A Holistic Presentation and recommendation of OpenFlow Its Challenges and Future Research Needed

Javier Coto

Abstract- OpenFlow could define flows and determine how packets are prioritized and forwarded through switches, reduce power consumption, and redesign data centers. You will find a holistic research of the current innovations, benefits, and future researched need in OpenFlow.

Index Terms- OpenFlow, New protocols for OpenFlow, How DDoS can be reduce through OpenFlow, Reduce datacenter power consumption through OpenFlow , Improved OpenFlow scalability with Ca-SDN, Implementing OpenFlow in a Wireless mesh network

I. Introduction

OpenFlow is one of the most important building blocks for software-defined networking (SDN). OpenFlow moves the control of the switch like routing to a centralized server, instead of the switch. With this a network can be programed allowing the system to be more flexible, and dynamic. With a centralized controller cloud computing power can be used.

OpenFlow can also be used to conserve power by turning off switches and ports off in peak hours, like at night. In data centers this reduces power consumption directly and indirectly in regards to cooling. With more demand for data centers and green computing OpenFlow provides and good solution.

The current solution to OpenFlow's issues will be presented here, many of the proposed solutions still raise more questions and research is needed. As you will read some solutions themselves create greater overhead that cancel out some of the benefits.

This rest of this paper is organized as follows. Section 2 looks at the Origins and Future of OpenFlow. Section 3 presents Power Consumption strategies to reduce carbon footprint. Section 4 takes a very important look into scalability. Section 5 explores the benefits of using the power of the cloud. Section 6 presents experiment with wireless mesh networks. Implementing and modifying OpenFlow headers for mobility. Section 7 shows how optical networks are taking advantage of OpenFlow-based control. Section 8 discusses the research environment and Tools to foster new innovations. Section 9 looks at domain-specific languages for OpenFlow. The final Section 10 uses OpenFlow to solve detection problems of denial of service attacks (DDoS).

II. Origins and Future

The origins of OpenFlow came from Martin Casado, a PhD student at Stanford University in 2006. Casado developed Ethane witch would later become OpenFlow.

2.1 Ethane

Casado 2006 paper [1] describes ethane as:

Ethane controls the network by not allowing any communication between end-hosts without explicit permission. It imposes this requirement through two main components. The first is a central Controller containing the global network policy that determines the fate of all packets. The second component is a set of Ethane Switches In contrast to the omniscient Controller, these Switches are simple and dumb. Consisting of a simple flow table and a secure channel to the Controller

This idea eventually leads to OpenFlow after joint research with Stanford and the University of Berkley.

2.2 Open Networking Foundation (ONF)

In February 2011 the Open Network Foundation was established, by Google , Facebook and Microsoft. It now includes many more members, like Cisco, Dell, HP, IBM, and many more.

ONF now oversees and retains controls over the specifications. In April 25, 2013 the latest version 1.3.2 was released [2].

III. Power Consumption

Large networks are typically provisioned for peak workloads, but the variation of workload varies greatly by day, week, or month. At night time the networks load could be 50% the load during the day [17]. In December the load will be higher than any other month due to Christmas and online shopping.

Figure 1 shows peak traffic during the day and night. Even though the traffic varies significantly with time, the rack, and aggregation switches associated with the 292 servers hosting an e-commerce application, the server draws constant power [9].

Power can be conserved by powering down switch or individual ports. One approach is Multilayer Traffic Engineering (MLTE) using adaptive link rates (ALR) and burst mode operation and another is Elastic tree. These are two ways propose to reduce power consumption

