ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

SEMESTER THESIS AT THE DEPARTMENT OF
INFORMATION TECHNOLOGY
AND ELECTRICAL ENGINEERING

Spring Term 2011

Performance Evaluation of OpenFlow Switches

Dany Sünnen

28^{th} February 2011

Abstract

Opening with the quest for useful metrics best characterizing an OpenFlow environment, continuing with the build-up of a test infrastucture before closing with the execution and interpretation of the measurement results, this work aims at evaluating the performance of OpenFlow switches. The measurements confirmed the assumption that OpenFlow switches can compete with legacy switches not incurring major performance losses. Only OpenFlow-specific features, like installing flow rules or requesting counters, yield some overhead which is fortunately still admissible.

Tutors:	Jose F. Mingorance-Puga,	jose.mingorance@tik.ee.ethz.ch
	Dr. Wolfgang Mühlbauer,	wolfgang.muehlbauer@tik.ee.ethz.ch
Professor:	Prof. Bernhard Plattner,	plattner@tik.ee.ethz.ch

Acknowledgements

This semester project was supervised by Jose F. Mingorance-Puga and Wolfgang Mühlbauer. Their support helped improving the quality of the results and I'm thankful for all their comments and advices. Furthermore I would like to thank Prof. Bernhard Platter for the opportunity to experiment with a state-of-the-art OpenFlow switch.

Contents

1 Introduction

Born in the academic world, OpenFlow is an emerging communication protocol aiming to separate the control plane and the forwarding plane of a switch. Thus all control plane algorithms are executed on a dedicated server dubbed controller. The duty of the switch is confined to forwarding data according to its fowarding table, which is subjected to the controller's operations. Controller and switch communicate over a secure channel using the OpenFlow protocol. OpenFlow does away with the onerous task of having to program on a low level by providing developers with the tools to work on a high level, paving the way for next-generation network applications. Applications may include smoothly adapting traffic routes in mobile wireless networks, load balancing or increasing the energy-efficiency of communication networks. Apart from OpenFlow's undoubted supremacy over legacy switches in terms of flexibility a comparison on the grounds of performance is justified and may reveal details about the switch's inner working mechanisms.

2 Basics of the OpenFlow Protocol

As mentioned in the introduction, the OpenFlow protocol constrains the switch's operation to parsing incoming packets for flow information, checking its flow table for matches and forwarding the data to the appropriate port. When data arrives for which no corresponding entry is found in the flow table the controller is consulted. The controller decides what should be done with the data: a new flow table entry could be installed, an existing one can be modified or deleted. Table 1 shows the form of a flow table entry. Figure 1 depicts how a possible set-up might look like. Each entry is assigned a multitude of counters for statistical purposes. Counters are also stored per table, per port and per queue. Especially for network administrators

Header fields	Counters	Actions

Table 1: Form of a flow table entry

the counters per table, which encompass the number of active entries, the number of packet look-ups and the number of matches, may be interesting. The action field may contain several instructions on how to deal with the packet. Options include forwarding data, dropping data, enqueuing data and modifying flow fields of the packet. The header fields specify the fields against which incoming packet is matched. Table 2 shows which fields exist. Flows are matched based on their priority. Exact flows have a higher priority than wildcarded flows.

Ingress Port	Ether src	VLAN id	IP src	TCP/UDP src
	Ether dst	VLAN prio	IP dst	TCP/UDP dst
	Ether type		IP proto	
			IP ToS	

Table 2: Fields to match against

Each flow table entry can be configured to automatically expire. This is achieved by setting the hard timeout and the idle timeout. Whereas the idle timeout timer starts as soon as the packet stream belonging to the flow in question ceases. If idle timeout and hard timeout are both set to zero the flow will permanently remain in the switch.

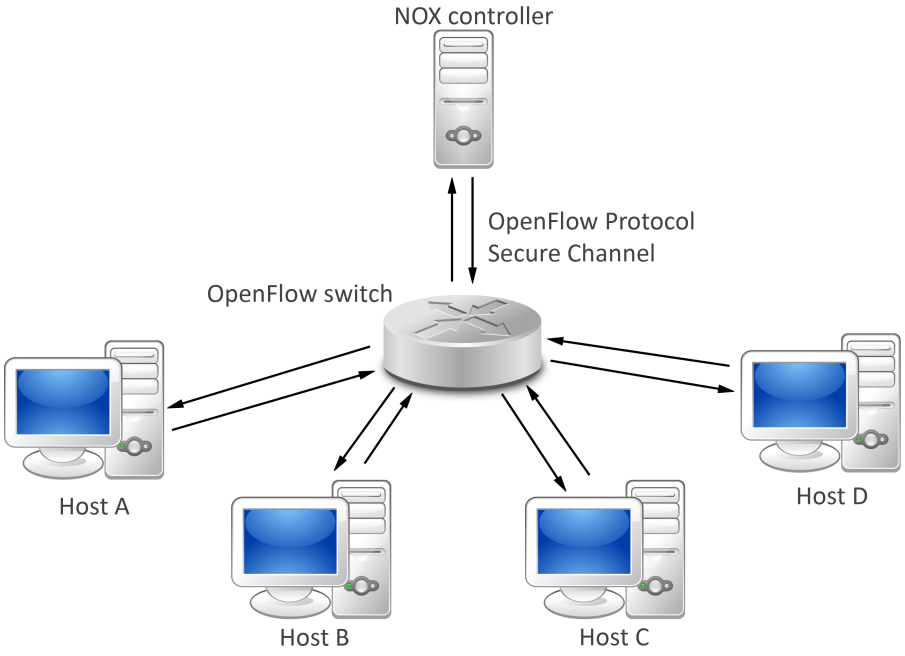

Figure 1: The OpenFlow Protocol[1]

3 Performance metrics

Evaluating the performance of an OpenFlow switch is an iterative process involving the determination of useful metrics, their measurement and analysis. Figure 2 serves as a reference figure to show the timing quantities

[1]Pictures were taken from [6], [1], [5].

