



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Computer Engineering and  
Networks Laboratory

# Traffic Measurement on OpenFlow-enabled Switches

## Semester Thesis

February 23, 2012

Jochen Mattes

Advisors: Dr. Wolfgang Mühlbauer and Jose F. Mingorance-Puga  
Supervisor: Prof. Dr. Bernhard Plattner

Communication Systems Group

Department Information Technology and Electrical Engineering  
Swiss Federal Institute of Technology Zurich (ETH Zürich)

### **Abstract**

Fundamental network management tasks like Intrusion Detection Systems (IDS), accounting or traffic engineering benefit greatly from accurate real-time traffic measurements. These measurements are normally expensive and inflexible, when using custom hardware, or inaccurate due to an aggressive sampling rate on the incoming packets (c.f. NetFlow). We exploit the separation of control and data plane, provided by OpenFlow, to install dynamic measurements on commodity switches. This gives us the opportunity to take all data packets into account without using custom hardware and therefore providing highly accurate and flexible measurements at reasonable cost.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Related Work . . . . .	4
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Flow Definition . . . . .	5
2.2	Counters . . . . .	5
2.3	FlowVisor . . . . .	7
<b>3</b>	<b>Measurement Framework</b>	<b>8</b>
3.1	Introduction . . . . .	8
3.2	Obtaining the Right Counter Values . . . . .	8
3.3	Install Measurement . . . . .	12
3.3.1	Prevent Rule Modification by FlowVisor . . . . .	12
3.3.2	Prevent Rule Explosion at the Switch . . . . .	13
3.4	Report Measurements . . . . .	13
<b>4</b>	<b>Sample Application: Hierarchical Heavy Hitters</b>	<b>15</b>
4.1	Test Set-Up . . . . .	15
4.2	Application . . . . .	16
4.3	Simulation . . . . .	17
4.4	Results . . . . .	17
<b>5</b>	<b>Conclusion</b>	<b>20</b>
5.1	Conclulsion . . . . .	20
<b>6</b>	<b>Appendix</b>	<b>23</b>
6.1	Task Description . . . . .	23
6.2	Declaration of Originality . . . . .	27

# Chapter 1

## Introduction

### 1.1 Introduction

Fundamental network management tasks like Intrusion Detection Systems (IDS), accounting, traffic engineering benefit greatly from accurate (real-time) traffic measurements.

Intrusion Detection Algorithms are rarely designed as packet streaming algorithms and proposed systems normally use some sort of aggregated key figures as input e.g. the traffic histogram (packets per flow) or the statistical entropy of the traffic histogram [1]. Furthermore, close-real-time knowledge of the traffic is necessary to capacitate the system administrator to make informed decisions. For Traffic Accounting an offline-calculation of the traffic produced / consumed by a specific client is sufficient, yet high accuracy is demanded.

In consequence we reason that a suitable measurement system (i) allows the administrator to dynamically change what is measured, (ii) delivers traffic measurements in close-real-time, (iii) produces accurate results with low error bound, (iv) introduces low overhead and (v) is cost-efficient.

Current solutions fail to attain these requirements: NetFlow enables fine-grained traffic measurements directly on the switch, but to keep the additional workload low, an aggressive sampling rate on the packet stream is necessary, which has a undesired effect on the error bound of the accuracy [2]. Data Streaming algorithms have been proposed [3] [4], but they rely on custom hardware, as commodity switches do not offer the required resources.

In this semester thesis we present a measurement framework that exploits the separation of control and data plane provided by the widely supported OpenFlow protocol and seek to meet the above-mentioned requirements.

The OpenFlow protocol allows live-manipulation of the flow tables of OpenFlow-enabled commodity switches [5]. Here the data plane and control plane run on separate hardware: the data is further-on handled by the hardware switch, which makes the system fast and the control plane runs on a separate controller that might be a Linux

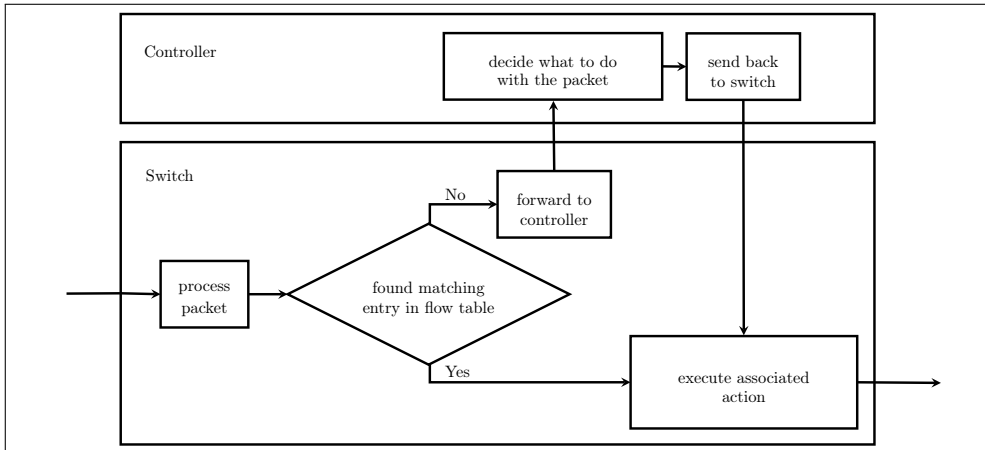


Figure 1.1: *The incoming packet is matched against the flow table and forwarded to the controller if no match is found. The controller decides what to do with the packet and sends it back to the switch. The switch then executes the action which the controller defined. If the packet matches an entry in the flow table the action is directly executed.*

server, which makes the system flexible.

OpenFlow-enabled switches treat incoming traffic similar to conventional switches: the incoming packet is matched against the flow table and the associated action (e.g. forwarding/dropping the packet) is executed. The main difference is that a normal switch would drop or maybe flood the packet if it has no matching rule in the flow table. An OpenFlow-enabled switch can be configured to encapsulate the packet and forward it to the controller, which can use the flexibility of software to analyze the packet and decide what to do with the packet. It can then send the packet back to the switch including the associated action. Figure 1.1 illustrates in a simplified way how the packet is processed.