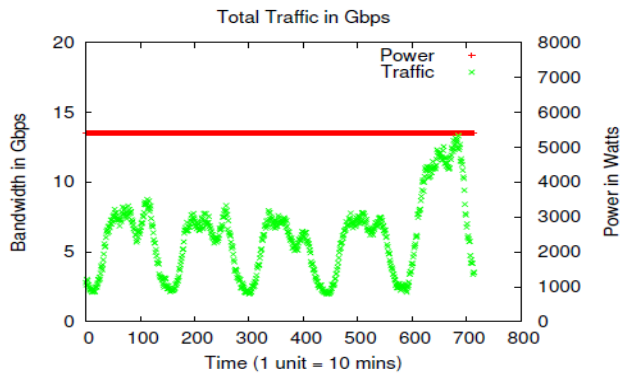

Fig. 1: Ecommerce website application Bandwidth and watts graph

3.1 MLTE with adaptive link rates and burst mode operation

MLTE can lead to power savings of 50% [9] Figure 2 illustrates how MLTE works

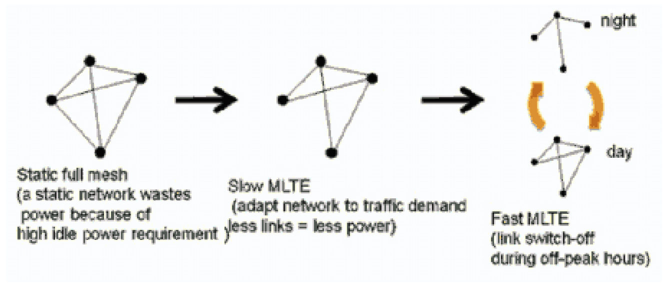
night
day
Static full mesh
(a static network wastes power because of high idle power requirement)
Slow MLTE
(adapt network to traffic demand less links = less power)
Fast MLTE
(link switch-off during off-peak hours)

Fig. 2: Multilayer traffic Engineering

Since OpenFlow currently has limited support for the control of power in the switches adaptive link rates and burst mode operation are used to reduce power on the device [21].

Adaptive Link Rates is based on the principle that lower link rates lead to lower power consumption in the network equipment. In burst Mode packets are buffered in a network node and then sent over the link at the maximal rate. In between the bursts the line can be powered down. This works in a very small time scale, so the number of links that can be turned off is limited. This can be implemented by using OpenFlow Header OFPT_* (featured_request , featured_reply, port_mod, port_down) or Open flows OFPC_BURST_MODE header [17, 19]

3.2 Elastic Tree

Elastic tree suggest having router that can be put in sleep mode to be efficient at low loads. It states it can reduce energy consumption by 60% when demands are low. Elastic tree uses open flow to control dynamic routing of flows and measure traffic matrix [9].

IV. Scalability

As OpenFlow uses a single controller that is centrally controlled, naturally question of scalability comes up. Several concerns related to scalability are: the amount of control traffic destined towards the centralized controller grows with the number of switches. If the network has a large diameter, no matter where the controller is placed, some switches will encounter long flow setup latencies. Since the system is bounded by the processing power of the controller, flow setup times can grow significantly as demand grows with the size of the network [11]. Cloud-assisted Software-defined Networking (Ca-SDN) can address the last problem but comes with other scalability issues. Some proposed solutions addressing scalability as well as other issues are, HyperFlow, DevoFlow but each has certain limitations. Another possible solution is Flowvisor. We will look at the benefits and challenges of each.

4.1 HyperFlow

HyperFlow is a distributed event-based control plane for OpenFlow. It is logically centralized but physically distributed, this gives it the ability to be scalable but retain the benefits of a centralized controller. HyperFlow does not require any change to the OpenFlow standard [22].

HyperFlow is implemented on top of NOX, NOX controllers will each be running an instance of the HyperFlow controller application. Each controller will have an event propagation system for cross-controller communication. Every controller operates as if it is in control of the entire system [11]. Figure 3 illustrates the High-level Overview of HyperFlow.

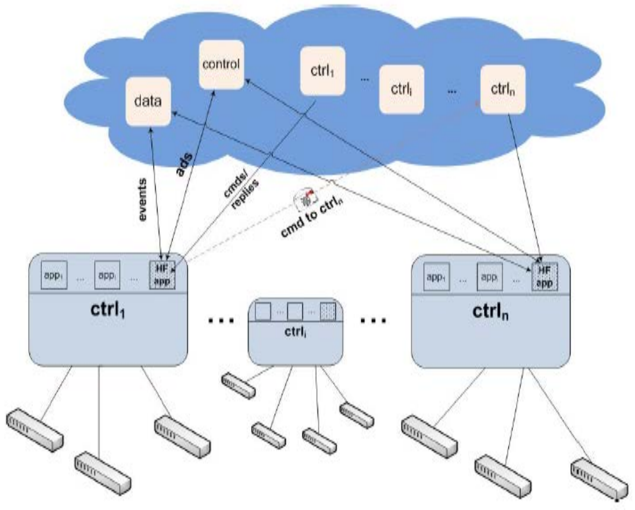

Fig. 3 HyperFlow Overview

HyperFlow's application uses a publish / subscribe system to let each controller achieve a constant network-wide view. Each controller publishes events that change the state of the system, while other controllers replay all published events to reconstruct the state.

