An Introduction to OpenFlow

Subir Varma

1.0 Introduction

A few years ago, Academics and Students at Stanford University, were frustrated by the fact that they were not able to test out their proposed network control algorithms in the environment of an actual production network. This was because almost all networking products were built using proprietary hardware and software, and did not feature any open interfaces that could be used to change the programming on the box. Motivated by this, they came up with a new way by which external software could be used to control the behavior of networking nodes. The protocol that is used on this new control interface that they designed was named OpenFlow.

Starting from these origins, in the last couple of years OpenFlow leapt out of Academia and has been embraced by a section of the networking industry that was looking for new solutions in areas where traditional networking architectures were running into their limits, such as Data Center Networks. The broader networking industry, including the Service Provider and Enterprise segments, is also beginning to take these ideas seriously, and in the coming years we are likely to see new products and technologies in these areas that make use of OpenFlow.

The objective of this White Paper is to give a brief introduction to OpenFlow and to the related areas of Software Defined Networking and Network Virtualization. These latter two technologies are often paired up with OpenFlow in the popular press, but there is a certain amount of confusion about their exact definition, that we will try to clear up. The White Paper is organized as follows: In Section 2 we motivate OpenFlow by describing issues and problems that cannot be solved within the framework of legacy networking. Section 3 is devoted to a description of the OpenFlow protocol. In Section 4 we describe Network Virtualization and how OpenFlow can be used for doing this function. In Section 5 we talk about how OpenFlow is influencing Service Provider networks, while in Section 6 is on the subject of Data Center networks, and finally in Section 6 we present our conclusions.

2.0 Why Do We Need OpenFlow?

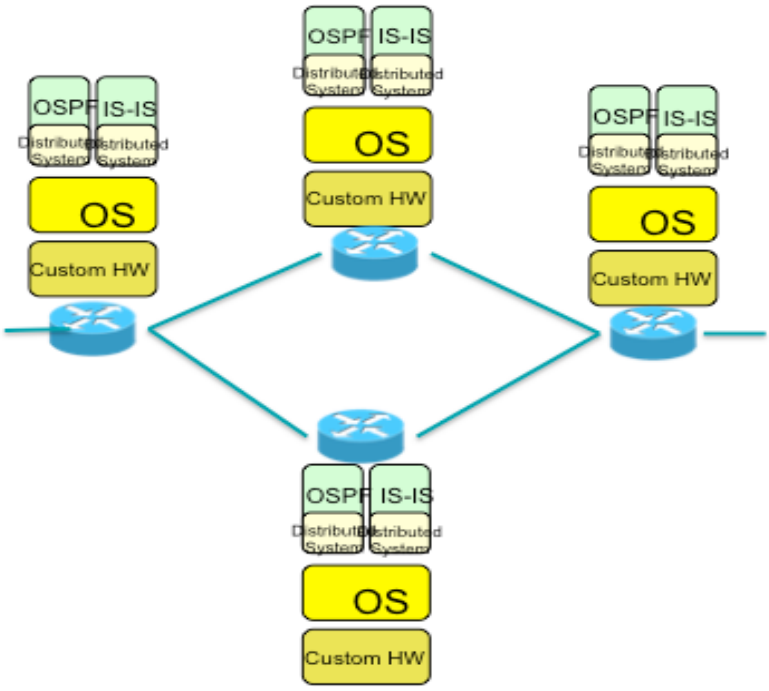

Figure 1

OpenFlow is often referred to as a way to Modernize the Network Control Plane, which begs the question as to why the control plane needs modernization and what is wrong with the way it works today.

Figure 1 shows an example of todays networking paradigm, consisting of nodes built using custom hardware designs and running a proprietary operating system. Each of these nodes features a Data Plane that does the actual forwarding of the data packets, and a Control Plane that influences the path that the packet takes, as well as other actions can be performed on the packet, such as tunneling, firewalling, traffic management etc. As shown in the figure, control plane algorithms make use of an abstract view of the network that is created by algorithms in the node, referred to as the Distributed System. The job of the Distributed System is maintain an current view of the network topology, by initially discovering the topology and later reacting to events such as link or node failures (or additions), that change the topology. The major portion of the work required to design a Control Algorithm, actually goes into the design of the Distributed System. Taking the example of the OSPF routing protocol: A description of the Distributed System takes 101 pages, while a description of Djikstra's algorithm takes up only 4 pages.

