University of Amsterdam

System and Network Engineering

Research Project 2

OpenFlow network virtualization with FlowVisor

Author:
Sebastian Dabkiewicz
Sebastian.Dabkiewicz@os3.nl

Supervisors:
Ronald van der Pol
rvdp@sara.nl
Gerben van Malenstein
Gerben.vanMalenstein@SURFnet.nl

February 6, 2013

Abstract

OpenFlow networks became popular in the past years for researchers and commercial companies. OpenFlow essentially separates the control and data plane of a switch. The control plane is moved to an external controller. Network researchers have wanted to virtualize networks for a long time. Therefore FlowVisor is developed. With FlowVisor, multiple isolated logical networks (slices) can share the same network topology and hardware. FlowVisor is placed between the OpenFlow switch and multiple OpenFlow controllers and acts as transparent proxy between them. Based on incoming switch traffic FlowVisor determines which OpenFlow controller is responsible and is able to create forwarding entries for that traffic. The purpose of this research was to identify the possibilities of network virtualization of OpenFlow networks using FlowVisor. The main research question of this project was: *Is the current FlowVisor implementation version 0.8.5 suitable to create stable virtual networks in production environments?*

The research question was answered by conducting several experiments that cover real life usage of FlowVisor. During these experiments the behaviour and working of FlowVisor was observed.

The results of the experiments show that the behaviour of FlowVisor is in most cases as expected when having simple setups. Creating more advanced setups with wildcarding (using fields that are not defined) results in situations where traffic is allowed regardless of the network slice to which it belongs. This may be caused by bugs in FlowVisor or not yet implemented features.

Based on the results of the experiments the author recommends that Flowvisor in the current state of development (version 0.8.5) is not suitable to run in a production environment. If one really wants to deploy FlowVisor, it is recommended to test it extensively before deploying it in the network.

Acknowledgements

I would like to thank my supervisors Ronald van der Pol and Gerben van Malenstein for their help, input and support during this research project.

I also want to thank Michiel Appelman, Jeffrey Bosma, Maikel de Boer and Joris Soeurt reviewing and proofreading the report.

Contents

List of Tables

List of Figures

1 Introduction

Networks have become business critical infrastructure to many companies to date. Doing experiments in these networks is not possible, because they can disturb or bring down the network. Since testing new features inside a testbed is not always possible and realistic, other techniques have been developed. One of them is Software Defined Networking (SDN) [1].

SDN is a form of network virtualization where the control plane is separated from the data plane and moved to an external software controller. OpenFlow [2] is the most used SDN solution.

OpenFlow is the protocol between the data plane inside the switch and the control plane in an external controller. Usually an OpenFlow controller is connected to multiple switches and has thus a centralised view of the network.

Worldwide there are several organisations and groups of organisations that are running OpenFlow (test-)networks, like: Google [3], NDDI [4], GENI [5], JGN-X [6] and Ofelia [7].

Researchers at Stanford University started a project called FlowVisor [8] a few years ago. FlowVisor is a piece of software that is placed between an OpenFlow switch and multiple OpenFlow controllers. By defining rules inside FlowVisor multiple isolated logical networks, called Slices, can be created that share the same network equipment. Each slice is controlled by a separate OpenFlow controller.

FlowVisor is actively developed at the moment, but not many papers are published yet. Papers that are available [9] [10], describe that FlowVisor can be used to create several network Slices to be able to separate production and test networks.

SARA [11] and SURFnet [12] are interested in this kind of network virtualization. SARA is currently running an OpenFlow testbed in the Lighthouse at SARA [13]. SURFnet is planning to build a OpenFlow network in the next few months, where connected organisations can get hands-on experience with the OpenFlow protocol. SURFnet wants to offer its connected institutions their own independent network slices. FlowVisor can provide a solution for creating slices on equipment by multiple parties. Therefore this research project is started to investigate the possibilities that FlowVisor offers and if they can be used in production.

In the next two chapters of this report a general introduction to OpenFlow and FlowVisor is presented. Chapter 4 describes the different conducted experiments as well the results. The results of the experiments, and how they relate to the research questions, will be discussed in chapter 5. Based on these results a conclusion is given in chapter 6. Finally in chapter 7 there are some recommendations for future research that can be done on this topic.

1.1 Research Question

FlowVisor is running in the production network of Stanford University for some years, but it should still be considered as a research project. Therefore FlowVisor should be well tested before deploying it in a production environment [8].

During this research project I want to examine how FlowVisor works in different kind of setups, to see if it is suitable to run FlowVisor in a production network.

This leads me to the following research question and sub-questions for this research project.

Is the current FlowVisor implementation version 0.8.5 suitable to create stable virtual networks in production environments?

- How stable is an FlowVisor environment?
- Is FlowVisor user friendly?
- In which way are the switch resources separated?
- How are switch events handled?
- Which kind of bugs are known and still present?

Answering some of these questions, like how user friendly is FlowVisor, is not easy and may depend on the user who is using FlowVisor. Other questions like "How are switch events handled?" or "In which way are the switch resources separated?" can be found out just by doing some experiments and analyse the results and the log-files.

1.2 Approach

To answer my research questions I conduct some experiments. The first six experiments are built towards some real life situations, while the 7th experiment is conducted to see how FlowVisor behaves in some advanced setups where wildcards are used.

In the following list of experiments a short description of each experiment is given as well which research question it tries to answer. Some research questions won't be answered by an experiment but they can be answered based on the user experience during the research.

List of experiments:

- **Experiment 1**: For this experiment two network slices were setup, the controller of slice 2 will try to push a flow entry inside slice 1 which should no be possible. With this experiment I want to answer the question how switch resources are separated.
- **Experiment 2**: This experiment uses Virtual Local Area Network (VLAN)s to share a single port on the switch. With this experiment I want to answer the question how switch resources are separated.
- **Experiment 3**: For this experiment some slices will be set up based on application port. Traffic will flow through the network based on the source or destination application port. With this experiment I want to answer the question how switch resources are separated.
- **Experiment 4**: This experiment is used to create a setup where a normal switch is simulated. So the flow entries are based on destination Media Access Control address (MAC-address). Furthermore a flow entry

is added to handle Address Resolution Protocol (ARP) traffic to translate IP-addresses into a MAC-address. With this experiment I want to answer the question how switch resources are separated.