HyperFlow uses WheelFS a distributed file system design [23] to propagate events. As illustrated in figure 3 each controller subscribes to three channels, the data channel, the control channel, and its own channel. All controllers can publish or subscribe to all channels. The data channel has the local network and applications events. The control channel is use to facilitate controller discovery and failure detection, each controller will periodically advertise itself to it. The controllers own channel is used for events and OpenFlow commands.

HyperFlow has several limitations. One, with applications that relies on temporal events, since different controllers perceive events in different orders. Two applications that query the switches perform poorly with HyperFlow. An example is discovery applications, they will need to be modified to use OpenFlow instead of protocols like LLDP.

Lastly and must important as regard to scalability. As HyperFlow uses WheelFS for event propagation. WheelFS has certain limitations with read speeds; it can read and deserialize 987 files [22]. This limitation can only guarantee abounded window of inconsistency among controllers, if the network changes trigger less than 987 events per second.

Future research is needed in developing an alternative to the publish / subscribe system, or modifications to WheelFS.

4.2 DevoFlow

As disused previously HyperFlow attempts to build on top of OpenFlow and NOX, without changing the basic premise. Contraire DevoFlow does not; DevoFlow believes OpenFlow excessively couples central control and complete visibility [24, 25]. DevoFlow states having full visibility over all flows is not quite the right goal. It focus instead, is to only have visibility over significant flows, while reducing the load of the controller. Its arguments are essentially an analysis of tradeoffs between centralization and cost; it is designed for simple and cost-effective hardware implementation.

DevoFlow introduces two new mechanisms for devolving control, Rule cloning and local actions. Rule Cloning augments the "action" past of the OpenFlow wildcard rule with a Boolean CLONE flag. If flag is clear, standard OpenFlow wildcard behavior is followed. If not the switch locally clones the wildcard rule to create new rules. With Local actions rules are augmented with a small set of possible "local routing actions". This will be done without invoking the controller. If the switch does not support the action only then will you invoke the controller.

DevoFlow also offers two different ways of collecting statistics information, sFlow and, Triggers and reports. sFlow uses sampling of header information at a rate of 1/1000 packets (this rate can be adjusted). Instead of OpenFlow's push-based or pull-based collection strategies. Since sFlow does not include the entire packet, the incremental load on the network is less than .1% [25]. An alternative to sFlow is Triggers and reports, this uses OpenFlow rules to include threshold-based triggers on counters. The switch only sends the report to the controller when the threshold is met. Research is still being done as to witch of the two collection strategies is better, as of now it is unsure.

DevoFlow uses the mechanisms mentioned above to reduce the number of flows that interact with the control-plane. By reducing the flows you enable scalability, greater than with OpenFlow itself. This does not solve your scalability issues; it only lets you have a bigger network then with OpenFlow alone. You still have all the constraints mentioned before. This is because you are only improving infrastructure for a system with one controller, as HyperFlow lets you add multiple controllers and is more scalable.

A good future research would be to implement HyperFlow with DevoFlow, expanding the network size of each controller, without addressing the limitations of WheelFS.

4.3 FlowVisor

FlowVisor has been address as a work around for scalability in OpenFlow [11]. FlowVisor was not specifically designs to address the scalability issues; it was design to enable multiple researchers to slice a production network for test bed and lab research [8]. We will go deeper into this in Section 8.

FlowVisor enables a way of placing multiple controllers on one physical network as shown in figure 4. Each controller will only have a globe view of its own network.

This is not a good solution is regards to scalability. Yes it allows you to use more controllers and reduce load but at the cost of sacrificing the overall goal of OpenFlow.

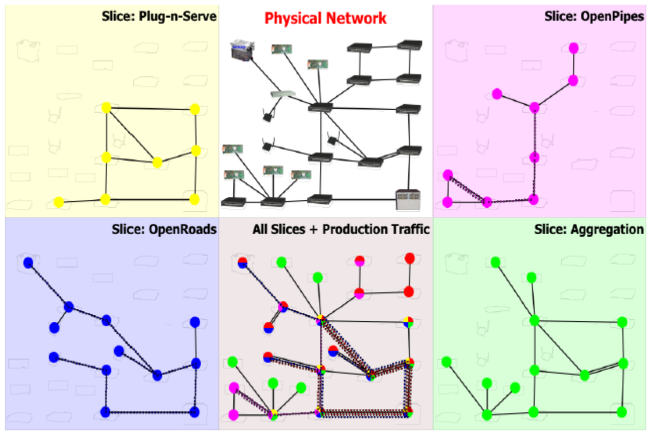

Fig. 4 FlowVisor network slicing

4.3 Global view and CA-SDN

In CA-SDN OpenFlow quires the switches to get a global view of the dynamic network, creating more packets gathering the information, and the extra overhead could outweigh the benefits.

