# An Intent-based Approach for Network Virtualization

Rami Cohen
and Katherine Barabash
and Benny Rochwerger
and Liran Schour
IBM Research Lab
Haifa, Israel

Daniel Crisan
and Robert Birke
and Cyriel Minkenberg
and Mitchell Gusat
IBM Research
Zurich, Switzerland

Renato Recio
and Vinit Jain
IBM
Austin, USA

*Abstract*—

**Virtualizing resources for easy pooling and accounting, as well as for rapid provisioning and release, is essential for the effective management of modern data centers. Although the compute and storage resources can be virtualized quite effectively, a comprehensive solution for network virtualization has yet to be developed. Our analysis of the requirements for a comprehensive network virtualization solution identified two complimentary steps of ultimate importance. One is specifying the network-related requirements, another is carrying out the requirements of multiple independent tenants in an efficient and scalable manner. We introduce a novel intent-based modeling abstraction for specifying the network as a policy governed service and present an efficient network virtualization architecture, Distributed Overlay Virtual Ethernet network (DOVE), realizing the proposed abstraction. We describe the working prototype of DOVE architecture and report the results of the extensive simulation-based performance study, demonstrating the scalability and the efficiency of the solution.**

## I. Introduction

Apart from becoming increasingly more complex, network management and configuration have not changed conceptually in the last few decades and involve dealing with control protocols, addresses, port properties, etc. In many cases, all these low level details are inherited from the underlying transport technologies and do not directly reflect the network functionality the administrator has endeavoured to achieve. For instance, in order to enforce a security policy with a firewall appliance, one is required not only to configure the filtering rules in the appliance, but also to consider multiple control aspects, such as routing rules, flooding overheads, etc. It is also required to arrange for path isolation, e.g. using VLANs, so that traffic requiring inspection is forced to pass through the appliance. Moreover, when the desired configuration is finally achieved, it often depends on addresses and locations of endpoints[1], so any endpoint lifecycle event, like addition, removal, or reconfuguration, requires updating the network, sometimes in more than one point of management. As it is hard to derive the intent with which the setup was created in the first place from the observable set of low level configuration details, making changes is cumbersome and risky and can cause severe and costly service outages, see [9], [7].

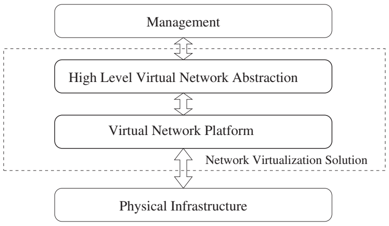

Management
High Level Virtual Network Abstraction
Virtual Network Platform
Network Virtualization Solution
Physical Infrastructure

Fig. 1. Network Virtualization Reference Architecture.

The growing demand to provide multi tenant network services in consolidated environments, so that each tenant can specify, deploy, and control its own virtual network, presents a golden opportunity to revise the network management state of the art. Instead of focusing on the infrastructure configuration and control, network management must become declarative and allow expressing the required network functionality. A comprehensive network virtualization solution must consist of two necessary components shown in Figure 1. One is a powerful infrastructure-independent abstraction for specifying the network functionality, while the other is an efficient and scalable virtualization platform translating abstract specifications into concrete infrastructure configuration primitives, thus enforcing the specified behaviour over a specific physical infrastructure. Network virtualization platform must host multiple, independent, and isolated virtual networks belonging to multiple tenants, so that tenants can define and manage their networks independently of other tenants and of the physical infrastructure characteristics like addresses, topology, policies, etc. Similarly to other virtualization technologies, the virtual layer must be decoupled from the infrastructure[2]. Clearly, the proposed solution must be efficient, scalable, and highly available, and take into account the dynamic nature of virtualized data centers where endpoints are dynamically created, deleted, and migrated.

[1] The term endpoint is used to denote network client that can either be a physical computer or a virtual machine.

[2] In networking, this means, for example, allowing the virtual networks to use different technologies and topologies than those deployed underneath.

978-3-901882-50-0 ©2013 IFIP

### A. Related Work

Network virtualization is a large research and technological field comprehensively surveyed in [6]. Survey authors state that although the network virtualization techniques are abundant, in most of the existing solutions the operational and the management aspects are either untouched or require further attention. Since the survey was published, quite a few new network virtualization technologies and architectures were proposed [12], [13], [5], [14]. Most of the proposed solutions focus on functional requirements for the network virtualization platform while delivering the traditional network management and configuration experience at the virtual level. For example, it is quite common for a virtual network to mimic or even to completely emulate an L2 broadcast domain or an L3 subnet. As a result, the complexity and the fragility of traditional physical network configuration are often copied into a virtual level. In addition, existing network virtualization solutions fail to answer the complete set of the network virtualization platform requirements. For example, VXLAN [12] heavily depends on physical infrastructure by requiring multicast support, while both VXLAN and NetLord [14] rely on data plane learning for location and address dissemination inheriting well-known L2 flooding and stabilization upon change limitations. Emerging SDN-based approaches [5], while promising, do not support advanced network services (policies) as an integral part of the network virtualization solution. In traditional SDN solutions, these services are superimposed later as 'applications' instead.

