SWIM: SWItch Manager For Data Center Networks

Chao-Chih Chen[‡*], Peng Sun[†], Lihua Yuan[‡], David A. Maltz[‡] and Chen-Nee Chuah[§] and Prasant Mohapatra[*]
[*]Department of Computer Science, University of California, Davis
{cchchen,pmohapatra}@ucdavis.edu
[†]Department of Computer Science, Princeton University
pengsun@cs.princeton.edu
[‡]Microsoft
{lyuan,dmaltz}@microsoft.com
[§]Department of Electrical and Computer Engineering, University of California, Davis
{chuah}@ece.ucdavis.edu

Abstract

As modern data center networks (DCNs) grow to support hundreds of thousands of servers and beyond, managing network equipment (e.g., routers, firewalls, load balancers) becomes increasingly complex. Network attributes (e.g., IP address allocations, BGP neighbor relations) are scattered amongst various network engineering groups, which makes troubleshooting the network a cumbersome task. In addition, network vendor diversity leads to an explosion of vendor-specific management systems or single-use automation scripts, limiting network scalability while increasing time to perform management tasks. This paper proposes a unified network management system, SWItch Manager (SWIM), to cope with the growth by 1) standardizing the language for describing network attributes, and 2) unifying the interface for executing management actions on the network equipment.

***Index Terms*—Data center Network, DCN, Network Management, Switch/router configuration**

I. Introduction

Network management is a crucial, yet challenging task for modern data centers that are growing dramatically in scale. A data center network (DCN) that supports 100,000 servers would typically require upwards of thousands of network equipment (NE). For brevity we will use NE to refer to both switches and routers. At such scale, operational agility is achieved through division of labor, with multiple network engineering groups managing different aspects of the network (e.g., physical wiring, routing design). In addition, as performance requirements such as oversubscription and fair-share bandwidth change at each layer of the DCN, NE from multiple vendors (or multiple NE models from the same vendor) are deployed to achieve optimal price-to-performance ratio.

Achieving operational agility and optimal price-to-performance ratio are not without their trade-offs. With each group managing only one aspect of the network, querying the network state incurs a communication overhead spanning multiple engineering groups. On the other hand, device diversity leads to a network environment rife with syntax/semantic disparities, and network engineers have to grapple with managing the DCN using multiple configuration languages. To put the two issues in perspective, consider the task of configuring peer-to-peer IP addresses between two routers (router X and Y) with different configuration syntax. In order to perform this task, network engineers need to know 1) the wiring map, to find all the interface pairs connecting X and Y, and 2) the IP subnet assignment, to know what addresses to configure onto each interface. Since both information are often kept by different groups, engineers would need to make two queries to gather all the required information. Furthermore, engineers commonly configure these addresses onto X and Y's interfaces via two single-use scripts, one tailored for X and the other tailored for Y, to handle the syntax disparity. Though it seems like a contrived example, the above process is common for many management tasks (e.g., establishing access control list, creating VLAN, setting up remote authentication). With the large number of NE in a DCN and multitude of tasks required in managing the NE, network engineers are easily overwhelmed by the communication overhead and explosion of single-use scripts.

For ease of discussions in the rest of the paper, we introduce the following two terminologies. **Network descriptions** refer to collections of NE attributes and their interconnections that together describe the network states, e.g., IP address or interface name of an NE or access control lists (ACLs). **Network implementation** refers to the action of configuring the NE to apply certain network attributes. For example, assigning an IP address to a specific interface of an NE through its command-line interface (CLI) is considered a network implementation. The example in previous paragraph highlights the challenge in gathering network description from various groups and the challenge in performing network implementation on two NE supporting different syntax.

In this paper, we propose SWItch Manager (SWIM) to cope with the afore-mentioned challenges. First, SWIM abstracts away the syntax and semantic disparities between different NE vendors by providing a centrally enforced standard language for network descriptions and network implementations. By doing so, it facilitates division of labor for network management among autonomous groups. Network descriptions specified by different groups can now be merged into a single global view that can be queried easily. Second, it defines a unified interface for performing management tasks on a diverse set of NE. Hence, network engineers can develop software to perform management tasks by invoking services provided by the unified interface without being exposed to vendor-specific details. With SWIM, network engineers who design the network (**network architects**) can express their designs in terms of network descriptions, while network implementation engineers develop software (**management clients**) to perform their management tasks via the unified interface.