A workaround has been proposed be putting a proxy controller that is co-located with the switch [4]. You can also consider your current topology and select a switch to gather information in a strategic location. This is still an open question that should be researched and a concern in regards to scalability

V. Using the Power of the Cloud

We have looked at how Cloud-assisted Software-defined Networking (Ca-SDN) helps with scalability and its shortcomings. Now we will look at the other advantages the cloud computing have for OpenFlow

There are several advantages one being Flexibility by functionality only requires a modification of the software implementation of the controller. In forwarding performances switches do most of the forwarding in hardware in contrast to the software routing. Additionally the time for setting up new entries can be reduced by utilizing the computational resources of the cloud, ease of administration, and cost reduction (by outsourcing complex functionality).

The flexibility and optimization of the disruption tree in Ca-SDN is excellent. The controller can calculate any kind of tree on the fly. This is where the real power is, calculating dynamic routing algorithms. By using the resources of the cloud you have many more routes that can be calculated compared to an inferior router or a single controller with finite resources. You now have the option of calculating thing like minimum spanning trees in parallel in different cores and can set a deadline for the calculation to avoid overloading [4].

CA-SDN uses two routing process reactive routing and proactive routing. In reactive a time is very critical as no distribution tree is installed on switch consequently this could cause the controller to get overloaded. An advantage of this is there is not redundant flooding and pruning like in MOSPE protocols. The distribution tree is calculated only once. The OpenFlow switches do not need to implement any multicast routing protocol at all.

Proactive routing advantage is in reducing the flow table size in switches. The two disadvantages are one the latency for the first packet and two the controller may become overloaded if it takes too long. This is especially true with UDP where packet will begin to send without a handshake like TCP.

VI. Experimenting with Mobility and OpenFlow

Implementing OpenFlow in a Wireless mesh network (WMN) faces many different obstacles than wired networks. Due to variations in link qualities and nodes joining and leaving the network, the network topology changes at a much higher pace than any wired network [3]. In addition, as wireless networks do not have the clear notion of a point-to-point link, neighbor and topology discovery need to be adapted to wireless networks. Handovers between station also need to be addressed.

The study of [3] demonstrates how with a few lines of python code, a reasonable and useful service can be implemented for WMN.

Fig. 5 illustrates the Initial association of a wireless laptop and architecture design.

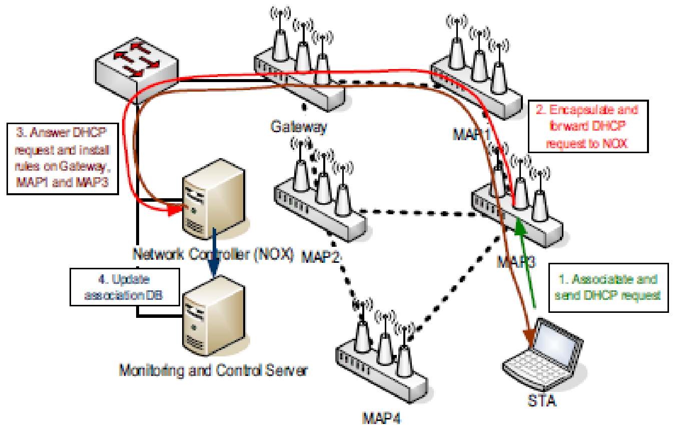
3. Answer DHCP request and install rules on Gateway, MAP1 and MAP3
Gateway
MAP1
2. Encapsulate and forward DHCP request to NOX
Network Controller (NOX)
MAP2
MAP3
4. Update association DB
1. Associate and send DHCP request
Monitoring and Control Server
STA
MAP4

Fig. 5 Wireless Mesh Network with Openflow

This design has been shown to work in a small scale. Future research is needed in developing an algorithm to calculate the optimal STA/MAP associations and flow paths and evaluate it in a large scale scenario

VII. OpenFlow-based Control of an Optical Network

Currently optical networks are controlled and managed through the element management system (EMS) and/ or the Network management system (NMS) as shown in figure 6 [5]. However this approach does not handle the rapid increase of dynamic networking traffic. An alternative choice to this has been developed called generalized multi-protocol label switching (GMPLS), but most network carriers seem to lack the confidence in it. OpenFlow has been proposed as a solution and received extensive attention worldwide. OpenFlow has been viewed as a positive replacement as illustrated in figure 7 , because of its centralized control scheme. It is easier to migrate and update current NMS/EMS architecture, unlike GMPLS.

