Implementing Software - Defined Networking (SDN) in a Campus Environment

L.Naga Durga Prasad Reddy
Information technology
Vel tech multi tech Dr.Rangarajan & Dr.Sakunthala engineering college
Chennai,India

B. Sivakumar
Asst.Prof
Information technology
Vel tech multi tech Dr.Rangarajan & Dr.Sakunthala engineering college
Chennai,India

***Abstract* - This paper researches in the area of software defined networking. Software Defined Networking was developed in an attempt to simplify networking and make it more secure. By separating the control plane (the controller)—which decides where packets are sent—from the data plane (the physical network)—which forwards traffic to its destination—the creators of SDN hoped to achieve scalability and agility in network management. The application layer (virtual services) is also separate. SDN increasingly uses elastic cloud architectures and dynamic resource allocation to achieve its infrastructure goals.**

I. INTRODUCTION

In the early days of basic Internet protocols development no native support for access control was provided at the network level. It was expected that applications would connect to each other in the global network without any restrictions. Along with the growth of commercial use of Internet mechanisms for L3 (and higher) network access control became necessary for normal operations, and packet filtering solutions were developed (including software implementations in operating systems) — firewalls, Intrusion Prevention Systems (IPS), network antiviruses, application
layer proxy servers, including WAF — web application firewalls.

Unfortunately, nowadays with evolved networking technologies, enormous growth of Internet throughput and a shift from fixed client devices towards mobile networking (we have over 1 billion connected smartphones already in early 2013, and only about 200 million fixed devices) efficiency of existing access control solutions reduces. More expensive devices are required for every new version of Ethernet protocol providing the same level of network granularity as five or ten years ago. In terms of client devices mobility, network configuration is changing rapidly and the information about network topology changes could not be used directly for access control. That is why the problem of network access control based on the information about the expected behavior (flows) of network applications is becoming more and more important. manages everything in a network. SDN is a step in the evolution towards programmable and active networking. SDN allows network administrators to have programmable central control over the entire network.

II. SOFTWARE DEFINED NETWORKING

Software-defined networking (SDN) is a new emerging technology for networking in which control is Decoupled a hardware and given to software part called a controller[1]. When a packet arrives at a switch in a foreseeable network rule built into the switch patented firmware tell the switch where to forward the packet. The switch sends every packet going to the same destination along the same path and treats all the packets the exact same way. In the campus network, smart switches designed with application-specific integrated circuits (ASICs) are Refined enough to recognize different types of packets and treat them differently, but such switches can be quite expensive.

The aim of SDN is to allow network administrators respond quickly to changing to the requirements. In a software-defined network, a network administrator can shape traffic from a centralized control software without having to touch individual switch. The administrator can change any network switch rules when necessary ordering, de-ordering or even blocking specific types of packets with a very level gritty of control. This is especially helpful in a cloud computing multi-tenant architecture because it allows the administrator to manage traffic loads in a flexible and more efficient manner. Essentially, this allows the administrator to use less expensive, commodity switches and have more control over network traffic flow than ever before. Currently, the most popular specification for creating a software-defined network is an open standard called Open Flow. Open Flow lets network administrators remotely control routing tables.

A . *Open Flow Switches*

We exploit the fact that most modern Ethernet switches and routers contain flow-tables that run at line-rate to implement firewalls, NAT, QoS, and to collect statistics. [5] While each vendor's flow-table is different, we've identified an interesting common set of functions that run in many switches and routers. OpenFlow exploits this common set of functions. OpenFlow provides an open protocol to program the flow table in different switches and routers. A network administrator can partition traffic into production and research

fellows. Researchers can control their own flows by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes, and even alternatives to IP. On the same network, the production traffic is isolated and processed in the same way as today. The data path of an OpenFlow Switch consists of a Flow Table, and an action associated with each flow entry. The set of actions supported by an OpenFlow Switch is extensible, but below we describe a minimum requirement for all switches. For high-performance and low-cost the data-path must have a carefully prescribed degree of flexibility. This means forgoing the ability to specify arbitrary handling of each packet and seeking a more limited, but still useful, a range of actions. Therefore, later in the paper, define a basic required set of actions for all Open Flow switches. An Open Flow Switch consists of at least three parts: A Flow Table, with an action associated with each flow entry, to tell the switch how to process the flow.