### B. Contributions

This work shows that it is both necessary and possible for a management abstraction at the virtual level to be conceptually different than that of a physical level, through two major contributions. The first is a novel *intent-based* network management abstraction, offering a basis for a paradigm shift in the way connectivity services are specified and consumed. The term 'intent-based' refers to the property whereby the management abstraction relates to the functionality of the network, allowing to express, formalize, and verify it. As will be shown in Section II, the proposed abstraction captures the network functionality as a blueprint that can be verified and approved prior to the deployement, and deployed prior to instantiating any of the endpoints. The second contribution is a network virtualization architecture created to carry out abstract network specifications down to the level of actual traffic delivery, policing, and control. The architecture is called Distributed Overlay Virtual Ethernet network (DOVE) and it builds upon two major principles—edge-terminated overlays and centralized control plane. Combining the intent-based network functionality abstraction with the DOVE architecture, we create a comprehensive network virtualization solution whereby the physical and the virtual infrastructure layers are loosely coupled and operationally independent. With this solution, any changes in abstract service definition layer do not require reconfiguration in the underlying infrastructure, while any transient failures in the infrastructure layer are transparent to the consumers of the virtual network services. The physical network infrastructure can remain relatively static while virtual networks transparently adapt as required to support large amounts of highly dynamic virtualized workloads belonging to multiple tenants.

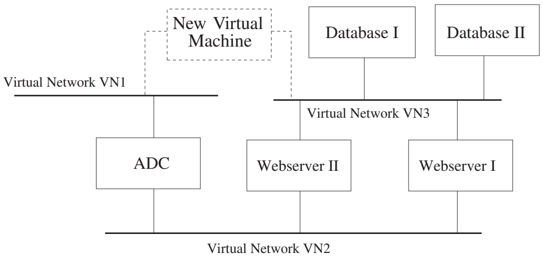


Fig. 2. Traditional network representation.

The rest of the paper is organized as follows. Section II introduces the novel intent-based virtual network abstraction, Section III gives an overview of the DOVE architecture, Section IV presents the working DOVE prototype for the open systems environment, and Section V reports the simulation-based performance evaluation results. The paper is concluded with a brief summary and future directions in Section VI.

## II. Intent-based Virtual Network Abstraction

Let us consider an example of a typical three-tier application comprising a set of web application servers connected to a public Internet through an Application Delivery Controller (ADC) and using a set of database servers as a back-end data store. Application specific policies[3] are usually associated with the network design for such applications. Let's say that in our example, all the traffic between the Internet and the ADC must pass through a set of firewall rules, and all the SSL traffic between the web servers and the ADC must be accelerated using an SSL accelerator. To enforce such policies, there is a need to configure the physical network infrastructure, the endpoints, and the appliances. Most of the required configuration steps are technology and vendor specific and are related to low level network control and not to its functionality. As a result, the configuration depends on the concrete physical network topology and technology and on the installed management tools and processes. Moreover, the connectivity is directly affected by the configuration of endpoints, in particular their network interfaces and addresses, making it sensitive to endpoint lifecycle events. Last, but not the least, the configuration always depends on the subjective interpretation and the skills set of the administrator in charge.

Using the abstraction, exposed by the existing network virtualization technologies, such as VXLAN, and by the virtualization management tools, such as OpenStack [1], each endpoint can be connected to one or more predefined Virtual Networks (VNs), each emulating either an L2 broadcast domain or an L3 subnet. One possible way to express the network design for the example above is presented in Figure 2. In this typical design, the application needs three VNs, ADC must

[3] Policies specify the network services beyond the simple connectivity and may include security, QoS, monitoring, etc

be connected with two virtual interfaces to $VN1$ and $VN2$, all the web servers must have interfaces connected to $VN2$ and $VN3$, and the databases must be connected to $VN3$. For policy enforcement, virtual interfaces are usually connected to the predefined port profiles that are in turn mapped to configuration rules, e.g. in virtual appliances deployed near the endpoint. Although the process is seemingly independent from the underlying topology and eliminates most of the physical network configuration, the abstraction behind it ties up the network functionality definition with the endpoints instantiation and thus is severely handicapped. First, the intended network functionality cannot be defined just by creating a set of VNs and a set of port profiles with their properties, because each newly instantiated endpoint must be correctly configured for the network[4]. Second, any endpoint misconfiguration can violate the policy criteria associated with other endpoints, as in Figure 2 where new VM can be configured to route between $VN1$ and $VN3$. Third, routing considerations, although they are part of the network control and do not relate to the functionality, can affect application deployement, for example when connectivity beyond a single data center is needed. In addition, due to lack of formal means for defining the network, the same functionality can be achieved in different ways, leading to ambiguity. While for simple networks these limitations can be overcame with careful bookkeeping and double checking, any of them can make it close to impossible to define, validate and deploy complex and dynamic networks.

So what is the good network abstraction? As shown above, it must capture the functionality of the network, best described in terms of the **connectivity** between endpoints and the **policies** associated with the connectivity, while leaving out the network control specifics. One possibility is to assign policies to pairs of communicating endpoints. Seemingly viable, this approach has drawbacks, e.g. the complexity explosion required to manage a pairwise mesh of endpoints. Far more important, this approach fails to decouple the network definition from the endpoints' instantiation, exactly as a traditional one presented above. Network abstractions with this property necessarily fail to separate the relatively static application lifecycle aspects, such as deployement, redesign, and end of service, from highly dynamic aspects, such as addition, removal, and migration of components. The combined static aspects, in what follows referred to as the *network blueprint*, fully and unambiguously specify the network functionality and reflect the intention behind the network design. Keeping the static and the dynamic aspects separate allows networks to grow and shrink, be redeployed over different infrastructure or upgraded to a new technology without affecting its functionality.

