Software Defined Networking – Shaping up for the next disruptive step?

Doan Hoang
iNEXT-Centre for Innovation in IT Services and Application, Faculty of Engineering and Information Technology, University of Technology Sydney.

Abstract: Software-Defined Networking (SDN) has emerged as a networking paradigm that can remove the limitations of current network infrastructures by separating the control plane from the data forwarding plane. As an immediate result, networks can be managed cost-effectively and autonomously through centralising the decision-making capability at the control plane and the programmability of network devices on the data plane. This allows the two planes to evolve independently and to open up separate horizontal markets on simplified network devices and programmable controllers. More importantly, it opens up markets for infrastructure providers to provision and offer network resources on-demand to multiple tenants and for service providers to develop and deploy their services on shared infrastructure resources cost-effectively. This paper provides an essential understanding of the SDN concept and architecture. It discusses the important implications of the control/data plane separation on network devices, management and applications beyond the scope of the original SDN. It also discusses two major issues that may help to bring the disruptive technology forward: the intent northbound interface and the cost-effective SDN approaches for the industry.

Introduction

The Internet has scaled enormously over the last five decades, partly due to its appropriate and universal level of abstraction at the IP layer (network layer), supporting any type of underlying physical networks and vast number of users and applications. The success is also brought about by the adoption of connectionless connectivity and resilient distributed routing mechanisms, allowing scalability and robust message delivery. However, the Internet has reached a point where it is extremely difficult to explore new architectures and flexible platforms that support emerging applications such as social networking, cloud, and big data platforms.

It has been difficult to introduce innovation in applications because legacy networks have become too rigid, yet too complex to manage, and no automation is available to handle the complexity. The decision-making capability (routing, managing, monitoring, and load balancing) is distributed across various network devices. The data plane and the control plane of networking devices are tightly coupled and vertically integrated, making it difficult for the Internet to evolve.

To support high-level network policies such as quality of service (QoS), network operators have to configure each network device manually and individually. This approach becomes impracticable, error prone and costly for large and dynamic virtual networks where network devices (virtual servers and virtual switches) may come into existence on-demand and move about within the infrastructure. In addition, it has been difficult for the Internet to deal with many important issues over the last few decades such as mobility, security, programmability, limitation of address space, and managing wireless sensor and optical networks.

Efforts have been undertaken to make the Internet more adaptable and manageable though active networks, programmable networks (Feamster et al 2013), and clean slate design (Greenberg et al 2005). All these projects realise that without separating the control plane (the part that is making network-wide decisions) from the data plane (the part that is responsible for forwarding data) it is difficult to innovate and revolutionise the Internet.

Software-defined networking (SDN) has emerged as a networking paradigm that can remove the limitations of current network infrastructures (Open Networking Foundation 2012). The intent of SDN is to separate the data forwarding plane from the control plane by centralising the network state and the decision-making capability, providing programmability on the control plane (SDN controller), simplifying the operation at the forwarding plane (SDN network devices), and abstracting the underlying network infrastructure to the applications (Gude et al 2008). The separation of the control plane and the data forwarding plane is through a programming interface between the SDN network devices and the SDN controller.

By separating the network's control logic from the underlying network devices, SDN enables network programmability, allows simplified and autonomous management, stimulates application, and hence opens up markets and opportunities for vendors, network and service providers.

SDN concepts and technologies have been developed to the point where the next step is crucial as all players (network equipment manufacturers, software vendors, original device manufactures, and enterprises) are evaluating appropriate strategies for their organisation in terms of cost-effective transition from legacy networks and the most beneficial SDN-enabled solution. This paper aims to provide a simple and clear explanation of SDN, its architecture, and the functionality of its components. The paper explores the scope of SDN in terms of the relationships of the technology with infrastructure providers, service providers, and cloud service providers. The paper also discusses emerging issues that are being addressed by SDN.

The rest of the paper is organised as follows. The fundamental architecture of SDN is introduced in the next section, followed by a concise description of the OpenFlow protocol that allows the separation of and the communication between the control and the data planes. The main components and the functionality of the SDN controller will be discussed in a subsequent section, followed by the description of several SDN-enabled applications. The paper discusses issues faced by this disruptive technology and concludes with remarks concerning the next step.