- **Experiment 5**: For this experiment switch events will be triggered by pulling out a cable from the switch and putting it back again. With this experiment I want to answer the question how switch events are handled.
- **Experiment 6**: This experiment is a combination of previous experiments, some traffic streams will be created based on application port, while some monitoring is done with a monitoring slice. With this experiment I want to answer how switch resources are separated and how switch events are handled.
- **Experiment 7**: This experiment is conducted to create several slices based on some layer 2, 3 or 4 field. Flow entries will be configured on the OpenFlow controllers using wildcarded fields. With this experiment I want to answer the question how switch resources are separated.

To be able to conduct the experiments I had to become familiar with the OpenFlow protocol, for which I followed an OpenFlow tutorial [14]. Thereafter some tests were done with a Pronto 3290 switch [15] and installed Open vSwitch software [16]. Open vSwitch has a built-in OpenFlow controller which was used to create simple rules.

In a later stadium several external Floodlight OpenFlow controllers were added as well the FlowVisor controller between the OpenFlow switch and the controllers that makes the experimental setup complete.

2 OpenFlow

In the last years a new protocol is developed named OpenFlow [2]. OpenFlow is the protocol between the data plane inside the OpenFlow capable switch and the control plane at the external controller. On this controller the flows are configured which will be stored in the OpenFlow switch. One OpenFlow controller can be used to control several OpenFlow switches.

The connection between the OpenFlow switch and the OpenFlow controller can be set up using Transmission Control Protocol (TCP), Secure Sockets Layer (SSL) and Transport Layer Security (TLS). SSL and TLS should be preferred for security reasons. However for research purposes a TCP connection is fine, to be able to intercept and analyse the OpenFlow packets on the wire.

Figure 1 shows a comparison between a traditional switch and an OpenFlow switch.

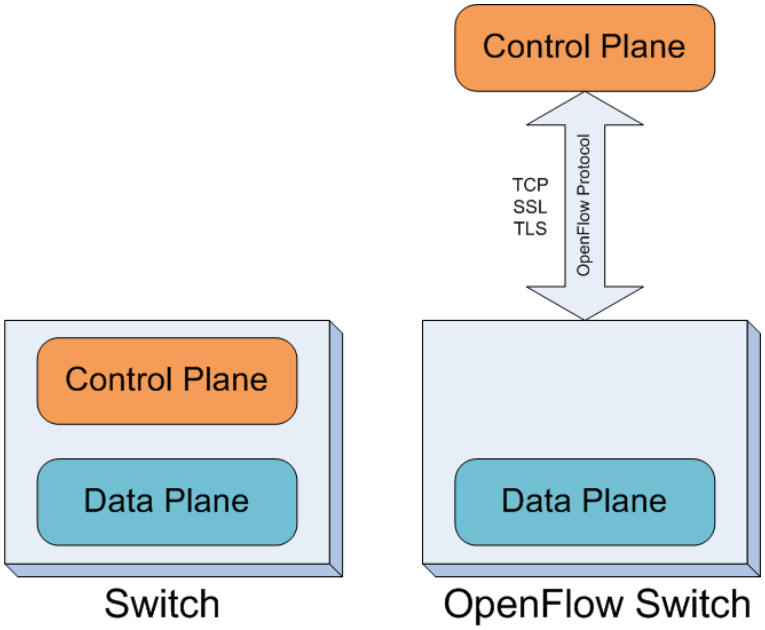

Figure 1: Difference between a traditional and an OpenFlow switch

The actual OpenFlow version is 1.0 [17], which was used during the research project, this version supports only Internet Protocol version 4 (IPv4). OpenFlow Version 1.2 [18], also supports Internet Protocol version 6 (IPv6), but a controller or switch supporting OpenFlow 1.2 is not yet available.

2.1 Flow entries

Flow entries are rules that are stored inside the OpenFlow switch and make the forwarding decision. Flow entries are not limited to layer 2 as in a traditional switch but can be based on a subset of the following fields:

- Ingress port

- Ethernet source/destination address
- Ethernet type
- VLAN ID
- VLAN priority
- IPv4 source/destination address
- IPv4 protocol number
- IPv4 type of service
- TCP/UDP source/destination port
- ICMP type/code

Fields that are not set will be wildcarded, which means that every value will match. Based on a match of one of these fields an action will be taken. This can be a very simple action by just sending the packet to an output port, or very complex by changing fields of the packet like the source MAC-address for example. A list of possible actions can be found in Table 1. A switch must support the actions that are marked as required, the other actions are optional. When a switch connects to an OpenFlow controller the switch reports which actions are supported.

Table 1: Possible OpenFlow actions

Action	Required	Optional
Forward	All	Normal
	Controller	Flood
	Local	
	Table	
	IN_PORT	
Modify Field		Set VLAN ID (or add VLAN tag)
		Set VLAN priority
		Strip VLAN header
		Modify Ethernet src/dst address
		Modify IPv4 src/dst address
		Modify IPv4 type of service bits
		Modify TCP/UDP src/dst port
Drop	X	
Enqueue		X

When a packet arrives at the switch, it will be matched against the flow entries that are stored inside the switch table. If there is a match an action is applied. When there is no match the packet is sent to the OpenFlow controller. If there is a flow entry for this packet inside the OpenFlow controller the match is stored inside the switch and the action is applied. Otherwise the packet will be dropped. A flowchart of this process can be found in Figure 2.

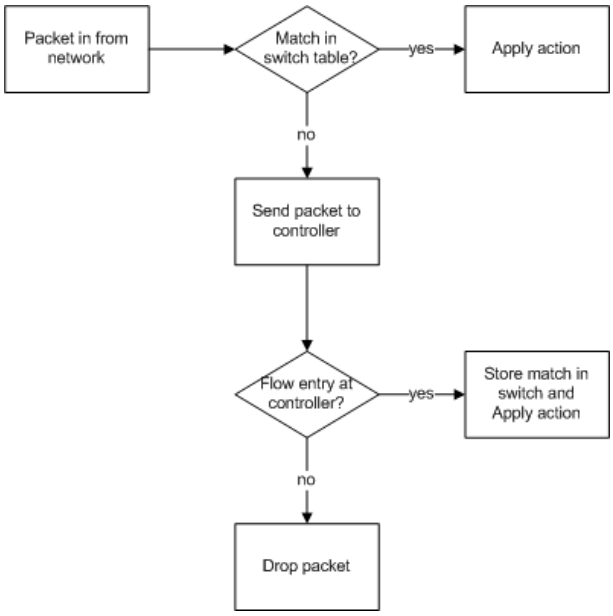

Figure 2: OpenFlow packet matching

2.2 OpenFlow Controller

An OpenFlow switch is controlled by an OpenFlow controller. This controller is a piece of software that pushes the flow entries to the OpenFlow switch.