Today, with every Control Algorithm comes with its own Distributed System, which leads to the following: Designing a new control algorithm is a major undertaking, which requires a lot of co-ordination between the hardware and software in the node. Also, all

the nodes need to be able to inter-operate with each other when using the algorithm, as a result it has to be approved by the Standard's Bodies. All this makes it very difficult to introduce new control algorithms into the network, which has slowed down the pace of innovation in the networking industry. This problem, in combination with the closed nature of the hardware and software in the networking nodes, has led to the situation that new features are introduced at the pace of hardware development, which is typically several years.

A problem created due to this state of affairs is that the Network Operator has very little control on the way in which his network works, and is not able to add any proprietary value added features to differentiate his network from that of the competition. He is forced to buy the same or very similar product from all vendors, whose price keeps going up due to the fact that the complexity of these products keeps increasing. The increase in complexity can be attributed to the fact that the vendor makes a generic product that incorporates all the standards that have been introduced in the last thirty years; the number of rfcs from the IETF has crossed the 6500 mark.

Another problem created due to this deficiency in network control arises for the cases in which the network control rules have to be manually configured into each node. This is actually the case for most network control rules, other than regular routing. Not only is this labor intensive and error prone, but also has to redone every time that there are changes in the network due to users or end nodes joining or leaving the network. A special case of this arises in modern Data Center networks, each of which may feature tens of thousands of Virtual Machines (VMs), distributed over multiple servers. Each of these VMs are subject to movement from one server to another in order to balance out server loads using technologies such as vMotion. However this also means that the network configuration is constantly changing, and control rules in the networking nodes such as VLAN rules or security policies have to be modified accordingly. There is no way to do this within the constraints of the current network control paradigm.

Current networks feature technologies such as VLANs and MPLS that enable the operator to split up the links at Layer 2 (using VLANs) or Layer 3 (using MPLS). However there is no way in which the network operator can create multiple virtual networks, in which not just the links, but also the node resources such as buffering, routing tables, ports etc are sliced into multiple virtual networks. Each of these virtual networks should be able to run their own control algorithms, for example Virtual Network 1 could be a bridged network, while Virtual Network 2 could a routed network.

3.0 OpenFlow Description

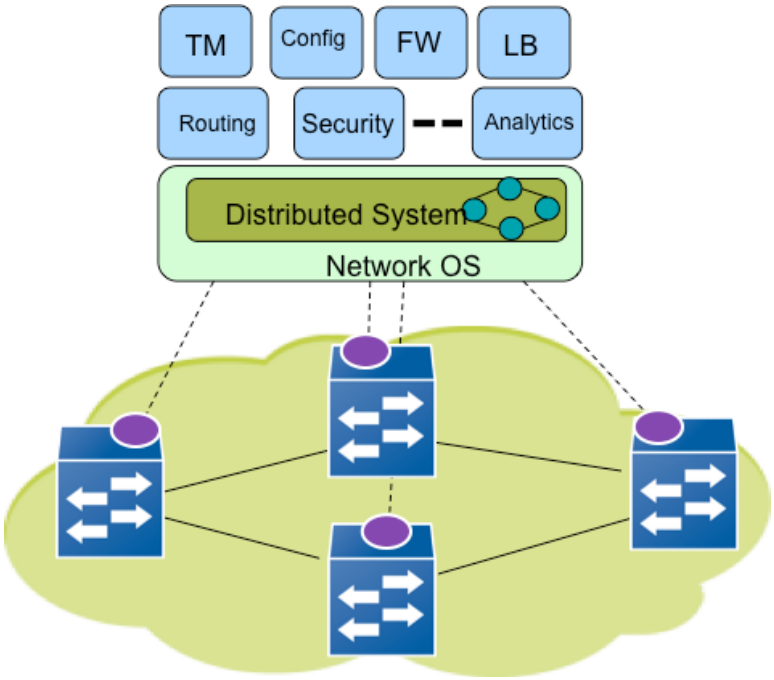

Figure 2

Figure 2 shows the basic architecture for an OpenFlow controlled network, which is made up of two principal components, namely OpenFlow Switches and the OpenFlow Controller. Note that the control plane, shown at the top of the figure, is centralized and exists separately from the data plane located in the network switches. The control node communicates with the switches using an out-of-band communication network (shown with the dotted line), using the OpenFlow protocol. We now describe the workings of the OpenFlow switch, followed by the Controller.