The contributions of this paper are:

- We propose and develop SWIM – SWItch Manager, which provides network management as a service to network operators. We present two key components of SWIM: (1) a Network Description Engine that maintains a global view of the network states, while allowing distributed maintenance, and (2) Network Implementation engine, which provides a common set of APIs to execute a management task on a switch/router.
- We demonstrate how SWIM can be leveraged to perform common management tasks such as configuring interface IP addresses or performing BGP routing control.

II. SWIM Design

A. Architecture

SWIM is designed based on two subsystems (Figure 1) – *Network Description Engine* and *Network Implementation Engine*. Network Description Engine controls how network engineers describe the network states (including attributes for the network and their interconnections), aggregates these descriptions, and presents them as a unified view that can be queried easily. Network Implementation Engine exposes a uniform set of API to management clients, regardless of the syntax/semantics on the NE. Together, Network Description Engine and Network Implementation Engine provide network architects a way to describe their DCN designs as a series of network attributes, while allowing network engineers to program/configure the NE at a high level of abstraction.

Figure 1 outlines the workflow in SWIM. Network architects create new network descriptions according to their designs and send them to the Network Description Engine. Upon recognizing these updates, management clients perform their management tasks by issuing a series of action requests to the Network Implementation Engine, with the goal of aligning NE's states to the updated descriptions. Based on these requests, Network Implementation Engine queries Network Description Engine for the information it needs to execute the actions, and then interacts with the underlying channel (e.g., telnet session, OpenFlow session) to execute these actions on the NE.

B. Network Description Engine

Network Description Engine is the information reservoir that aggregates various descriptions at SWIM and presents a unified view of the network to management clients. Some examples of network descriptions are NE hostname (describing the NE in human readable string), interface IP address

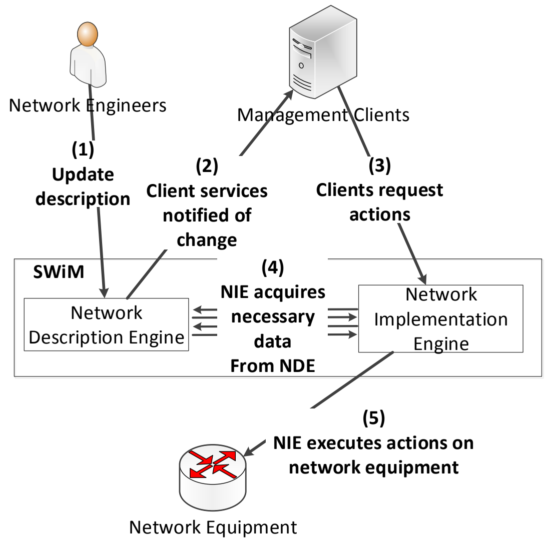
Network Engineers
Management Clients
(1) Update description
(2) Client services notified of change
(3) Clients request actions
SWiM
(4) NIE acquires necessary data From NDE
Network Description Engine
Network Implementation Engine
(5) NIE executes actions on network equipment
Network Equipment

Fig. 1: SWIM subsystems and workflow. (1) Network engineers update the network descriptions. (2) Management Clients pick up network description update. (3) Management Clients call Network Implementation Engine to request actions. (4) Network Implementation Engine makes internal calls to Network Description Engine to acquire the information it needs to execute the actions. (5) Network Implementation Engine executes actions on NE.

(describing the interface's L3 identity), and authentication configuration (describing accounts eligible to access the NE). Network descriptions can be more complicated, describing relationship between devices. Examples include physical wiring (describing physical connectivity between two NE ports) and BGP sessions (describing peering relationship between two BGP speakers). To achieve this, Network Description Engine institutes Description Manager to enforce syntax and semantics on network descriptions, and Network View Manager to present network descriptions as a unified view of the network.