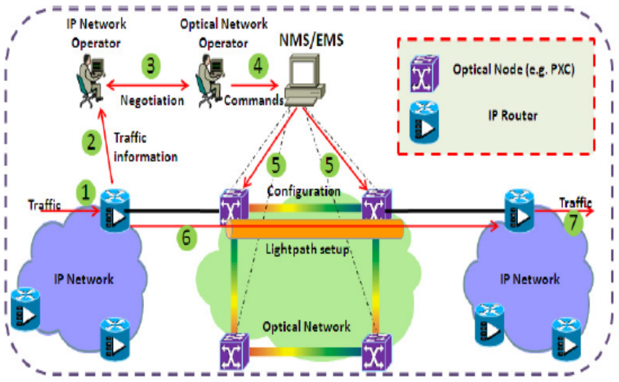

Fig. 6 NMS/EMS in current optical networks

To implement OpenFlow in optical network, they first created OpenFlow-enabled PXC (OF-PXC). PXC is a one of the devices used to switch high-speed optical switches. OF-PXC enables the NOX to control the cross-connections by using PXC.

To enable control of the node and a globe network view, virtual Ethernet interfaces were introduced to the OpenFlow switches (veths) [5]. Veths are virtualized from the physical interfaces of the PXC and each veth exactly corresponds to a physical interface of the PXC.

With the above mention methods in place the controller can effectively control flow in optic networks figure 7 illustrates the OF-PXC.

Now we will look at how the light path is setup and released. For each there are two proposed approaches, for light path setup, we can use the sequential approach or the delayed approach. For light path release we can use the active approach or the passive approach.

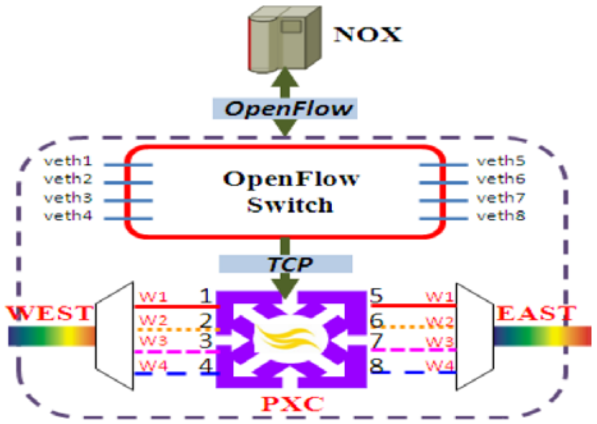

Fig. 7 OpenFlow-enabled PXC(OF-PXC)

The difference between the sequential and delayed approach is obvious by its name. The delay approach waits for an appropriate time delay for the successful lightpath in the optic domain and then inserts a new flow entry. Where the sequential approach does not.

The sequential approach is the most straightforward but there is no guarantee that the lightpath in the optical domain is completely provisioned before the flow arrives. As the latency of the PXC needs to be considered, the delayed approach is recommended, but can also cause bottleneck with protocols like UDP.

With lightpath release the active approach is only applicable when the amount of data for the arriving traffic is known in advance. In the passive approach the lightpath is only release after the NOX receives confirmation that is was received.

Future research is still needed in restoring lightpath restoration, and investigating unified OpenFlow-based control for heterogeneous multi-layer optical switching networks. OpenFlow and optic networks is still in the early stage compared to GMPLS, but has very promising expectation and worldwide support.

VIII. Research Environment and Tools

In order to design and test new ideas in OpenFlow. Network administrators need a way of giving more than one researcher access at a time. We will again look at FlowVisor as a solution in carving research slices out of a production network [8]

After giving you a slice of the network, we will introduce a tool for finding bugs in OpenFlow

8.1 Carving research slices

OpenFlow has some limitation from the perspective of researching and testing, because only one researcher can innovate on the network at a time [8]. FlowVisor has been proposed as a way to “slice” the network resources to allow researchers to use them in parallel. Typically most networks would be “slice” with VLAN’s, this approach complicates certain research like IP mobility and of wireless handoffs [3]. As mentioned in 4.3 FlowVisor has also been used as a workaround for scalability [11].

FlowVisor is a transparent virtualization layer between the OpenFlow switch and the controller. FlowVisor acts as a virtual controller to the switches and as a network of virtual switches to the research controller figure 8 and figure 4 illustrate the architecture.