We introduce the term *policy domain* to denote an entity aggregating endpoints with a common policy criteria, so that policies are specified for pairs of policy domains. Policies are unidirectional and it is possible to define a default policy as well as a policy from a policy domain to itself. Network blueprints are represented as directed graphs with vertexes standing for policy domains and directed edges—for policies, as in Figure 3, where the blueprint for the network traditionally deployed as in Figure 2 is shown. When new endpoint is created, it is associated with a single policy domain, whereby all the domain's policies are applied to it[5]. Network blueprints approach decouples the network management from the endpoints management and opens up opportunities for creating formal validation schemes ensuring correctness prior to deployement or migration to a new physical infrastructure.

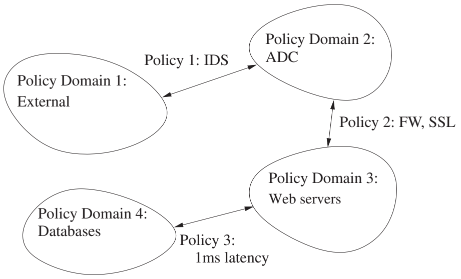


Fig. 3. Network blueprint example

## III. *DOVE* Architecture

*DOVE* (Distributed Overlay Virtual Ethernet network) is the network virtualization architecture, created for providing multi tenant network services in consolidated environments. DOVE carries out the network functionality, specified in a form of blueprints described in the previous section, into the underlying physical infrastructure. The detailed description of DOVE, complete with special cases and enhanced features treatment (e.g. External connectivity and NAT, multicast services, traffic engineering, DHCP, etc.), can be found in [19]. This section outlines the major architectural concepts DOVE builds upon, shows how DOVE provides policy based connectivity and isolation, and how it overcomes the limitations of existing network virtualization architectures discussed in Section I.

Employing overlays for interconnecting virtual machines, where the data sent by a virtual machine is encapsulated using IP based tunneling protocol was recently advocated in [3], [14], [16], [12]. Some of the benefits of this usage of overlays are: First, physical switches do not longer need to deal with a large and dynamic set of virtual network endpoints, but only with a much smaller set of static physical servers, so they need to support less addresses and less configuration and control protocols; Second, with overlay, virtual network endpoints are isolated from the physical infrastructure, enabling virtual network to be created over different physical network technologies (e.g. Ethernet, Infiniband, IPv4, IPv6), and topologies (e.g. multiple networks and subnets); Third, overlay achieves full isolation between different virtual networks, enabling each virtual network to define its own network characteristics independently, including its topology

[4] In the above example, all the ADC servers must have two interfaces connected to $VN1$ and $VN2$, otherwise, the three tier application connectivity will be broken

[5] It must be assured by the virtualization platform that all the communications between two endpoints follow the policy determined between the policy domains containing these endpoints.

and address scheme (e.g. IPv4 and IPv6). As a result, several virtual networks can share the same address space in the virtual domain and different virtual networks can coexist on a shared physical infrastructure. We refer to addresses defined in virtual networks as to *virtual addresses* and to addresses used in the physical infrastructure as to *physical addresses*.

Instead of reproducing the complexity of the traditional Ethernet learning mechanisms, based on flooding (STP) and broadcasting (ARP, NDP), DOVE ensures connectivity through employing a centralized controller, used, among other things, for address and policy dissemination. This approach to network forwarding control has received lots of attention after the introduction of Software Defined Networking (SDN) paradigm and the invent of the OpenFlow for communicating the control information between the centralized network controller and the forwarding devices [10]. In DOVE, a similar approach was employed for creating multi-tenant virtual networks interconnecting Virtual Machines (VMs). The major difference is that the SDN paradigm was not conceived for virtualizing the network, but has adopted it as one possible, although prominent, use-case. DOVE architecture, on the contrary, was designed specifically for network virtualization and natively includes the rich network services support and the intent-based modeling abstraction. In DOVE, the controller is responsible not only for configuring flow tables in forwarding devices, but also for the policy enforcement and for maintaining a mapping between the virtual network specifications and the instructions controlling the physical infrastructure.

With DOVE, network virtualization is achieved with two main entities, DOVE Switches (dSWitches) and DOVE Policy Controller (DPC), described below:

*dSwitch* is a data plane component serving a (dynamic) set of endpoints and acting as an overlay edge, responsible for enforcing the connectivity and policy requirements between endpoints as specified through a management abstraction at a virtual level. A dSwitch must reside in a data path of every virtual network endpoint[6] and must support all the packet handling functionality required for policy enforcement, e.g. traffic shaping, data path control, QoS primitives handling, etc. dSwitch intercepts all outgoing and incoming data and performs overlay encapsulation and decapsulation according to instructions it obtains from the *DPC* and caches locally.