There are a number of OpenFlow controllers available like:

- Beacon [19]
- Floodlight [20]
- Maestro [21]
- NOX / POX [22]
- Trema [23]

Floodlight is the OpenFlow controller which is used currently at SARA, so this controller will be used during this research project. Floodlight is a Java based open source controller with an Apache license. SARA uses Floodlight because it is actively developed at the moment and it has a big user base and a very active mailing list.

Floodlight is able to add flow entries proactive as well reactive into the switch. Proactive means that the flow entry is inserted to the switch before traffic arrives on the switch. So directly after configuring the flow entry on the OpenFlow controller the entry is sent and stored in the switch.

Reactive means that when a packet arrives at the switch, the packet is sent to the OpenFlow controller. On the OpenFlow controller the forwarding decision is made and the packet is sent back to the switch. The forwarding entry is then stored in the flow table of the switch.

Flows are added to the controller using an REpresentational State Transfer (REST) Application Programming Interface (API) named Static Flow Pusher [24]. This is a JavaScript Object Notation (JSON) interface to configure Floodlight. At the moment a more extended version of this API is in development and will be released soon [25].

Floodlight also offers a web-interface that gives information about the switches and the attached devices. However, no configuration can be done using the web interface. A screenshot of the Floodlight web interface can be found in Figure 3. It shows the dashboard with information about the controller, the connected switch as well the attached devices.

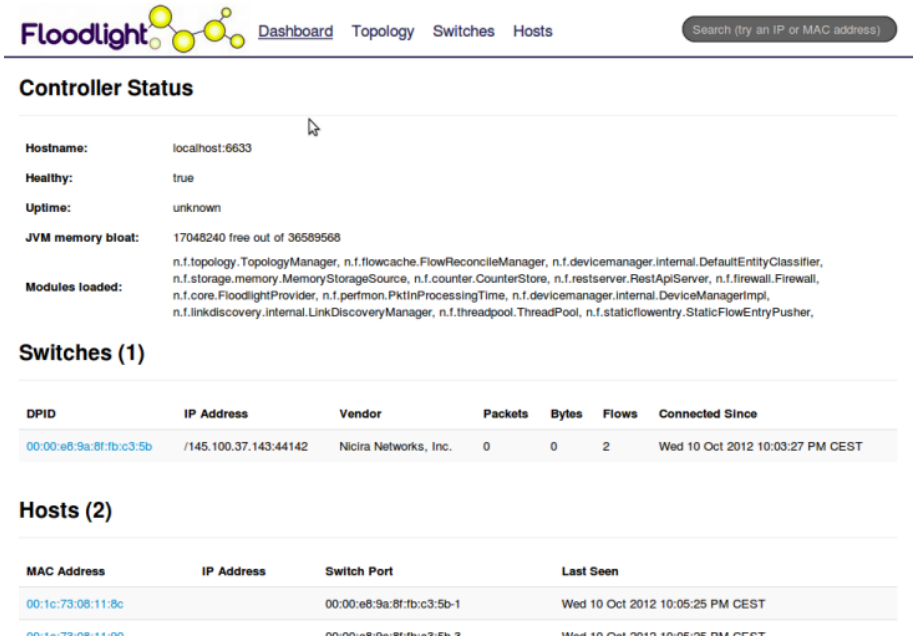

Figure 3: A Screenshot of the Floodlight web interface

Each OpenFlow device is identified by its Datapath Identifier (DPID). Each DPID is 8 bytes long and can be specified as a decimal number or as 8 hex octets, e.g., `00:00:00:23:10:35:ce:a5`. The DPID `ff:ff:ff:ff:ff:ff:ff:ff` is a "wildcard" DPID that matches all DPIDs inside an OpenFlow network.

3 Virtualisation with FlowVisor

A network consisting of switches capable to run OpenFlow can be separated into different independent networks using FlowVisor [8]. FlowVisor will be placed between the OpenFlow switch and the OpenFlow controller and act as a transparent proxy between them. The switch as well the OpenFlow controller are not aware that FlowVisor is in between them.

With FlowVisor separate networks can be created by defining different network slices. Each slice is connected to its own OpenFlow controller. In fact a slice is a reference to such an OpenFlow controller.

By defining FlowSpaces traffic can be classified and attached to a slice. A FlowSpace is created based on DPID, a match on incoming traffic like a regular OpenFlow match, a priority as well a slice action.

Fields can belong to multiple FlowSpaces, i.e. a FlowSpace based on `in_port 1` can belong to slice 1, while the FlowSpace with `in_port 1` and source IP-address `192.168.1.1` belongs to slice 2.

Flowspaces are given a priority in the range from 0-2^{31} and the highest priority will match. So referring to the example above, the FlowSpace with `in_port 1` and `192.168.1.1` should get a higher priority than the other.

The sliceaction determines to which slice the FlowSpace belongs and what rights the slice has. Possible values are DELEGATE, READ and WRITE, a short description can be found in Table 2.

Table 2: Possible Slice actions

Action	value	description
DELEGATE	1	delegate control to another slice.
READ	2	read the packet_in matches for the flow entry.
WRITE	4	Same as read and ability to write to flow table.

Each slice has its own view on its part of the network topology. Doing experiments inside one slice should not affect the proper working of other network slices. So in theory experiments can be done safely inside the production network.

Depending on the mode in which the OpenFlow controller is running the traffic is handled different by FlowVisor.

In reactive mode the incoming traffic from the switch to the OpenFlow controller will be intercepted by FlowVisor and depending on the configured FlowSpaces, the traffic is sent to the designated OpenFlow controller. The OpenFlow controller then makes the forwarding decision and sends a forwarding entry to the switch. On the way to the switch FlowVisor will check if the forwarding entry of the OpenFlow controller is allowed and send it to the switch.