- A Secure Channel that connects the switch to a remote control process allowing commands and packets to be sent between a controller and the switch.
- The OpenFlow Protocol, which provides an open and standard way for a controller to communicate with a switch.

By specifying a standard interface the OpenFlow Protocol through which entries in the Flow Table can be defined externally, the OpenFlow Switch avoids the need for researchers to pro-gram the switch. It is useful to categorize switches into dedicated OpenFlow switches that do not support normal Layer 2 and Layer 3 processing, and OpenFlow-enabled general purpose commercial Ethernet switches and routers, to which the Open Flow Protocol and interfaces have been added as a new feature.

B. OpenFlow Controllers

The open Flow controller is a bid that manages flow mechanism in a software-defined networking (SDN) environment. The present SDN controllers are based on the Open Flow protocol where SDN controller serves as a sort of operating system (OS) for the network. Entirely communications between applications and devices have to go through the controller. The OpenFlow protocol connects controller software to network devices so that server software can tell switches where to send packets. The controller uses the OpenFlow protocol to configure network devices and choose the best path for application traffic. Because the network control plane is implemented in software, rather than the firmware of hardware devices, network traffic can be managed more dynamically and at a much more granular level.

III NETWORK MANAGEMENT PROBLEMS

In numerous respects the large enterprise networks of today are reminiscent of the islands of automation that were common in manufacturing during the 1980s and 1990s the experiment in front of manufacturers was in linking together the islands of microprocessor-based controllers, PCs, minicomputers, and other components to allow end-to-end actions such as aggregated order entries leading to automated production runs. This optimism be located islands of automation could be joined so that the previously isolated intelligence could be leveraged to manufacture better products. Comparable difficulties harassed network operators at the beginning of the 21st century as traffic types and volumes continue to grow. In the similar with this the range of deployed NMS is also growing. Numerous NMS adds to operational expense. To hand a strong needs to reduce the cost of ownership and improve the return on investment for network equipment. The factual not just during periods of economic downturn, but has become the norm as service level agreement are applied to both enterprise and SP networks. Network management system technology provides the network operator with some increasingly useful capabilities. Unique way from error-prone, tedious, physically intensive operations to software assisted automated end to end operations.The network operators needed to execute automated end to end management operations taking place their networks. An sample of this is Virtual local area ne management in which an NMS GUI provides a visual picture such as a cloud of MAC addresses, VLAN members ports, VLAN IDs. The network management system can also provide the ability to easily delete, modify and add virtual local area network members as well as indicate any faults as and when they happen. An additional example is enterprise WAN management in which ATM or FR virtual circuits are used to carry the traffic from branch offices into central sites. In this situation the enterprise network manager wants to be able to easily create, delete, modify, and view any faults on the virtual circuit to the remote places. Supplementary examples include storage including SANs management and video/audio conferencing equipment management.[8] The advantage of this type of end to end competence is a large discount in the cost of managing enterprise networks by SLA fulfillment less need for arcane NE know how smooth enterprise business processes and happy end users. Exposed purveyor independent NMS are needed for this and later we look at ways in which software layering helps in designing and building such systems. Humble ideas such as always using default MIB values pragmatic database design and technology sensitive menus also play an important part in providing NMS vendor independence. The problems presenting menu options appropriate to a given selected NE provides abstraction for example if the user wants to add a given NE interface to an IEEE 802.1Q VLAN then that device must support this frame tagging technology. The Network management system should be able to figure this out and present the option only if the underlying hardware supports it. Through awarding only appropriate options the NMS reduces the amount of data the user must sift through to actually execute network management actions.

Automated flows through actions are required for as many network management operations as possible including the following FCAPS areas:

• Provisioning
• Detecting faults
• Checking performance
• Billing/accounting
• Originating repairs or network upgrades
• Maintaining the network inventory

A more and more common problem faced by network management system software developers is implementing network management support for new NE features, such as MPLS. The following four steps provide a linked overview of the management aspects of an NE feature:

- Obtain a basic understanding of the managed technology what it does how it operates and so on. Designed for MPLS the basic purpose is to carry traffic across an MPLS core so connections LSPs and tunnels are important. These tunnels can be traffic-engineered and can also support QoS, so path and resource blocks are needed.

- Use the energy management system to get an understanding of how the NE provides the feature for example for MPLS the user can separately create objects like paths and resource blocks. Then, these can be combined to create MPLS LSPs or tunnels. User manuals are often very useful during this process.