1) Description Manager: With Description Manager creating and enforcing descriptions' syntax and semantics, network architects can use them as guides to create structured network descriptions. The division of labor among different groups (that maintain their respective network descriptions in a distributed manner) is transparent to the management clients. Instead, the management clients are only exposed to a unified view of the network without having to know how the network descriptions are actually managed.

Description Manager exposes a set of grammars to the network architects to enforce syntax and semantics on network descriptions. A well-known benefit of using grammars is that languages expressed from the grammars (i.e., network descriptions) are structured data. This eases syntax and semantics validation, as well as extracting information from network descriptions.

To avoid having too many sets of grammars, SWIM exposes grammars under two basic constructs: **graph** and **NE**

attributes. The graph construct (Grammar 1) is used to describe relationship between NE, be it physical or virtual. For example, a BGP graph would use BGP speakers' identity (e.g., Router ID, Autonomous System Number or ASN) as nodes and the BGP session between them as a graph link. Extension to support such a BGP graph is shown in Grammar 2.

Grammar 1 Graph grammar.

```
GRAPH := NODES LINKS
NODES := NODE*
NODE := NODE_ATTRIBUTE
NODE_ATTRIBUTE := string
LINKS := LINK*
LINK := START_NODE END_NODE
START_NODE := NODE
END_NODE := NODE
```

Grammar 2 Extending the basic graph grammar to support BGP.

```
GRAPH := NODES LINKS
NODES := NODE* BGP_NODE*
NODE := NODE_ATTRIBUTE
BGP_NODE := HOSTNAME ROUTER_ID ASN PEER*
NODE_ATTRIBUTE := string
HOSTNAME := string
PEER := [1-255].[1-255].[1-255].[1-255]
ROUTER_ID := [1-255].[1-255].[1-255].[1-255]
ASN := [1-65535]
LINKS := LINK* BGP_LINK*
LINK := START_NODE END_NODE
START_NODE := NODE
END_NODE := NODE
BGP_LINK := HOSTNAME PEER HOSTNAME PEER
```

For attributes that do not describe relationship between NE, the NE attributes construct can be used. Examples of NE attributes are host name, authentication and interface IP addresses. An NE attributes grammar supporting host name is shown in Grammar 3.

Grammar 3 An attribute grammar.

```
NE_ATTRIBUTE := ATTRIBUTES*
ATTRIBUTES := HOSTNAME
HOSTNAME := string
```

2) Network View Manager: Description Manager polices how architects create the descriptions. Network View Manager controls how the descriptions are stored internally and presented to clients. Recall that, based on the grammar imposed by Description Manager, descriptions are expressed as a collection of graphs and NE attributes. Network View Manager stores this information as-is. The reason for not merging them is that each graph already defines well-known relationships between NE in the network (e.g., wiring for physical relationships, BGP peering for virtual relationships). Storing them as overlay graphs instead of a single graph allows Network View Manager to preserve those relationships. Performing queries about specific graph then becomes easier, while queries involving multiple graphs can be performed in parallel and aggregated before returning to clients. For similar reasons NE attributes are also stored separately.

C. Network Implementation Engine

In this section, we describe the components that realize the management actions requested by management clients. We will describe how the actions are executed on NE after clients' action requests have been submitted to the Network Implementation Engine. Network Implementation Engine institutes Network Abstraction Manager and Implementer to enable management clients to work at a high level of abstraction, regardless of the underlying diversity in terms of vendor-specific syntax and semantics.

1) Network Abstraction Manager: To let clients operate at a high level of abstraction, SWIM exposes NE as generic NE with a set of common entities (e.g., routed interface, switching interface) to act on. Network Implementation Engine achieves this goal through a well-known mechanism called namespacing.

Conceptually, one can think of management tasks as actions supported under a specific namespace. In SWIM, actions are subclass of the generic NE namespace. For example, actions supported for routers will be exposed under the generic router class, and actions supported for switches will be exposed under the generic switch class. Interior nodes in the namespace hierarchy correspond to entities on the NE (e.g., the NE itself, routed interface, switching interface), while leaf nodes define the supported actions. One example of using namespace to expose supported actions is shown in Figure 2.