The controller can determine the behavior of the network switches by installing flow table rules that consist of three parts: (a) the *match fields*, which define the flow space for which the rule is valid (e.g. IP range: 192.168.0.0/24), (b) a *actions part* which defines what to do with packets that match the matching fields and (c) a *counters section* which comprises statistics about the rule [6].

The presented measurement framework runs as NOX-application (NOX being a controller implementation) on the OpenFlow controller and makes use of the counter values the switch keeps for every flow entry. The measurement application can install an initial measurement flow that is inserted to the flow table once a switch registers at the controller. The framework then repeatably requests the counter values and informs the measurement application - using the framework - about the change of counter values. Moreover it offers a safe way of installing and replacing measurement

flows while theoretically ensuring that all packets are counted.

## 1.2 Related Work

Measurements in Computer Networks is a well-studied field in the Computer Network Community. Most research done relies on the collection of 5-tuple-statistics (Source IP, Source Port, Destination IP, Destination Port, Protocol) via NetFlow [7] or Data Streaming Algorithms that involve custom hardware [8],[9]. The approaches proposed range from the solution for a very specific problem [9],[10] up to a whole framework [11].

During the work on this semester thesis Lavanya Jose et al. elaborated a very similar problem in [12]. They developed a measurement framework that uses the per-flow counters provided by OpenFlow to calculate the number of packets that were forwarded by the switch in the last time period. They evaluate the results by writing a measurement application that dynamically modifies the measurement rules to find the hierarchical heavy hitters in the network (in the dimension of source or destination IP). Even though the work is very similar, the results are hard to compare as they use a different error measure. In their work the error is measured by the number of misclassifications, meaning the false positives and false negatives whereas we use the absolute difference of the counter values obtained in the simulation and the measurement. This error measure seems more appealing as we are directly measuring the error introduced by our framework and not the error introduced by the algorithm used to find the hierarchical heavy hitters.

# Chapter 2

## Background

### 2.1 Flow Definition

A flow is commonly defined as an individual, unidirectional data stream between two applications, and can be uniquely identified by the 5-tuple, (source IP address, source port, destination IP address, destination port and the transport protocol)[13].” The OpenFlow specifications allow a more flexible definition which we will refer to in the later parts of this work:

**A flow is a subspace of all possible header values in which a subset of the fields in Table 2.1 are fixed to a specific value.**

This definition has the advantage that the flow can be characterized by a single entry in the flow table of an OpenFlow-enabled switch<sup>1</sup>[15].

### 2.2 Counters

The switch will keep a packet and a byte counter for every flow that is installed in the flow table. If an incoming packet matches a rule, the packet counter is incremented and the size of the packet is added to the byte counter. These counters overflow with no indicator and appropriate measures have to be applied. Furthermore the switch will keep a counter per table, port, queue, group and bucket but they are not relevant for our work <sup>2</sup>.

The OpenFlow Specifications state: “OpenFlow compliant counters may be implemented in software and maintained by polling hardware counters with more limited ranges [6]”. One consequence of that is that the current counter values associated with a flow entry can only be calculated with the knowledge of the software and the

---

<sup>1</sup>Rob Sherwood et al. states in [14] that certain switches will internally expand rules that match multiple input ports due to hardware limitations. We experienced that in our experiments as well. The implications for our project are discussed in Section 3.3.2.)

<sup>2</sup>These counters are introduced in the OpenFlow Specifications 1.1.0.

- Ingress Port
- Meta-data (new in OpenFlow Specifications v 1.1.0)
- Ethernet type
- Ethernet source
- Ethernet source mask
- Ethernet destination
- Ethernet destination mask
- VLAN id
- VLAN priority
- MPLS label
- MPLS traffic class
- IPv4 source
- IPv4 destination
- IPv4 protocol / ARP opcode
- IPv4 ToS bits
- TCP / UDP / SCTP source port, ICMP Type
- TCP / UDP / SCTP destination port, ICMP Code

Table 2.1: *We define a flow to be the subspace of all possible header values in which a subset of the fields in this list are fixed to a value. The fields can only be fixed a value in the valid value range (refer to the OpenFlow specifications [15]).*



hardware counters. The Specifications leave it open to the switch designer, when the software counters are updated. In consequence we expect varying accuracy depending on the hardware used. The time resolution of measurements with OpenFlow is directly limited by the update circle of the software counters and appropriate care has to be taken when selecting the hardware.

## 2.3 FlowVisor

FlowVisor is an application that limits the flow-space that individual controller can influence. To do so the switches are connected to FlowVisor which then is connected to the controller and theoretically masquerades the communication between controller and switch in a transparent way<sup>3</sup>. The system administrator defines a flow space  $M$  (e.g.  $M=\{\text{vlan\_id: } 13\}$ ) and assigns the flow-space to a controller. When this controller sends the command to install a flow  $f$ , FlowVisor applies the flow space mask  $M$  and installs the flow  $f_2 = \langle f_1, M \rangle$  on the switch, where  $\langle f_1, M \rangle$  is the intersection of flow space  $f_1$  with flow space  $M$ . For a more profound explanation please refer to [14].

---

<sup>3</sup>Refer to Section 3.3.1 for the explanation of an imperfection.

## Chapter 3

# Measurement Framework

This chapter describes the details of our measurement framework and is organized in the following way: Section 3.1 gives an overview of the functionality of our measurement framework. Section 3.2 illustrates the timing problems we face and how we circumvent them. Section 3.3 depicts the method used to install a new measurement and finally Section 3.4 describes how the reply of the switch is rehashed and reported to the measurement application.

### 3.1 Introduction

The custom measurement application is running as a NOX-coreapp on the controller and can make use of our measurement framework to install, replace and delete measurement rules on the switches, while the framework ensures that no packets go uncounted. Every time instance (set by the application) the framework will report the number of packets of the according rules to the application.

The framework provides an easy to use python interface as illustrated in the minimal example shown in Algorithm 3.1 . This sample application asks the framework to report the number of IP packets (dl\_type 0x800) seen by the switch every second.