- Vision the germane MIBs using a MIB browser.

- Write simple test code to view and modify the MIB objects.

Step 4 essentially automates the actions in steps 2 and 3. The software produced in step 4 can then form the basis of Java classes that can eventually be imported into the finished NMS. The final stage in development is then adding code to the NMS to implement the overall MPLS management feature. A significant observation about the above is that it depicts NMS development as a type of reverse engineering. The network management is provided at the end of NE development then it has a reverse engineering flavor. If the two occur in parallel then no real reverse engineering effort is required. As a result view a linked overview as the resulting knowledge emanating from following the above four steps.

Overview of network problems

At hand some serious problems affecting network management. A taking beside managed data and code together is one of the central foundations of computing and network management. Accomplishing this union of data and code in a scalable fashion is a problem that gets more difficult as networks grow. Management information base tables expand as more network resident managed objects such as virtual circuits are added. Our main Management information base note records a useful object that can help in managing additions to large Management information base tables. The grander than before size networks is matched by ever more dense devices. Ending both help and hinder operators. The originators of management systems need a rarified skill set that matches the range of technologies embedded in NEs and network supplementary emphasis is needed on solutions than on technology particularly as the components of the technology are combined in new and complex ways for instance in layer 2 and layer 3 VPNs. Resolutions should try to hide as much of the underlying network complexity as possible. Network management system technology can help in hiding unnecessary complexity. The liberal use of standards documents and linked overviews are some important tools for tackling the complexity of system development managed object derivation and explanation. Ethics documents can be used in conjunction with UML to inform and open up the development process to stakeholders.

IV LIST OF OPEN FLOW SOFTWARE'S

This paper suggest that Ns3 is the best simulation tool for making you research on implementation of SDN & too for making the perforns analysis on SDN.

Switch Software and Stand-Alone OpenFlow Stacks

- **Open vSwitch**: (C/Python) Open vSwitch is a an OpenFlow stack that is used both as a vswitch in virtualized environments and has been ported to multiple hardware platforms. It is now part of the Linux kernel (as of 3.3).

- **OpenFlow Reference**: (C) The OpenFlow reference implementation is a minimal OpenFlow stack that tracks the spec.

- **Pica8**: (C) An open switch software platform for hardware switching chips that includes an L2/L3 stack and support for OpenFlow.

- **Indigo**: (C) Indigo is a for-hardware-switching OpenFlow implementation based on the Stanford reference implementation.

- **Pantou**: (C) Pontou is an OpenFlow port to the OpenWRT wireless environment.

- **OpenFaucet**: (Python) OpenFaucet is a pure Python implementation of the OpenFlow 1.0.0 protocol, based on Twisted. OpenFaucet can be used to implement both switches and controllers in Python.

- **OpenFlowJ**: (Java) OpenFlow stack written in Java.

- **Oflib-node**: (Javascript) Oflib-node is an OpenFlow protocol library for Node. It converts between OpenFlow wire protocol messages and Javascript objects.

- **Nettle**: (Haskell) OpenFlow library written in Haskell.

Controller Platforms

- **POX**: (Python) Pox as a general SDN controller that supports OpenFlow. It has a high-level SDN API including a queriable topology graph and support for virtualization.
- **MUL**: (C) MūL, is an openflow (SDN) controller. It has a C based muli-threaded infrastructure at its core. It supports a multi-level north bound interface for hooking up applications. It is designed for performance and reliability which is the need of the hour for deployment in mission-critical networks.
- **NOX**: (C++/Python) NOX was the first OpenFlow controller.
- **Jaxon**: (Java) Jaxon is a NOX-dependent Java-based OpenFlow Controller.
- **Trema**: (C/Ruby) Trema is a full-stack framework for developing OpenFlow controllers in Ruby and C.
- **Beacon**: (Java) Beacon is a Java-based controller that supports both event-based and threaded operation.
- **Floodlight**: (Java) The Floodlight controller is Java-based OpenFlow Controller. It was forked from the Beacon controller, originally developed by David Erickson at Stanford.
- **Maestro**: (Java) Maestro is an OpenFlow "operating system" for orchestrating network control applications.
- **NDDI - OESS**: OESS is an application to configure and control OpenFlow Enabled switches through a very simple and user friendly User Interface.
- **Ryu**: (Python) Ryu is an open-sourced Network Operating System (NOS) that supports OpenFlow.
- **NodeFlow** (JavaScript) NodeFlow is an OpenFlow controller written in pure JavaScript for Node.JS.
- **ovs-controller** (C) Trivial reference controller packaged with Open vSwitch.