3.1 The OpenFlow Switch

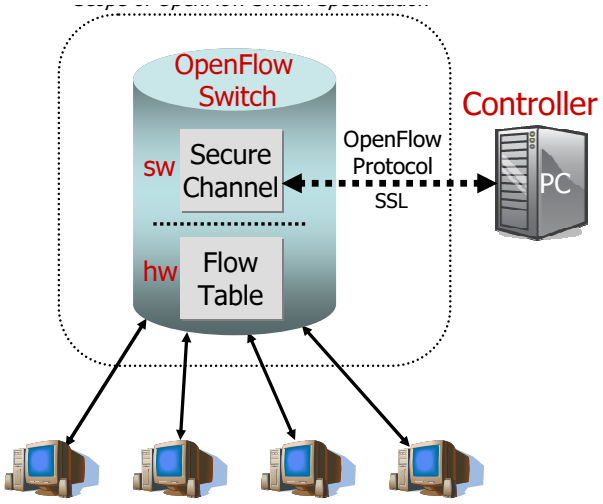

Figure 3a

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

Figure 3b

A modern Ethernet switch or router contains Flow Tables that run at line rate to implement firewalls, NAT, QoS etc and to collect statistics (Figure 3a). A Flow Table consists of a list of Packet Flows, accompanied by an action that the switch has to perform on that flow. This basic construct is also used in an OpenFlow switch.

A Flow Table is typically implemented using TCAMs in order to get the line rate performance. If we are able to directly control the entries in the Flow Table, then we can bypass the control logic in the switch, and be able to influence packet flows, which is precisely the function performed by the OpenFlow protocol.

An OpenFlow switch consists of at least three parts: (1) A Flow Table, (2) A Secure Channel that connects the switch to a controller, allowing command and packets to be sent between the controller and the switch using (3) The OpenFlow protocol, which provides an open and standard way for a controller to communicate with a switch. By specifying a standard interface through which entries in the Flow Table can be defined externally, the OpenFlow switch avoids the need for network operators to program the switch.

The set of actions supported by the Flow Table in the switch is extensible, but consists of the following minimum set:

- Forward this flow's packets to a given port (or ports). This allows packets to be routed through the network.
- Encapsulate and forward this flow's packets to the controller. The packet is delivered to the Secure Channel module in the switch, where it is encapsulated

and sent to the controller. This is typically used for the first packet in the flow, so that the controller can decide if the flow should be added to the Flow Table.

- Drop this flow's packets. This can be used for security or to curb DoS attacks.

An entry in the Flow Table has three fields: (1) A Packet Header that defines the flow, (2) The action, which defines how the packets should be processed, and (3) Statistics, which keep track of the number of packets and bytes for each flow, and the time since the last packet matched the flow (to help with the removal of inactive flows).

In the first generation of OpenFlow switches (referred to as "Type 0" switches), the flow header is a 10-tuple as shown in Figure 3b. A TCP flow could be specified using all 10 fields, while an IP flow might not include the TCP ports in its definition. Each header field can be a wildcard to allow for aggregation of lows. For example if only the VLAN ID is specified, then it would apply to all traffic on a particular VLAN.

OpenFlow Enabled Legacy Switches

Instead of building an OpenFlow switch on a new hardware platform, existing commercial switches and routers can be enhanced with the OpenFlow feature by adding the Flow Table, Secure Channel and OpenFlow protocol. The Flow Table can re-use existing hardware, such as a TCAM, while the Secure Channel and OpenFlow Protocol can be ported to run on the Switch's operating system. A number of networking vendors, including HP, Juniper and NEC have taken precisely this approach to make their existing products into OpenFlow switches. However this approach does come with some performance impacts, which we will review in Section 3.3.

3.2 The OpenFlow Controller

The OpenFlow controller can be implemented in software on a standard server platform. The high level structure of a controller is shown in the top portion of Fig 3a. It is implemented using a layered architecture, with the bottom layer called the Network Operating System or NoS, and the upper later consisting of one or more Applications used to control the OpenFlow switches. Analytics etc. In this architecture neither the NoS nor the interface between the NoS and the Applications has been standardized yet, only the interface between the NoS and the switches has been specified with OpenFlow.