### 3.2 Obtaining the Right Counter Values

In order to generate the requested traffic statistics, we accumulate the packets belonging to a flow in the time dimension. This part is done by the switch. In the simplest scenario the controller has a perfect clock and requests the statistics every time interval (e.g. every second), there is no time delay introduced between the clock interrupt and the dispatch of the TCP package, the request is sent through an ideal network which introduces no time delay, the processing queue of the switch is idle, the request is handled immediately and no data packets are processed by the switch while generating the statistics (so that the counter values of the first flow entry are as old as the counter values of the last flow entry) and the returned counter values represent the

---

**Algorithm 3.1** *Minimal example for the use of our framework.*

---

```
from nox.lib.core import *
from flow_measurement.flow_measurement import *

class Example():
    def __init__(self, ctxt):
        Component.__init__(self, ctxt)
        self.ctxt = ctxt

        # register the method that is called when
        # the framework reports a measurement
        self.measurement.register_handler( XidEvent.NAME, \
            self._report_measurements)

        # initialize a measurement with measurement period 1sec
        self.measurement = FlowMeasurement(ctxt, 1)

    def install(self):
        # tell the framework to install this measurment every time
        # a new datapath joins
        self.measurement.install_initial_measurement(
            MeasurementRule({'dl_type': 0x0800}, openflow.OFP_DEFAULT_PRIORITY))

        # start the measurment
        self.measurement.start_measurement()

    def _report_measurements(self, event):
        for dpid in event.pyevent.dpid_list:
            print "flow stats for dpid %d: %s" %(dpid, \
                event.pyevent.flow_stats[dpid])

# refer the the NOX API documentation for an expl.
# of the following lines
def getFactory():
    class Factory:
        def instance(self, ctxt):
            return Example(ctxt)

    return Factory()
```

---

current state of these counters. In that case the controller has to subtract the statistics of two subsequent time intervals to calculate the number of packets that were seen by the switch in the last measurement period. In reality none of the above idealizations holds. We are describing the problems with the individual points and present possible ways to minimize the introduced error.

**Controller timing** Installing clock interrupts on a non-real-time system with no dedicated hardware always introduces a certain inaccuracy in precision. There are ways to stabilize the interval time e.g. by calculating the sleep time for the next time instance by  $(1 + i) * t_{interval} - t_{elapsed}$  where  $i$  is the iteration variable,  $t_{interval}$  is the theoretical time between requests and  $t_{elapsed}$  is the time elapsed since the start. But that still does not take into account the time shift that the local PC clock experiences. The necessary precision is way smaller than the timing inaccuracy introduced by this part of the system. Still we suggest to keep the workload of the controller low so that the in-determinism introduced with the scheduler is kept small.

**Network** The communication between controller and switch relies on the normal TCP/IP protocol stack, with all its benefits and drawbacks. Relevant to us is that packets might be delayed in a non-deterministic way and can be lost in the network so that they have to be sent again by the TCP layer. This adds to the inaccuracy of measurements. To keep these effects low it is advisable to connect controller and switch by a direct link.

Time is divided into time slices of constant length and ideally at the end of each time slice the framework would report the number of packets that were counted in this time slices. This aspiration is challenged by the asynchronous communication between controller and switch. Suppose the controller has a perfect non-shifting clock and requests the statistics in constant intervals. The request of the framework will generate a TCP/IP packet that is forwarded through the network either directly to the switch or first to FlowVisor and then to the switch. As the in-between networks and the controller are likely to experience a change in load, we cannot guarantee that the requests arrive with constant time gaps. Even if they do, the requests are queued at the switch and the time of processing depends on the internal state of the switch. Figure 3.1 illustrates the communication between controller for the case that FlowVisor is used as a mediator. None of the durations listed in the figure is fully deterministic. Everything that happens after the counter value is copied from the internal memory of the switch to the packet that is to be sent, does not alter the counter value reported to the controller and therefore we concentrate on the time elapsing between triggering a statistics request and the time the message is generated. The standard deviation of the time that has an influence on the counter values  $T_{influencial} = t_{net:con,fv} + t_{comp:fv,1} + t_{net:fv,sw} + t_{queue:sw}$  (c.f. Figure 3.1) can be reduced by connecting the controller directly to the switch and by ensuring that the load of the network between controller and switch is sufficiently low. The OpenFlow specifications state that with the flow statistics the duration since installation of the rule is transmitted to the controller. [15]. This information could be used to examine the precision of the received counter

