration may trigger new resolutions. In order to avoid database inflation by virtual addresses, it is possible to limit the number of virtual addresses per endpoint (subject to virtual network SLA agreement). Moreover dSwitches may periodically initiate ARP requests to their endpoints and update the DPS with invalid addresses.

4.5 Multicast and Broadcast

As shown in the previous Section, DOVE leverages the DPS for eliminating broadcast and multicast messages required by control protocols. However, application level broadcast and multicast support might be required in order to support legacy applications, to provide enhanced services (e.g. IPTV), or to serve as a basis for other broadcast/multicast based control protocols. DOVE supports such cases by mapping application level broadcast domains and multicast groups into multicast groups in the physical infrastructure[12].

In a physical network, the broadcast domain is derived from the physical network configuration (e.g. an Ethernet based LAN is a broadcast domain that can be divided into several domains using VLAN) and determines the set of endpoints sharing a single broadcast infrastructure. In DOVE, on the other hand, this term is not associated with a physical constrains, and it can be interpreted differently according to the logical network description. Thus, while the entire virtual network can be considered to be a broadcast domain, one may refer to each policy domain as a broadcast domain. In this case, in order to provide broadcast service, one multicast group should be assigned in the physical infrastructure for each policy domain[13]. Upon intercepting a broadcast packet sent by an endpoint, the source dSwitch encapsulates it using the corresponding physical multicast address (received from the DPS as part of a policy resolution reply). As a result, the packet is forwarded to the entire multicast group, that is to all the dSwitches hosting endpoints from the target broadcast domain. It is the responsibility of dSwitches to join and leave relevant multicast groups (using multicast control protocols such as IGMP [6]) to correctly serve their hosted endpoints.

In order to support multicast the DPS should maintain a multicast table for each virtual network, with all the virtual multicast groups currently used and their members (i.e. endpoints joined the group). In addition, each group contains a physical multicast address to be used to disseminate data associated with this group. Thus, multicast based policy acquisition (triggered by a multicast packet sent by an endpoint) is similar to the unicast flow described above. The multicast table is updated by the dSwitch upon intercepting IGMP messages. When an endpoint joins multicast group, its hosting dSwitch updates the DPS that creates new entry or adds the endpoint to an existing entry.

Considering the policy enforcement discussed above, the encapsulated multicast packet may be subject to path control in which the multicast dissemination is done only after the packet passes through the entire set of required appliances (in IP source routing based path control, for example, only the last address contains the multicast address). In the extended version of this paper we explain how a single virtual multicast group can be mapped to multiple physical multicast groups in order to provide multicast service between endpoints located of different policy domains.

5. IMPLEMENTATION

We have designed, implemented and deployed fully functional DOVE system. Envisioning heterogeneous data center environments, we have dSwitch implementations on three different hypervisor platforms. Our DPS is implemented as Linux user space application where the logical networks information is stored in a set of in-memory hash tables. At this point, we did not have a chance to deploy DOVE in a large scale test bed. We are preparing a comprehensive simulation-based study of DOVE performance which will be covered in the extended version of this paper. Here we give an evaluation of DPS performance, observing that DPS is essential component in the DOVE architecture and has direct implication on the overall system performance [14].

While DPS can be implemented in a distributed fashion to provide high availability and resiliency, the actual way it is designed, implemented, and deployed is subject to many factors including the data center size, the characterization of the network applications including their connection distribution, the physical infrastructure, the service level to be provided, etc. We examine the DPS performance by evaluating the latency of the policy acquisition process on a single DPS server. To obtain these results, we have executed our Linux-based DPS server on a single core *Intel Xeon* CPU X5570 running at 2.93GHz with 8MB cache and DRAM size of 8GB. We have measured the policy acquisition latency (including the network and client and server latency) using a user space client executed on a similar

[12]In order to provide broadcast/multicast services the underlying infrastructure should support multicast.

[13]These groups can be assigned statically or dynamically and have to be maintained by the DPS.

[14]While DOVE performance is also affected by the overhead associated with the overlay infrastructure, this aspect was studied and examined in previous work dealing with evaluating tunneling and overlay overhead (see for example [19]).

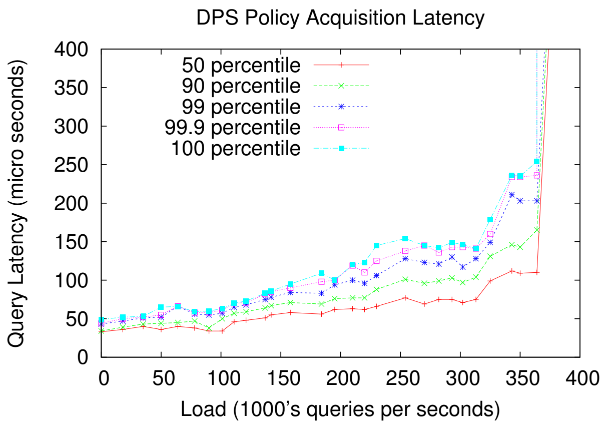


Figure 5: Policy acquisition latency in μseconds, including the wire delay, as a function of total load on a single DPS server.

server connected with 10G NIC [15]. Figure 5 depicts the latency in μseconds for different loads and different percentiles. It can be seen that DPS handles up to 380K requests per second, with an impressive latency of under 150μseconds even at the rate of 250,000 requests per second. One should take into account that with distributed DPS deployment strategies, it is possible to achieve the required latency defined by factors like the amount of dSwitches to be supported, the expected policy resolutions rate, as well as QoS and service level agreements with the customers.