SDN Architecture

The basic elements of SDN include: SDN devices, SDN controller, and applications. The SDN devices contain components for deciding what to do with incoming traffic (frames or packets). The SDN controller programs the network devices and presents an abstraction of the underlying network infrastructure to the SDN applications. The controller allows an SDN application to define traffic flows and paths, in terms of common characteristics of packets, on the network devices to satisfy its needs and to respond to dynamic requirements by users and traffic/network conditions. The Open Networking Foundation defines a high-level architecture for SDN (Berde et al 2014) with three main layers as shown in Figure 1.

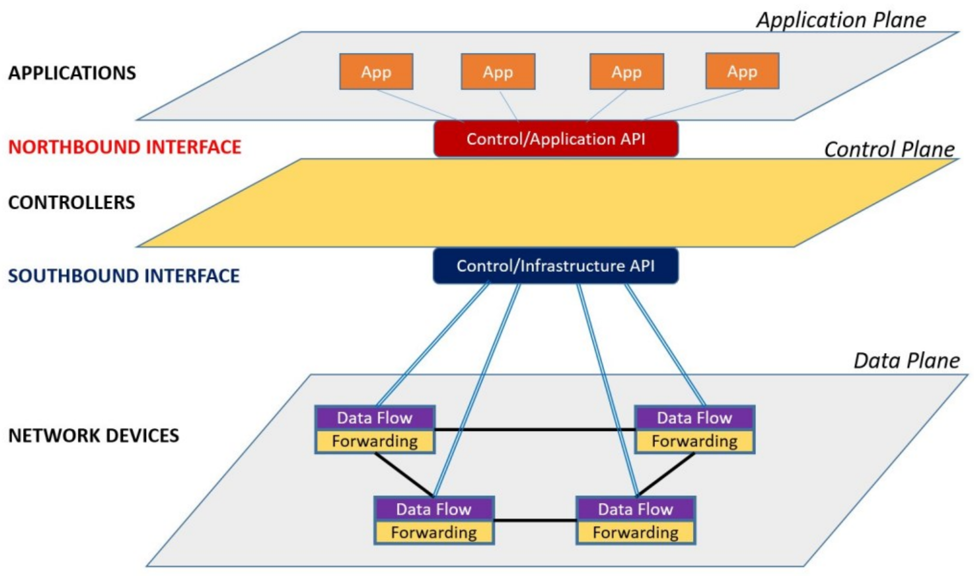

Figure 1 – Software-defined networking – a high level architecture

Infrastructure Layer

This layer consists of SDN devices (both physical and virtual) that perform packet switching and forwarding. Specifically, an SDN device is composed of an application program interface

(API) for communication with the controller, an abstraction layer, and a packet-processing component. The abstraction layer abstracts an SDN device as a set of flow tables. The packet processing function decides on actions to be taken, based on the results of the evaluating incoming packets relative to flow entries in the flow tables.

Control Layer

This layer provides the logically centralised control functionality that supervises the network forwarding behaviour through an open interface. An SDN controller controls, through a southbound API, all SDN devices that make up the network infrastructure; and implements policy decisions such as routing, forwarding, load balancing, etc. It provides an abstract view of the entire network to the applications through a northbound interface.

Application Layer

This layer consists of end-user applications that utilise the SDN communications and network services (Goransson & Black 2014). Through the controller the applications are able to affect the behaviour of the underlying infrastructure by configuring the flows so as to route packets through the best path between two endpoints, balancing traffic loads across multiple paths or destined to a set of end points, reacting to changes in network topology such as link failures and the addition of new devices and paths, or redirecting traffic for purposes of inspection, authentication, segregation, and similar security-related tasks.

The following sections describe the standardised southbound interface, OpenFlow, and an SDN controller.

OpenFlow

OpenFlow is a standardised protocol (Open Networking Foundation 2013) that defines the southbound communication between a controller and an OpenFlow switch. The communication messages between the two are transmitted over a secure channel that is implemented via a Transport Layer Security (TLS) connection over TCP. Through the exchange of commands and packets, the controller defines and programs the switch's packet forwarding behaviour, and the switch performs the packet forwarding accordingly and reports its configuration status and traffic conditions.

User traffic is classified into flows based on its characteristics. An OpenFlow switch performs packets lookups and forwards according to the flow they belong. A **flow** is a set of packets transferred from one network endpoint (or set of endpoints) to another network endpoint (or set of endpoints). The endpoints may be defined as IP-TCP/UDP address pairs, VLAN endpoints, or switch input-output ports, etc.