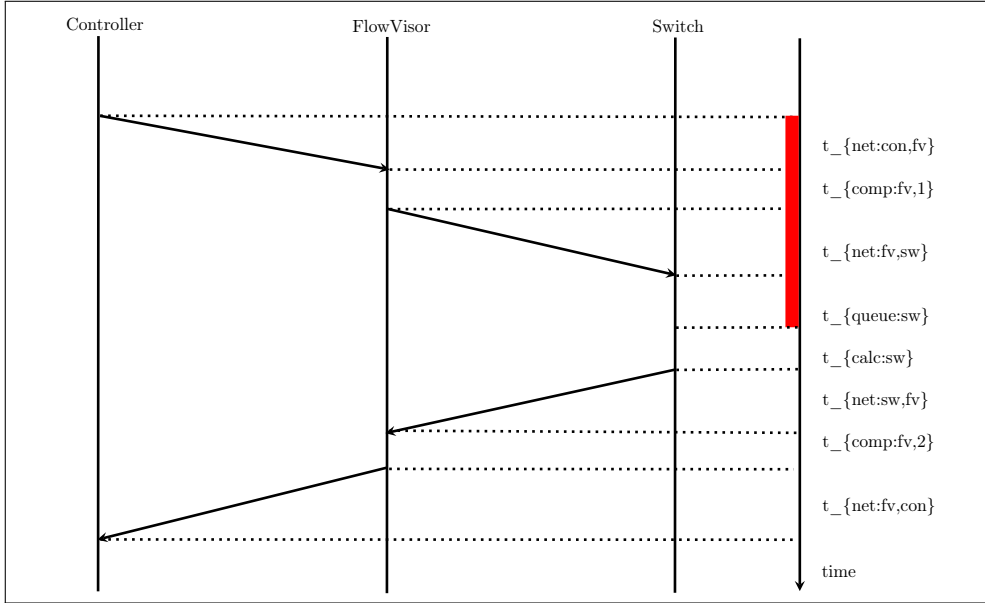


Figure 3.1: This figure illustrates the request and response of flow statistics. The controller sends the request, with delay  $t_{net:con, fv}$  FlowVisor receives the TCP/IP packet and processes and forwards it after  $t_{comp:fv, 1}$ . After  $t_{net:fv, sw}$  the packet is received by the switch and queued for  $t_{queue:sw}$ . Then the counter value is copied and the response is sent after  $t_{calc:sw}$ . The response is transmitted to FlowVisor ( $t_{net:sw, fv}$ ), forwarded ( $t_{comp:fv, 2}$ ) and transmitted to the controller ( $t_{net:fv, con}$ ). The red colored part has influence on the counter value that is reported to the controller.

values<sup>1</sup>. Yet not all implementations fully implement the standard and we therefore omit the treatment of this feature.

**Switch** Besides the expected effects e.g. the switch might drop the request due to a full queue or delay it because other requests in the queue have to be processed first. But there is a more severe problem that introduces noise into the counter values as the OpenFlow Specifications state: “OpenFlow compliant counters may be implemented in software and maintained by polling hardware counters with more limited ranges [6]”. In consequence the counter values that are returned by the switch after a statistics request represent the state of the counters at the time of the last hardware counter polling. This part of the switch-implementation directly influences the minimal attainable noise level.

<sup>1</sup>The switch we used in our experiment (NEC IP8800/S3640) sets this value to zero. We therefore do not use this feature and have to come up with a different solution.

**Measuring with artificial time** The time elapsing between the triggering of the timer interrupt to generate the statistics request and the generation of the statistics response message at the switch is hard to estimate and as the OpenFlow Specifications do not allot a time-stamp for the statistics reply it is hard to tell how far in time two statistics replies lie apart. To circumvent this problem we introduce an artificial time. Therefore another python script is used to generate a known number of packets per time unit and send these packets through the switch. The controller installs a measurement rule specifically for this flow (we call it clock flow) and uses this counter-value as time-stamp.

We can now program the measurement framework to send several statistics request in a small time window around the theoretical end of each measurement period. The measurement reply for which the counter value of the clock flow is closest to the ideal value is then chosen for further calculations.

Our experiments suggest that the update circle of the switches of our testbed is about one second with a variance of about 5 milliseconds. This renders our approach to send several statistic-requests within a small time window useless, as the returned statistics will be identical. The value of the clock flow is interesting never the less, as it gives us an estimate of the time of the last hardware counter poll and might be used in a post-processing step.

### 3.3 Install Measurement

The measurement application can request the framework to install a new measurement at any time independent of the measurement period. The framework will instantly translate the measurement request to a set of rules (c.f. Section 3.3.1 and Section 3.3.2) and install these rules on the switches that are connected to the controller.

#### 3.3.1 Prevent Rule Modification by FlowVisor

Even though FlowVisor is designed to be transparent, the current version (v 0.8.2) does not handle the flow statistics correctly<sup>2</sup>. Assume a switch is connected to FlowVisor which itself is connected to our controller. Suppose further that the controller aims to install a flow  $F_1$  (c.f. Section 2.1). FlowVisor will receive the request, intersect the flow with the assigned slice mask  $M_{fv} = \bigcup_j m_j$  (where all  $m_j$ 's are slices) and send the request with the modified flow definition  $R_1 = \langle F_1, M_{fv} \rangle$  to the switch. When the switch sends a flow statistics response, FlowVisor will not re-translate the flow definitions<sup>3</sup>, but forward the statistics to the controller as they are. The measurement framework will then fail at the attempt to match the flows  $\{F_i\}$  against the rules  $\{R_i\}$ .

To overcome this problem we exploit the fact that  $\langle \langle F_i, M_{fv} \rangle, M_{fv} \rangle = \langle F_i, M_{fv} \rangle$  as follows: when the connection between data-path and controller (via FlowVisor) is established for the first time, the controller installs the flow  $I$  that covers the whole

<sup>2</sup>The development team of FlowVisor is aware of this issue c.f. <https://openflow.stanford.edu/bugs/browse/FLOWVISOR-9>.

<sup>3</sup>Be aware that in general:  $\langle \langle F_i, M_{fv} \rangle, M_{fv} \rangle^{-1} \neq F_i = R_i$

space (all wild-card). The installed rules on the switch will then be  $\langle I, M_{fv} \rangle = M_{fv}$  and the controller obtains  $M_{fv}$  with the next flow statistics request and sets  $M_{con} = M_{fv}$ . To prevent FlowVisor from changing the measurement rules, the framework will internally translate the flows to rules ( $R_i = \langle F_i, M_{con} \rangle$ ), store the mapping and send  $R_i$  to FlowVisor, which will not alter the flow as  $\langle R_i, M_{fv} \rangle = R_i$ . With this method, re-translation of the flows is easily done at the controller.

### 3.3.2 Prevent Rule Explosion at the Switch

In the technical report of FlowVisor [14] Sherwood et al. states that: “Due to hardware limitations, certain switches will internally expand rules that match multiple input ports”. This might unnecessarily increase the number of rules that are used to measure a flow. In the case where the flow is installed to measure the traffic generated by a specific sub-net, it is likely that only one incoming port will contribute to the flow. We can use this to reduce the number of rules installed on the switch. When the switch connects to the controller for the first time it sends a feature reply message to the controller which contains details about the physical port of the switch. From the list of active ports  $\{p_i\}$  we construct the new controller mask  $M_{con} = \bigcup_i \bigcup_j m_i^{con} \cap p_j$ . This will increase the number of flow installation requests that are sent to the switch, but reduce the number of rules in the flow-table.