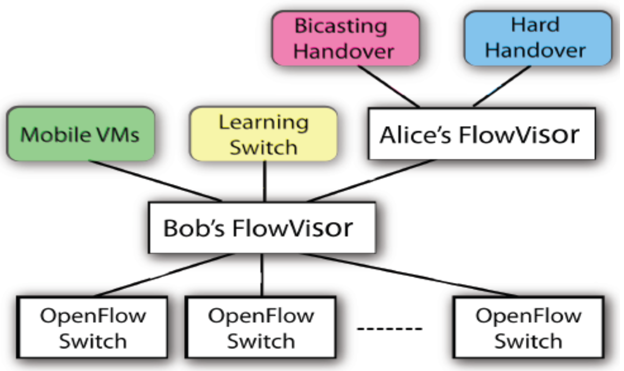

Fig. 8 FlowVisor architecture

FlowVisor is intentionally architecturally neutral, it does not know or make any assumptions about either the switches or the controller. FlowVisor was setup up this way for three reasons. One to make it centralized policy enforcement; all traffic passes through FlowVisor giving it a globe network view. Two recursive delegation, to allow FlowVisor to cascade instances and make recursive delegation when it needs to reclaim a subnet. Three Decouple control and virtualization technologies, this makes it possible to have advancement in each and independently, avoiding new forms or changes.

8.2 Testing OpenFlow for Faults (Bugs)

As Software Defined Networking (SDN) moves the control plane from the switches to the controller, software needs to be tested for faults (bugs). Even large corporation that extensively test software, release version with major bugs that affect and sometimes shutdown offices. Testing OpenFlow application is challenging because you are looking for bugs in a large environment that behavior dynamically.

A make the problem simple OpenFlow can require programmers to use domain-specific languages. Most OpenFlow applications have been written in Java and Python and adaptation of a domain-specific language will be difficult.

8.2.1 NICE

A tool has been developed to test OpenFlow application, NICE (No bugs In Controller Execution [6]. NICE test controller programs by generating carefully-crafted streams of packets under many possible event interleaving.

NICE test application written in Python that works with NOX platforms. To use the NICE tool a programmer will enter three thing; One the controller program. Two, the topology to use with all the switches and host. Lastly what to check for like no forwarding loops or no black hole. The programmer can also write his own properties to check. After NICE is done, it will output the results of the traces. Figure 9 illustrates each step.

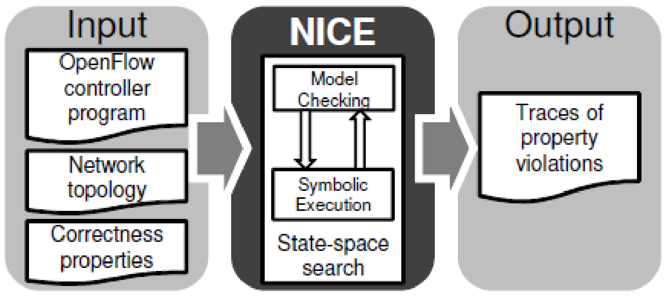

Fig. 9 How NICE is setup to use

NICE is built around two major components, the symbolic engine and the model checker. The symbolic engine is called by the model checker when the network model requires the generation of new packets to inject [27]. The model checker describes the network topology in terms of clients, switches, controller and links between them.

8.2.2 OFTEN

OpenFlow Testing Environment (OFTEN) is a tool for systematic testing of integrated OpenFlow networks with the goal of gaining confidence if controllers and real switches work together correctly in a deployment-like setting. OFTEN is built on top of NICE, but extends NICE by enabling communication between the model checker and real switches [26].

OFTEN adds necessary glue to synchronize the state used in NICE with a dual environmental model of the real switches. It gets the flow tables from the real switches and controls the timing of events. Than it reports both the testing and real switches correctness issues and inconsistencies.

This comes with several challenges; first the switches should be treated as a black-box. Since the OpenFlow switch is expected to grow rapidly, testing process should rely on a common standardized interface to reduce overhead. Second a corrective definition needs to be defined of the expected behavior for each test case. This should be defined two fold, one in network-wide and at a low level to aid in debugging.

OFTEN approach come with some limitation, since it is built on NICE the model checking forces sequential executions, this limits the ability to force high load situation for peak performance testing. Additional OpenFlow is not design with testability in mind. By introducing new mechanisms like barrier request forcing synchronization, standardized interface to get information about internal state, and something to determine when packet processing has ended. This is a good starting point for future research and discussions.