To the controller, an OpenFlow switch is presented by a set of flow tables. A **flow table** consists of flow entries. A **flow entry** comprises header fields, counter fields, and action fields. The header fields are used to identify the flow of a packet, the counter fields are used to collect statistics of the identified flow, and the action fields provide the instructions and actions the switch has to perform on the packets of that flow (Figure 2).

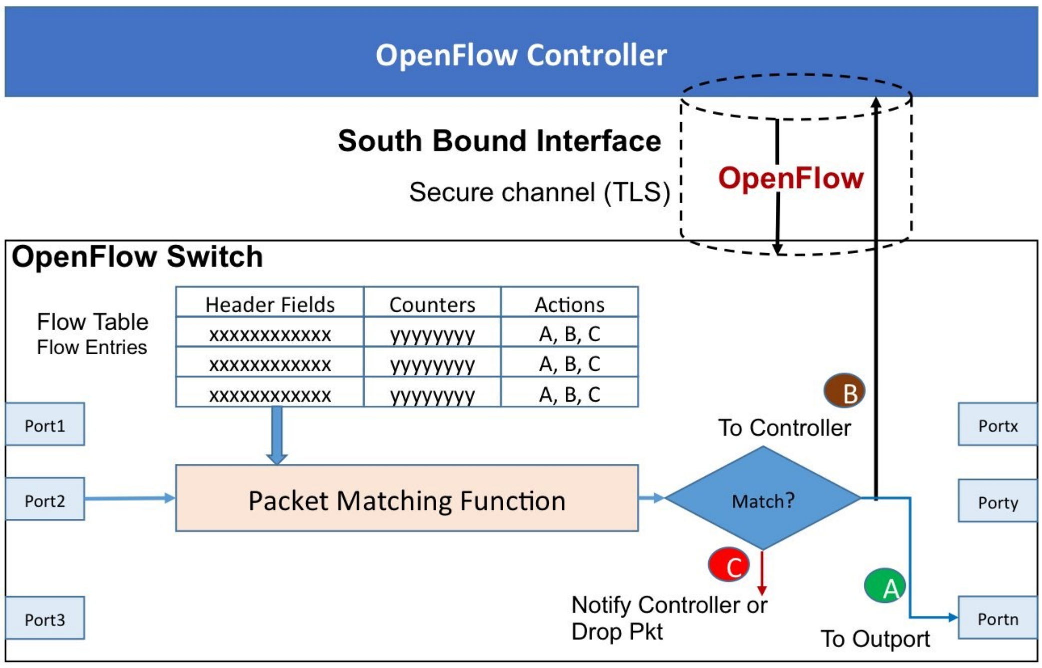

Figure 2 – OpenFlow switch – basic operation

The controller, based on its knowledge of the underlying network topology, device capability, and application requirements, defines a flow with appropriate fields and uses the OpenFlow protocol to program the switch with necessary flow tables and entries.

OpenFlow defines three types of messages:

- controller-to-switch,
- asynchronous, and
- symmetric messages.

Controller-to-switch messages are used to manage and program the switch (e.g., setting switch configuration, sending flow tables, requesting traffic statistics). The asynchronous messages are from the switch to the controller without having been solicited by the controller. They are used to notify the controller of changes in the switch's state and to report network events including errors. The symmetric messages are used by both the switch and the controller for ascertaining the liveness of the connection.

The main functions of an OpenFlow switch include interacting with the controller through the OpenFlow protocol, identifying traffic flows through packet matching, performing packet forwarding, and reporting statistics and switch state to the controller.

When a packet arrives at the OpenFlow switch, it is matched against the flow table to determine whether there is a matching flow entry. The match fields associated with the incoming packet cover layer 2 to layer 4 headers. They include Port (switch ingress port); VLAN (ID and priority); Ethernet (source and destination addresses, and frame type); IP (sources and destination addresses, protocol, type of service); TCP/UDP (source and destination ports). If a matching flow entry is found, the packet may be dropped, modified, or forwarded depending on the instructions and actions associated with that flow (see Figure 2). A table-miss is the term used to describe the situation when no matching entry is found and in this case appropriate actions may include dropping the packet, forwarding it the controller, or continuing to the next table. Readers are referred to (Open Networking Foundation 2013) for additional features.