Special Purpose Controllers

- **RouteFlow** RouteFlow, is an open source project to provide virtualized IP routing services over OpenFlow enabled hardware. RouteFlow is composed by an OpenFlow Controller application, an independent RouteFlow Server, and a virtual network environment that reproduces the connectivity of a physical infrastructure and runs IP routing engines (e.g. Quagga).
- **Flowvisor** (Java) FlowVisor is a special purpose OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers.
- **SNAC** (C++) SNAC is an OpenFlow controller builton NOX, which uses a web-based policy manager to manage the network.
- **Resonance** Resonance is a Network Access Control application built using NOX and OpenFlow.
- **Oflops** (C) OFlops (OpenFLow Operations Per Second) is a standalone controller that benchmarks various aspects of an OpenFlow switch.

Misc

- **STS** SDN Troubleshooting Simulator.
- **FlowScale** FlowScale is a project to divide and distribute traffic over multiple physical switch ports. FlowScale replicates the functionality in load balancing appliances but using a Top of Rack (ToR) switch to distribute traffic.
- **NICE-OF** NICE is a tool to test OpenFlow controller application for the NOX controller platform.
- **OFTest** OFTest is a Python based OpenFlow switch test framework and collection of test cases. It is based on unittest which is included in the standard Python distribution.
- **Mirage** Mirage is an exokernel for constructing secure, high-performance network applications across a variety of cloud computing and mobile platforms. Apparently, it supports OpenFlow.
- **Wakame VDC** (Ruby) IaaS platform that uses OpenFlow for the networking portion.
- **ENVI** ENVI is a GUI framework that was designed as an extensible platform which can provide the foundation of many interesting OpenFlow-related networking visualizations.
- **NS3** (C++/Python) NS3 is a network simulator. It has openflow support built in to emulate an openflow environment and also it can be used for real-time simulations.

IV SDN STRUCTURE

The logical structure of a Software defined networking is explained as below. The main controller gets done all difficult functions, together with security checks, naming, course of action assertion, and routing. This plane set up the SDN Control Plane and be thru of one or more SDN servers. A SDN Controller describes the data flows that follow on SDN Data Plane. The bit flow through the network must

first get permission from the controller verifies that the communication is permissible by the network guiding principle.

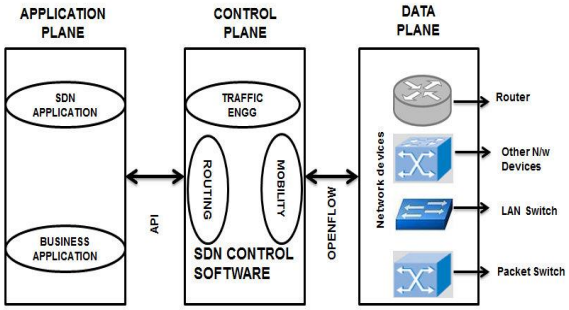

Fig. 1. SDN Structure

A controller allows a flow the situation computes a route for the flow to proceeds and adds an entry for that flow in each of the switches along the path. By way of means all complex functions subsumed by the controller switches simply manage flow tables whose entries can be populated individual by the control and communiqué stuck between the controller and the switches uses a standardized protocol and API. Utmost commonly this interface is the OpenFlow specification conversed consequently.SDN architecture is extraordinarily flexible. [7] It can operate with different types of switches and at different protocol layers. Software defined networking controllers and switches can be implemented for Internet routers, Ethernet switches, transport switching, or application layer switching and direction-finding. SDN be dependent on the common functions found on networking devices which essentially involve forwarding packets based on some form of flow definition. SDN architecture, a switch performs the following functions:

- Switch reduces and straight on the first packet of a flow to an SDN controller, facilitating the controller to decide whether the flow should be added to the switch flow table.
- Switch forwards received packets out the suitable port based going on flow table the flow table might include importance information dictated by the controller.
- Switch can drop packets on a specific flow in the short term or permanently, as dictated by the controller. Packet dipping can be used for security purposes limit Denial-of-Service attacks or traffic management necessities.
- In inconspicuous terms the SDN controller manages the forwarding state of the switches in the Software Defined networking.