The NOS layer is responsible for managing the communications between the controller and the switches. In addition it also responsible for creating and maintaining a current topology map of the network. This topology map is exposed to the applications running in the controller on its northbound interface.

Note that the structure of the NoS results in the creation of a common Distributed System which can be shared by all the applications running in the controller. As a result of this design, every control application shares a common Distributed System, unlike in current networks where Applications need to implement their own Distributed System (Fig 1). Moreover, the problem of designing a distributed control algorithm is reduced to a pure software problem, since the control algorithm can operate on the network abstraction created by the NOS rather than the real network. This aspect of the Controller design gives this approach a lot of its power, since now it is a much easier task to design a new control application for the network, for the following reasons: (1) The designer does not have worry about taking his algorithm to a Standards Body, since all aspects of inter-operability between nodes are taken care of by OpenFlow, (2) The

problem of designing new Applications is now reduced to a problem in Software Engineering, rather than a more difficult problem in Distributed Algorithms. This ability to control the network by purely software means is referred to as *Software Defined Networking*.

Examples of Applications include Routing, Network Configuration and Security, Firewall Rules, Load Balancing, Traffic Management etc

3.3 Performance Issues

An extensive study of OpenFlow performance on legacy switches was done at HP Labs [3]. Their main findings can be summarized as follows:

- More than the central controller, the switches themselves are a bottleneck in flow set up.
- The effective bandwidth available on the control-plane between the switch and the controller was much less than expected, which adds unacceptable latency to flow setup, and cannot provide flow statistics timely enough for traffic management tasks as load balancing.
- Maintaining complete visibility in a large OpenFlow network requires hundreds of thousands of flow table entries at each switch. However commodity switches are not built with such large flow tables, making them inadequate for many high performance OpenFlow networks.

These limitations are due to the fact that switches have finite bandwidths between their data and control planes, and finite compute capacity. This study leads to the conclusion that switches that are optimized for OpenFlow will require a new hardware design with larger TCAMs and an increase in bandwidth between the data and control parts of the switch.

There have been several projects that have been undertaken to scale up the OpenFlow controller, including Onix [5] and Maestro [6].

4.0 Network Virtualization using OpenFlow

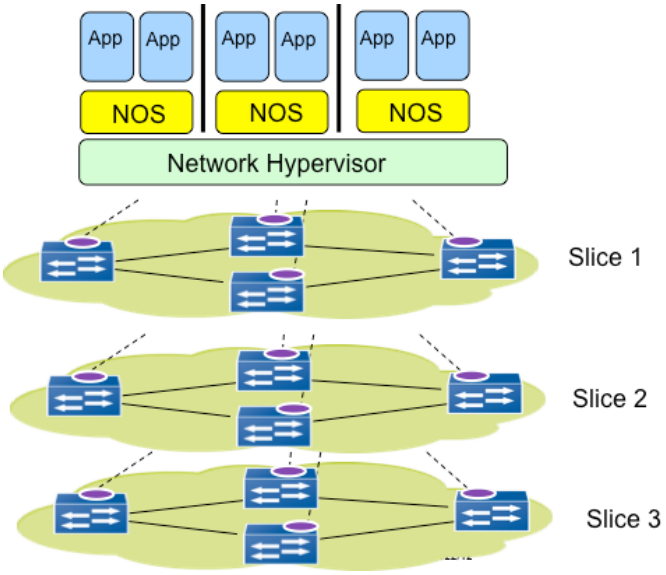

Figure 4

Recall that in the OpenFlow model described in Section 3.0, the Applications running on the OpenFlow Controller operate on an abstraction of the network topology that is created by the NOS layer. This network topology was based on the actual physical nodes in the network. But what if instead of exposing the physical topology to the Applications, they instead see a logical topology instead. Instead of interfacing directly to the networking hardware as is done today, the Applications read and write to the virtual forwarding elements making up the logical topology. This approach allows network state (forwarding and configuration) to be largely de-coupled from the underlying hardware.