To further reduce the number of rules installed at the switch, the framework constantly observes the flow statistics and deletes rules that are not applied. As unknown packets are by default send to the controller the framework can install a port-specific rule, should traffic for a certain rule arrive on a new physical port.

## 3.4 Report Measurements

The framework collects all the flow statistics from the various switches that might be connected to the controller. After the response from the last switch arrived, the framework takes the absolute counter values and subtracts the stored counter values of the last time slice. If the rule has been installed in the last measurement period there will be no stored counter value and a value of 0 is assumed. The measurement is marked as invalid, because the rule was not installed for the whole measurement period and the real number of packets seen in the last measurement period is likely to be higher than the observed counter value.

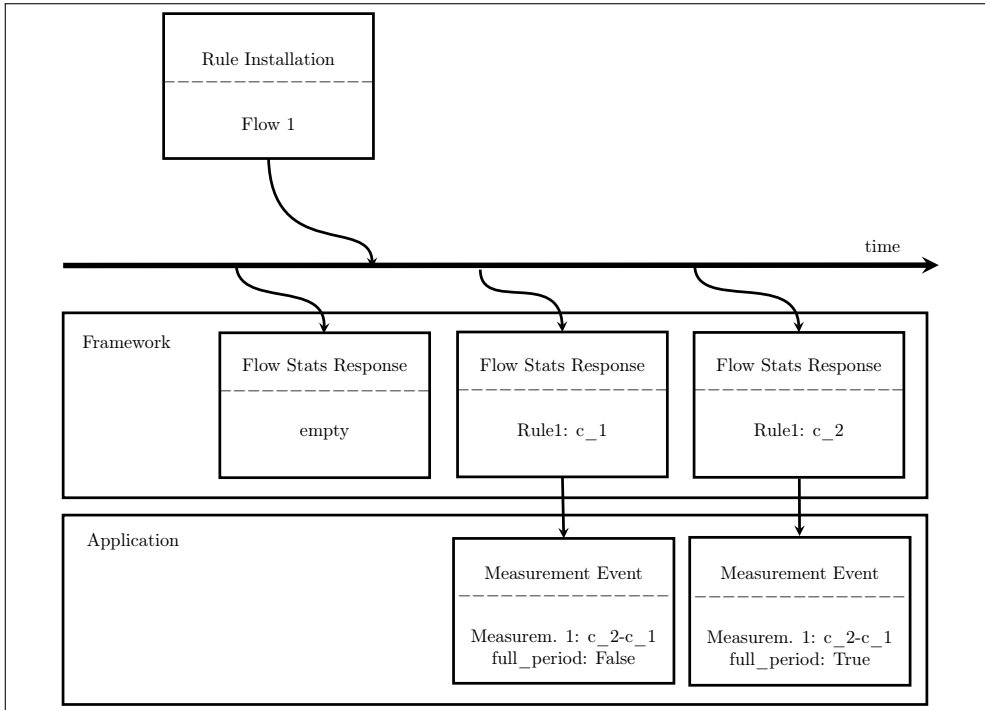


Figure 3.2: Before the rule is installed the flow statistics response received from the switch obviously does not include statistics for the flow. After the installation the counter value will be  $c_1$ , but the measurement rule was not active for the whole measurement and therefore the `full_measurement` will be set to false.



## Chapter 4

# Sample Application: Hierarchical Heavy Hitters

In order to elaborate the performance of our framework we wrote a simple application that searches the hierarchical heavy hitters and compares the results to the ones of the simulation.

### 4.1 Test Set-Up

The set-up is as simple as possible in order to reduce unrelated impacts. As physical Switch we use a NEC IP8800 S3640, the controller is running on a virtual Xen machine on physical host I and for traffic generation we use another virtual Xen machine (hereafter called traffic generator) on host II. The devices are connected with RJ-45 cables and run the standard IP protocol. Figure 4.1 depicts the architecture.

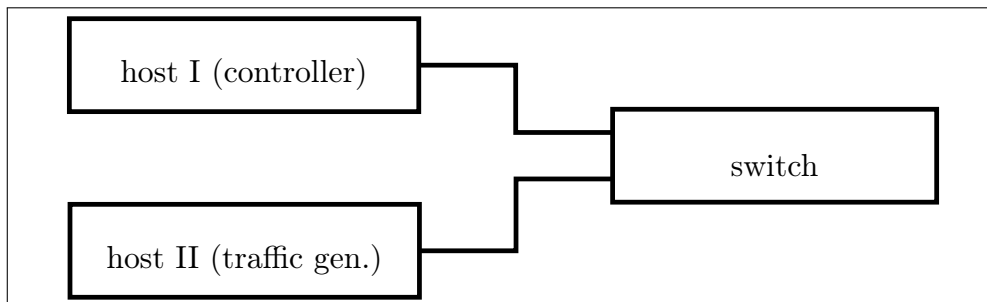


Figure 4.1: *Setup.tex*

**The traffic generator** replays a tcpdump file<sup>1</sup> at the fixed speed of 9000pps<sup>2</sup>. The clock stream is generated on the traffic generator by a simple python script that generates 100pps destined to a specific IP at a specific port. We choose 192.168.100.1:1 because the traffic traces we use, are of a backbone router and this address should not occur there. Analysis of the traces confirms this assumption. To obtain a basis for the simulation of the system we are running tcpdump on the traffic generator and record both the traffic of tcpreplay and the clock flow script.

## 4.2 Application

The application divides the IP address space in sub-nets so that each of these sub-nets contributes at most  $x$  percent of the traffic. Initially it installs two rules  $0/31^3$  and  $2^{31}/31$ . If the traffic contribution of one of these parts is larger than  $x$  percent of the total traffic, the application will split that specific address space into two parts of equal size that cover the whole space of the replaced rule. Algorithm 4.1 describes the application with pseudo code.