*DPC* maintains virtual network blueprints, described in Section II, and provides management interfaces for their creation, modification, and deletion. In addition, *DPC* 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, *DPC* resolves and validates policy requests received from *dSwitches* and maps these requests to packet sending instructions reflecting the connectivity and the policy criteria. *DPC* 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. To meet the above mentioned non-functional requirements, DOVE architecture relies on a clustered DPC solution, allowing no single point of failure in the DOVE environment. For the DPC cluster design and the distributed state management considerations, please refer to [19].

```
$pkt \leftarrow get\_packet(vm\_if_i)$
$doveContext \leftarrow get\_dove\_ids(vm\_if_i)$
$type \leftarrow get\_type(pkt)$
$vSrcIP \leftarrow get\_src\_IP(pkt)$
$vDestIP \leftarrow get\_dest\_IP(pkt)$
$policy \leftarrow cache\_lookup(doveContext, vSrcIP, vDestIP)$
if policy is NULL then
   $send\_policy\_request(doveContext, vSrcIP, vDestIP)$
   $policy \leftarrow get\_policy\_reply()$
end if
if policy is NULL then
   $drop\_packet(pkt)$
end if
if type is ARP then
   $vDestMac \leftarrow get\_dest\_MAC(policy)$
   $create\_and\_send\_ARP\_reply(vm\_if_i, vDestMac)$
else if type is IP then
   $pDestIP \leftarrow get\_dest\_pIP(policy)$
   if destination is hosted locally then
      $dest\_port \leftarrow get\_vm\_if(doveContext, vDestIP)$
      $deliver\_packet\_locally(dest\_port, pkt)$
   else
      $encPkt \leftarrow encapsulate(pkt, policy)$
      $send\_to\_network(encPkt)$
   end if
else
   $drop\_packet(pkt)$
end if
```

Fig. 4. The *dSwitch* algorithm for VM generated packets processing.

```
$encPkt \leftarrow get\_packet()$
$pSrcIP \leftarrow get\_source\_ip(encPkt)$
$dHeader \leftarrow get\_dove\_header(encPkt)$
$pkt \leftarrow remove\_header(encPkt)$
$doveContext \leftarrow get\_dove\_ids(dHeader)$
$vDestIP \leftarrow get\_dest\_IP(dHeader)$
if destination is hosted locally then
   $dest\_port \leftarrow get\_vm\_if(doveContext, vDestIP)$
   $deliver\_packet\_locally(dest\_port, pkt)$
else
   $notify\_sender(pSrcIP)$
end if
```

Fig. 5. The *dSwitch* algorithm for processing packets received from a physical network.

DOVE data plane processing algorithms are presented in Figures 4 and 5. Upon intercepting an endpoint generated data packet, the hosting *dSwitch* must acquire the associated sending instructions, also called *DOVE policy*, either from the local cache or from the *DPC*, as will be described later. To acquire the correct policy for a specific data packet, *dSwitch* must match the packet into the context of a specific *Policy Domain*, as well as to extract address information from the packet itself. Apart from data required for encapsulation, *DOVE policy* may include traffic shaping and traffic engineering information for QoS based policies, path control information for ensuring

[6] In deployments where endpoints are VMs, dSwitch replaces a hypervisor virtual switch and connects VMs hosted by the server to a DOVE network.

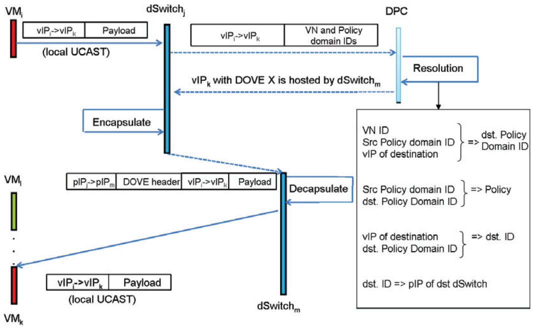


Fig. 6. Policy request-response cycle and policy resolution by DPC.

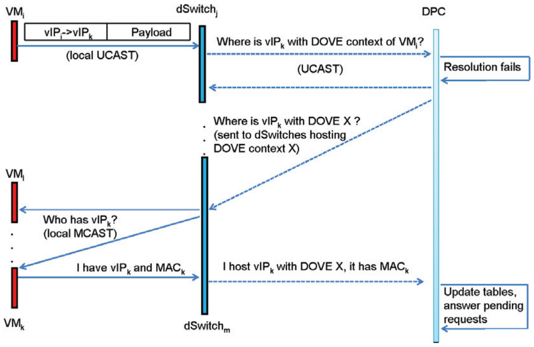

$VM_i$
$dSwitch_j$
DPC
$vIP_i$->$vIP_k$ | Payload
(local UCAST)
Where is $vIP_k$ with DOVE context of $VM_i$?
(UCAST)
Resolution fails
Where is $vIP_k$ with DOVE X ?
(sent to dSwitches hosting
DOVE context X)
$VM_l$
Who has $vIP_k$?
(local MCAST)
I have $vIP_k$ and $MAC_k$
I host $vIP_k$ with DOVE X, it has $MAC_k$
Update tables, answer pending requests
$VM_k$
$dSwitch_m$

Fig. 7. DOVE Address Resolution Protocol round involves the controller, a set of dSwitches, and a set of endpoints hosted by these switches.