SDN Controller

The controller is the software that manages all shared resources of the underlying network infrastructure among applications. The controller communicates with the network devices to obtain information about them in order to build the global information state of the underlying network infrastructure, and programs the application-specific configurations and policies onto these devices in order to control their forwarding behaviour. Furthermore, the controller also offers an execution environment for programming of the network (Gude et al 2008). Figure 3 depicts core functional modules and interfaces of an SDN controller.

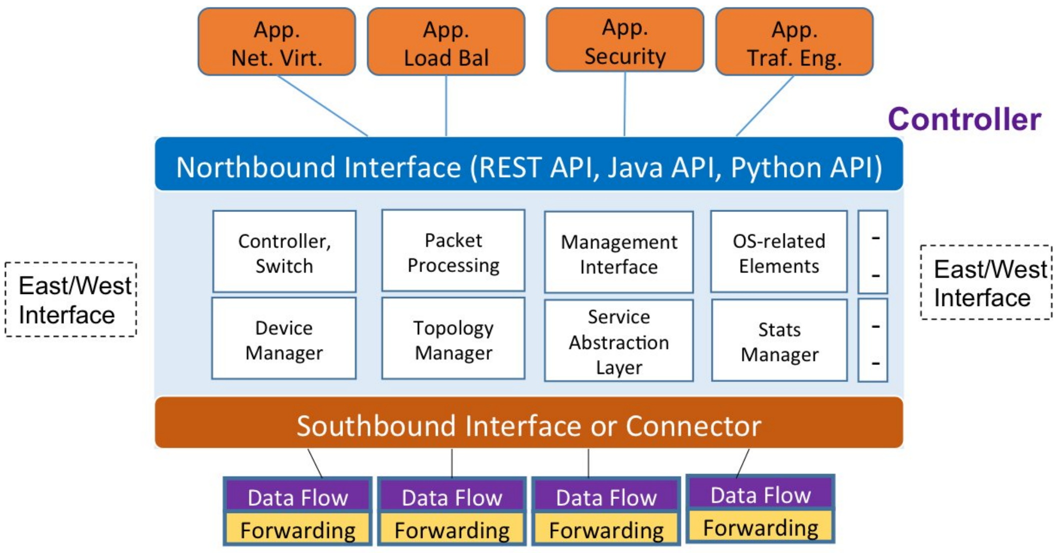

Figure 3 –Components of an SDN controller

Core Component Functions

The basic set of components of the SDN controller may include:

Device Manager. Device Manager manages devices in the network. Device manager registers to the device listeners so that it is notified when a device is added to or removed from the network. It is also notified when the device IP addresses have been added, updated or removed.

Packet Processing Unit. Packet processing processes packet headers and payloads. Each packet contains source MAC addresses, destination MAC addresses, message type, its parent and payload. Usually packets are handled for different protocols such as Ethernet, IPv4, LLDP (Link Layer Discovery Protocol), UDP (User Datagram Protocol), so there are functions to create packets for every protocol that the controller can handle.

Topology Manager. The topology manager identifies topology changes in the network. It sends out the LLDP messages with the switch and its port addresses. If the received LLDP messages match a known switch then a new link is established in the network. The topology manager maintains an up-to-date topology of the network and sends the topology updates to network applications.

Routing. Depending on the protocol the routing manager implements the routes between the source and destination addresses provided.

Openflow Implementation. An OpenFlow module exists in every controller to provide functions related Openflow messages, actions, table entry, flow rules, matching of flow rules, message queues and statistics. In principle, a controller may support other protocols to manage its switches and underlying networks as required by the application.

Controller Interfaces

A controller interacts with the infrastructure layer and the application layer through open interfaces: Southbound and Northbound interfaces.

Southbound Interface or Control/Infrastructure Interface

The interaction between the controller and its network device is realised through a communication channel, within which is an application programming interface (API). A Transport Layer Security (TLS) connection between the device and its controller is often established as the secured communication channel. The southbound API is basically a layer of device drivers. This allows the controller to use different southbound APIs and multiple protocol plug-ins to manage a whole range of physical and virtual devices including SDN devices and legacy devices.