Network Abstraction Manager also keeps track of per-action variables that need to be instantiated by data from Network Description Engine. For example, for the **SetASN** action in Figure 2, Network Abstraction Manager would maintain an ASN variable that can be instantiated from the BGP graph. Variables in the tree are instantiated from data in Network Description Engine, if possible. This removes the need to validate data from management clients that have already been verified when they were updated to Network Description Engine.

The basic namespace tree in Figure 2 specifies actions that can be performed on entities, but it does not specify the exact entity to operate on. For example, while the namespace in Figure 2 shows SetIP as a supported action on a router's interfaces, it does not specify which interface should have its IP address set. This basic namespace tree exposes the entity **category** but not the **naming**. To add support for naming, Network Abstraction Manager augments the basic namespace tree with entity-specific names under each interior node. For example, if router X has two interfaces and one BGP peer with ID 10.0.0.1, its instantiated namespace tree would look like Figure 3. Entity-specific names can be instantiated from

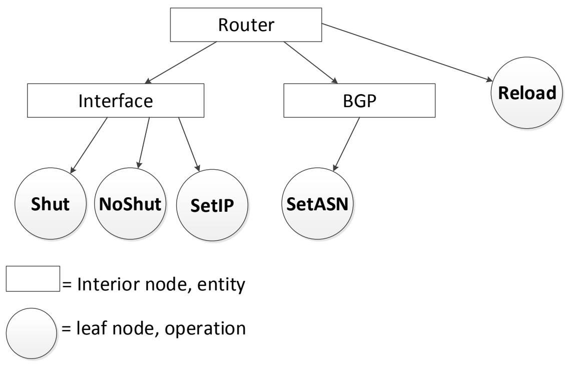

Fig. 2: An example of using namespace to specify supported actions for routers.

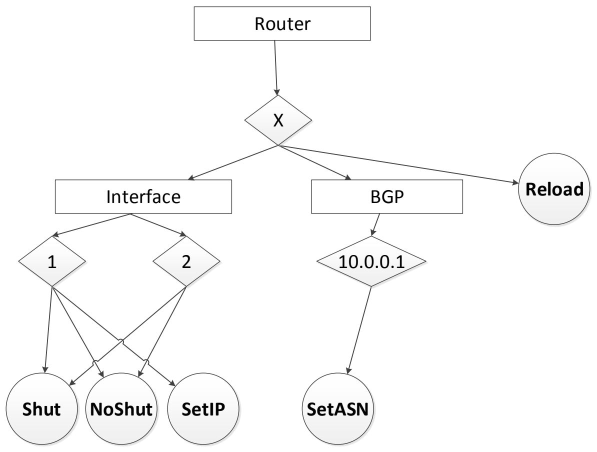

Fig. 3: Example of an augmented namespace tree for a router X that has two interfaces and a BGP peer with identifier 10.0.0.1.

network descriptions. For example, if the network description for router X contains two interfaces, Network Abstraction Manager can augment two interface nodes in router X's namespace tree.

2) Implementer: With Network Abstraction Manager exposing management tasks as a series of actions to perform on a generic NE, SWIM needs a way to actually execute these actions on different NE. There are many methods to accomplish this, such as templates [8] and declarative language [7]. We defer readers to these studies for more detail. It is worth noting that with Network Abstraction Manager exposing a generic NE to clients, Implementer is free to choose the actual mechanism to execute actions on the NE.

III. SWIM Implementation

We implemented a working prototype of SWIM based on the designs in Section II. SWIM is developed as a multi-threaded process, employing endpoints based on the Windows Communication Foundation [1] platform to serve requests from management clients.

Network Description Engine: Network Description Engine is implemented to accept network descriptions in the form of Extensible Markup Language (XML) documents and are validated against XML schema definition (XSD) files. The XML files contain descriptions based on either the graph grammar or the NE attributes grammar construct. The XML tags in the XML documents correspond to non-terminals in the grammar, while XML attributes/contents correspond to grammar terminals. Each grammar construct has a corresponding XSD file, so that their syntax and some of their semantics can be validated. Note that XML/XSD are not the only method to maintain descriptions, they are used here for their simple interface to construct grammar and validate descriptions against the grammar.