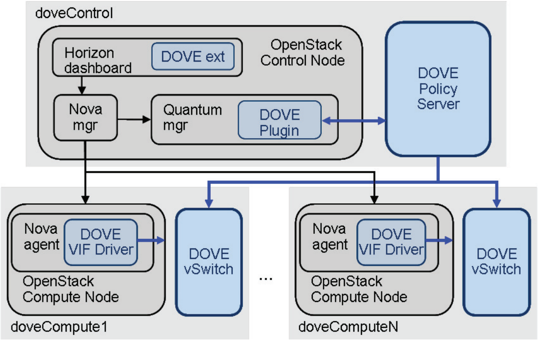


Fig. 8. DOVE virtual network environment prototype integrated into the OpenStack cloud management software.

that the packet will pass through a set of appliances, ACLs for security based policies, etc. *dSwitch* must be capable of carrying out all the instructions, either by itself, or by using the preconfigured support in the underlying infrastructure. Once the policy is acquired, *dSwitch* encapsulates the packet so that the outer headers contain the physical addresses of the source and the destination *dSwitch*. Any, preferably standard, encapsulation header can be used, for example, VXLAN, STT, GRE, as soon as they are capable of carrying the virtual network identifier, in our case, the DOVE Policy Domain identifier. Destination *dSwitch* decapsulates the packet, matches it into the context of specific *Policy Domain* using the encapsulation header, and delivers it to the destination endpoint. Upon VM migration, the destination *dSwitch* informs the source *dSwitch*, either directly or though the controller, that the destination endpoint is no longer served by it and the source *dSwitch* must acquire the new location information from the *DPC*.

Beyond the simple connectivity, DOVE is architectured to natively support a rich set of policies, subject to the physical infrastructure capabilities. For example, to support QoS in the infrastructure with QoS enabled forwarding devices, *dSwitch* may perform traffic shaping and set the ToS field in the header. Similarly, supporting security policies may require deploying and configuring physical or virtual network appliances, e.g. firewalls, intrusion detection systems, etc. The supported policies set necessarily depends on the specific physical infrastructure, appliances deployment, and configuration.

As mentioned above, policy resolution requests sent by *dSwitches*, contain the virtual addresses extracted from data packets as well as the Policy Domain identifiers, required to correctly identify the source and the destination endpoints. Upon receiving a policy resolution request from a *dSwitch*, the *DPC* performs a simple series of lookups in its internal database. First, the Policy Domain identifier is used to lookup the containing Virtual Network identifier. Next, the Virtual Network identifier together with the virtual L3 address of destination endpoint are used to lookup the destination Policy Domain identifier. Then, the pair of Policy Domain identifiers is used to look-up the policy. In addition (as well as in parallel), virtual L3 address of destination endpoint together with the destination Policy Domain identifier are used to lookup the physical L3 address of *dSwitch* hosting the destination endpoint, and, optionally, some other information regarding the destination, e.g. its virtual MAC address in a case L2 services must be provided in the virtual domain. With all the required information at hand, the *DPC* sends policy resolution reply message to the requesting *dSwitch*, to be cached and used to handle packets belonging to the same flow as the one that triggered the request. Figure 6 presents both the flow sequence and the policy resolution algorithm run in DPC in order to resolve the request.

To correctly answer policy resolution requests, the *DPC* needs to maintain a mapping between the virtual addresses and the physical locations of endpoints. In addition, to provide L2 services to endpoints, a mapping between virtual L2 and L3 addresses is required. Required address and location data can be obtained by the *DPC* from two sources: first, from the management, if, for example, virtualization orchestrator communicates VM lifecycle events to the *DPC*; and second, from the *dSwitches* that are positioned in a data path of DOVE endpoints and thus can learn addresses locally and report them to the *DPC*. For a corner case where there is a need to obtain whereabouts of a silent DOVE endpoint through *dSwitches*, *DPC* initiates a *DOVE Address Resolution Protocol* round as shown in Figure 7. Note that once *DPC* has performed DOVE ARP round for a certain endpoint, this endpoint's information is retained through subsequent migrations, so this (costly) resolution can be required at most once in endpoint's lifecycle.

## IV. Implementing the DOVE Architecture

DOVE architecture prototype for open systems is integrated with OpenStack cloud management software [1], to achieve a fully functional management plane, where tenants can specify abstract network blueprints for their workloads and to subsequently deploy workload components into these blueprints to achieve the desired communication behaviors.

The prototype setup is presented in Figure 8 and comprises the *doveControl* box and several *doveCompute* boxes, each running as a bare-metal application on a separate virtualization-enabled x86 computer. All the computers are running Ubuntu 12 and have OpenStack components deployed and configured: the *doveControl* box runs the OpenStack control services and each *doveCompute* box runs an instance of the OpenStack compute node. The OpenStack components are extended to accommodate DOVE integration: Horizon dashboard is extended to expose the available virtual network blueprints in a context of VM deployment, DOVE Quantum plugin—to communicate network management information to the DPC, and the DOVE virtual interface driver (VIF)—to communicate network management information to dSwitches. In this simplified setup, a single DPC engine is solely responsible for all the control plane functionality. DPC is implemented as a Linux user space application and it stores all the logical network information in a set of in-memory hash tables. The DPC is capable of controlling multiple dSwitches in a small scale OpenStack environment where clients are running simple isolated LAMP applications. dSwitch implementation is based on Open vSwitch (OVS) [18], augmented with DOVE overlay capabilities. To communicate DOVE control information to the OVS datapath, we have extended the most recent OpenFlow (OF) protocol features, namely, extensible match, tunneling, and flow entry metadata capabilities. DOVE control protocol, although can be carried by the OF as in the prototype featured here, is inherently a higher level protocol and incorporates more than just forwarding control in overlay endpoints. DOVE control protocol is designed to support the intent-based network management and to specify more than just simple connectivity[7].

## V. Modeling and Evaluating the DOVE Architecture

At the core of our simulation environment lies Venus [11], an Omnet++ [20] based network simulator. Venus operates at flit-level, accurately modeling the switch and network adapter micro-architecture including, but not limited to, queuing, buffering and scheduling. DOVE is mapped on top of a common datacenter physical network infrastructure.