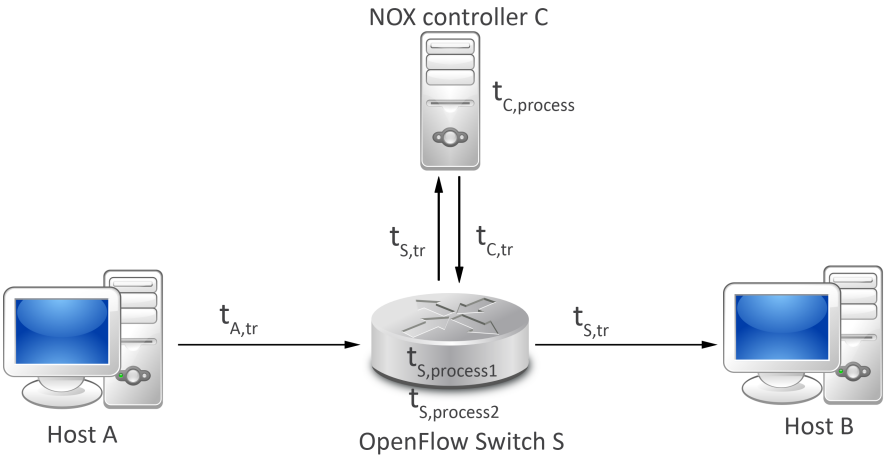

Figure 2: Packet transmission with flow rule installation

governing a packet transmission across a single OpenFlow switch. Assuming a transmission line of 1m the propagation delay approximately is 5ns.

$$t_{prop} \approx \frac{distance}{2/3 \cdot speed\ of\ light} = \frac{3}{2} \cdot \frac{1m}{3 \cdot 10^8 m/s} = 5ns \tag{1}$$

Therefore it is safe to consider t_{prop} as negligible. $t_{A,tr}$,$t_{B,tr}$,$t_{C,tr}$ and $t_{S,tr}$ are the transmission times, i.e. the time the sender is spending to put the bits on the link. All transmission lines in the setup have the same link speeds. As the packet is encapsulated in the secure channel, the packet size is not the same on all links. The following approximation is however precise enough. $t_{tr} = t_{A,tr} = t_{B,tr} = t_{S,tr} = t_{C,tr}$.

$$t_{tr} = \frac{packet\ size}{link\ bandwidth} \tag{2}$$

$t_{S,process1}$ is the time it takes to process a new flow before sending the packet to the controller. $t_{S,process2}$ is the time that elapses from the receipt of the flow installation command until the packet is forwarded. The controller needs time $t_{C,process}$ to inspect an incoming packet and send the pertaining response back to the switch. The following metrics were deemed expressive of OpenFlow's nature.

3.1 Cost of installing flow rules

Installing flow rules is a crucial operation inherent to OpenFlow enabled switches. The cost of installing flow rules $t_{cost,inst}$ is defined as the amount of time elapsing from the point in time when a new flow is received to the instant when a corresponding flow rule is installed and effectively working.

One can speculate that this measurement depends both on the number of fields that are matched and which fields are matched. This scenario is drawn in figure 2.

$$t_{cost,inst} = t_{S,process1} + t_{S,tr} + t_{C,process} + t_{C,tr} + t_{S,process2} \tag{3}$$

3.2 Delay to match against different flow table entries

To measure this metric flow table entries are pre-installed. This metric again aims to represent the processing times in the switch. In this case the controller will not intervent. Figure 3 shows the involved actions.

$$t_{delay,match} = t_{S,process} \tag{4}$$

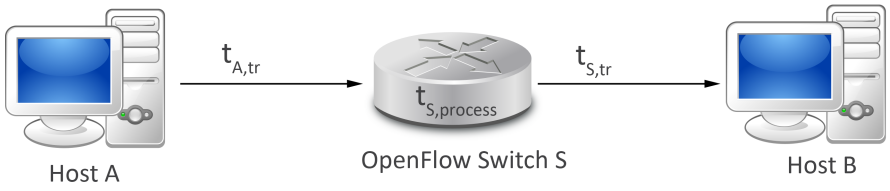

Figure 3: Delay to match against flow table entry

3.3 Comparison between OpenFlow mode and legacy switch mode

Comparing the OpenFlow switch with a legacy switch allows to draw conclusions about both the performance of OpenFlow and about how the switch works internally. The performance is assessed by varying the amount of additional traffic in the network once with the switch working in OpenFlow mode and once with the switch acting as legacy switch. As in the previous subsection the delays to match against a certain flow rule is recorded. The idea is the same as demonstrated in figure 3.

3.4 Behavior upon flow table overflow

It might be interesting to discover what happens when the flow table is overflowed. The switch will then process the packets in software which should to take longer than processing them in hardware. To report on the behavior the cost of installing flow rules amongst a surging flow table and the cost to forward a new flow with an empty and a full flow table is measured. The situation is the same as pictured in figure 2.

3.5 Requesting the counters

The counters provide invaluable information for administering and monitoring a network. Aggregate flow statistics requests are sent at some fixed rate from the controller to the switch while monitoring the CPU utilization and the number of pending requests. Requesting the counters is repeated with a multitude of frequencies.

4 Experimental setup

4.1 Devices used

The performance analysis was conducted on a NEC IP8800/S3640-24T2XW switch. The experimental servers have a 64 bit quad core Intel Xeon E5620 processor clocked at 2.4GHz and 36GB of RAM. The bandwidth of all experimental links is limited to 1 Gbit/s.