---

**Algorithm 4.1** *Sample application in pseudo code. `adapt_address_spaces()` is called after each measurement interval.*

---

`address_spaces = (0/31 , 231/31)`

```
function adapt_address_spaces(measurements){
    for measurement_data in measurements{
        if measurement_data > x% of total traffic {
            cur_address_space = address_spaces[measurement_id]
            remove cur_address_space from address_spaces

            new_address_space1 = first half of cur_address_space
            add new_address_space1 to address_spaces

            new_address_space2 = second half of cur_address_space
            add new_address_space2 to address_spaces
        }
    }
}
```

---

<sup>1</sup>We have been testing with the tcpdump files of <http://mawi.wide.ad.jp/mawi/>

<sup>2</sup>This is the parameter given to tcpreplay. The analysis of the generated tcpdump shows that we are only sending about 8800pps

<sup>3</sup>integer representation of the ip address

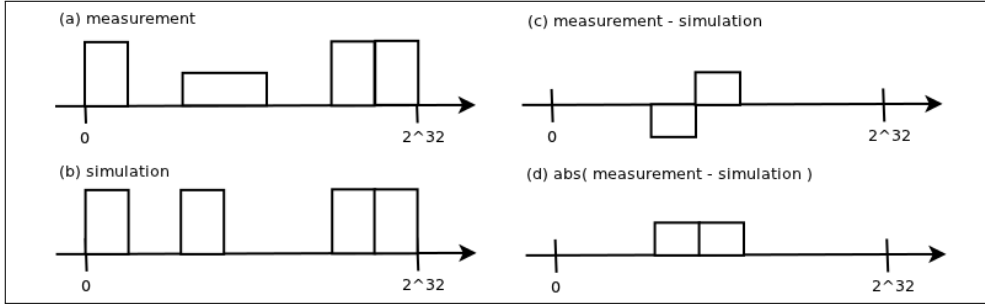


Figure 4.2: *Calculating the relative error.*

### 4.3 Simulation

As ground truth for the validation of our framework we use the tcpdump files of the traffic generator. We wrote a simple python script that analyzes the dump file and generates a “perfect” measurement and stores it in the same format as the measurement results. We use the number of measured packets in the first time interval to synchronize measurement and simulation. That means if the framework measured  $n$  packets in the first time interval, the simulation will skip the first  $n$  packets and start the simulation with the second time interval and a synchronized clock.

### 4.4 Results

The results of measurement and simulation are two distributions that can be represented by two bar charts in which the bars have variable width as illustrated in Figure 4.2 (a) and (b). We calculate the relative error by subtracting the simulation from the measurement results (Figure 4.2 (c)) and using the mass (Figure 4.2 (d)) as description of the error made. In order to make the results comparable we normalize our results by the mass of the simulation results (Figure 4.2 (b)) and obtain the relative error shown in Figure 4.3.

Figure 4.3 shows the development of that relative error over time. The curve has comparable shape to the one of the error before normalization, as the mass of the simulated distribution is fixed in the range between 8600 and 8850pps.

Figure 4.4 (a) suggests the rate limited traffic generation by tcpreplay is not perfect: even though the rate with which tcpreplay is supposed to be sending is 9000pps the analysis of the tcpdump file shows that it is only sending about 8800pps. Moreover we can see that the measured value of the number of packets in the last time unit sometimes oscillates around the simulated number. This might be explained by the uncertainty of the measuring intervals. If the interval is too long, the number of counted packets is too high; if it is too short, the number is too low. So if the time difference between measurement one and three is correct, but the measurement two