For the first quantitative evaluation of DOVE we investigate the two dimensional elastic scalability and performance isolation. We measure the performance of a variable number of independent and identical tenants, each running a 3-tier workload, with varying loads. We model an anti-collocation random placement policy typically employed for highly reliable applications. This is a worst case VM placement scenario from DOVE's perspective with respect to the generated traffic patterns, as it provides no benefits from VM collocation, clustering affinity and internal communications.

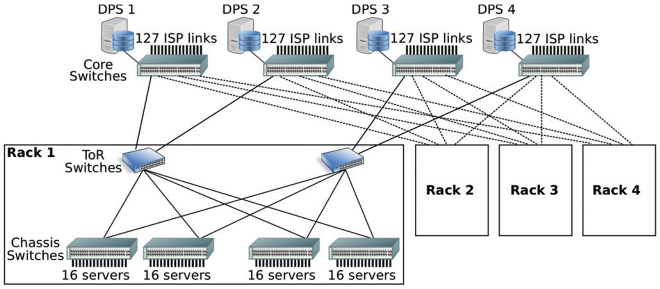


Fig. 9. Datacenter network topology with three layers of switches forming an Extended Generalized Fat Tree [17] *XGFT(3;16,4,4;1,2,2)*. The top ISP links are used by the external clients to inject HTTP queries that are served by the tenants' VMs. The bottom 256 servers, grouped in 4 racks each rack containing 4 chassis, are virtualized holding up to 16 VMs each. The DPC is distributed and attached to the aggregation switches.

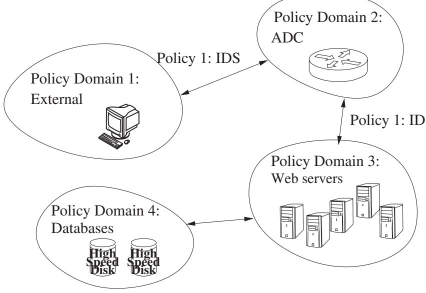


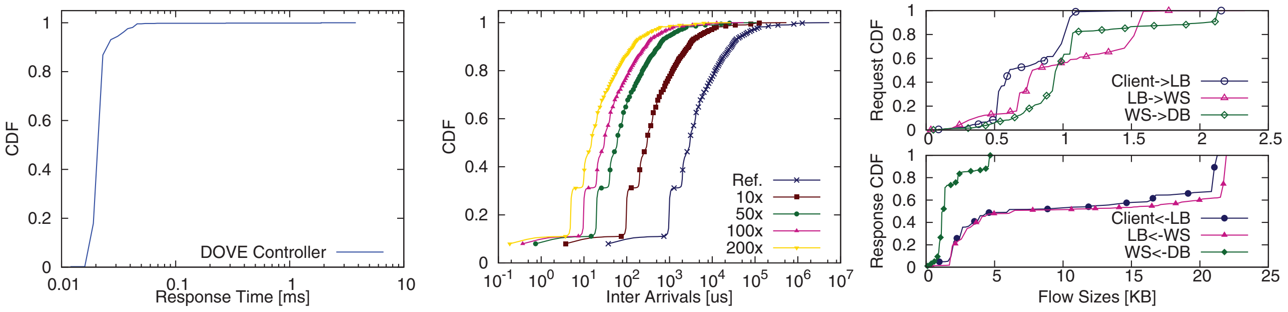
Fig. 10. Each tenant deploys a 3-tier workload consisting of 8 VMs: 1 load balancer, 5 web servers and 2 database servers.

To characterize DOVE's performance scalability we propose three key figures of merits: 1) DOVE *aggregate throughput*, expressed in transactions per second, as a datacenter operator metric; 2) DOVE *query completion times*, as a primary performance metric for tenants; 3) DOVE *packet loss ratios*, as a metric for the network service quality.

### A. Models: DCN, DOVE, Protocol Stack, and Workload

*1) DCN:* The modeled infrastructure fabric is based on 10G commodity Ethernet. To avoid the low order head-of-line blocking, the physical network adapters use one Virtual Output Queue for each destination. The switches have an input-buffered output-queued architecture: incoming frames are stored in the input buffer corresponding to the reception port, and enqueued in parallel at the corresponding transmission port. We abstract the internal complexity of a modern switch design by assuming an $N$-fold ideal speedup and a full buffer sharing. The network topology is an Extended Generalized Fat-Tree (XGFT) [17] with three layers of physical switches as shown in Figure 9, representative for current datacenter network deployments [2], [15], [8].

[7] Enforcing advanced network services may require controlling more devices than OF possibly can control, e.g. physical and virtual appliances.



(a) DOVE Controller response times.

(b) Queries inter-arrivals for one tenant.

(c) Flow size distributions.

Fig. 11. Size and delay distributions measured using real life applications running on physical machines.