Northbound Interface or Control/Application Interface

The interaction between the controller and the applications can also be realised through a communication interface. However, the communication is more efficient by a software API rather than a formal protocol. This API enables the programmability of the controllers by exposing network abstraction data models and other functionalities for use by applications at the application layer. There is currently no standard northbound API. Various APIs have been implemented in various shapes and forms (REST API, Java API, Python API, etc.) Some present a low-level interface, providing access to the network devices. Others may provide high-level APIs that give an abstraction of the network itself. The controller may use the API to inform the application of events that occur in the underlying network (e.g., state or topology changes). Applications may use the API to respond to a received event and alter the behaviour of the network (e.g., flow modification).

In cases where multiple controllers are deployed to handle cross-domain networks, an East/Westbound interface is required to provide the logically centralised control functionality.

SDN featured applications

Ultimately, business applications and network services are the driving forces behind the development of the network infrastructure. In this section, however, only several applications that highlight specific features of SDN will be discussed: ease of network management, enabling network virtualization in cloud computing context, and embracing network function virtualisation.

Network Management

Today's networking software is typically embedded within the network device, managing network devices requires manual operation on individual network devices. Network management is thus difficult, laborious, error-prone, and costly; and yet it is not flexible to deal with the dynamic nature of network conditions. Software defined networking offers an effective solution to this network management problem. The separation of the control plane from the data forwarding plane allows a centralised network controller to control the overall behaviour of the network. Network policies concerning security, topology, quality of service, etc., can be translated to network management tasks and management commands. The controller then can use an API between the control plane and the data plane to program the devices *autonomously* to obtain the desired network behaviour. The network devices in the data plane can also use the same interface to inform the controller the changes in its status and operating conditions, allowing the controller to respond to these changes by working out appropriate responses and direct the device to adapt to the new situation.

Procera (Kim & Feamster 2013) has been developed as a network control framework using SDN paradigm. Procera has been deployed in Georgia Tech campus and several home networks. Google deployed OpenFlow in their production network (Sushant et al 2013). Operations are scheduled depending on the available bandwidth in the network. Sushant demonstrated that effective resources utilisation was achieved with SDN and OpenFlow. Sivaraman (Sivaraman et al 2013) proposed an architecture for virtualising the access network via open APIs to enable dynamic service quality management.

SDN Network Virtualisation for Cloud Computing

IT resources have been virtualised, provisioned and offered as cloud services over the last few years through a cloud computing model. Openstack is an open-source platform for orchestrating and maintaining clouds. OpenStack creates an abstraction layer above compute, storage, and network resources. This abstraction provides APIs that applications can use to provision virtual resources to a cloud service, independent of the hardware resources. An SDN network virtualisation can be implemented as a component of OpenStack. This component could be an interface to an OpenFlow controller that would control physical/virtual switches (Leon-Garcia et al 2015). An example of network virtualisation is illustrated with the following scenario where a service provider submits a request for a virtual network (VN) to OpenStack (see Figure 4).