In proactive mode FlowVisor checks if the flow entries that the OpenFlow controller want to push to the OpenFlow switch match the configured FlowSpace. In this case, the flow entry will be forwarded and stored inside the OpenFlow switch and if not the entry is discarded as can be seen in Figure 4.

Configuring FlowVisor is done using the FlowVisor configuration tool `fvctl`. This tool sends the commands to an Extensible Markup Language - Remote Procedure Call (XML-RPC) running on port 8080. There are plans to replace the XML-RPC with a JSON interface.

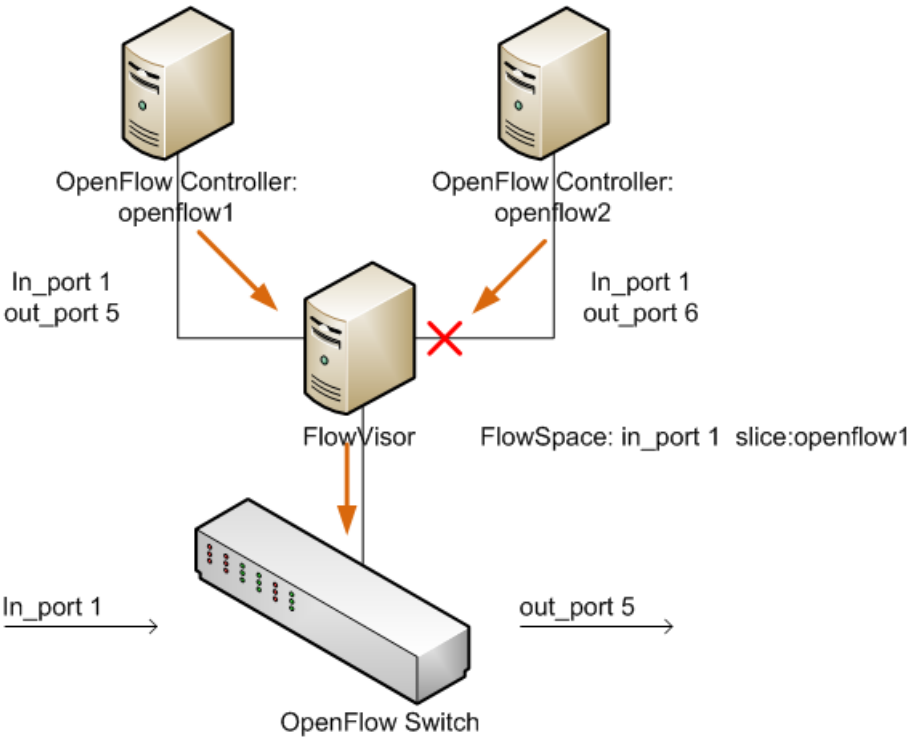

Figure 4: The working of Flowvisor in proactive mode

During this research project version 0.8.5 of FlowVisor was the latest stable version of FlowVisor and was used to conduct the experiments. However, after conducting the experiments and while writing the report version 0.8.6 of FlowVisor was released [26].

4 Experiments

The experiments during this research are conducted with a basic OpenFlow topology. It consist of one Pronto 3290 switch [15], some Floodlight OpenFlow Controllers, one FlowVisor controller and several virtual machines running Ubuntu 12.04 LTS [27].

By using different setups, several situations were created to test the working and behaviour of FlowVisor.

The equipment was placed in the Lighthouse [28], a research lab located at SARA.

The logical topology of the experiments is shown in Figure: 5. Note that this is a sample topology, in some experiments more OpenFlow controllers were used, and in other experiments a laptop was attached to the switch instead of a server.

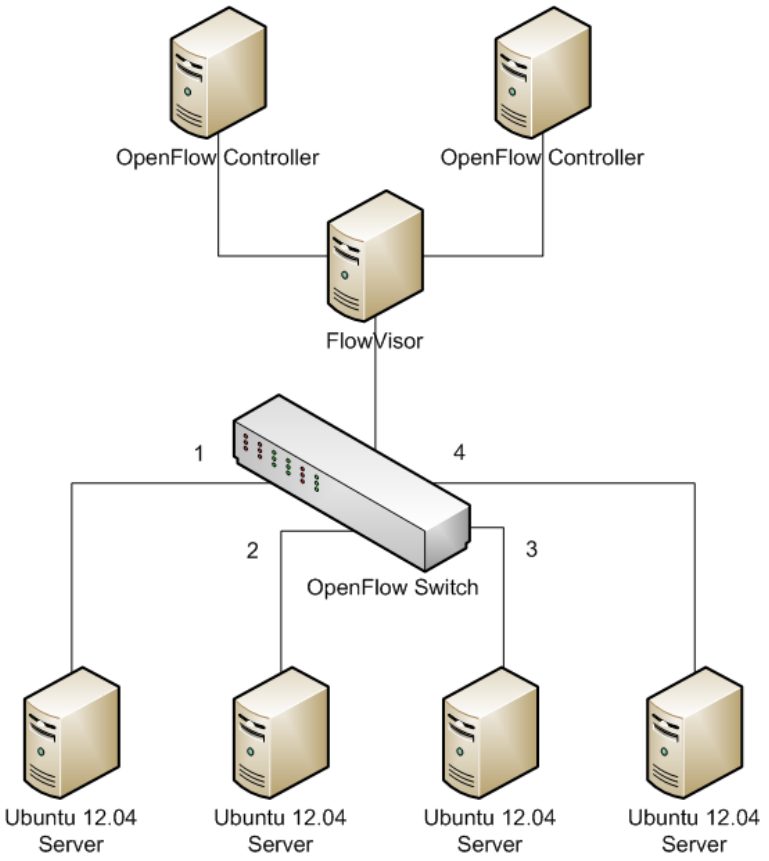

Figure 5: Logical setup of the used OpenFlow topology

Each slice is connected to one Floodlight OpenFlow controller. Because most of the controllers are used in every experiment they stay the same.

At the begin of the experiments all slices were created. Which are `slice1`, `slice2`, `slice3` and `slice4`. The default configuration of FlowVisor and the

Floodlight OpenFlow controllers can be found in appendix B.1. The configurations of each individual experiment can be also found in the appendix.

4.1 Experiment 1: Switch-port based slice membership

This experiment is conducted to see and verify that a slice can only control attributes of its own FlowSpace. This shows in which way the switch resources are separated, when talking about separation of the switch hardware into several virtual OpenFlow networks. This is done by FlowSpaces based on switch port.

