Improving Cloud Datacentre Scalability, Agility and Performance using OpenFlow

Charlie Baker, Ashiq Anjum, Richard Hill, Nik Bessis
School of Computing and Mathematics
University of Derby, Derby, UK
C.Baker2@unimail.derby.ac.uk, {a.anjum, r.hill, n.bessis}@derby.ac.uk

Saad Liaquat Kiani
Faculty of Engineering and Technology
University of the West of England, Bristol, UK
Saad2.Liaquat@uwe.ac

***Abstract* — With network infrastructure reaching capacity limits, new ways of managing network traffic and devices in Cloud data centres need to be investigated to better scale and manage the data centre resources. OpenFlow can be a good fit for the usage within a datacentre due to its promise to reduce the cost of operations by reducing the time needed to configure network resources and flexibly managing the operations. OpenFlow allows a programmable *Software Defined Network* to be implemented and network resources such as switches can be remotely and collectively configured, controlled and monitored. With increasing number of resources in data centres, and by consequence the networking hardware and software, OpenFlow can be used to manage the network devices and the flow of traffic throughout the network. This paper investigates if OpenFlow based switches allow for flexible networking and administration to enable quick network segmentation, packet re-routing, permit scalability and allow for quick network setup in Cloud data centres. With the help of experiments, this work also investigates what design and implementation considerations need to be made to achieve scalability, agility and performance in Cloud data centres.**

Keywords-component; openflow; networking; cloud computing; datacentres

I. INTRODUCTION

Data centres are expensive to setup, manage and operate. On average each rack can cost around $120,000 (~£75,000) over the Total Cost of Ownership (TCO) - approximately half of which is hardware and the other half, operating costs [1]. The average cost of a switch is about $450 (~£285) per 10Gbps Ethernet switch port i.e. the cost of a 10Gbps 48-port switch can be about $21,600 (~£13,540) [11]. One of the approaches towards reducing the operating cost of data centres is to eliminate unneeded hardware and make them virtualised. Administrating the increasing number of switches in a data centre is a non-trivial task. Using OpenFlow based switches would allow all of the switches to be administered from one central location resulting in quick setup and configuration which can reduce the costs of setting up new or modifying existing network configurations in data centres thus reducing operational, maintenance and setup costs

The main purpose of switching hardware is to propagate packets - receiving from a sender and forwarding it on towards the intended recipient. Switches operate on various layers of the network stack from layer 1 (repeaters) up to layer 7; their functional characteristics include network bridging, routing, address translation, content filtering, redundancy and load balancing. Switches that carry out routing functionality usually compartmentalise their functions into two parts that control the processing of the incoming packets. These are known as the *Control Plane* and the *Forwarding or Data Plane*. The Control Plane is responsible for generating the network map or an in-built routing table, which lists the routes that should be used to forward packets. The Forwarding/Data Plane is used to determine, with the help of the table or map produced by the Control Plane, where the packet should be sent based on various criteria such as destination, port or the contents of the packet. Usually the Control and Forwarding planes in a device are vendor specific, proprietary and closed source. This means they are not modifiable easily.

OpenFlow is an open standard that allows researchers and network administrators to design, test and implement experimental protocols in networks. On a switching device that does not run OpenFlow, the Control and the Forwarding planes operate on the same device whereas an OpenFlow configured device separates these two planes with the routing decisions being controlled by a *controller*. A controller is typically a standard server running on the same or different device. There can be one or many controllers in a network, allowing one controller to control a large number of OpenFlow devices. The controller can be programmed to route packets depending on various control sets [4]. This also means that switches can be controlled easily from one remote location such as a Network Operating Centre (NOC). Custom control sets allow the switch to route packets in various ways that do not exist currently in today's hardware such as legacy switches.

Switching devices are part of almost every network infrastructure used in commercial, technology, education and residential settings. Whilst this is a success for implementation, success in developing and then deploying new switching technologies is much more remote due to the large install base. The sheer number of devices in widespread deployment has caused network innovation to stall and stagnate due to the reluctance to experiment with new ideas. New and alternative approaches are needed due to the traffic increasing considerably each year [10]. With network infrastructure slowly reaching capacity limits, new ways of routing packets may need to be investigated if there is reluctance to upgrading physical infrastructure. This is especially the case as Cloud-based applications gain more

978-0-7695-4808-1/12 $26.00 © 2012 Crown Copyright
DOI 10.1109/iNCoS.2012.118

popularity, possibly increasing the bandwidth usage [3]. Due to the highcost of entry to implementation in realistic environments, new ideas need to gain confidence to enable widespread deployment, which is currently a problem for new protocols. OpenFlow can allow researchers to experiment in such environments due to its ease of access and programmable nature whilst running alongside and without affecting the production networks. Such experimentation and subsequent analysis can help new switching and routing approaches gain traction.