TABLE I
MODEL PARAMETERS

| Parameter | Value | Unit | Parameter | Value | Unit |
|---|---|---|---|---|---|
| ***Network hardware*** | | | | | |
| link speed | 10 | Gb/s | adapter delay | 500 | ns |
| frame size | 1518 | B | switch buffer size/port | 100 | KB |
| adapter buffer size | 512 | KB | switch delay | 500 | ns |
| ***DOVE Overlay*** | | | | | |
| request size | 86 | B | encap. overhead | 58 | B |
| reply size | 86 | B | request RTO | 10 | ms |
| delay | 20 | $\mu$s | | | |
| ***TCP*** | | | | | |
| buffer size | 128 | KB | TX delay | 9.5 | $\mu$s |
| max buffer size | 256 | KB | RX delay | 24 | $\mu$s |
| timer quanta | 1 | $\mu$s | reassembly queue | 200 | seg. |
| base RTO | 20 | ms | RTO slop | 20 | ms |
| min RTO | 2 | ms | congestion control | *NewReno* | |

*2) DOVE Architecture:* To model the DOVE data plane, we increase packet size by 58B to stay for encapsulation: 22B outer Ethernet header + 20B outer IP header + 8B UDP header + 8B VXLAN header. To avoid fragmentation, we accordingly decrease the MTU value on the endpoints from 1518B to 1460B. The DCP query response time distribution, measured in [4], is shown in Figure 11(a).

*3) Protocol Stack:* For the TCP and UDP transports, we extended Venus with a model of TCP that allows evaluating the performance of socket-based applications. To be as close as possible to reality, we ported the TCP stack from a current FreeBSD v9 kernel into Venus, adding only a fixed delay to each segment instrumented from real hardware.

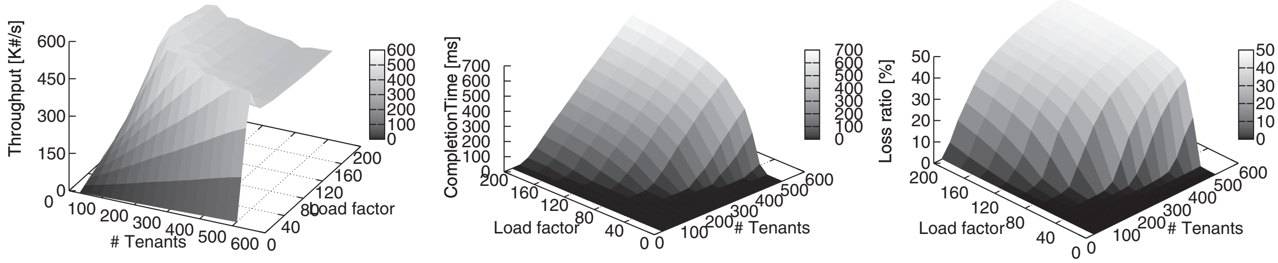
*4) 3-Tier Workload Model:* Each tenant of the DOVE datacenter runs a typical 3-tier workload described in Section II, with 8 VMs deployed and assigned to the policy domains as shown in Figure 10. The 8 VMs of a workload are randomly placed across the physical servers, subject to the cold spot load balancing, with a limit of up to 16 VMs per host. Each tenant receives external HTTP queries through a dedicated ISP link, so that bottlenecks on the ISP links do not affect the experiments. The size of the requests and the replies transmitted between the different tiers are drawn from the distributions shown in Figure 11(c) and were obtained by instrumenting a 3-tier workload based on RUBiS v1.4.3. Only 1 in 3.57 HTTP queries required dynamic content obtainable by the means of a cascading SQL query.

The system is installed on 4 physical machines: one for each tier, plus one for RUBiS emulating 80 external clients. The size and the inter-arrival times of the flows generated between tiers were measured. To model heavier workloads, with more external clients, we scaled the inter-arrivals distribution by a load factor ranging from 1x to 200x as shown in Figure 11(b). Table I summarizes the framework parameters chosen to match the current generation of server CPUs, as well as the delays reported in modern data centers. The base RTO was chosen to be larger than the worst case RTT of virtualized network – less than 10 ms – to which we add the RTO of the DOVE policy resolution – another 10 ms. The RTO slop was set to 20 ms, also matching the current server processors.

### B. Results and Discussion

Figure 12 presents the results of our 2D investigation across two variables. The first variable the *consolidation factor* (CF) represented by the number of tenants, varying from 32 to 512 in increments of 32. Each tenant deploys a single 3T workload containing 8 VMs as described above. Since all tenants share the same physical network, the aggregated load increases with the number of tenants. The second variable parameter is the *load factor* (LF) that controls the inter arrival time between the two consecutive requests of the same tenant's workload. As shown in Figure 11(b), we start from the reference load factor, LF=1, and progressively increase it up to 200x, by increasing the number of external users per tenant. In all the experiments, each tenant served 1000 HTTP queries before the simulation was stopped and statistics were gathered. For each request, the completion time, accounting for all the delays induced by the network, was measured. To abstract the end node processing variability and the VM scheduling side-effects, we assume here that all the servers have infinite processing power and are able to serve a reply instantaneously. While in future experiments we shall remove this arguably limiting abstraction, here we focus on DOVE's scalability and the virtualized datacenter networking performance.

*1) Aggregate Throughput:* Figure 12(a) presents the DOVE throughout, $Tput$, computed by dividing the total number of HTTP requests by the time needed to serve them. To be more robust to outliers, we use the 99th percentile. Increasing the number of tenants is beneficial to the aggregate throughput that increases steadily until a peak is reached at around 500K



(a) Aggregate throughput.

(b) Median query completion time.

(c) Packet loss ratios.

Fig. 12. DOVE virtualized datacenter performance. Elastic Scalability: aggregate throughput, median completion time and packet loss ratios.

requests/s. The peak is reached earlier when the inter-arrivals are short corresponding to a high per tenant load.