IX. Domain-Specific Programing Language

As mentioned in section 8.2, most OpenFlow application is written in general-purpose languages like Python or java [6]. Tools like NICE and OFTEN [26] have been developed to find bug in these languages as they are more prone to errors, than domain-specific languages that prevent certain classes of bug. One of this domain-specific languages is Frenetic witch is an extension of Python.

9.1 Frenetic

Frenetic is a domain-specific language for OpenFlow that aims to eradicate a large class of programing faults. Frenetic simplifies the task of programing OpenFlow networks, without compromising flexibility and efficiency [10].

Frenetic is based on functional reactive programing (FRP). By using FRP you do not need to write programs that are event drive as FRP see every packet.

Frenetic architecture consists of three pieces illustrated in figure 10. The Frenetic program witch implements the FRP operations. The run-time system and the NOX.

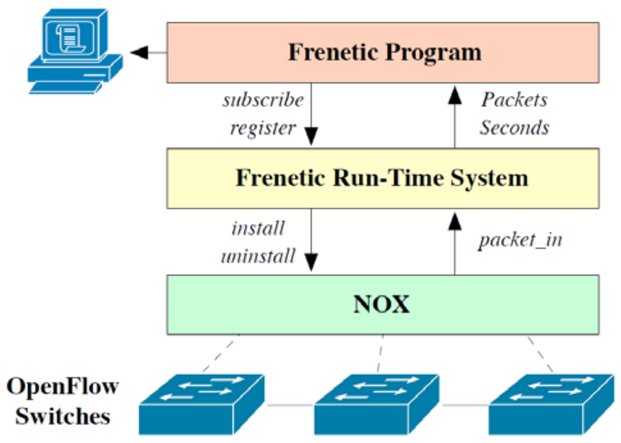

Fig. 10 Frenetic Architecture

Frenetic focuses exclusively on discrete stream. It uses a push-based strategy that propagates values from input to output streams. Even though Frenetic sees every packet, it does not send then to the controller, as this will limit frenetic scalability. Instead it developed optimizations that capture some common idioms. The run-time system is the back end that installs and uninstalls rules and communicates between the switch and the controller.

X. Internet Security and OpenFlow

Flooding-based distributed denial of service attacks (DDoS) have been difficult for security administrators to detect. Since packet headers fields are modified to look like normal traffic. So to tell the difference between a legitimate packet and a useless one is quite hard. Also with the overwhelming amount of packets sent in a DDoS, it makes it difficult to analyze each one. These two factors make detection of DDoS attack problematic.

The method proposes to detect DDoS attacks using OpenFlow switch and NOX is divided into three methods placed within the detection loop of the NOX controller.

The three modules are the flow collector, the feature extractor, and the classifier (SOM) (illustrated in figure 11). The flow collector periodically request flow entries from all flow tables of the OpenFlow switches. It communicates and transmits through a secure channel isolated from host connect to the switch. The feature extractor receives the collected flow from the flow controller, and extracts certain features important to analyze DDoS flooding attacks. The classifier analyzes weather the packet received by the featured extractor of DDoS flooding attacks or legitimate traffic. If it is legitimate traffic the classifier send the information to the flow collector to update tables appropriable, if not the classifier alerts detection of an attack [16].

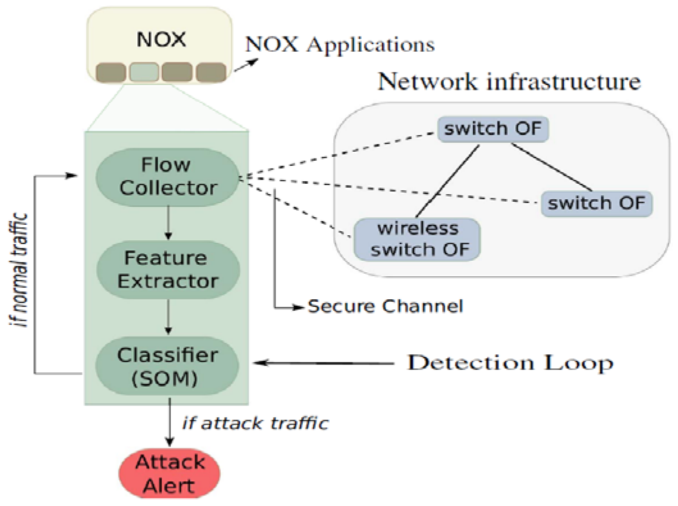

Fig. 11 Detection loop Operation

XI. Conclusion

As this paper shows OpenFlow has been embraced and implemented in many areas. It has been shown it can be applied in areas not initially intended for like optic networks and wireless mesh networks.