The most compelling use case for this architecture arise when we consider a many-to-many mapping between the logical and physical forwarding elements. For example, multiple logical forwarding elements may share the same physical switch (shown in Fig 4), or a single logical forwarding element may span multiple physical switches. This latter approach is being applied to data centers networks, where an entire data center can be managed as a single switch instance. The first model shown in Fig 4, is also very attractive to Network Operators, since it allows them to slice up their network into multiple segregated parts, each of which may have their own topology, and could be running their own control algorithms. An Operator can slice up a network in this manner between multiple protocols (IPv4, IPv6 for example), multiple MVNO customers or even between multiple applications (VoIP, IPTV).

Note that current networks also support virtualization in the form of VLANs, tunnels of various types (GRE, IPSec), VRF contexts etc. However these do not provide an adequate abstraction to build a physical topology independent control software. Managing tunnels, VLAN configurations and VRF contexts requires significant manual network management, and component failures result in disruption of the virtual configurations. The OpenFlow based virtualization approach makes use of the sophisticated hardware support for these existing virtualization techniques, but adds

more comprehensive virtualization solution in which hardware can be treated generically as a resource pool of forwarding capacity and hardware changes do not disrupt the logical view of the system.

In order to carry out the virtualization, a new layer is introduced at the controller, known as the Network Hypervisor (Fig. 4). It communicates with the physical network nodes using OpenFlow which gives it a view of the physical network topology, while from above it is given one or more logical views of the network. The main job of the Hypervisor is to implement the desired logical functionalities through configuration of the physical network. Thus the hypervisor is the point where the logical network(s) is mapped to the physical network.

The hypervisor achieves this mapping by creating logical forwarding elements, where each forwarding element has a set of logical ports, a set of lookup tables and other basic forwarding primitives such as counters, en/de-capsulation etc. When a packet comes into a physical switch, it gets mapped to its logical context (or network slice), which determines the forwarding rule which leads to the logical port for forwarding the packet. The logical port then gets mapped back to the physical next-hop address. Note that the logical port may correspond to a physical port on another physical switch (for the case when multiple physical switches form a logical switch). Packets can be mapped to logical contexts using a variety of mechanisms, such as identifying tags like the VLAN or MPLS headers, or the ingress port etc.

5.0 OpenFlow in Service Provider Networks

Service Providers today face a number of challenges for which they are urgently looking for solutions, including:

- Rapidly increasing of deploying new infrastructure to keep up with traffic growth, which is not matched by a corresponding increase in service revenues. If this trend continues, then in a few years, the Service Provider business model will become un-profitable, with costs exceeding revenues. OpenFlow promises to reduce infrastructure costs by replacing expensive, feature-rich proprietary networking nodes by commodity switches controlled centrally through OpenFlow.
- Inflexibility in Network Control: The Service Provider is not able to add value to his network which limits his differentiation when compared to the competition, since everyone is essentially deploying an identical network architecture. The reason for this is that under the current rules, a new network control protocol requires that it be blessed by a Standards Body in order to ensure inter-operability. This is typically a multi-year process, at the end of which network vendors deploy the new protocol, but also make it available to all Operators, thus limiting differentiation. With OpenFlow on the other hand, Service Providers can deploy their own value added proprietary network control protocols, without having to worry about standardization. Moreover, since the process of designing new protocols is reduced to a software engineering problem, it is subject to much faster development cycles as compared to the current paradigm in which new protocols are constrained by the hardware development cycle, which is typically a multi-year effort.
- Support for Multi-Tenancy: Service Providers are constrained by existing network technology in area of slicing their network into multiple virtual pieces. This capability is important to them for a number of reasons: (1) It will enable Service Providers to better support MVNO customers on their network, leading to increased revenues. (2) It will enable Service Providers to test out new protocols on their production network, without affecting revenue generating traffic. (3) Several operators are moving towards providing Cloud based services, for which they are converting their Central Offices into Data Centers. Multi-tenancy through network virtualization is one of the essential technologies required in Data Centers (more on this topic in Section 6).

An area, which is of more immediate interest to Tellabs, is the implication of OpenFlow for Optical Transport Networks; this is described in more detail in Section 5.1.