NEC IP8800/S3640-24T2XW	
Max. switching capacity	88 Gbps
Max. packet processing performance	65.5 Mpackets/s
1Gbit ports 1000BASE-T	24
1Gbit ports 1000BASE-X	4
10Gbit ports 10GBASE-R	2

Table 3: NEC switch characteristics

4.2 Finding an adequate experimental setup

At the beginning of this project the experimental setup looked like shown in figure 4. The first step consisted in pinging host B from host A. This worked fine as long as the switch operated as a legacy switch. With OpenFlow enabled however no ping ever reaches host B, although the NOX controller can successfully connect to the switch. Unfortunately this setup was doomed to fail, since data from and to the secure channel is not matched against the flow table [3, p. 12].

The next setup looks far more promising, it is shown in figure 5. In this setup the NOX controller running on host A is directly connected to host C via a dedicated link. Thanks to FlowVisor [7] Host C can relay the controller's messages to the OpenFlow switch. As a result this setup does not suffer from the problem experienced in the first setup. And as expected pinging now works in OpenFlow mode.

The final setup is shown in figure 6. This setup only needs a single machine to perform the measurements. All previous setups possessed one

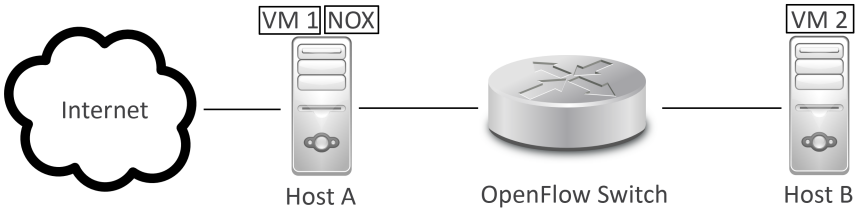

Figure 4: First setup

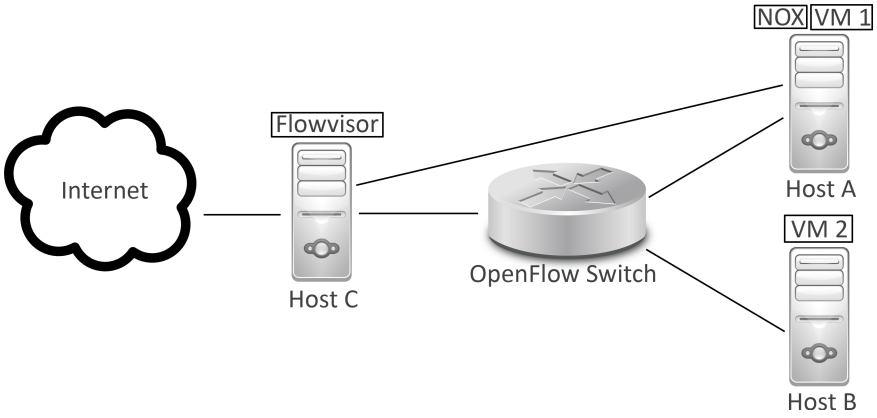

Figure 5: Second setup

machine to initiate the transmission of data and another machine to receive the data, perform a minor computation and send back a response. Host A has 2 virtual machines that are each assigned a single network interface directly connected to the switch. The NOX controller again communicates with the switch via FlowVisor on host B. Another advantage of this setup is the direct connection to the Internet without a detour, which tremendously simplifies installation of software on the servers.

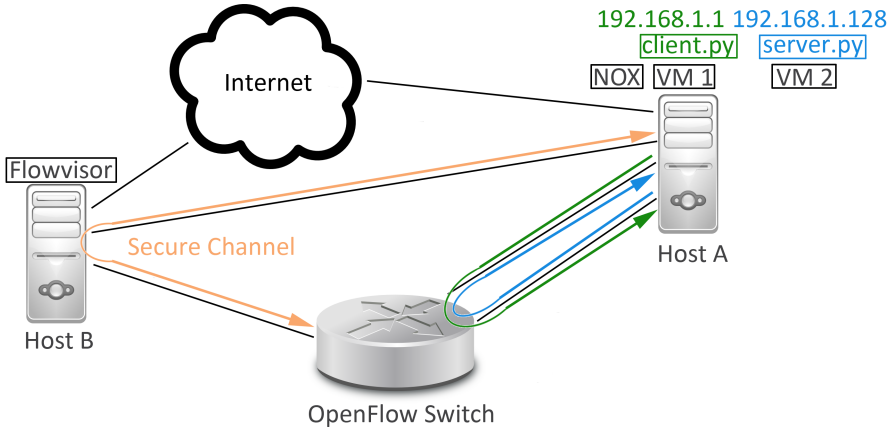

Figure 6: Final setup

4.3 Measurement approaches

All metrics can be assessed by gauging the delays to send a message. To deduce the metrics a clever approach to measure the delays is therefore crucial.

4.3.1 Measuring on 2 machines

This approach consists in sending a message containing the current timestamp to the second machine. Upon receipt this machine records the current time and subtracts the received time from it to obtain the delay. Although this approach seems simple and straight forward it is not easy to get accurate results. One would expect 2 VM's running on the same host OS to be perfectly synchronized, but even in this scenario the attempt to get precise delay figures fails. During the experiments it was found that the clocks of the 2 VMs were always some 3.5ms apart and as a consequence all measurements are obviously worthless.