Consider the scenario of Cloud datacentres, where many Cloud-based applications are migrating from physical servers to virtual servers. Hao et al. [3] show that utilising OpenFlow enables virtual servers to be seamlessly migrated. However they conclude that whilst possible in datacentres, numerous issues remain and require further research.

This work investigates if OpenFlow based switches allow for easy networking and administration to enable quick network segmentation, packet re-routing, permit scalability and allow for quick network setup in Cloud data centres. Furthermore, this utility of OpenFlow needs to cater for metrics such as performance and bandwidth in comparison with existing switches and explore any potential reduction in cost of configuring network resources. Establishing the viability of OpenFlow in these aspects can help persuade the cloud data centre community to implement OpenFlow in their infrastructures.

The rest of the paper is organised as follows: Section II presents the work related to our discussion. In Section III, we present the design of a data centre, its network configuration and OpenFlow scripts that form the basis of our experiments, which are discussed in Section IV. Experimental results and their evaluation is presented in Section V, and the discussion is concluded in Section VI.

II. Related Work

There are a number of works aimed at improving OpenFlow including [4], which suggests that using network based acceleration cards to perform the OpenFlow switching to improve performance. In [13], the authors argue that load balancers are expensive, but using OpenFlow tables to perform the switching can require hundreds or thousands of rules. They suggest that OpenFlow should allow wildcard rules and present a prototype implementation of their approach in Mininet and NOX. CleanSlate (http://cleanslate.stanford.edu) at Stanford tested the implementation and deployment of an actual OpenFlow network. A related project is OpenRoads [14] that aims to provide a wireless extension of OpenFlow with their final goal of deploying it in production networks that incorporate wireless technologies such as WiMAX and Wi-Fi. Their wireless base stations incorporate flow tables so that they can be controlled remotely via the OpenFlow protocol. There have also been some developments with alternatives to OpenFlow such as the Arista Networks EOS (Extensible Modular Operating System) [15]. Arista Networks EOS is made up of components such as CloudVision, which allows single point administration of hundreds of thousands of cloud node switches.

Frenetic [16] is high-level programming language for programming software-defined networks. Frenetic is based on the argument that most of the languages currently available for Software Defined Networks are too low level, complicated and hard to learn. Frenetic uses the functional reactive programming paradigm, in contrast to the event driven programming used in OpenFlow. NetCore [17] is another such programming language that is used for software-defined networks. NetCore is a high-level, declarative programming language and is mainly used for expressing packet-forwarding rules for software-defined networks. RouteFlow [18] is a project following the software-defined networking paradigm, utilising an OpenFlow controller with a RouteFlow server that manages virtual network environments that connect to virtualised IP routing engines. The goal of RouteFlow is to centralise remote IP routing eventually allowing developments such as virtual routers or “Routers as a Service”. As you can see there are a number of differing projects that centre on either OpenFlow or Software Defined Networks and improving them or providing alternatives to current technologies and implementations. However, none of these aims to improve the cloud data centre operations with respect to traffic management and segmentation, scalability and administrative control over the resources.

III. Design

To assess the viability of OpenFlow in data centre routing and how it compares to existing switches in real world data centres, our work will devise a set of experiments that utilises OpenFlow along with all the available tools that come with the testing suite. We will design a model data centre network, housing a large number of nodes. The experiments will generate and route traffic under the control of OpenFlow and test metrics such as performance, throughput and bandwidth.

A. Network Design

For the experiment, a sample segment of a data centre network is going to be designed and implemented using the OpenFlow tools (explained in the next section). Our data centre design is based on established conceptual and experimental data centre designs [2][6] (see Fig. I), with the aim of supporting the generalizability and reproducibility of our experimental results. A custom network topology has been designed, based on [2][6] and illustrated in Fig. II that implements features of each to provide an adequate design that can be used and tested throughout the experiment.

Fig. I shows a network similar to the ones proposed in [2][6] but they are also linked up to a SAN (Storage Area Network). It depicts more redundancy features such as the interconnections between routing, switching and firewall nodes. A SAN is a separate, high-speed network for storing data as efficiently and as fast as possible. Most SANs reside on a separate network due to the specialised hardware needed. Initially our test implementation, of a design included a SAN, however was removed from the eventual design to simplify the experimental setup. Figure I shows our experimental data centre network layout.