5.1 OpenFlow and Optical Transport Networks

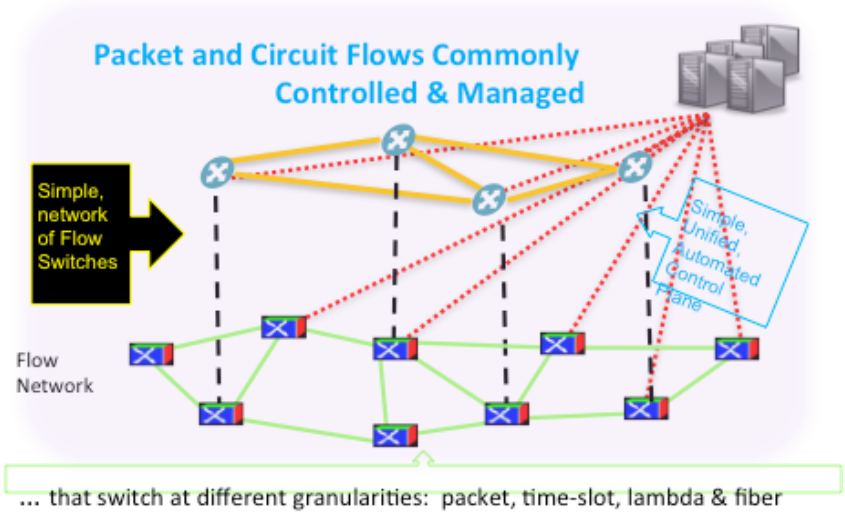

Figure 5a

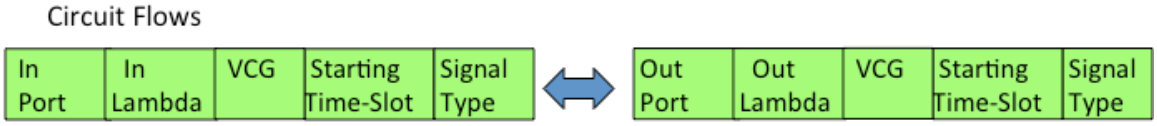

Figure 5b

We provide a brief summary of a joint research project between Ciena Corp. and Stanford University on the use of OpenFlow for controlling Optical Transport Networks [7].

The Optical Transport Network today is made up of statically configured switches in which bringing up a new circuit is a manual operation that can take days. The IP and Optical networks are separately managed and operated independently, which results in duplication of functions and resources in multiple layers in addition to capex and opex burdens. If it were possible to make the Optical Network more dynamic, then it can lead to several benefits that are hard to provide in packet-only networks, including:

- *Dynamic Packet Link Creation/Deletion*: This allows the operator to change the underlying circuit link topology in response to changing traffic demands.
- *Application Aware Routing*: Instead of forcing all applications to traverse the same circuit link between routers, it allows operators to dynamically create circuits tailored to the application. For example, VoIP traffic can be sent over low latency circuit path, while video traffic can be sent over a low-jitter circuit path.
- *Variable Bandwidth Packet Links*: By monitoring the bandwidth usage of the circuits that make up a packet link, the operator can dynamically vary the bandwidth allocated to those circuits. This would reduce congestion on the packet link.
- *Intelligent Failure Recovery*: The failure recovery mechanism can be tailored to the type of traffic. For example video traffic could circuit protected with pre-

provisioned bandwidth, voice could be dynamically re-routed in the circuit topology and http traffic could be re-routed in the packet topology.

As we pointed out in Section 3.1, OpenFlow abstracts a data-plane switch as a Flow Table, and defines a flow to be any combination of L2-L4 packet headers for packet flows. In a similar fashion, an Optical Switch can be abstracted as a Cross Connect Table, with the circuit identification fields as shown in Fig 5b. Using these abstractions, it becomes possible for OpenFlow to control the switching between the packet and circuit domains, thus leading to a Unified Control Plane for them. Additional details on the extensions to the OpenFlow protocol to enable this are provided in [8].