4.1.1 Slices

Slices `slice1` and `slice2` are used, each slice consists of two ports with attached a Virtual Machine (VM) to each port. The slice configuration can be found in Table 3.

Table 3: Network Slices of experiment 1

Slice	Ports
slice1	1 and 2
slice2	3 and 4

Since FlowVisor is a transparent proxy between the OpenFlow switch and the OpenFlow controller it should not be possible for an OpenFlow controller to apply rules for ports which do not belong to the slice. The OpenFlow controller of `slice2` should not be able to insert a flow entry for a port in `slice1`. However the OpenFlow controller of `slice2` should be able to create a flow entry for its own port. The setup of this experiment can be found in Figure: 6.

4.1.2 Flow entries

During this experiment the OpenFlow controller floodlight 1 was configured to create a flow for its part of the network (Rule #1). OpenFlow controller floodlight 2 was configured with a rule that matches the configuration of his slice (Rule #2) as wel a rule that does not match the FlowVisor configuration (Rule #3) as described in Table 4.

Table 4: Created Flows for experiment 1

Rule #	Controller	Flow
1	Floodlight1	in_port 1 out_port 2
2	Floodlight2	in_port 3 out_port 4
3	Floodlight2	in_port 2 out_port 4

4.1.3 Result

Configuring the flow entries on the Floodlight controller yielded the following flow table inside the OpenFlow switch:

```
root@XorPlus#ovs-ofctl dump-flows br0
in_port=1 actions=output:2
in_port=3 actions=output:4
```

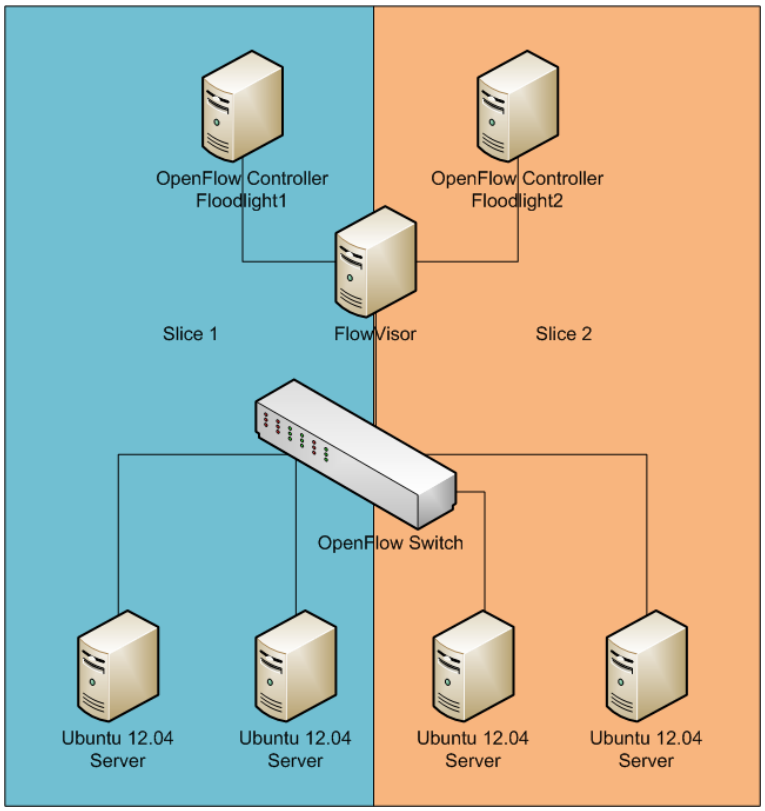

Figure 6: Setup of experiment 1: Switch-port based slice membership

As seen rule #1 and rule #2 are inserted correctly to the flow table according to the configured FlowSpaces, only rule #3 is missing.

But actually no error message was seen while configuring the flow entries. The floodlight2 controller showed for both rules a entry pushed message, which means that the entry is stored in the database of the OpenFlow controller. After looking into the log-file of floodlight one can see the FlowVisor has blocked rule #3.

```
10:15:53.208 [New I/O server worker #1-1] ERROR n.f.core.internal.
Controller - Error OFPET_FLOW_MOD_FAILED OFPFMFC_EPERM from
[OFSwitchImpl /145.100.37.143:55771 DPID[00:00:e8:9a:8f:fb:c3:5b]]
```

According to the OpenFlow 1.0 specification the error says that there is a problem modifying flow entry (`OFPET_FLOW_MOD_FAILED`), and that the error is cause due a permissions error (`OFPFMFC_EPERM`)

The message is originating from IP-address `145.100.37.143` which belongs to FlowVisor and not to the OpenFlow switch.

It seems that Floodlight checks and stores the entry in the database and sent it at the same time to the switch. However it does not check if the pushed entry is allowed by the switch. So the flow entry stays in the floodlight database but does not get in to the OpenFlow controller. This can give some problems

when checking entries.

4.2 Experiment 2: VLAN based slice membership

This experiment is conducted to examine how FlowVisor handles shared ports. These are ports that are present in two different network slices. To achieve this VLANs are used. There will be one trunk interface that carries traffic for two VLANs and two access ports that have untagged network traffic. This experiment shares a switch resource (port) but is separated based on the VLAN that is used.

4.2.1 Slices

`Slice1` and `slice2` were used as showed in Figure: 7. `Slice1` will handle traffic for VLAN 60 and `slice2` will handle traffic for VLAN 50.