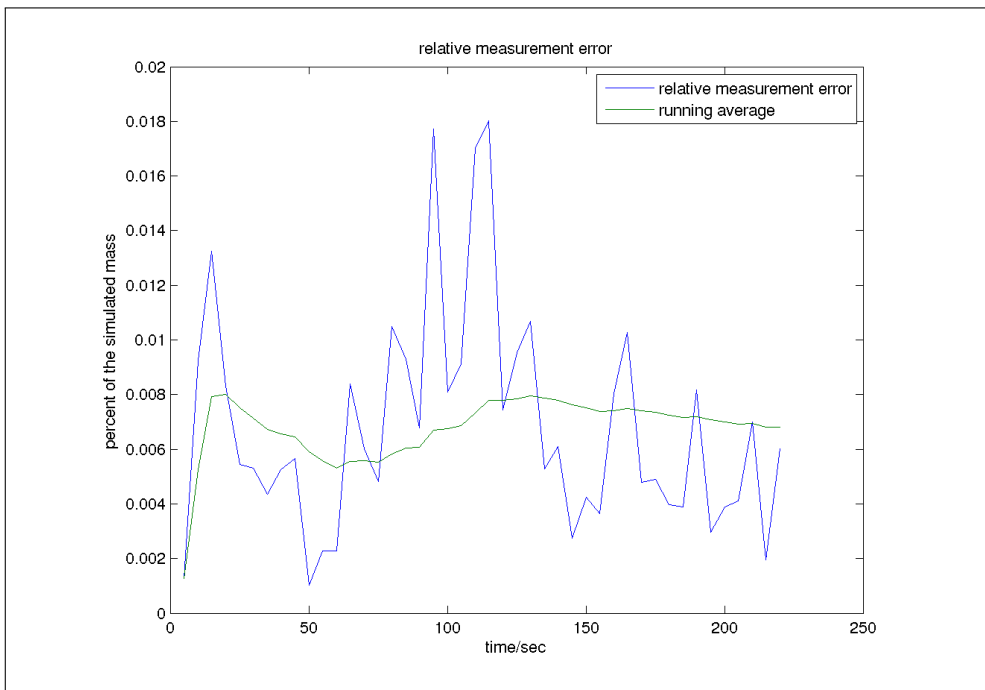


Figure 4.3: *Relative error of the measurement.*

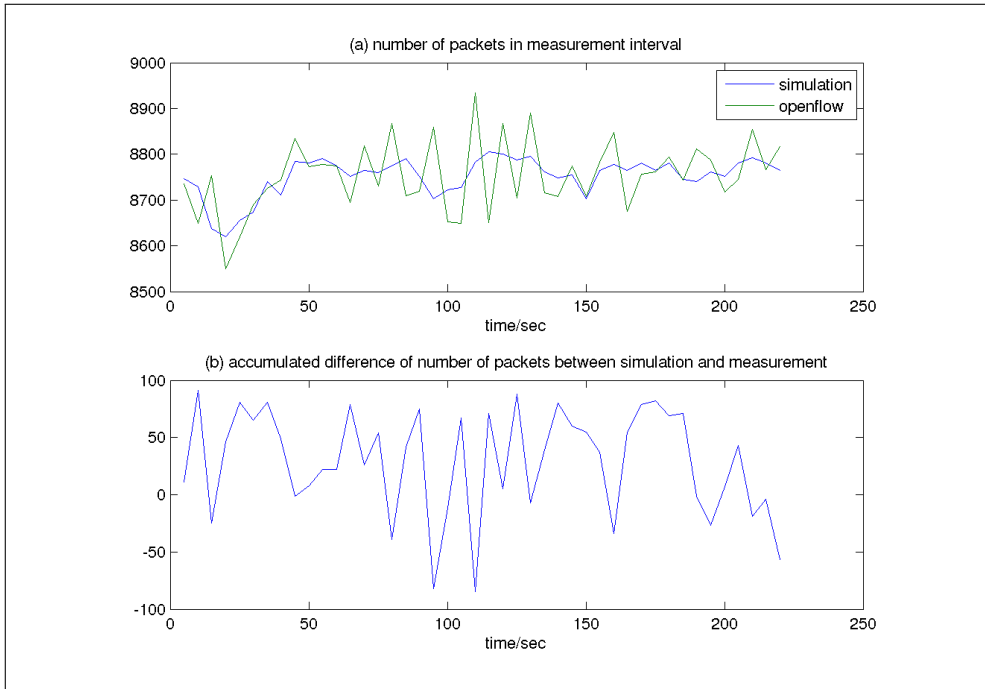


Figure 4.4: (a) number of packets in a measurement interval (b) accumulated difference in packets

is a bit early, we will first observe a measurement with too few packets and then one with too many. Figure 4.4 (b) is the running integral of the difference of the curves in 4.4 (a) and depicts the accumulated difference in the number of counted packets. That this curve is bounded between -100 and +100 packets suggests that the packets sent by the traffic generator are received and counted by the switch. If this curve had a negative trend then we had evidence that some packets remain uncounted. As this is not the case we conclude that the only source of inaccuracy is the uncertainty of the interval length.

# Chapter 5

## Conclusion

### 5.1 Conclusion

Even though measurements were not the main focus during the definition of the OpenFlow specifications v1.0.0, the offered capabilities might still be used to obtain meaningful measurements. In our experiments we observed that the measurement error was never larger than 2%, which makes our system a reasonable solution for a lot of applications, such as intrusion detection or maybe even accounting.

The main problem with version 1.0.0 of OpenFlow is that in order to use the switch as measurement-unit while maintaining the forwarding-functionality we have to apply the cross-product between the measurement rules and the forwarding rules. This leads to a massive amount of entries in the flow table, which is limited in size. It is therefore not a practical solution. The Specifications v1.1.0 allow it to define multiple tables. Therefore it would be possible to have a first table that does the forwarding and after having a hit in the primary forwarding table, clone the packet, deliver the first version and forward the copy to the measurement table. Yet, according to Nicholas Bastin from the FlowVisor development team, version 1.1.0 is unlikely to be implemented so it depends on the protocol specifications v1.2.0 whether or not traffic measurements with OpenFlow will become a practical solution in the foreseeable future.

# Bibliography

- [1] A. Lakhina, M. Crovella, and C. Diot, “Mining anomalies using traffic feature distributions,” *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 217–228, August 2005.
- [2] B.-Y. Choi and S. Bhattacharria, “On the accuracy and overhead of cisco sampled netflow,” in *ACM Sigmetrics Workshop on Large-Scale Network Inference (LSNI)*, (Banff, Canada), June 2005.
- [3] A. Lall, V. Sekar, M. Ogihara, J. J. Xu, and H. Zhang, “Data streaming algorithms for estimating entropy of network traffic,” in *IN ACM SIGMETRICS*, pp. 145–156, 2006.
- [4] A. Kumar, M. Sung, J. J. Xu, and J. Wang, “Data streaming algorithms for efficient and accurate estimation of flow size distribution,” 2004.
- [5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, March 2008.
- [6] *The OpenFlow Switch Specifications 1.0.0*.
- [7] C. Estan, K. Keys, D. Moore, and G. Varghese, “Building a better netflow,” in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM ’04, (New York, NY, USA), pp. 245–256, ACM, 2004.
- [8] A. Kumar, M. Sung, J. J. Xu, and J. Wang, “Data streaming algorithms for efficient and accurate estimation of flow size distribution,” *SIGMETRICS Perform. Eval. Rev.*, vol. 32, pp. 177–188, June 2004.
- [9] H. C. Zhao, A. Lall, M. Ogihara, O. Spatscheck, J. Wang, and J. Xu, “A data streaming algorithm for estimating entropies of od flows,” in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC ’07, (New York, NY, USA), pp. 279–290, ACM, 2007.

- [10] J. Mikians, P. Barlet-Ros, J. Sanjuàns-Cuxart, and J. Solé-Pareta, “A practical approach to portscan detection in very high-speed links,” in *Passive and Active Measurement* (N. Spring and G. Riley, eds.), vol. 6579 of *Lecture Notes in Computer Science*, pp. 112–121, Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-19260-9\_12.
- [11] M. Singh, M. Ott, I. Seskar, and P. Kamat, “Orbit measurements framework and library (oml): motivations, implementation and features,” in *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pp. 146 – 152, feb. 2005.
- [12] L. Jose, M. Yu, and J. Rexford, “Online measurement of large traffic aggregates on commodity switches,” in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services, Hot-ICE’11*, (Berkeley, CA, USA), pp. 13–13, USENIX Association, 2011.
- [13] S. S. Nalatwad, *Self-Sizing Techniques for Locally Controlled Networks*. PhD thesis, North Carolina State University, Dec 2005.
- [14] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, N. McKeown, and G. Parulkar, “Flowvisor: A network virtualization layer,” tech. rep., Stanford University, 2009.
- [15] *The OpenFlow Switch Specifications 1.1.0*.