4.3.2 Measuring on a single machine

In this case a single machine is commissioned to measure and compute time differences. This way problems related to imprecisely synchronized machines are solved. However a new way to calculate the delay is sought. One method involves sending a message containing the current timestamp to the second machine which will immediately return the message back to the sender. The first machine which initiated the transmission is then in a position to compute the round trip time. Assuming that a message needs the same amount of time to be sent from machine A to machine B than back from machine B to machine A, the time to send the message in one direction can be found by calculating $\frac{RTT}{2}$.

4.4 General measurement approach

The second measurement approach was deemed the only viable one and was thus adopted. The experiments now consist in computing $\frac{RTT}{2}$ in a multitude of different situations described in more detail in the section 5. All sent probe messages are UDP messages since TCP introduces too much overhead which will interfere with and falsify the measurements.

4.5 Setting up the NOX controller

A thorough guide on how to install the NOX controller software can be found on the web [4]. Although the installation of the NOX controller posed no problems, the start-up did not work at all. The error message "Cannot change the state of 'python' to INSTALLED" popped up upon launching the application. To solve this it suffices to change line 133 in nox/build/src/nox/coreapps/pyrt/pyoxidereactor.py to "signal.signal(signal.SIGCHLD, lambda:self.callLater(0,reapAllProcesses))".

4.6 Setting up the VMs

The Xen Virtual Machine Monitor software was used to run the Virtual Machines. The two VMs were assigned IP addresses in two different subnets. VM1 was given the IP address range 192.168.1.1/25 and VM2 the address range 192.168.1.128/25. Furthermore VM1 and VM2 were bridged two different interfaces which were connected to the OpenFlow switch. An entry in the VMs' routing table ensures that VM1 and VM2 can exchange messages. Moreover the VMs were assigned 128 MB of RAM memory. The experiments were conducted with python scripts stored on both VMs. The "client" script computes the various metrics and automatically writes all measurements and a summarizing report to a file. The "server" script listens for incoming packets and sends them back to the originator.

5 Measurement methodology and performance results

All the experiments required recording a large number of samples. To describe the distribution of the numbers, the mean and standard deviation are computed. The distribution is neither a Gaussian distribution nor a uniform distribution. Therefore the mean is assumed to be the average value and the standard deviation is approximated using following definition, prevalently dubbed the sample standard deviation:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{5}$$

Where

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{6}$$

and x_i are the measured sample values. The frame size is known to be 55 bytes long. With a link speed limited to 1Gbit/s both the switch and the VM spend

$$t_{tr} = \frac{packet\ size}{link\ bandwidth} = \frac{55 \cdot 8bit}{1000 \cdot 10^6 bit/s} = 0.44\mu s \tag{7}$$

to put the bits on the link. Thus we can compute the processing time in the switch according to

$$t_{pr} = \frac{RTT}{2} - 2 \cdot t_{tr} \tag{8}$$

5.1 Cost of installing flow rules

For this experiment the idle timeout is set to 0 and the hard timeout is set to 1. Flows will expire 1 second after their insertion. The actual measurement is performed by sending 1000 packets and waiting 4 seconds after the reception of the last packet before sending the next packet. Every sent packet will therefore be treated as a new flow and the controller has to be consulted. Table 4 summarises the results for the different flows. The sign 'x' means that this field is matched against. It was found that not all possible combinations are accepted by the switch which then refused to install a flow rule. Only the configurations showed in the table induced the installation of a flow rule in the switch. Exact flow rules need the least time to install, while wildcarded flow rules just matching against 2 or less fields take the most time to install. The only two flow rules not follwing this logic are "Src IP" and "Inport". The time required to install a flow thus depends on the fields that we match against and the number of fields that are matched.

	Matching fields									Time in ms	
Description	in_port	dl_src	dl_dst	dl_type	nw_src	nw_dst	nw_proto	tp_src	tp_dst	Mean	σ
Exact	x	x	x	x	x	x	x	x	x	2.598	0.437
Inport	x									3.275	1.001
Src IP				x	x					5.468	1.013
UDP port				x	x	x	x	x	x	5.556	1.671
Src UDP port				x	x	x	x	x		5.645	2.056
IP src and dst				x	x	x				5.656	1.989
Dst UDP port				x	x	x	x		x	5.795	2.214
MAC		x	x							8.675	1.764
Src MAC		x								8.754	1.735
Dst IP				x		x				8.993	1.768
Dst MAC			x							9.022	2.043

Table 4: Cost of installing flow rules

5.2 Delay to match against different flow table entries

This experiment is about measuring t_{pr} when the packet's flow table entry is already installed in the switch. The average values have been deduced from 10000 samples. The results are diplayed in table 5. The processing times are all roughly the same no matter which is the flow rule that is matched against.

5.3 Overflowing the flow table

The maximum number of flow table entries the switch can store depends on several parameters: the device, the flow detection mode and the flow statistics. The experiments were conducted on a NEC IP8800/S3640-24T2XW with flow detection mode set to "openflow-3" and flow-statistics assuming the value "single". Thus according to the "OpenFlow Feature Guide" the switch is configured to accommodate a maximum of 1024 hardware based flow table entries and 2048 software based flow table entries [2, p. 30]. While filling the flow table a warning message is triggered as soon as the threshold value specific to the inserted flow entry is exceeded. The results show that at this point the behaviour of the switch is not changing yet. Experiments revealed that the threshold values and the maximum number of table entries vary significantly with respect to which kind of flow entries are installed.

In order to overflow the flow table 1000 packets are sent from VM1 to VM2. Each of these packets has exactly the same flow details except for the destination UDP port which is steadily incremented by 1 in every packet. As each packet triggers a response packet at VM2, 2 different but symmetric flows (source and destination numbers are swapped) are being inserted into the flow table. After overflowing the flow table the behaviour of the switch changes abruptly.