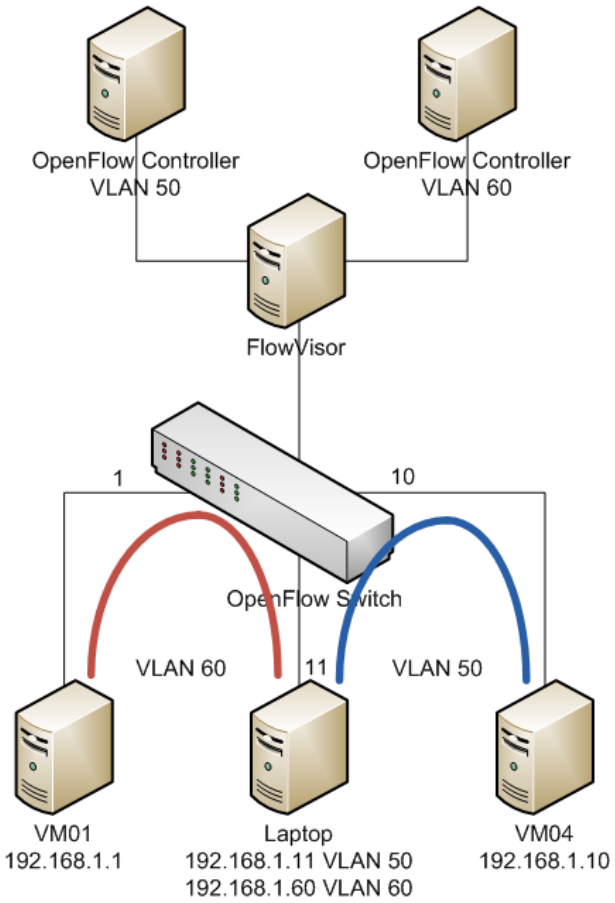

Figure 7: Setup of experiment 2: VLAN based slices

Each slice will get two FlowSpaces, one for the access port and one for the trunk port according to Table 5. Port 11 will be shared by `slice1` and `slice2`.

Traffic from switch port 1 should go to switch port 11 and will be tagged with VLAN 60 and traffic from switch port 10 should go to switch port 11 and will be tagged with VLAN 50. Traffic back from the trunk port will be handled based on the VLAN-ID of the packet.

Table 5: Network Slices for experiment 2

Slice	FlowSpace
slice1	port 1
slice1	port 11 & VLAN 60
slice2	port 10
slice2	port 11 & VLAN 50

The OpenFlow switch will take care of stripping the VLAN from the packet. Using the OpenFlow strip-VLAN option will cause a connection reset between the OpenFlow controller and FlowVisor, which seems to be a (old) bug in FlowVisor [29].

4.2.2 Flow Entries

This setup translates to the following flow entries for the OpenFlow controllers as shown in Table 6.

Table 6: Created Flows for experiment 2

Controller	Match	Action
floodlight2	in_port 11,vlan-id:50	out_port 10
floodlight1	in_port 11,vlan-id:60	out_port 1
floodlight2	in_port 10	set-vlan-id=50 out_port 11
floodlight1	in_port 1	set-vlan-id=60 out_port 11

4.2.3 Result

After configuring the rules on the OpenFlow controllers, the traffic inside the VLANs was possible. The screenshot in Figure: 8 shows the traffic flow of VLAN 60. The traffic was captured on the trunk interface therefore the VLAN is showing up in the packet dump. The IP-address `192.168.1.1` is the IP of VM01 and IP-address `192.168.1.60` is the IP of the laptop that was connected to the trunk interface.

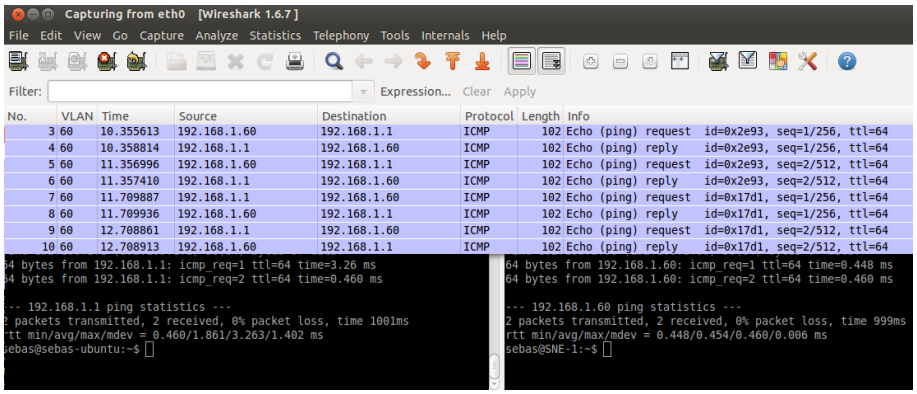

Figure 8: Traffic of VLAN 60 between VM01 and the laptop

The flow table on the switch looked correct corresponding to the configured flows on the OpenFlow controllers.

```
root@XorPlus#ovs-ofctl dump-flows br0
in_port=1 actions=mod_vlan_vid:60,output:11
in_port=10 actions=mod_vlan_vid:50,output:11
in_port=11,dl_vlan=60 actions=output:1
in_port=11,dl_vlan=50 actions=output:10
```

4.3 Experiment 3: Application-port based slice membership

This experiment is conducted to have flows based on application ports. Goal of this experiment is to separate parts of the network to increase security. Certain application ports will be forwarded to a specific switch port.

4.3.1 Slices

This experiment is setup using three slices, `slice1`, `slice2` and `slice3`. Each slice handled an other kind of traffic.

Figure 9 shows the setup of the experiment. `Slice1` will handle web traffic on port 80 (green), `slice2` is for web traffic on port 8080 (blue) and finally `slice3` handles MySQL traffic on port 3306 (red) as shown in Figure 9.