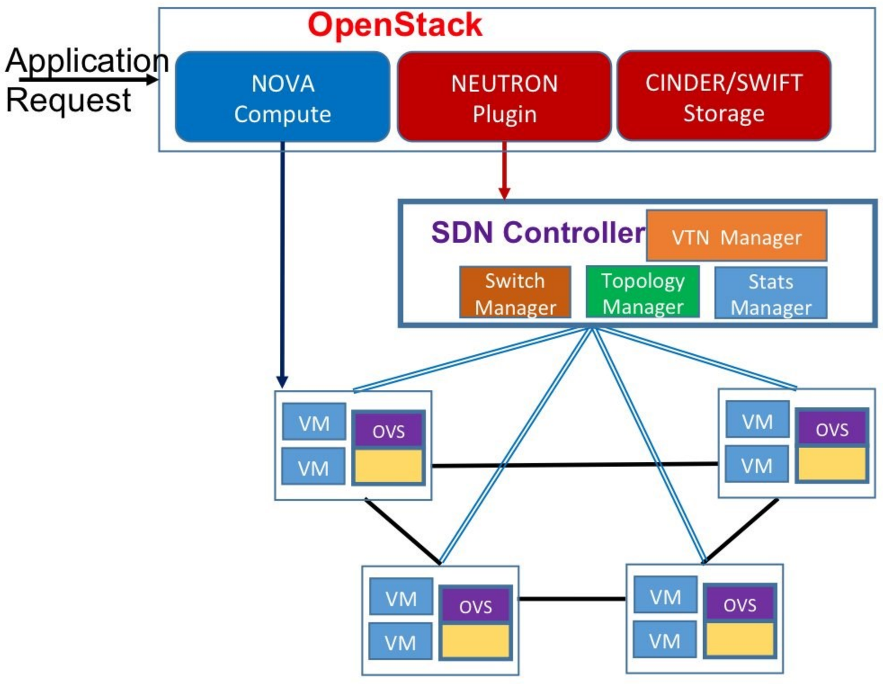

Figure 4 – SDN network virtualisation for cloud computing with open stack

The request specifies the topology of the VN, the resources required for each virtual node (number of virtual machines and virtual switches), and bandwidth requirement for each virtual link. The OpenStack computing orchestrator responds to the request by scheduling the virtual machines (VMs), where virtual network nodes and switches (OVS or Open vSwitch in Figure 4) reside, in the data centre. The OpenStack orchestrator delegates an SDN controller (through its Virtual Network Tenant (VTN) manager) to schedule the VN by sending the request to the SDN controller through its northbound interface. The SDN controller creates and maps the required virtual switches and links to available physical resources. The controller creates and installs forwarding policies in OpenFlow-enabled switches.

Network Function Virtualisation (NFV)

According to the Open Data Center Alliance:

> "NFV is a network architecture concept in which the network is implemented through software, virtualising classes of network node functions. Under the NFV concept, virtualisation technologies are used to implement network node functions on industry-standard commodity hardware, including servers, switches, and storage devices that can be moved to or instantiated in, various locations in the network as required, without the need for installation of new equipment" (Open Data Center Alliance 2013).

Functionality of network appliances and devices such as load balancers or intrusion detection systems, often realised by specialised hardware, can be implemented under NFV. Multicore processors and network interface controllers are powerful enough to be programmed to be many different types of network devices. SDN may be used to implement certain parts of NFV. Usually security firewalls are being implemented in a dedicated hardware component at the network edge. They are now being virtualised and deployed anywhere in a network using OpenFlow-based devices (physical or virtual). Firewalls inspect incoming packets and make a decision to forward them, to drop them, or to steer them to another destination for further analysis. SDN with its programmability and flow matching ability can easily accommodate these firewall actions. Similarly, with SDN and OpenFlow technology, a network device can act as a load-balancing appliance by inspecting incoming packets and forward them to appropriate replica servers.

Discussion

Within less than 50 years the Internet, through its universal connectivity, has reached every corner of the globe with killer applications such as world-wide-web, email, and social network applications. Software-Defined Networking (SDN) has emerged as a networking paradigm that can remove the limitations of current network infrastructures by separating the control plane from the data forwarding plane. As seen from the discussion of featured application of SDN, the significance of the control/data separation includes:

- *Opening up network devices and controller markets*. The controller (implementing the control plane) and the network device (implementing the data plane) can be developed independently by different manufacturers/providers and this opens up their market horizontally for innovation and opportunity.
- *Simplifying network devices*. The network device becomes much simpler as it does not have to deal with complicated and distributed information and decision-making. Inexpensive but high performance switches dedicated to forwarding packets can be developed for applications that need speed such as real-time video streaming.
- *Facilitating autonomous management*. The network can be programmed to a desired configuration to deal with changes in the network conditions such as topology or traffic conditions *autonomously*.
- *Centralising control of network behaviour*. The controller is implemented in software and it controls all the devices within its network infrastructure *centrally*.
- *Enabling network virtualisation*. The controller provides an abstraction of the underlying network infrastructure to the applications, and this provides a mechanism for partitioning the network resources and allocates them to relevant applications/tenants through virtualisation.
- *Embracing network function virtualisation*. Network functions and services can be developed and virtualised and installed from the controller to appropriate devices and network strategic points.
- *Allowing network programmability*. The controller understands the needs of the applications through its northbound interface and resources available of the underlying network infrastructure through its southbound interface. It can program the network for new deployment possibilities.

Software-defined networking promises to be a disruptive technology that revolutionises the Internet with the ability to generate and schedule huge number of virtual networks on demand, to program and to manage them autonomously, and to support business applications and network services. Whether the potential of this technology is realised depends on several factors: the ease at which innovative and emerging business applications and services can be developed over the north bound interface; and the specific SDN-based approach that allows cost-effective transition to the new networking paradigm. These two issues are discussed below.

Development of Intent Northbound Interface.