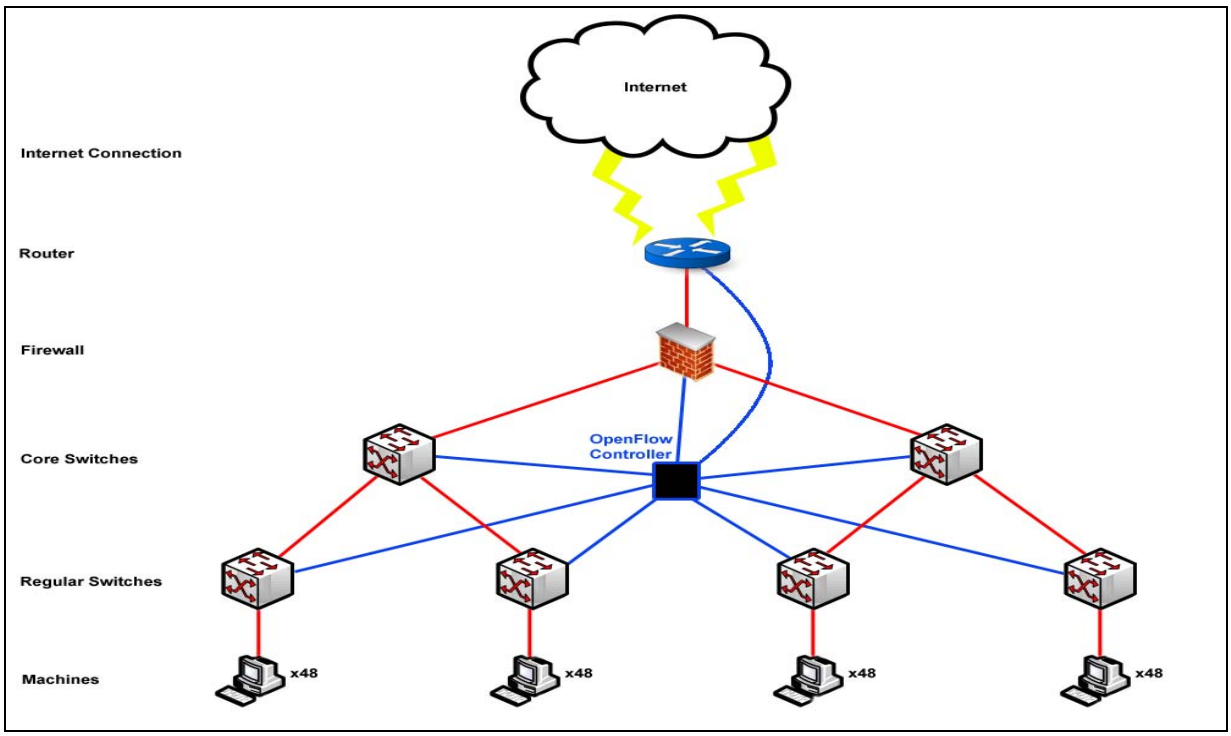


FIGURE I Design of the Cloud data centre used in the experiments

B. *Code / Algorithm Design*

Our experiments are directed through a script file that is used to set up the environment for the experiment. The script contains OpenFlow API based instructions. Fig. II describes an algorithm that is used to set up the network. Line 1 imports the required libraries such as the OpenFlow API. Line 2 and 3 are initialisation methods that are part of the structures for the Python script. Lines 4-8 instruct all nodes to set up a unique ID in the network. Lines 9-13 add all the switching devices in the network. Lines 14-15 do the same for the host nodes. Lines 16-20 set up connections between all the nodes, in this case, the main nodes such as the Firewall, Internet and Core Switches. Each set of Line 21-22 up to Line 28 loop through every set of 48 hosts and connect them to the correct switch. Finally on Line 29, we enable all the nodes and then end the definition in Line 30.
This algorithm sets up a network with 192 nodes (host machines) with 9 switches in the network, producing the end result of a network topology illustrated earlier in Figure I. The Internet Switch will act as a host outside of the network. An important consideration taken into account was how many nodes would be an adequate in the experimental network. Too many would prohibitively increase the network testing time whereas too few would cause the network to be inadequate for correctly simulating a segment of a data centre.

C. *Alternative Algorithm*

Originally this pseudo code included the code to incorporate a simple SAN setup in the network. Fig. III shows example code of how the original design was implemented.

This set of code continues on from Line 27 and 28 and finishes before the original Line 29 i.e. adding the SAN features before continuing with the existing script. Lines 31 and 32 set up all the SAN Switches unique IDs that are similar to the previously defined Switches. Next Lines 33 and 34 loop through all the SAN hosts and add them to the network. After which the Switches get added to the network

on Line 35. After this, each set of SAN hosts are connected to their corresponding SAN Switch during Lines 36-39. Finally, it has to loop through every original set of Hosts from the network and then connect them to the correct SAN Switch and then carry on executing the existing script.