*2) Query Completion Time:* Figure 12(b) presents the query completion time, $T_c$, as tenant's primary performance metric. Given the non-gaussian delay distributions collected from our measurements, the *median* query completion time is more rigorously descriptive than the mean - albeit we thus miss the benefits of standard deviation and central limit theorem.

*3) Packet Loss Ratios:* DOVE packet loss ratios, plotted in Figure 12(c), are required for completeness because the modeled commodity 10GigE infrastructure does not employ link level flow control (i.e. Priority Flow Control), and are obtained by dividing the total number of bytes sent by all VMs to the total number of bytes dropped by all the switches.

After the throughput peak is reached, the network saturates, as well as the queues' occupancies at the physical switches. The percentage of losses grows accordingly, leading to longer execution times. Whereas the number of tenants grows linearly, the completion times grow with higher slopes, and the aggregate throughput decreases, indicating the overload. The 3T flows contain up to 10-15 segments: far too short for the TCP congestion control loop to react properly. For low to medium load factors ($LF < 80 - 100x$), the measured throughput monotonically increases with a higher slope than the query completion time (delay). As can be seen in all 3 metrics above, the saturation peak is variable with both the consolidation and load factors. In the linear region below the overload peak, increasing the consolidation factor (adding tenants) does not influence the general completion times past the set threshold. Thus, in the linear region of DOVE datacenter operation, each new tenant positively contributes to the DOVE aggregate throughput – while diminishing it (and increasing the latency) beyond the saturation with a relatively smooth roll-off. At higher load factors ($LF > 120x$), the new tenants saturate the network earlier, hence DOVE's global efficiency in this scenario peaks around 320 tenants before decreasing. The extended linear region of performance isolation, where workloads of different tenants do not affect each other, demonstrates DOVE elastic scalability across a wide dynamic range of consolidation and load factors.

## VI. Conclusion

In this work we have presented a complete network virtualization solution. We have proposed a novel intent-based virtual network abstraction whereby network blueprints are created for deterministic and verifiable specification of network functionality. Network blueprints allow managing the complete application lifecycle independently of managing the endpoints lifecycle. We believe our approach to be a major breakthrough in achieving the separation of concerns between the network and the virtualization administration and to open up new horizons in advancing the data center management plane. In addition, we have presented the network virtualization architecture to serve as a platform carrying out the abstractly specified network blueprints down to actual infrastructure configuration and control. We described a working DOVE prototype for open systems environment, and presented the results and the analysis of the performance evaluation study based on a large scale simulated DOVE network running contemporary multi-tenant workloads. Our performance evaluation results demonstrate the performance isolation provided by the solution and scalability that can be achieved with it.

This work provides a foundation for important advancements in the way network services are provided and consumed, and in how interconnected applications are created, deployed, and maintained. Among many future research directions, we actively explore the network modeling abstraction evolution, the service insertion framework creation, performance improvements, and others.

## References

[1] Openstack:Open source software for building private and public clouds.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proc. ACM SIGCOMM 2008 Conference on Data Communication*, Seattle, WA, August 2008.

[3] Katherine Barabash, Rami Cohen, David Hadas, Vinit Jain, Renato Recio, and Benny Rochwerger. A case for overlays in dcn virtualization. In *Proceedings of the 3rd Workshop on Data Center - Converged and Virtual Ethernet Switching*, DC-CaVES '11, pages 30–37. ITCP, 2011.

[4] R. Birke, D. Crisan, K. Barabash, A. Levin, C. DeCusatis, C. Minkenberg, and M. Gusat. Partition/aggregate in commodity 10g ethernet software-defined networking. In *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on*, pages 7 –14, June 2012.

[5] 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.

[6] Mosharaf Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 54(5):862–876, April 2010.

[7] 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.

[8] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, and Parveen Pat. VL2: A Scalable and Flexible Data Center Network. In *Proc. ACM SIGCOMM 2009 Conference on Data Communication*, Barcelona, Spain, August 2009.

[9] Teemu Koponen, Mohit Chawla, Byung G. Chun, Andrey Ermolinskiy, Kye H. Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM Computer Communication Review*, 37:181–192, 2007.

[10] 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.

[11] Cyriel Minkenberg and German Rodriguez. Trace-driven Co-simulation of High-Performance Computing Systems using OMNeT++. In *Proc. 2nd SIMUTools International Workshop on OMNeT++*, Rome, Italy, March 2009.

[12] 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.

[13] M.Sridharan, K.Duda, I.Ganga, A.Greenberg, G.Lin, M.Pearson, P.Thaler, C.Tumuluri, N.Venkataramiah, and Y.Wang. Nvgre: Network virtualization using generic routing encapsulation, 2011.

[14] 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.

[15] 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 *Proc. ACM SIGCOMM 2009 Conference on Data Communication*, Barcelona, Spain, August 2009.

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

[17] Sabine R. Öhring, Maximilian Ibel, Sajal K. Das, and Mohan Kumar. On Generalized Fat Trees. In *Proc. 9th International Parallel Processing Symposium (IPPS 1995)*, Santa Barbara, CA, April 1995.

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

[19] Cohen Rami, Barabash Katherine, Rochwerger Benny, Jain Vinit, and Recio Renato. Dove: Distributed overlay virtual network architecture. In *H-0315*. IBM, 2012.

[20] Andras Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. European Simulation Multiconference (ESM 2001)*, Prague, Czech Republic, June 2001.