Innovation and rich applications are recognised as the driver of the revolutionised Internet. However, this happens only if the developers are free to concentrate on developing their applications rather than being burdened by the complexity of networking. This is where a standardised Northbound API is needed. The debate is whether one API is adequate for all types of applications. Clearly, network services such as network virtualisation and load balancing are closely linked to characteristics of networking and require intimate knowledge of network resources such as traffic load, routing scheme, link bandwidth. Other business applications make use of the underlying network for simple connectivity without strict requirements on network resources. Recently, it is realised that it is *unreasonable* or even *unproductive* to require an application to express its requirements of network resources in a **network-specific language**. The application needs not to know how the network is being deployed to satisfy its needs. The application, however, knows exactly its objectives, its policies, and its requirements and constraints. An NBI should be designed to reflect this understanding in order to support a large class of applications and services. The intent-based networking is in that spirit and the promising **Intent Northbound Interface** (NBI) is being developed (Janz 2015) by the Open Networking Foundation (Figure 5).

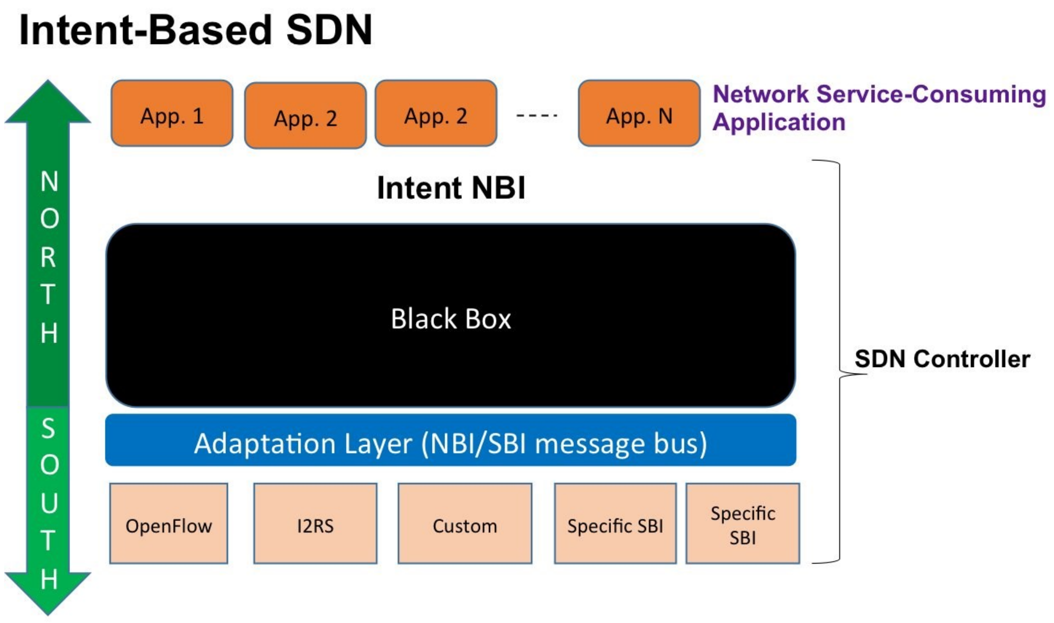

Figure 5 – Intent Northbound Interface in intent-based SDN (Janz 2015)

It is for an application to express its “intent” and for the control plane (controllers) to translate the intent to the network setting and resources necessary to satisfy the needs of the application. The intent is for the application to express its request in terms of information and directives in a simple and common language, and for the controller to obtain adequate inputs to map the request down to network specific instructions. As it is difficult to translate diverse application needs, expressed in application-specific language, to the expressions the controllers can understand and function, it is suggested that that API be composed of iterative levels of refinement until the application needs can be decomposed into a set of base API functions for the controller. A group of support such as dictionary database, negotiation procedure, and compiler may be needed at various levels of refinement.

Cost-effective SDN-based approaches?

The concept of SDN can be accomplished in other ways than the OpenFlow-centric approach coupled to Open SDN as discussed so far, but which would be the most beneficial for the industry?

SDN via Existing APIs.

With this approach, the behaviour of the network can be manipulated by sending management commands to the network devices to configure them. Existing CLI, SNMP or RESTful API or their enhanced versions can be used to program legacy switches but they do not allow fine-grain control of individual flow. This approach requires no controller or upgrading existing switches to OpenFlow-enabled switches. However, it does not solve all the problems that can be addressed by an Open SDN solution. The network programmer still has to interact directly with *each individual switch*. The approach does not revolutionise the networking industry as most issues remain unaddressed.