OpenFlow is still in its early stages. It will require further research but it seems like the future of network will derive from it. Especially in data centers where its application on large network, give the best return on investment.

Its biggest challenge still seems to be scalability. As it is a centralized approach, HyperFlow and DevoFlow is a good start but lacks solution adequate enough to implement on a large network.

XII. Future Research

As this paper has covered many topics related to OpenFlow. I will give a recap of where specifically research is needed by topic.

In power consumption reduction, more research is needed in hardware control of the switches. Sleep state, individual ports and the switch as a whole, in both powering down and up quicker and with remote control.

In Scalability, creating an alternative to HyperFlow's use of WheelFS seems very promising. Also as mentioned in the paper, I feel implement HyperFlow with DevoFlow may be a good combination.

In using the Ca-SDN, co-located controllers and addressing UDP protocol challenges.

In Wireless Mesh network, developing an algorithm to calculate the optimal STA/MAP associations and flow paths and evaluate it in a large scale scenario.

In Optical Networks, restoring lightpath restoration, and investigating unified OpenFlow-based control for heterogeneous multi-layer optical switching networks

In tools and research environments, implementing features from NICE and OFTEN into OpenFlow, creating a more testable design.

In Domain-Specific Programing Language, I feel if ONF should pick a standard for all switches. Giving consistency, and promote open source collaboration.

References

[1] M. Casado, Ethane: taking control of the Enterprise, Stanford University, 2006

[2] Open Network Foundation, www.opennetworking.org

[3] P. Delay , OpenFlow for wireless mesh networks, Karlstad Universality , 2009

[4] F. Durr, Toward cloud-assisted software-defined networking, IPVS Instite of parallel and distributed systems, 2012

[5] L. Liu, Experimental Validation and performance evalyation of OpenFlow-based wavelength path control in transparent optical networks, Universaty of Post and Telecommunications, 2011

[6] M. Canini, A NICE way to test OpenFlow applications, Princeton university, 2011

[7] S. Das, Packet and circuit network convergence with OpenFlow, Stanford university, 2010

[8] R. Sherwood , Carving Research slices out of your production Network with OpenFlow, ACM SIGCOMM, January 2010

[9] B. Heller, Elastic tree: Saving energy in data center networks, Stanford University, 2009

[10] N. Foster , Frentic: A high-level Language for OpenFlow networks, Princeton University, 2011

[11] A. Tootoonchian, HyperFlow: A distributed control plane for OpenFlow, University of Toronto, 2010

[12] N. Gude, NOX: Towards an operating system for network, Stanford University, ACM SIGCOMM, 2008

[13] www.computerweekly.com/feature/the-history-of-OpenFlow

[14] A. Bianco, OpenFlow switching: Data plane performance, IEEE, 2010

[15] Y. Luo , Accelerating OpenFlow witching with network processors, University of Massachusetts, 2009

[16] R. Braga, Lighweight DDoS flooding attack detection using NOX/OpenFlow, Universidade federal do amazonas, 2010

[17] D. Staessens, Software defined networking: meeting carrier grade requirements, Ghent university- 2008

[18] N.Mckeowen et al Openflow: Enabling innovation in campus networks, SIGCOMM, April 2008

[19] Open Networking Foundation, OpenFlow Switch Specification version 1.3.2, April 2013.

[20] J. He, Towards robust multi-layer traffic engineering: optimization of congestion control routing, Princeton University, 2007

[21] B. Nordman et al, reducing the energy consumption of networked devices, IEEE 802.3 tutorial, 2005

[22] OpenFlow Consortium, OpenFlow switch specifications version 1.1.0, www.openflow.org/documents , 2011

[23] Stribling, J., wide-area storage for distributed systems with WheelFS, USENIX Symposium on Networked Systems Design and Implementation, April 2009

[24] A. Curtis, DevoFlow: Scaling flow management for high-performance networks, University of Waterloo, 2010

[25] J. Mogul, DevoFlow cost-effective flow management for high performance enterprise networks, 2010

[26] M. Kuzniar, OFTEN testing OpenFlow Networks, European workshop on software defines networking, 2012

[27] NICE-OF, code.google.com/p/nice/of

Authors

First Author – Javier Coto – Coto@bellsouth.net – (305) 733-5858