This script will be used to develop a software-defined network utilising tools, which consist mainly of a virtual

FIGURE II PSEUDO CODE - SETUP SCRIPT

```
1    Import required libraries

2    Begin definition of my topology:

3      Define initialisation method:

4        Set up ID for Internet node
5        Set up IDs for Router and Firewall nodes
6        Set up IDs for Core Switches
7        Set up IDs for Regular Switches
8        Set up IDs for Host machines

9        Add Internet to Network
10       Add Router to Network
11       Add Firewall to Network
12       Add Core Switch 1 and 2 to Network
13       Add Regular Switch 1, 2, 3 and 4 to
             Network

14       Loop through every node.
15         Add node to the network

16       Connect Internet to Router
17       Connect Router to Firewall
18       Connect Firewall to Core Switch 1 and 2
19       Connect Core Switch 1 to Regular Switch
             1 and 2
20       Connect Core Switch 2 to Regular Switch
             3 and 4

21       Loop through first set of nodes.
22         Connect 48 Hosts to Regular Switch 1

23       Loop through second set of nodes.
24         Connect 48 Hosts to Regular Switch 2

25       Loop through third set of nodes.
26         Connect 48 Hosts to Regular Switch 3
27       Loop through last set of nodes.
28         Connect 48 Hosts to Regular Switch 4
29       Enable all nodes.
             30      End definition
```

D. Tools & Environment

Figure IV illustrates the interaction between the components involved in our experiment. The host machine consists of a 2.7 GHz Intel Core i5 Quad Core system with 16 GB of RAM executing Windows 7 64-Bit Professional. Oracle's VirtualBox virtualization software has been used to host the OpenFlow virtual machine. The OpenFlow VM testing environment, derived from the Ubuntu Linux distribution, is executed in VirtualBox. XMing server [19] running on the machine running the OpenFlow testing image (an Ubuntu / Debian based Unix distribution modified with the OpenFlow tools). Finally, our experiments will use this test topology to assess the viability of the use of OpenFlow in a switch and its feasibility in terms of performance related metrics.

host machine is used to pipe graphical applications from the VM to the host machine over an SSH tunnel.

Mininet allows creation, interaction with, customization and sharing of prototype software-defined networks from the command line or scripting using the OpenFlow protocol. It allows creation of different types of networks and performing various tests (throughput, ping, etc.) on individual nodes and the whole network. The main advantage of using Mininet is that it allows us to rapidly prototype real networks without having to resort to setting up a lab environment; though this is also possible if required. Mininet also incorporates various test functions for thorough network testing; these include Iperf, Ping, PingAll, PingPair and CBench.

FIGURE III PSEUDO CODE - SAN SETUP SCRIPT

```
         ...
27       Loop through last set of nodes.
28        Connect 48 Hosts to Regular Switch 4
         // End of existing Code

31       Set up IDs for SAN Hosts Set #1 and #2
32       Set up IDs for SAN Switch #1 and #2

33       Loop through SAN set of Hosts
34         Add SAN Host to Network

35       Add San Switch #1 and #2 to Network

36       Loop through SAN Hosts Set #1
37         Connect Host to Switch #1

38       Loop through SAN Hosts Set #2
39         Connect Host to Switch #2

40       Loop through every original host Set #1 and #2
39         Connect SAN Switch #1 to Set #1 and
           #2 Hosts

40       Loop through every original host Set #3 and #4
41         Connect SAN Switch #2 to Set #3 and
```

These tools, which will be used in our experimental analysis, are described below:

a) Iperf

Iperf is used to find out the bandwidth between two nodes. It uses the two 'farthest away' nodes in the network and then finds the maximum amount of bandwidth possible between them. This is useful for determining how fast the network can be.

b) Ping

The Ping programs tests the connectivity between two devices by sending a message to the recipient from the

sender and then awaits a response from the recipient back to the original sender.

c) PingAll

PingAll utilises the Ping program and loops through every host in the network and then pings every other host in the network to check for connectivity. This can take a considerable amount of time when dealing with a large number of hosts.

d) PingPair

PingPair only pings between two deepest nodes in the network.

e) CBench

CBench is used to determine how quickly flows can be changed on all the switches in a network from the controller. This is measured in total flow modifications per second.