The controlling is done through a vendor-neutral API that allows the controller to address a wide variety of operator requirements without changing any of the lower-level aspects of the network, including topology. The decoupling of the control and data planes Software defined networking enables applications to deal with a single abstracted network device without concern for the details of how the device operates. The network solicitations see a single API to the controller.

As a result it is possible to quickly create and deploy new applications to orchestrate network traffic flow to meet specific enterprise requirements for performance or security.Motives for using Software defined networking domains include the following:

Scalability: The numeral of devices an SDN controller can feasibly manage is limited. Thus, a reasonably large network may need to deploy multiple SDN controllers.

Privacy: A transferor may choose to implement different privacy policies in different SDN domains. For example, an SDN domain may be dedicated to a set of customers who implement their own highly customized privacy policies, requiring that some networking information in this domain not be disclosed to an external entity.

Incremental deployment: A transporter's network may consist of portions of traditional and newer infrastructure. Dividing the network into multiple, individually manageable SDN domains allows for flexible incremental deployment.

V IMPLEMENTATION AND PROCESS

Fig. 2 depicts the SDN architecture. There is a centralized SDN block which is a software governed protocol that manages the entire network. It is connected to the OpenFlow Controller which in turn are connected to other OpenFlow switches, routers and other networking components. It uses a unique protocol known as “Open Flow Protocol” that controls and manipulates the entire network. The open flow controller controls the open flow switches. These switches monitor the networks performance and sends periodical reports to the Network Administrator as per his preference. In case of any network failures or node failures in the network this Open Flow Controller will trigger an alert message to the network administrator.

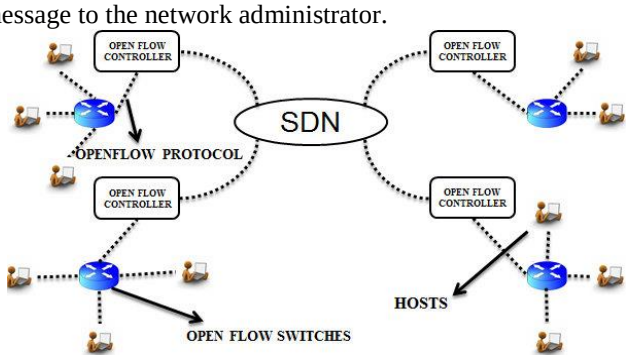

Fig. 2. SDN Design Architecture

It also has the self healing process but if the issue goes beyond the scope of the Open Flow Control protocol then an alert message is triggered to the administrator. This whole process can be manipulated by the Network administrator according to the environment of the network or

according to the outcome required. This structure is applicable from a small home network to a complex commercial network.

IV. CONCLUSION

This paper deals with the identification of current network problems and inaddition to that the theoretical implementation of SDN technology to enhance the performance of network management. SDN benefits the Networks in many ways. It mainly simplifies the job of a Network Engineer and it also renders ubiquitous support of the network related issues. We can also set preferences like controlling the bandwidth, restricting access according to time, Probing for security issues, reporting a bug, healing the problems automatically and many more complex functions thereby reducing the work of a human labour.In this paper we have given the indepth idea about SDN and we have also shown the design architecture of SDN. As a further work the SDN will be implemented in the network which we designed and the performance analysis is done through network stimulator (Ns3).

REFERENCES

[1] Controlling an SDN Via Disturbed Controller, by Europen Commission Number PIRG06-GA-2009-256326, Volkan yazici, Ozyegin University.
[2] Improving Network Management With Software Defined Networking, by Hyojoon Kim and nick feamster, 0163-6804/13/ IEEE Communication magazine 2013.
[3] Software Defined Networking Overview And Implementation, by Manal Algarni, Vinayak Nair, David Martin. George Mason University IFS 2013.
[4] Searchsdn.techtarget.com/definition/OpenFlow-controller. March 2013.
[5] OpenFlow enabling innovation in campus networks, by Tom Anderso University of Washington, hari Balakrishnan MIT, Larry Peterson Princeton university,march 2008 Openflow switch.org.
[6] http://yuba.stanford.edu/~casado/of-sw.html
[7]Software-Defined Networks and OpenFlow - The Internet Protocol Journal, Volume 16, No. 1 - Cisco Systems.
[8]http://www.pearsonhighered.com/samplechapter/0131011138.pdf