6. CONCLUSIONS

DOVE architecture presented in this paper fully answers the set of requirements listed in Section 1. Being based on overlay technology, DOVE supports multiple collocated tenants so that data traffic of each tenant is fully isolated and recognizable. DOVE network abstraction is easy for the clients to grasp while allowing them to define rich set of network policies besides the simple connectivity. There is no architectural prohibition to support more services that we have described here, either natively by the system or by providing hooks for third party tools to add them. For example, host configuration services and QoS levels can be easily defined and we plan to add them to our DOVE implementation.

DOVE does not have any complex requirements from the underlying infrastructure apart from a simple connectivity between the dSwitches as well as between the dSwitches and the DPS. In DOVE, network state synchronization is achieved through careful separation of duties between the components, so that information is delivered strictly in "have-to-know" basis and is never flooded, making the system scalable and flexible.

DOVE finally achieves the proper network virtualization, whereby conceptual infrastructure agnostic application blueprints can easily be created by the application architects and turned, by the DOVE network virtualization layer, into concrete deployments on any type of infrastructure.

[15]Note that our implementation is very simple and many optimizations can be done including kernel based implementation, optimizing networking parameters such as interrupt coalescing, offloading, etc.

7. REFERENCES

[1] *IEEE 802.1q: VLAN*, 2005.

[2] Cisco vn-link: Virtualization-aware networking, 2009.

[3] IEEE 802.1qbg - edge virtual bridging, June 2011.

[4] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM*, volume 41, pages 242–253. ACM, October 2011.

[5] Theophilus Benson, Aditya Akella, Anees Shaikh, and Sambit Sahu. CloudNaaS: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011.

[6] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. RFC 3376: Internet group management protocol, version 3, October 2002.

[7] B. Carpenter and S. Brim. RFC 3234: Middleboxes: Taxonomy and issues, February 2002.

[8] Martín Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Natasha Gude, Nick McKeown, and Scott Shenker. Rethinking enterprise network control. *IEEE/ACM Transactions on Networking (TON)*, 17(4):1270–1283, 2009.

[9] Martín Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: taking control of the enterprise. In *ACM SIGCOMM*, pages 1–12. ACM, 2007.

[10] Martín Casado, Teemu Koponen, Rajiv Ramanathan, and Scott Shenker. Virtualizing the network forwarding plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, 2010.

[11] S. Deering and R. Hinden. RFC 2460: Internet Protocol, Version 6 (IPv6) specification, December 1998.

[12] R. Droms. RFC 2131: Dynamic host configuration protocol, March 1997.

[13] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.

[14] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim,

Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. *Commun. ACM*, 54:95–104, 2011.

[15] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 2008.

[16] Dilip A. Joseph, Arsalan Tavakoli, and Ion Stoica. A policy-aware switching layer for data centers. In *ACM SIGCOMM*, pages 51–62. ACM, 2008.

[17] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. SEATTLE: A scalable ethernet architecture for large enterprises. *ACM Trans. Comput. Syst.*, 29, 2011.

[18] Teemu Koponen, Martín Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Others. Onix: A distributed control platform for large-scale production networks. 2010.

[19] Alex Landau, David Hadas, and Muli Ben-Yehuda. Plugging the hypervisor abstraction leaks caused by virtual networking. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, pages 16:1–16:9. ACM, 2010.

[20] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[21] M.Mahalingam, D.Dutt, K.Duda, P.Agarwal, L.Kreeger, T.Sridharand M.Bursell, and C.Wright. Vxlan: A framework for overlaying virtualized layer 2 networks over layer 3 networks, August 2011.

[22] Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C. Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *USENIX conference on Networked systems design and implementation*, pages 18–18. USENIX Association, 2010.

[23] Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. NetLord: a scalable multi-tenant network architecture for virtualized datacenters. In *ACM SIGCOMM*, pages 62–73. ACM, 2011.

[24] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM*, SIGCOMM '09, pages 39–50. ACM, 2009.

[25] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. RFC 4861: Neighbor discovery for ip version 6 (ipv6), September 2007.

[26] T. Narten and M. Sridharan. Problem statement: Using l3 overlays for network virtualization, 2011.

[27] Ping Pan and Thomas Nadeau. Software-defined network (sdn) problem statement and use cases for data center applications, 2011.

[28] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *HotNets*, 2009.

[29] David C. Plummer. RFC 826: An ethernet address resolution protocol, November 1982.

[30] Lucian Popa, Minlan Yu, Steven Y. Ko, Sylvia Ratnasamy, and Ion Stoica. CloudPolice: taking access control out of the network. In *HotNets*. ACM, 2010.

[31] J. Postel. RFC 791: Internet Protocol, September 1981.

[32] E. Rosen, A. Viswanathan, and Callon R. RFC 3031: Multiprotocol label switching architecture, January 2001.

[33] Alan Shieh, Srikanth Kandula, Albert Greenberg, and Changhoon Kim. Seawall: performance isolation for cloud datacenter networks. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 1–1. USENIX Association, 2010.

[34] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 23–23. USENIX Association, 2011.

[35] Xin Sun, Yu-Wei Sung, Sunil Krothapalli, and Sanjay G. Rao. A systematic approach for evolving vlan designs. In *IEEE INFOCOM*, pages 1451–1459. IEEE, 2010.

[36] M. Yu, J. Rexford, X. Sun, S. Rao, and N. Feamster. A survey of virtual LAN usage in campus networks. *IEEE Communications Magazine*, 49(7):98–103, July 2011.