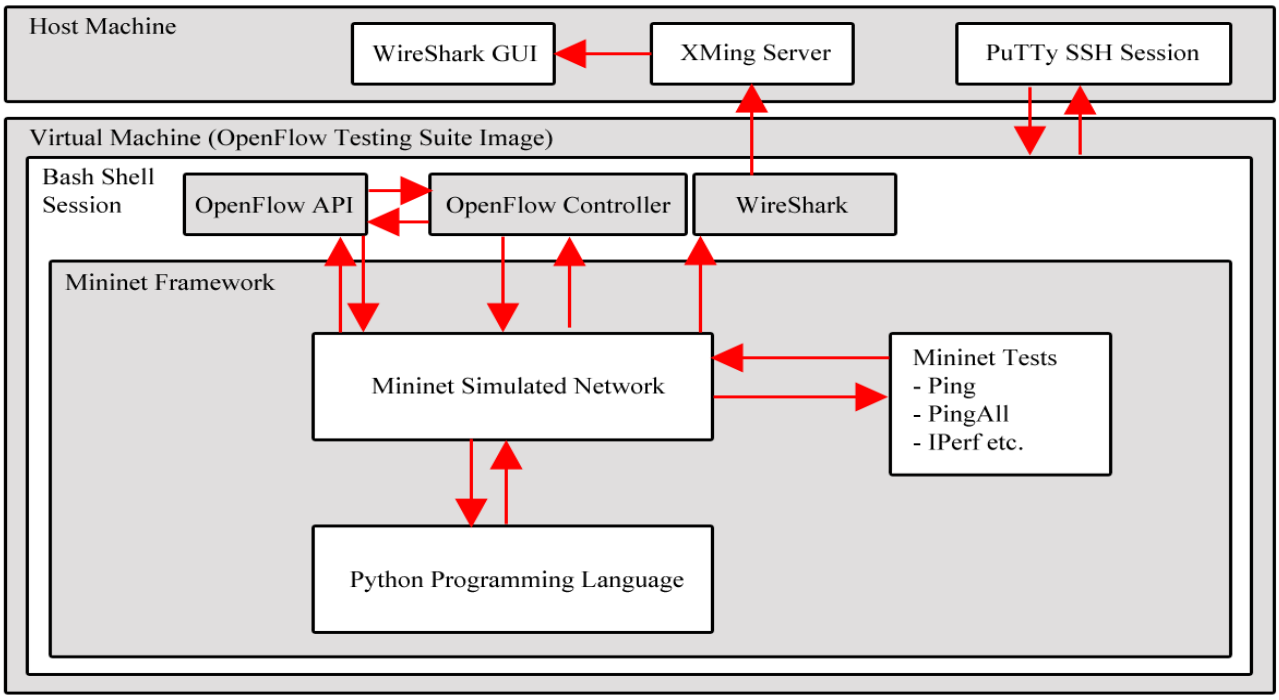


FIGURE IV Detailed Image of Components and how they interact

IV. Implementation and Experimental Setup

The experiment uses this main set of software/tools to eventually implement the prototype cloud datacentre utilizing the OpenFlow protocol. The experimental is driven through a python file and an execution file with instructions on how to execute it.

A. Execution

To specify the experimental execution in Mininet, a custom topology is produced that is executed to automatically produce networks of various types. Correctness of the network topology used in the experiments was initially tested between a smaller number of nodes. Afterwards, additional nodes were continually added to build up the network to the full scale. Fig. II shows the pseudo code for the topology file that was created and used. By using the custom topology, start-up and setup (including the generation of the network) of Mininet was achieved using commands shown in Fig. V. These two commands start the controller and then proceed to start up the Mininet environment.

```
openflow@openflowvm:$ controller ptcp:
openflow@openflowvm:$ sudo mn --custom
/home/openflow/mininet/examples/dctopo.py --topo dctopo -
-mac --controller remote
```

FIGURE V Start-up commands

The first command starts up the remote Controller, which determines on how packets are routed or flowed. The packet flow determining rules can be custom made programmatically, though a default set of rules is also available. Our experiment uses the learning controller supplied with the Mininet environment. The second command starts up Mininet with our custom topology file.

Testing the experimental network required iterating through each section of code and ensuring the correctness of individual command executions. Once the correctness of the script's execution was verified, the WireShark

application [20] was utilised to view the traffic being generated in the network. WireShark is a packet sniffer that allows the user to analyse packets that are passing through a network. This particular version of WireShark was pre-setup to include OpenFlow specific features to allow snooping on OpenFlow related traffic. The use of this application enabled network inspection to ensure that the implementation of the designed network was operating correctly provided a real time view of the effects of the packets as well as all messages sent between the switches and controllers. The results of the network testing are discussed in the following seciton.

B. *Testing*

To test the design of the network, connectivity tests were carried out. Tools available from within the Mininet framework were used for this test. The Pingall function in Mininet was used to test that each node can connect to every other node. Due to the number of nodes – 192 in total – this takes a considerable amount of time as it tests each node's connectivity with every other node in the network. The results show that overall there was a total of 37,056 pings between every host in each test batch. This shows that this topology is being constructed properly and that all the nodes are communicating with one and other. WireShark was used to make sure the OpenFlow controller was having an effect on any packets at all and routing them to the correct destination. The PingPair tool pings pairs of nodes that are the furthest down the tree of nodes to check for connectivity. Because most of the nodes at the bottom of the tree are on the same level or depth, it only checks the connectivity between two nodes – in this case host 1 and 48. As the output in Fig. VI illustrates, the PingPair test completed successfully.

```
h1 -> h48
h48 -> h1
*** Results: 0% dropped (0/2 lost)
```

FIGURE VI PingPair Testing Output 1

V. RESULTS & EVALUATION

A. *Bandwidth*

Testing the bandwidth can be particularly useful as it demonstrates how fast the packets are routed through the network. Also this shows how fast the controller makes decisions depending on various situations. For example, the controller can be running in the kernel space, the user space and various other ways. Each has its own sets of pros and cons and issues relating to speed. For this example, it was executed in the kernel space to benefit from faster execution of the controller. To test the bandwidth, *iperf* tool supplied with the Mininet environment was used.