	Matched fields									Time in μs	
Description	in_port	dl_src	dl_dst	dl_type	nw_src	nw_dst	nw_proto	tp_src	tp_dst	Mean	σ
Exact	x	x	x	x	x	x	x	x	x	94.4	13.0
Inport	x									93.4	13.0
Src UDP port				x	x	x	x	x		94.4	13.4
Dst UDP port				x	x	x	x		x	93.3	12.9
IP src and dst				x	x	x				94.0	13.1
Src IP				x	x					94.0	13.2
UDP port				x	x	x	x	x	x	93.4	13.2
Dst MAC			x							93.7	13.1
MAC		x	x							93.6	13.0
Src MAC		x								93.9	13.0
Dst IP				x		x				93.8	13.4

Table 5: Delay to match against different flow table entries

Hardware flow table entries		
Flow	Threshold	Maximum number
Exact	800	1008
UDP port dst	52	67
UDP port src	52	67
UDP port	52	67

Table 6: Maximum number of table entries

Time used to forward new flow in ms					
Flow	Empty table		Full table		OpenFlow overhead factor
	Mean	σ	Mean	σ	
Exact	2.598	0.437	34.669	2.622	13.3
UDP port	5.556	1.671	40.667	3.121	7.3
UDP port src	5.645	2.056	40.710	3.181	7.2
UDP port dst	5.795	2.214	40.674	3.052	7.0

Table 7: Forwarding new flow with empty and full flow table

When the flow table is completely full the switch is as fast as before, dealing with known flows. But when a new flow appears the switch runs into trouble. The new flow should be installed, at the same time all flows in the table are required to remain persistent. The switch will refrain from evicting a flow rule from the full hardware flow table and forwards the message using the software flow table. The plots show that this significantly inflates the forwarding delay of the message.

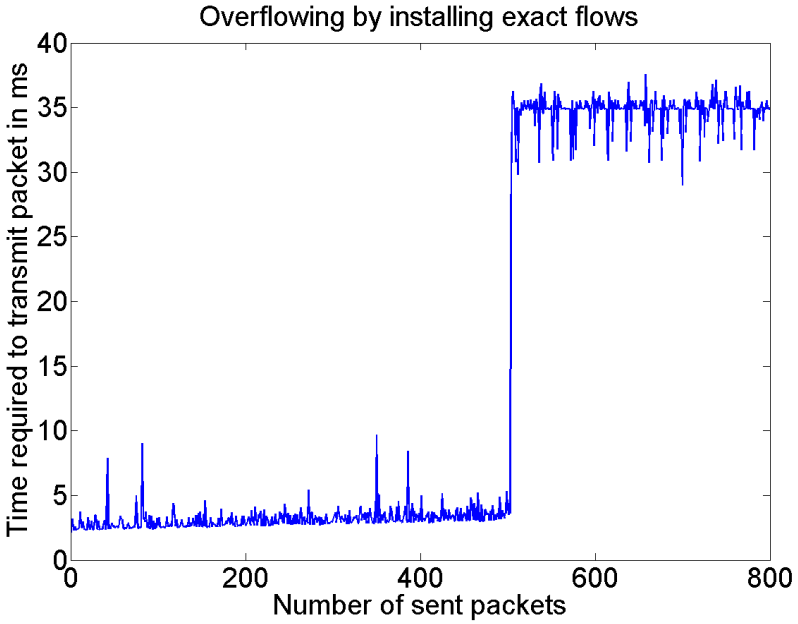

Figure 7: Flow table overflow

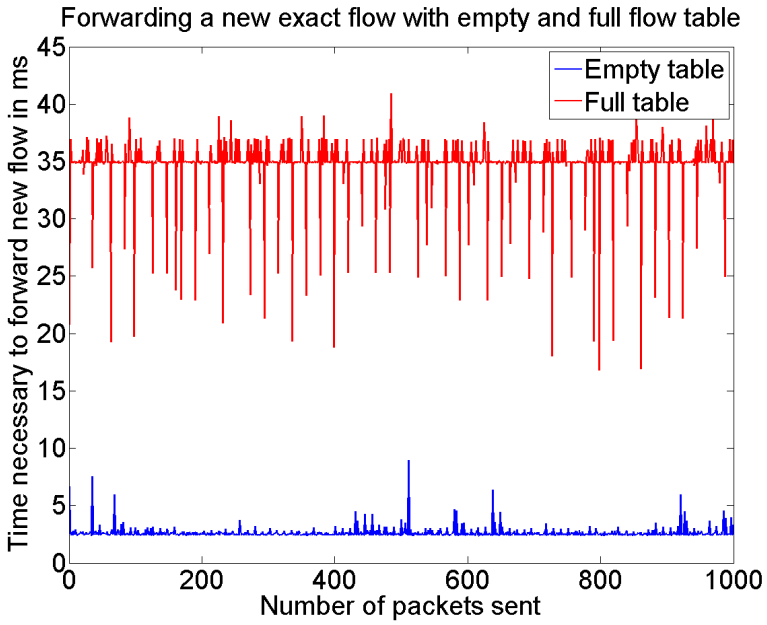

Figure 8: Forwarding a flow in a full and empty table

5.4 Comparison between OpenFlow mode and legacy switch mode

In this experiment the controller pre-installs flows only containing the inport into the switch. The switch in the experiment can work as an OpenFlow switch and as a legacy switch, therefore all experiments can be conducted on the same device. The performance is measured with different loads in the network, ranging from no additional load up to 1000Mbit/s of supplementary traffic.

Figure 9 and table 8 show that OpenFlow and the legacy switch behave quite similary. Any deviations are in the order of a percent. This may confirm the assumption that OpenFlow is working with the same circuitry than the simple legacy switch.