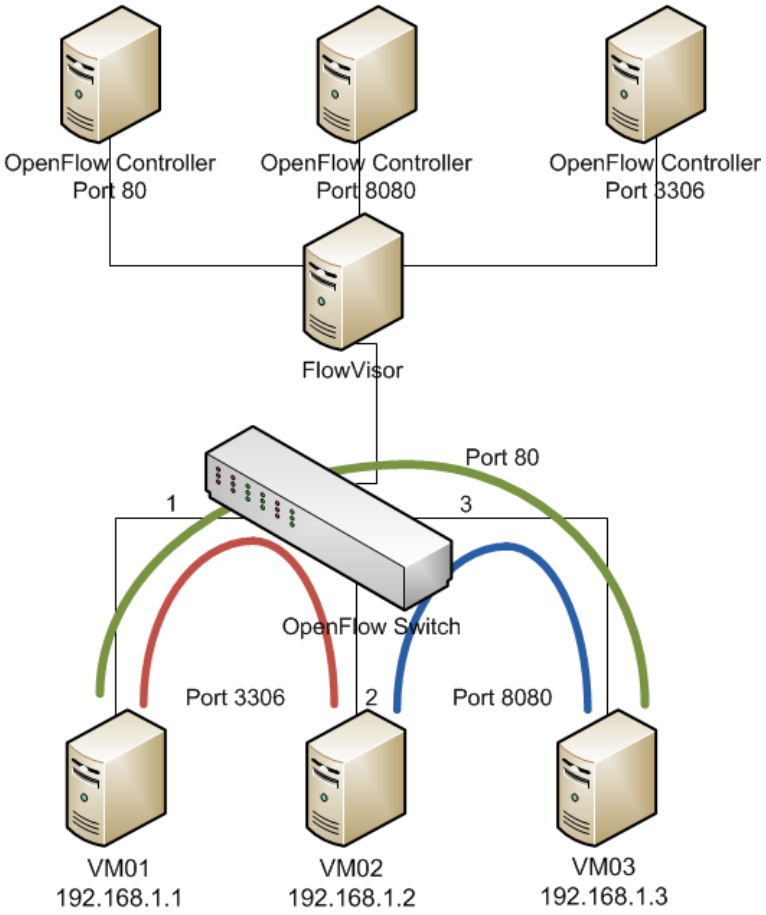

Figure 9: Setup of experiment 3: Application-port based membership

Eight FlowSpaces based on `in_port` and application source or destination port were created as described in appendix B.4

On VM01 a PHP-enabled Apache2 [30] web server was configured. This webserver serves a file which gets a field of data from a MySQL [31] database

back-end that is running on VM02.

When a page is accessed from VM03, the web server will connect to the database server to get a value from a field stored in the database. The result of this then is served to the end user on VM03.

On VM02 also an Apache2 web server is running on port 8080. The web server serves a static file and allows traffic from VM03.

4.3.2 Flow Entries

The described set up can be translated into the Flow entries as shown in Table 7.

Table 7: Created Flows for experiment 3

Controller	Flow	Action
Floodlight1	ether-type=0x0800,protocol=6, src-port=80, in_port 1	out_port 3
Floodlight1	ether-type=0x0800,protocol=6, dst-port=80, in_port 3	out_port 1
Floodlight2	ether-type=0x0800,protocol=6, src-port=8080	out_port 3
Floodlight2	ether-type=0x0800,protocol=6, dst-port=8080, in_port 3	out_port 2
Floodlight3	ether-type=0x0800,protocol=6, src-port=3306	out_port 1
Floodlight3	ether-type=0x0800,protocol=6, dst-port=3306	out_port 1

4.3.3 Result

Figure 10 shows that web traffic between VM03 and VM01 was possible. The file `index.php` includes the value from the field of the MySQL database, which means that the MySQL connection between VM01 and VM02 functions correctly.

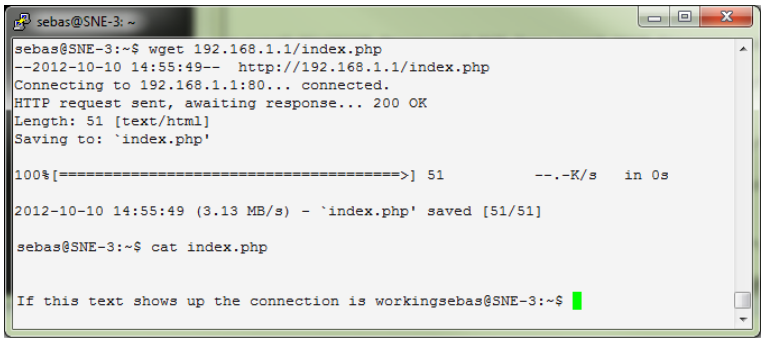

Figure 10: Get page from server and show contents

Traffic on port 8080 between VM03 and VM02 was also possible, the file could be downloaded. But VM03 could not access the server on port 3306 to access the MySQL database, as one can see in the screenshot in Figure: 11.

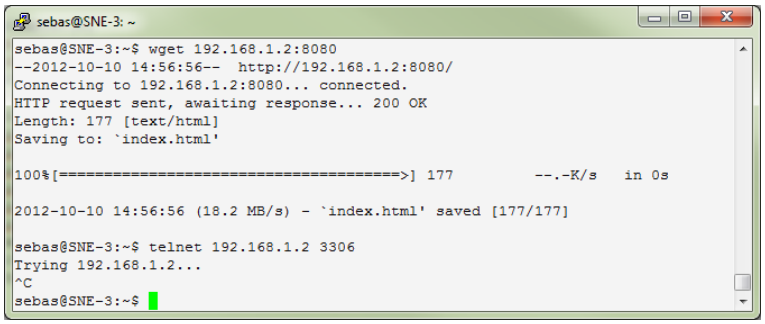

Figure 11: Connection to port 8080 is possible but not to port 3306

4.4 Experiment 4: MAC-address based slice membership

This experiment is conducted to have flows based on destination MAC-address. Traffic to a destination MAC-address will be sent to the port where the host is attached. This could simulate the working of a traditional switch.

4.4.1 Slices

For this experiment only `slice1` is used. Two FlowSpaces based on destination MAC-address for the hosts were created and one FlowSpace for ARP traffic. The configured FlowSpaces can be found in Table 8.

Table 8: Network Slices for experiment 4

Slice	FlowSpace
Slice1	dst-mac: 52:54:00:b2:0d:d6
Slice1	dst-mac: 52:54:00:23:8b:87
Slice1	dl_type: 0x0806

4.4.2 Flow Entries

After configuring the FlowSpaces the flow entries on the OpenFlow controller could be inserted. Every host is attached to its own port, therefore a direct mapping can be done of destination MAC-address and switch port.

To be able to translate the IP-address of a host to its MAC-address a flow entry that handles ARP-traffic is also inserted. The configured flow entries can be found in Table 9.

Table 9: Created Flows for experiment 4

Controller	Flow	Action
Floodlight1	dst-mac: 52:54:00:b2:0d:d6	out_port 1
Floodlight1	dst-mac: 52:54:00:23:8b:87	out_port 2
Floodlight1	ether-type: 0x0806	out_port all

4.4.3 Result

After pushing the flow entries to the OpenFlow switch, a ping from VM01 on port 1 to the VM02 on port 2 was possible. Inside the FlowTable of the switch one could find the following flow entries:

```
root@XorPlus#ovs-ofctl dump-flows br0
dl_dst=52:54:00:23:8b:87 actions=output:2
dl_dst=52:54:00:b2:0d:d6 actions=output:1
arp actions=ALL
arp,dl_dst=52:54:00:23:8b:87 actions=output:2
arp,dl_dst=52:54:00:b2:0d:d6 actions=output:1
```

As one can see, there are five flow entries inside the switch table present. However, there were only three flow entries configured. FlowVisor did rewrite both destination MAC-address flow entries to match the FlowSpace.