TABLE I. BANDWIDTH MEASUREMENTS DATASET

Test #	Bandwidth (MBits/sec)					
	User Space Switch			OpenVSwitch (Kernel) Switch		
	Attempt 1	Attempt 2	Attempt 3	Attempt 1	Attempt 2	Attempt 3
1	22.8	23.6	27.1	496	585	433
2	23.8	22.0	24	584	580	513
3	25.9	22.6	24.2	589	572	575
4	25.8	22.7	25.5	587	576	572
5	23.5	22.1	24.7	582	579	573
6	22.7	21.9	24.2	578	577	578
Ave.	24.08	22.48	24.95	569.33	578.16	540.67

Table 1 shows the results of doing each test twice using the User and Kernel Space Switches as well as using the OpenVSwitch implementation that is bundled together as part of the OpenFlow testing image. Some of the switches that are bundled are used for learning purposes only and will not do anything "intelligent" such as routing or building route tables. The idea is that you yourself manually build a flow table and install it on the switch for it to process flows. This could be a reason as to why the bandwidth is so low.

B. *Flow Modifications Per Second*

CBench is a utility for testing the number of flow modifications per second in a Controller. Obviously the higher the result the better as the Controller can push out flows quicker resulting in the flow of packets being altered quicker. Here the utility test has been run three times to determine how fast the modifications were running in the virtual machine.

TABLE II. CBENCH RESULT SET

Test #	Min (fmods/s)	Average (fmods/s)	Max (fmods/s)
1	2648.66	2835.58	2736.25
2	2615.00	2800.17	2714.03
3	2622.98	2797.99	2704.18
4	2638.99	2830.22	2733.15
5	2687.99	2882.17	2759.18
6	2573.99	2785.92	2713.84
Average	2631.27	2822.01	2726.77

The results in Table II show the number of flow modifications these Controllers can propagate per second. As this test was executing in a virtual machine, we estimate that the propagation rate would increase considerably when executed on real world networking hardware. However, there may not be a real need to require such a high number of flow modifications per second unless in stress testing scenarios.

C. *Visualising the Results*

As a companion to the result sets, graphs are going to be generated that illustrate the information from the result sets generated in a more understandable form. As the

graph in Fig VII illustrates, the network bandwidth during these tests is low, especially if this setup is compared with actual datacentre deployments.. The User switch runs in the User space memory rather than in the kernel space, which may explain the low bandwidth achieved in these tests.

1) Full Bandwidth Results

Figure VIII illustrates a considerable difference in the speed between the Userspace switches and the OpenVSwitch switches. This is mainly because of the fact that the OpenVSwitch switches execute in the kernel space. Intermittent results of over 1Gbps were also observed in some of the experiemnts (outliers). Our presumption is that it relates to the execution in a virtual machine in contrast to native hardware and we are investigating this aspect further in our future work. Figure IX illustrates the bandwidth difference between the Userspace Switch and the OpenVSwitch.

FIGURE VII USER SWITCH RESULTS FOR TABLE II

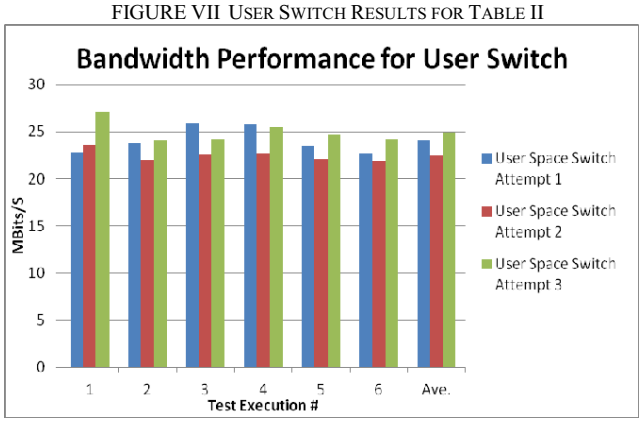


Figure VII shows only the User Switch bandwidth performance in relation to Table II.