The description files are read into SWIM's in-memory objects through Window .Net's XML serialization [12]. Using Windows .Net, XML tags, attributes, and contents can be serialized into class variables. These classes are then used as the data structure for querying about the network.

Network Implementation Engine: Parallel to the container classes for network descriptions, another set of classes represent the actions that are supported under the namespace. SWIM processes management clients' requests through a dispatcher, which tracks and instantiates the variables using data gathered from Network Description Engine. It then invokes additional API or external tools to execute the actions on the NE.

In our implementation of SWIM, we only expose **SetIP** and **AddRoute** actions to clients. In reality many more actions can be exposed to clients, and we leave it to SWIM practitioners to determine what to expose. Based on our experiences, network topology and vendor diversity often determine what actions are exposed.

IV. Experiments

Based our prototype of SWIM, we performed two macro benchmarks to demonstrate its feasibility in practice and SWIM's performance on end-to-end scenarios.

The goal of our macrobenchmarks is to gauge the client-perceived end-to-end delay for executing management tasks/actions. To validate that SWIM can abstract away syntax/semantic differences, we expose two APIs for routers, each with a different underlying implementation. The first action is SetIP, which assigns the IP address to an interface; this is implemented by using telnet to connect to the router, and uses the router's CLI to push the configuration. The second action is AddRoute, which adds a network route to the router; which is implemented as a Routing-as-a-Service (RaaS) client [5] requesting to insert a route. For each experiment, results from 10 representative runs are shown, with the client perceived end-to-end delays broken into three components: time for SWIM to perform the critical task, time for the underlying tool to complete the action, and any client overhead.

For configuring interfaces (Figure 4), the time is dominated by the tool SWIM uses to log into the router and push the configuration. The client overhead for calling the API and the

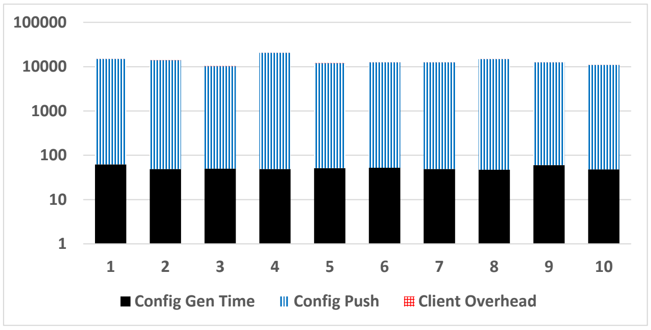

Fig. 4: Overall delay (in milliseconds, plotted on a log scale) of SetIP action through SWIM and underlying CLI. The delay is broken into the configuration generation time (critical task time), time to log into the router and push configuration via CLI (underlying tool time), and client-server communication time (client overhead).

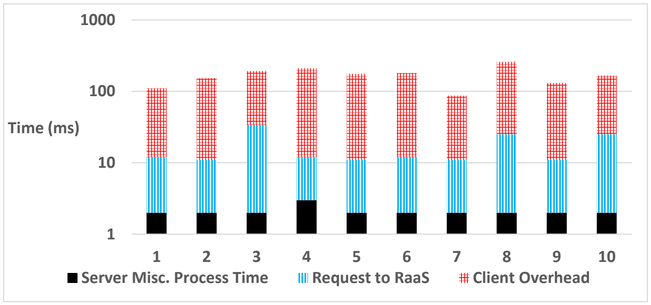

Fig. 5: Overall delay (in milliseconds, plotted on a log scale) of AddRoute action. The delay is broken into server process time (critical task time), time for server to send request to RaaS controller (underlying tool time), and client overhead.

time it takes SWIM to generate the appropriate configuration are both negligible, with configuration generation taking less than 100ms and client API call taking less than 200ms. The result is similar for route addition (Figure 5), except the total time is much shorter because RaaS can quickly generate and send the BGP UPDATE message. This result shows that while SWIM can quickly perform its task, the choice of underlying technology could dominate server-side latency.

V. Related work

There is a rich literature on network management, spanning from configuration management and generation to accounting, and hence it is infeasible to cover them all. Instead, we focus on studies that are most related to SWIM.