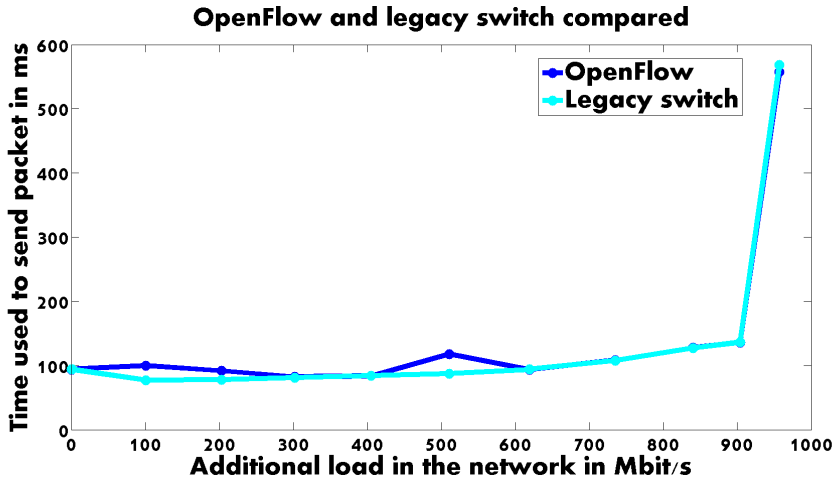

Figure 9: Comparison between OpenFlow and legacy switch mode

5.5 Requesting the counters

To discover how often the counters can be successfully retrieved, the controller requests the counters at some fixed rate and at the same time the CPU utilization and the number of pending requests is monitored. Varying the counter request frequency the blue curve in figure 10 is obtained. At the same time the average number of pending requests is measured. The number of pending requests is increasing with the frequency however by far not as significantly as the CPU utilization. Around 50 Hz some requests cannot be answered before the next request is sent, as a consequence the average number of pending requests surpasses 0.

Switch mode	Additional load in the network in Mbit/s											
	0		0.988		101		203		302		405	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
OpenFlow [μs]	93.4	13.0	94.6	12.4	100.0	50.0	91.7	36.6	81.3	12.8	83.9	8.3
Legacy switch [μs]	93.9	13.1	94.0	12.1	77.3	67.0	77.9	17.2	81.2	7.2	84.3	13.6
Variation [%]	-1.48	-0.76	0.63	2.48	29.03	-25.37	17.51	112.79	0.12	77.78	-0.47	-39.12

Switch mode	Additional load in the network in Mbit/s											
	511		619		735		840		904		957	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
OpenFlow [μs]	117.9	14.6	93.5	14.4	108.0	27.0	127.8	22.8	135.7	26.1	557.0	67.5
Legacy switch [μs]	87.3	18.4	94.0	13.9	107.9	22.4	127.4	19.4	136.3	30.4	568.2	66.8
Variation [%]	34.69	-20.65	-0.53	3.6	0.09	20.54	0.31	17.53	-0.44	-14.14	-1.97	1.05

Table 8: Comparison with legacy switch

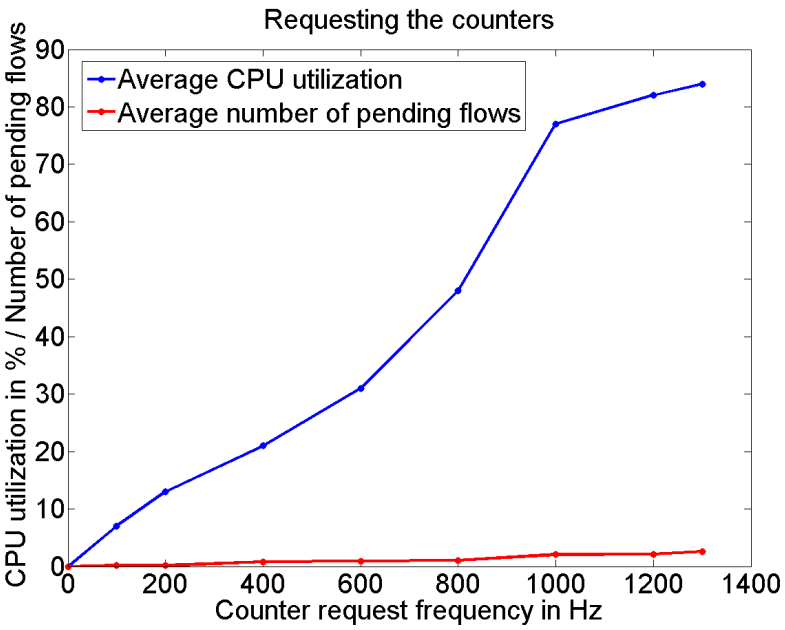

Figure 10: Request the counters

5.6 Measurements with OFlops

'OFlops (OpenFlow Operations Per Second) is a standalone controller that benchmarks various aspects of an OpenFlow switch' [8]. We used the tool to find out the time between port stats request and response.
The output of the program was the following:
Experiment has 10236 packets sent and 6543 received – 0.360785 dropped (i.e., loss = 3693) with average delay of 5018.362525 us.

6 Conclusion

This project showed that OpenFlow-enabled switches are serious competitors to legacy switches, paralleling their performance and offering great flexibility. Although OpenFlow inherent operations like installing flow rules or reading counters may introduce additional delays, its overhead still seems tolerable for future applications. Furthermore the results revealed that when providing the same conditions and functionality as a legacy switch the OpenFlow switch does not carry overhead.