FIGURE VIII OPENVSWITCH RESULTS FOR TABLE II

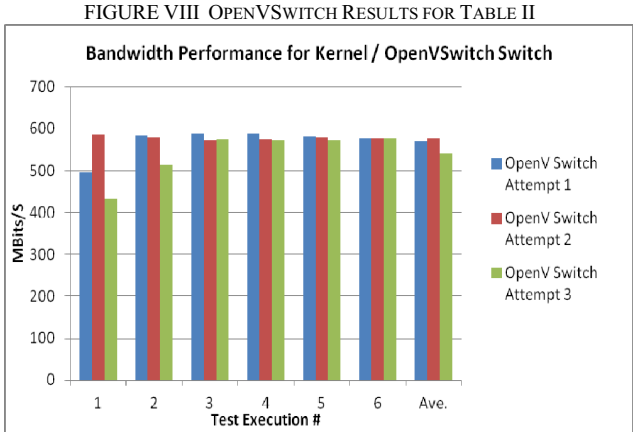
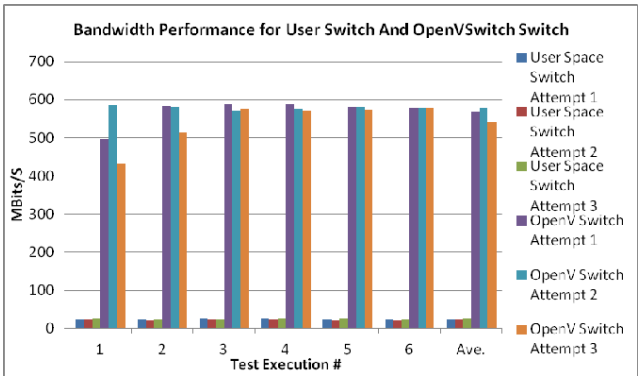


Figure VIII shows the OpenVSwitch data set from Table II.

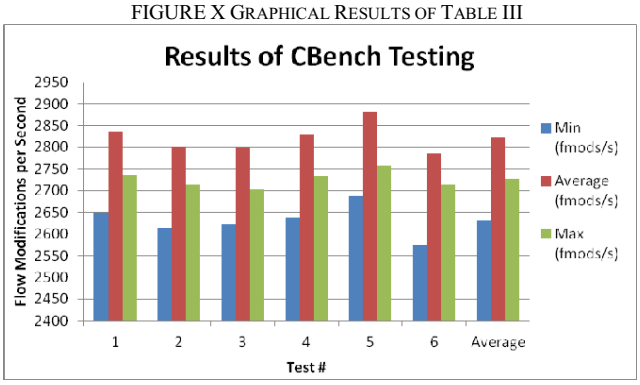
FIGURE IX COMBINED RESULTS OF TABLE II



The graph in Figure X displays the number of flow modifications per second. A *flow modification* is what occurs when the Controller has a rule for a particular packet and knows what it wants to do with it. It will propogate this rule between the switches or the switch will ask the controller what to do with it, if unknown. The graph shows a rather large number of modifications per second were achived in our experiments.

2) CBench Results

FIGURE X GRAPHICAL RESULTS OF TABLE III



3) Summary of Results

Whilst the experiment has had varying degrees of results relating to the performance of the network, it has shown that a simulated model of a datacentre segment can be implemented to analyse OpenFlow behaviour.–Whilst the bandwidth performance in kernel space execution switch was better than the Userspace switch – by about 15-20x– it may not be comparable to hardware switches deployed in datacentres. This can be explained by the fact that our experiments are being carried out withing Mininet, which itself is being executed in a virtual machine.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

OpenFlow and Software Defined Networks are growing in popularity and use in distributed infrastructures. Datacentres deployment sizes are increasing due to demand as more Cloud services are created and adopted. With the number of devices increasing within the datacentres – including switches -

alternative management mechanisms such as OpenFlow need to be considered and evaluated against existing mechanisms to improve scalability, agility and performance of data centre operations. The experiment presented in this involved the design, implementation and testing of a simulated datacentre network segment utilising the OpenFlow and the associated tools.

The results show that the User space switches offer a lower performance in comparison to the kernel space (OpenVSwitch) switches. The results for the kernel space switches' performance is not comparable to commercial switches in a real datacentre due to the framework used for the simulation (Mininet) and the usage of a virtual machine to perform the experiment.

OpenFlow could be a good fit for the usage within a datacentre - to reduce the cost of operations by reducing the time needed to configure switches. They can be remotely controlled from one place rather than having to configure and control them independently. Due to the early stage of development of these tools, it cannot be recommended for datacentre switches to be completely replaced with OpenFlow switches until the software platform matures. Further research needs to be carried out using commercial switches with OpenFlow technology built in rather than replacing them completely. We specifically propose the following additions in carrying out experimental analysis in this domain.. Different OpenFlow controller implementations, e.g. the NOX controller, Learning controller, the Beacon Controller, can be used to analyse their performance in an experimental networks such as the one presented in this paper. Beacon is a Java based modular controller with support for threaded and event based operations. Secondly, comparative studies can be performed in the area of Software Defined Networks using alternatives to OpenFlow e.g. RouteFlow.