Many management studies model and abstract the network in some way. In CONMan [2], protocols are abstracted as multiple modules, with results piped from one module to another. SWIM's level of abstraction are actions that can be performed on the NE, and can leverage CONMan as a way to manage protocol-related changes.

In the space of configuration, there are plethora of works that leverage techniques such as templates [8], procedural [7] or declarative [6] specification. SWIM abstracts away the underlying mechanisms used to execute actions onto NE, so management clients can manage NE at a high level of abstraction (Section II). In addition, SWIM jointly considers the problem of distributed network description maintenance.

There are also other studies focusing on managing the flow of packets. Many of these studies leverage some aspect of the OpenFlow API [10] [9] while others leverage API on the NE [4]. A few studies leverage existing protocol to achieve some form of route control [3], [5]. This body of work target a part of NE's functionality that is vital in ensuring packets are delivered, but are specialized to just routing. SWIM on the other hand can support a wider range of management tasks through its abstractions.

From the point of view of network abstraction, Open vSwitch [14], VMWare vNetwork Distributed Switch [13], and Hyper-V vSwitch [11] are some of the commercial products that provide network abstraction through virtualization. In these products, switching/routing/policing capabilities are provided by virtual switches residing on the physical servers, below virtual machines (VM). Using virtual switches, many of the network functionalities can be implemented onto a common NE type (the virtual switch), and the underlying physical network becomes a simple data transport network. While having virtual switches move some of the complexities away from the physical network, physical switches still have to be managed, and in these cases SWIM can run alongside virtual switches to provide a comprehensive network management solution.

VI. Conclusion

We present SWIM, a general switch manager that enables network engineers and operators to develop software to perform network management tasks and distributedly maintain their view of the network. DCNs often rival or surpass ISP networks in size and have a diverse portfolio of NE. In such networks, maintaining coherent views manually and performing management tasks via single-use scripts could easily overwhelm network engineers and introduce errors. SWIM decouples the maintenance and enforcement of network descriptions via Network Description Engine and provide a common interface for executing management tasks via Network Implementation Engine. While allowing network engineers to independently maintain network descriptions, SWIM presents a global view of the network by merging these disparate sources network descriptions to support common queries. Network engineers can also independently develop software clients that interact with an NE to perform management tasks through a common interface provided by SWIM. Through a prototype of SWIM, we show that SWIM can perform management tasks through different underlying technology without exposing such details to network engineers.

References

[1] Windows communication foundation.

[2] Ballani, H., and Francis, P. CONMan: taking the complexity out of network management. In *Proceedings of the 2006 SIGCOMM workshop on Internet network management* (New York, NY, USA, 2006), INM '06, ACM, pp. 41–46.

[3] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. Design and implementation of a routing control platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2* (2005), USENIX Association, pp. 15–28.

[4] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. Ethane: taking control of the enterprise. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2007), SIGCOMM '07, ACM, pp. 1–12.

[5] Chen, C.-C., Yuan, L., Greenberg, A., Chuah, C.-N., and Mohapatra, P. Routing-as-a-Service (RaaS): A framework For tenant-directed route control in data center. In *INFOCOM, 2011 Proceedings IEEE* (2011), pp. 1386–1394.

[6] Chen, X., Mao, Y., Mao, Z. M., and Van der Merwe, J. Declarative configuration management for complex and dynamic networks. In *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies* (New York, NY, USA, 2010), Co-NEXT '10, ACM, pp. 6:1–6:12.

[7] Chen, X., Mao, Z. M., and Van der Merwe, J. PACMAN: a platform for automated and controlled network operations and configuration management. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies* (New York, NY, USA, 2009), CoNEXT '09, ACM, pp. 277–288.

[8] Gottlieb, J., Greenberg, A., Rexford, J., and Wang, J. Automated provisioning of BGP customers. *Network, IEEE 17*, 6 (2003), 44–55.

[9] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev. 38*, 3 (July 2008), 105–110.

[10] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (April 2008), 69–74.

[11] Microsoft. Hyper-V Virtual Switch Overview.

[12] Microsoft. XML Serialization in the .NET Framework.

[13] VMWare. vNetwork Distributed Switch.

[14] vSwitch, O. Open vSwitch.