6.0 OpenFlow in Data Center Networks

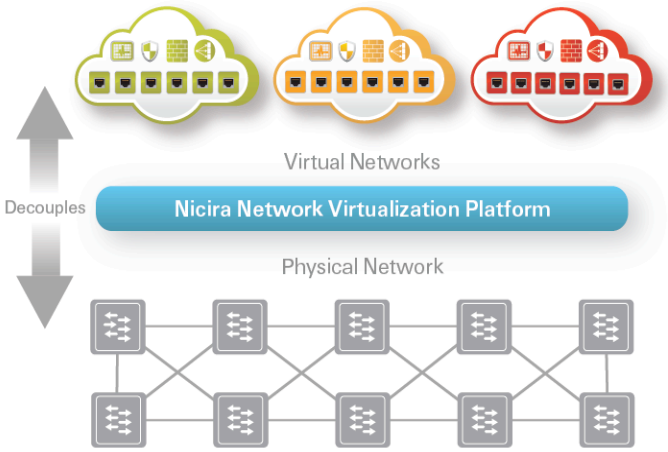

Figure 6

Data Centers have undergone enormous changes in the last decade, mostly as a result of the rapid adoption of Hypervisor technology in servers, leading to the proliferation of Virtual Machines (VMs). A large data center for example may have as much as 400,000 VMs spread over thousands of servers. This has led to new networking challenges, that current networks are having difficulty solving, including:

- VMs need to be connected together at Layer 2, in order to allow VM migration between servers (this required to do server load balancing for example). However existing Layer 2 protocols are not able to handle the scale of the network thus created. For example Layer 2 bridging can create broadcast storms and the lack of hop count field in the Ethernet header can result in infinite loops in the routing.
- Each VM comes with a Virtual Interface with its own MAC and IP addresses. In a layer 2 environment, the number of MAC addresses thus created is beyond the capacity of the bridging tables in current Ethernet Switches.
- If the Data Center Operator wishes to segregate the VMs into multiple Virtual Networks (see Figure 6), then using available VLAN technology, at most 4096 Virtual Networks can be created. This number is not sufficient to support the number of users in a large Data Center.

As a result, Data Center networks have seen many recent innovations by vendors in an attempt to solve these problems. The industry seems to be converging around an architecture that consists of the following elements:

- A Virtual Ethernet Switch (VES) in Server Hypervisors. All the VMs in the server connect to the VES through their virtual interfaces, rather than to the external switch. The first VES implementations were basic bridges, but the trend has been to make them more intelligent with higher functionality.
- All VESs are connected together using tunneling technologies such as VxLAN, NvGRE, TRILL or SPB. These tunneling protocols do either L2 in L3

encapsulation (VxLAN, NvGRE) or L2 in L2 encapsulation (TRILL, SPB), and provide a way for the L2 frames from the VMs to traverse the Data Center network without running into the problems mentioned above, such as broadcast storms or infinite loops. They incorporate routing protocols such as IS-IS or OSPF to find shortest paths, rather than use Spanning Tree.

- An external controller that is connected to the VESs. Companies such as VMWare and Cisco are using a proprietary protocol on this interface, while a Nicera uses OpenFlow. The external controller is required to configure the Virtual Networks, and also dynamically re-configure them when VMs move between servers.

As Nicera has shown, OpenFlow can be used to provide an open interface to control the Data Center network. However there are other proprietary solutions that solve this problem, so it remains to be seen who wins out in the marketplace.

7.0 Conclusions

This White Paper has provided a brief introduction to OpenFlow and some background into the reasons why it is being taken seriously by the networking community. Its initial target market has been Data Center networks, but Service Providers are showing increasing interest in the protocol and we can expect to see more products targeting this segment of the market soon.

References

1. N. McKeown et. al., “OpenFlow: Enabling Innovation in Campus Networks,” Available at www.opennetworking.org.
2. “OpenFlow Switch Specification”, Version 1.2, Available at www.opennetworking.org, Dec. 2011.
3. J.C. Mogul e.al., “DevoFlow: Scaling Flow Management for High Performance Networks,” SIGCOMM 2011.
4. M. Casado et. al, “Virtualizing the Network Forwarding Plane.” Available at yuba.stanford.edu/~casado/virt-presto.pdf
5. T. Koponen et. al, “Onix: A Distributed Control Platform for Large Scale Production Networks,” OSDO 2010.
6. Z. Cai et. al., “Maestro: A System for Scalable OpenFLow Control,” Rice Univ. 2010.
7. S. Das et.al., “Application-Aware Aggregation and Traffic Engineering in a Converged Packet-Circuit Network,” Available at www.stanford.edu/~yiannisy/cgi-bin/docs/pac-circ.pdf.
8. S. Das, “Extensions to the OpenFlow Protocol in Support of Circuit Switching,” Available at http://www.openflow.org/wk/images/8/81/OpenFlow_Circuit_Switch_Specification_v0.3.pdf.