VII. REFERENCES

[1] American Power Conversion, 2005, Determining Total Cost of Ownership for Data Center and Network Room Infrastructure Revision 3. [PDF] Available at: <http://www.linuxlabs.com/PDF/Data%20Center%20Cost%20of%20Ownership.pdf> [Accessed: 04/04/2012]

[2] Athenaeum Limited, 1999-2011, California Solar Powered Datacentre Specification, [Image Online] Available at: <http://www.ecologicalhosting.com/company/data-centre/solar-powered> [Accessed 20/12/2011]

[3] Fang Hao, T.V. Lakshman, Sarit Mukherjee, Haoyu Song, Enhancing Dynamic Cloud-based services using network virtualization. Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, VISA '09, Pages 37 - 44.

[4] Luo. Y, Accelerating OpenFlow Switching with Network Processors, ANCS '09 Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. , 2009

[5] McKeown, N, 2008. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Computer Communication Review, Volume 38, Number 2, Pages 69.

[6] Mohammad Al-Fares, 2008, A Scalable Commodity Data Center Network Architecture, Proceedings of the ACM SIGCOMM 2008 conference on Data communication, Volume 38, Issue 4, Page 65.

[7] Nate Foster, Michael J. Freedman, Rob Harrison, Jennifer Rexford, Matthew L. Meola, and David Walker. 2010. Frenetic: a high-level language for OpenFlow networks. In Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO '10). Volume 6 , Pages 1-6

[8] Okamura, K, 2010. Design and implementation of application based routing using OpenFlow. Proceedings of the 5th International Conference on Future Internet Technologies, CFI '10, Pages 60 - 67. [Online] Available at: <http://doi.acm.org/10.1145/1853079.1853096> [Accessed 30 October 2011].

[9] OpenFlow, 2011, OpenFlow: Learn More. [Online] Available at: <http://www.openflow.org/wp/learnmore> [Accessed: 18/12/2011]

[10] Othman M., Design and Implementation of Application Based Routing Using OpenFlow, 2010, Proceedings of the 5th International Conference on Future Internet Technologies, [Online] Available at: <http://dx.doi.org/10.1145/1853079.1853096> [Accessed 01/01/2012]

[11] Popa L, Ratnasamy S, Iannaccone G, Krishnamurthy A, Stoica I. 2010. A Cost Comparison of Data Center Network Architectures. ACM CoNEXT 2010. [Online] Available at: <http://www.cs.washington.edu/homes/arvind/papers/datacenter_comparison.pdf > [Accessed 18/04/2012]

[12] Tanyingyong V., 2010, Improving PC-based OpenFlow Switching Performance, ANCS '10 Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, [Online], Available at: <http://dx.doi.org/10.1145/1872007.1872023> [Accessed 01/01/2012]

[13] Wang. R., 2011, OpenFlow-Based Server Load Balancing Gone Wild, Proceeding Hot-ICE'11, Proceedings of the 11th USENIX conference on Hot Topics in management of internet, cloud and enterprise networks and services, [Online] Available at: < http://static.usenix.org/events/hotice11/tech/full_papers/Wang_Richard.pdf> [Accessed 26/02/2012]

[14] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. Openroads: empowering research in mobile net-works. SIGCOMM Comput. Commun. Rev., 40(1):125–126, 2010. (Best Poster at SIGCOMM 2009).

[15] Arista Networks EOS, (http://www.aristanetworks.com /en/products/eos, Last accesed: July 07, 2012

[16] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A Network Programming Language. In *ACM SIGPLAN International Conference on Functional Programming (ICFP), Tokyo, Japan*, September 2011.

[17] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker, A compiler and run-time system for network programming languages. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '12). ACM, New York, NY, USA, 217-230.

[18] C. Esteve Rothenberg, M. Ribeiro Nascimento, M. R. Salvador, C. Corrêa, S. Lucena, A. Vidal, and F. Verdi. "*Revisiting IP Routing Control Platforms with OpenFlow-based Software-Defined Networks*" In III Future Internet Experimental Research Workshop (WPEIF), Ouro Preto, MG, Brazil, May 2012.

[19] XMING X Server, http://sourceforge.net/projects/xming/, Last accesed: July 07, 2012

[20] A. Orebaugh, G. Ramirez, J. Burke, and J. Beale. Wireshark and Ethereal network protocol analyzer toolkit. Syngress Media Inc, 2007.