7 Appendix

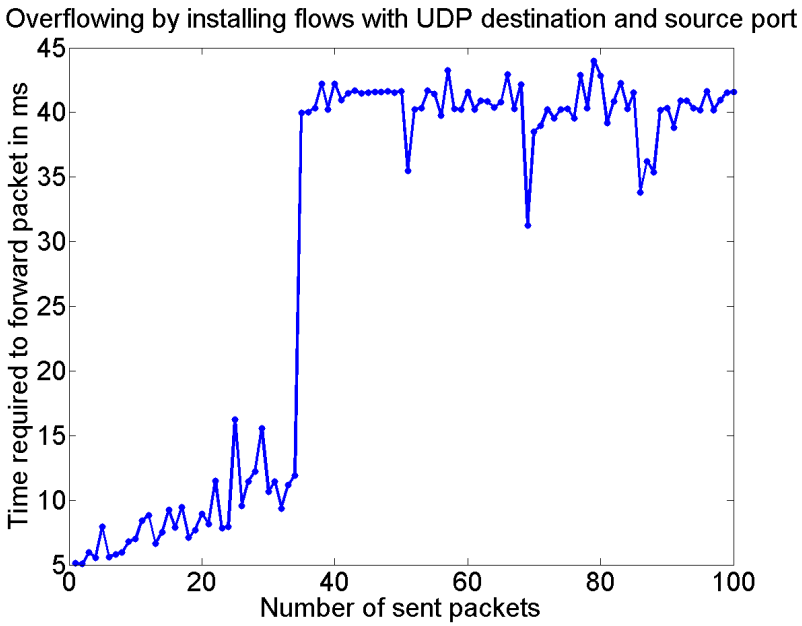

Figure 11: Flow table overflow - UDP port

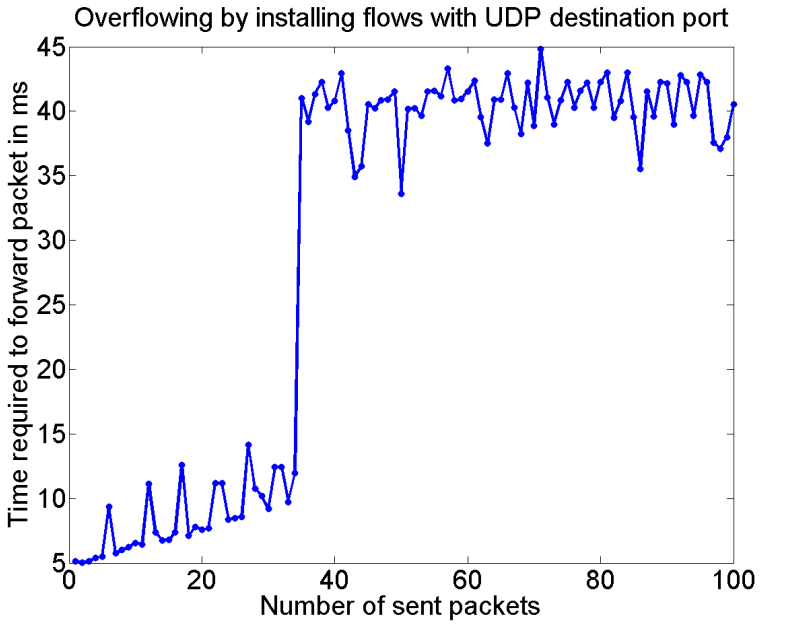

Figure 12: Flow table overflow - UDP port dst

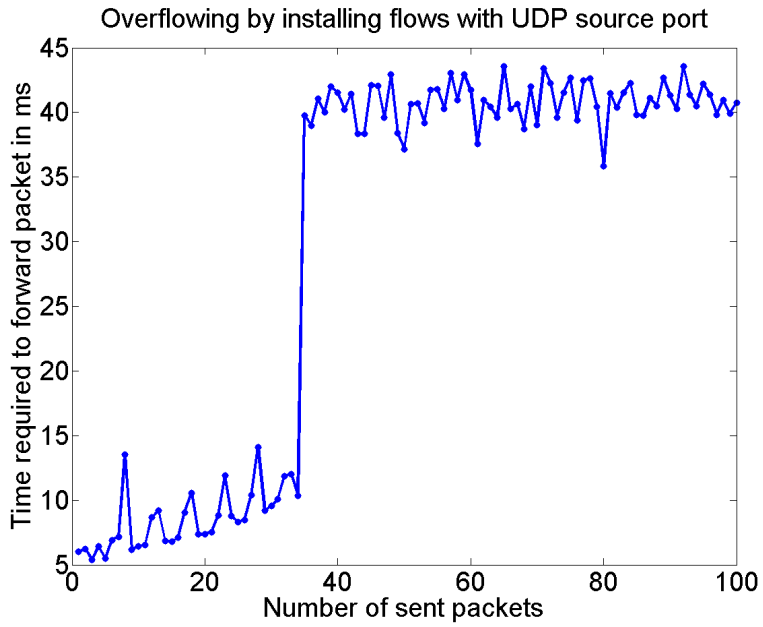

Figure 13: Flow table overflow - UDP port src

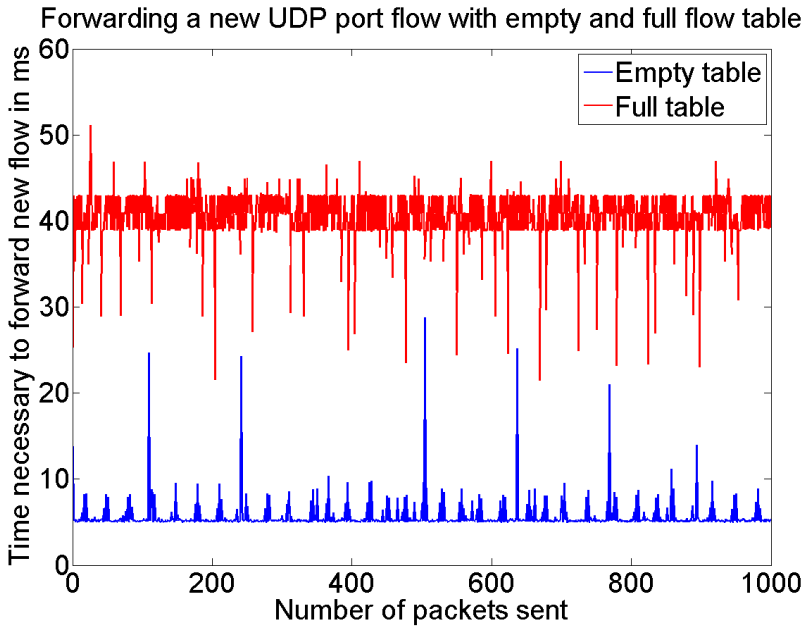

Figure 14: Forwarding a flow with full and empty table - UDP port

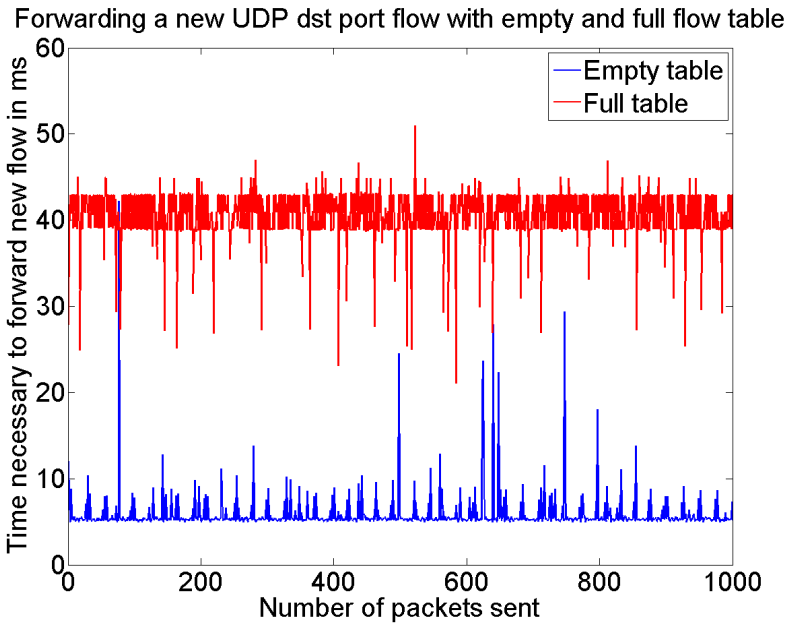

Figure 15: Forwarding a flow with full and empty table - UDP port dst

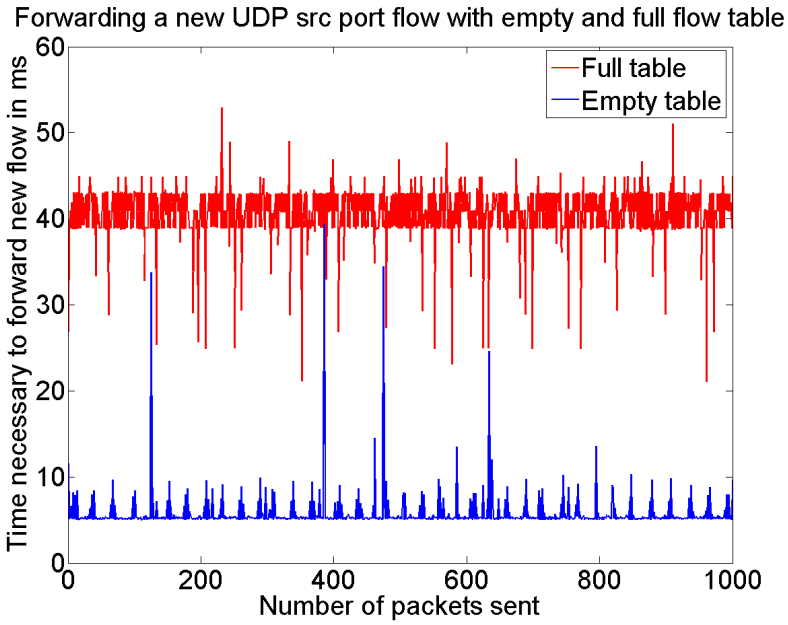

Figure 16: Forwarding a flow with full and empty table - UDP port src

References

[1] Everaldo Coelho. Computer and screen image. http://2.bp.blogspot.com/_7i8uZFjV1_k/Sp3bdfrxD8I/AAAAAAAABWU/7vW6MpWnIbU/s1600-h/500px-Computer_n_screen.svg.png.

[2] NEC corporation. Openflow feature guide. Technical report, NEC corporation, 2010.

[3] Brandon Heller. Openflow switch specification. Technical report, Standford University, 2009.

[4] NOX. Nox installation. http://noxrepo.org/noxwiki/index.php/NOX_Installation.

[5] OCAL. Cloud image. http://www.clker.com/cliparts/2/7/1/0/11949849491786662466cloud_jon_phillips_01.svg.med.png.

[6] rg1024. Router image. http://colouringbook.org/www/COLOURINGBOOK.ORG/Artists/rg1024/rg_1_24_router_coloring_book_colouring-555px.png.

[7] Rob Sherwood, Glen Gibb, Srini Seetharaman, Nick Bastin, and Anne Struble. Flowvisor. https://openflow.stanford.edu/display/flowvisor/Home.

[8] Rob Sherwood and Kok-Kiong KK Yap. Oflops. http://www.openflow.org/wk/index.php/Oflops.