SDN via Hypervisor-Based Overlay Networks.

With this approach, virtualised overlay networks are built on top of and without modifications to the underlying physical network. An overlay network interconnects virtual machines (nodes) by virtual links (tunnels between two end points of the overlay network). The data sent by a virtual machine through a virtual link is encapsulated using IP-based tunnelling protocol. The tunnelling mechanism is referred to as MAC-over-IP tunnelling whereby the whole MAC frame is encapsulated within an IP packet. The network edge switches (virtual switches) will serve as virtual tunnel endpoints of these virtual networks. Each node in the overlay network knows its adjacent edge switch through a central overlay controller. Cisco offers Virtual eXensible Local Area Network (VXLAN), Microsoft uses Network Virtualization using Generic Routing Encapsulation (NVGRE), and Nicira offers Stateless Transport Tunnelling Protocol (STT) (Goransson & Black 2014). This approach provides an effective way for data centres to extend their VNs over and cross-domain data centres without altering the underlying physical infrastructure. However, it does not to address issues in the physical infrastructure such as programmability and automation. It still requires manual configuration.

Conclusion

SDN makes the network open and programmable, allowing new capabilities and services to be simply created on demand. It has the potential to be a disruptive technology in many market segments: network device vendors, service and infrastructure providers, network operators, and data centres. This paper provides a high-level understanding of software-defined networking through its concepts, architecture, and interfaces. It highlights the features and implications of the Technology with several SDN-based applications. Finally it discusses two major issues that may help to bring the technology to the next step: the intent northbound interface and the cost-effective approaches for the industry.

References

Berde, P; Gerola, M; Hart, J; Higuchi, Y; Kobayashi, M; Koide, T; Lantz, B; O'Connor, B; Radoslavov, P; Snow, W; Parulkar, G. 2014. 'ONOS: Towards an Open, Distributed SDN'. Proceedings of HotSDN'14. August; Chicago, USA.

Feamster, N; Rexford, J; Zegura, E. 2013. 'The Road to SDN: An Intellectual History of Programmable Networks'. ACM Queue, Technical Report, New York, USA.

Goransson, P; Black, C. 2014. *Software Defined Networks - A Comprehensive Approach*. Morgan Kaufmann.

Greenberg, A; Hjalmtysson, G; Maltz, D; Myers, A; Rexford, J; Xie, G; Yan, H; Zhen, J; Zhang, H. 2005. 'A clean slate 4D approach to network control and management'. *Computer Communication Review*, Volume 35, Issue 5, 41–54.

Gude, N; Koponen, T; Pettit, J; Pfaff, B; Casado, M; McKeown, N; Shenker, S. 2008. 'NOX: Towards an operating system for networks'. *Computer Communication Review*, Volume 38, Issue 3, 105–110

Janz, C. 2015. 'Intent NBI – Definition and Principles'. Open Networking Foundation, Version V2, July 2015.

Kim, H; Feamster, N. 2013. 'Improving Network Management with Software Defined Networking'. *IEEE Communications Magazine*: 114-119.

Leon-Garcia, A; Bannazadeh, H; Zhang, Q. 2015. 'OpenFlow and SDN for Clouds'. *Cloud Services, Networking, and Management*, Chap. 6. Editors: da Fonseca N; R. Boutaba R. IEEE Press.

Open Networking Foundation. 2012. White Paper. 'Software-Defined Networking: The New Norm for Networks'. Palo Alto, USA.

Open Networking Foundation. 2013. 'OpenFlow Switch Specification'. Version 1.4.0

Open Data Center Alliance. 2013. 'Master Usage Model: Software-Defined Networking'. Rev. 2.0.

Sushant J; Kumar, A; Mandal, S; Ong, J; Poutievski, L; Singh, A; Venkata, S; Wanderer, J; Zhou, J; Zhu, M; Zolla, J; Hölzle, U; Stuart, S; Vahdat, A. 2013. 'B4: Experience with a Globally-Deployed Software Defined WAN', Proceedings of the ACM SIGCOMM. August 2013; Hong Kong, China, 2-14

Sivaraman, V; Moors, T; Habibi Gharakheili, H; Ong, D; Matthews, J; Russell, C. 2013. 'Virtualizing the access network via open APIs'. Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. December 2013; Santa Barbara, CA, USA, 31-42