## Chapter 6

# Appendix

### 6.1 Task Description

The Task description can be found on the following pages.

Semester Thesis  
for  
Jochen Mattes

Tutors: Jose Mingorance, Wolfgang Mühlbauer

---

Issue Date: XX.06.2011  
Submission Date: 15.08.2011

---

## Detecting Traffic Anomalies on OpenFlow-Enabled Switches

---

### 1 Introduction

The OpenFlow framework enables fine-grained flow-level control of Ethernet switching. It offers high flexibility, since forwarding rules can be dynamically adjusted at run-time. At the same time, OpenFlow has been supported by hardware vendors (e.g., NEC and HP). Therefore, there exist commercial OpenFlow-enabled switches with high forwarding performance.

A plethora of research proposals have been made that could benefit from the OpenFlow approach. We focus on measurement-related applications. In addition to matching incoming packets against a small collection of rules, OpenFlow-enabled switches update byte/packet counters if a packet matches a certain rule. Importantly, the set of rules can be dynamically adjusted during run-time. For example, one could dynamically tune the wildcard rules to quickly “drill down” to identify large traffic aggregates [1]. Overall, OpenFlow can pave the way for novel approaches in the area of network measurements, in particular for smaller “edge” networks.

### 2 Requirements

The goal of this thesis is to design a measurement framework for OpenFlow-enabled switches and to demonstrate its usefulness for one example application (e.g., anomaly detection, heavy hitter detection).

This task is split into four major subtasks:

1. **Literature study:** The student should actively study the literature and include a short survey in the final report. The focus of this survey should be on OpenFlow-related measurements and on topics that are related to the example application (e.g., anomaly detection).

2. **Measurement framework:** The student has to design and implement a measurement framework that can be run on an OpenFlow controller (e.g., NOX). The framework should continuously collect statistics from the OpenFlow switch by reading out byte/packet counters. In addition, it should provide an easy-to-use interface to increase the granularity of flow rules (“drill down”) or to decrease it (“collapse”). The interface should be designed in a generic such that it can be useful for different types of applications.
3. **Example application:** The student shows for one example application the usefulness of OpenFlow-related measurements. To this end, he will implement a logic that runs in the controller and that adjusts the granularity of flow rules depending on the needs of the example application. This will be only a preliminary, proof-of-concept-like implementation. Possible example applications include anomaly detection (e.g., computing entropy values) or heavy hitter detection.
4. **Evaluation:** Finally, the student will do a preliminary evaluation of this approach, either on productive or on synthetic traffic.

### 3 Deliverables

The following results are expected:

- Short survey on literature research.
- Measurement framework, see above
- Implementation of a small example application
- Evaluation of the example application
- A final report, including a concise description of the work conducted in this project (motivation, related work, own approach, implementation, results and outlook). The abstract of the documentation has to be written in both English and German. The original task description is to be put in the appendix of the documentation. The documentation needs to be handed in electronically. The whole documentation, as well as the source code, slides of the talk etc., need to be archived in a printable, respectively executable version on a CDROM.

### 4 Assessment Criteria

The work will be assessed along the following lines:

1. Knowledge and skills
2. Methodology and approach
3. Dedication
4. Quality of results
5. Presentations
6. Report

### 5 Organisational Aspects

#### 5.1 Documentation and presentation

A documentation that states the steps conducted, lessons learned, major results, and an outlook on future work and unsolved problems has to be written. The code should be documented well enough such that it can be extended by another developer within reasonable time. At the end of the project, a presentation will have to be given at TIK that states the core tasks and results of this project. If important new research results are found, the results can be published in a research paper.

## 5.2 Dates

This project starts on June XX, 2011 and is finished on August YY, 2011. At the end of the second week the student has to provide a schedule for the thesis, that will be discussed with the supervisors.

An intermediate presentations for Prof. Plattner and all supervisors will be scheduled after one month.

A final presentation at TIK will be scheduled close to the completion date of the project. The presentation consists of a 20 minutes talk and reserves 5 minutes for questions. Informal meetings with the supervisors will be announced and organized on demand.

## 5.3 Supervisors

Jose Mingorance, mingorance@tik.ee.ethz.ch, +41 44 632 70 52, ETZ G 94

Wolfgang Mühlbauer, muehlbauer@tik.ee.ethz.ch, +41 44 632 70 17, ETZ G 90

## References

- [1] L. Jose, M. Yu, J. Rexford, Online Measurement of Large Traffic Aggregates on Commodity Switches, in: Proc. 11th USENIX conference on hot topics in management of Internet, cloud, and enterprise networks and services

21st June 2011

## 6.2 Declaration of Originality

The Declaration of Originality can be found on the following page.

## Declaration of originality

This signed "Declaration of originality" is a required component of any written work (including any electronic version) submitted by a student during the course of studies in Environmental Sciences. For Bachelor and Master theses, a copy of this form is to be attached to the request for diploma.

I hereby declare that this written work is original work which I alone have authored and written in my own words, with the exclusion of proposed corrections.

Title of the work

**Traffic Measurement on Openflow-enabled Switches**

Author(s)

Last name

First Name

**Mattes**

**Jochen**

With my signature, I hereby declare:

- I have adhered to all rules outlined in the form on „Citation etiquette“, [www.ethz.ch/students/exams/plagiarism\\_s\\_en.pdf](http://www.ethz.ch/students/exams/plagiarism_s_en.pdf).
- I have truthfully documented all methods, data and operational procedures.
- I have not manipulated any data.
- I have identified all persons who have substantially supported me in my work in the acknowledgements.
- I understand the rules specified above.

I understand that the above written work may be tested electronically for plagiarism.

Albstadt, 21 November 2011

Place, Date

Signature\*

\* The signatures of all authors are required for work submitted as a group. The authors assert the authenticity of all contents of the written work submitted with their signatures.