The returning ARP traffic, should not be flooded to all ports of the switch but should only be sent to the designated host.

4.5 Experiment 5: Switch events

This experiment is conducted to find out in which way switch events are handled by FlowVisor and the attached OpenFlow controllers. Therefore a simple network setup is created based on switch ports. To trigger a switch event a cable will be pulled from the switch.

4.5.1 Slices

Port 1 and port 2 of the switch belong to `slice1` and port 3 and port 4 belong to `slice2`. The topology can be found in Figure 12.

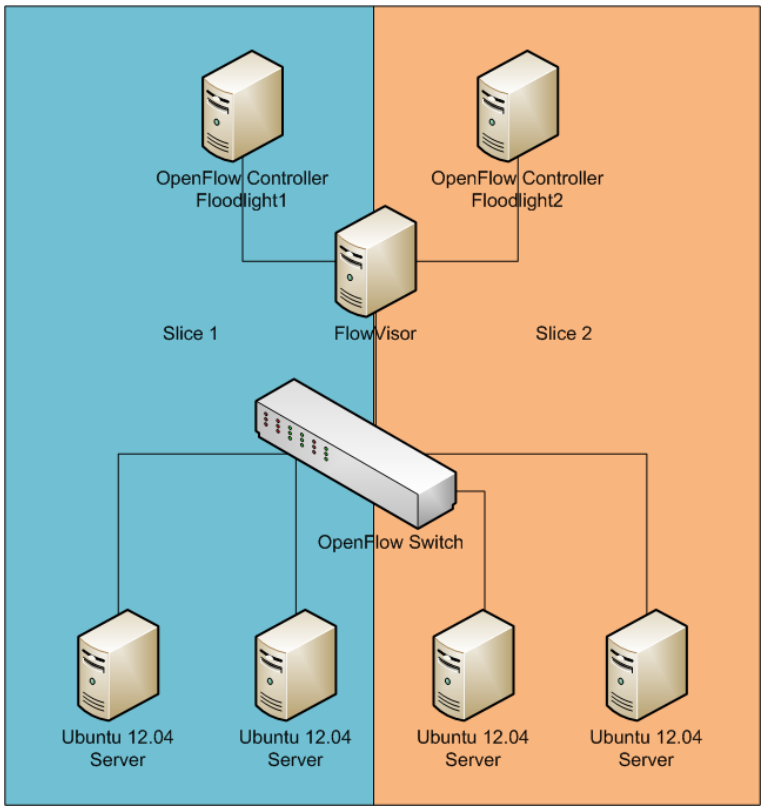

Figure 12: Setup of experiment 5: Switch events

4.5.2 Flow Entries

Traffic between the ports which are assigned to the same slices is allowed. In Table 10 one can find the created flow entries that are pushed to the OpenFlow switch from the OpenFlow controllers. Traffic between port 1 and 2 and traffic between port 3 and 4 is allowed.

Table 10: Created Flows for experiment 5

Controller	Flow
Floodlight 1	in_port1 out_port 2
Floodlight 1	in_port2 out_port 1
Floodlight 2	in_port3 out_port 4
Floodlight 2	in_port4 out_port 3

4.5.3 Result

After configuring the flow entries and finding them in the switch the cable of port 2 is pulled out of the OpenFlow switch, which causes the port to go down. After some minutes the cable is put back into the switch and the port became up again.

Then the log-files of both Flowvisor as well the OpenFlow controllers are examined. Inside FlowVisor two messages regarding a changed port status were logged:

```
1 Oct - 13:33:38 INFO org.flowvisor.log.AnyLogger.log(AnyLogger.java:38)
modifying port 2
1 Oct - 13:36:33 INFO org.flowvisor.log.AnyLogger.log(AnyLogger.java:38)
modifying port 2
```

The first message is regarding the port that goes down, and the second message belongs to the upcoming port.

In the log file of Floodlight 1 also messages belonging to a port status are found, but not in the log file of Floodlight 2, since this controller belongs to the slice where the port is not located.

Below the messages from the Floodlight log-file:

Remove port:

```
13:33:38.057 [New I/O server worker #1-1] DEBUG n.f.core.internal.Controller
- Port #2 modified for OFSwitchImpl [/145.100.37.143:42990
DPID[00:00:e8:9a:8f:fb:c3:5b]]
13:33:38.058 [pool-3-thread-13] DEBUG n.f.d.internal.DeviceManagerImpl
- Triggering update to attachment points due to topology change.
13:33:38.059 [pool-3-thread-13] DEBUG n.f.devicemanager.internal.Device
- DEVICE_MOVE: Old AttachmentPoints:  [],New AttachmentPoints:  []
13:33:38.059 [pool-3-thread-13] DEBUG n.f.d.internal.DeviceManagerImpl
- Attachment point changed for device:  Device [deviceKey=1,
entityClass=DefaultEntityClass, MAC=00:1c:73:08:11:8d, IPs=[], APs=[]]
```

Add port:

```
13:36:33.056 [New I/O server worker #1-1] DEBUG n.f.core.internal.Controller
- Port #2 modified for OFSwitchImpl [/145.100.37.143:42990
DPID[00:00:e8:9a:8f:fb:c3:5b]]
13:36:33.056 [pool-3-thread-5] DEBUG n.f.d.internal.DeviceManagerImpl
- Triggering update to attachment points due to topology change.
```

```
13:36:33.057 [pool-3-thread-5] DEBUG n.f.devicemanager.internal.Device
- DEVICE_MOVE: Old AttachmentPoints:  [],New AttachmentPoints:  []
13:36:33.057 [pool-3-thread-5] DEBUG n.f.d.internal.DeviceManagerImpl
- Attachment point changed for device:  Device [deviceKey=1,
entityClass=DefaultEntityClass, MAC=00:1c:73:08:11:8d, IPs=[], APs=[]]
```

These messages belong to each the port down and the port up status. In Figure 13 one can see the port down OpenFlow message that is sent from FlowVisor to the OpenFlow controller.

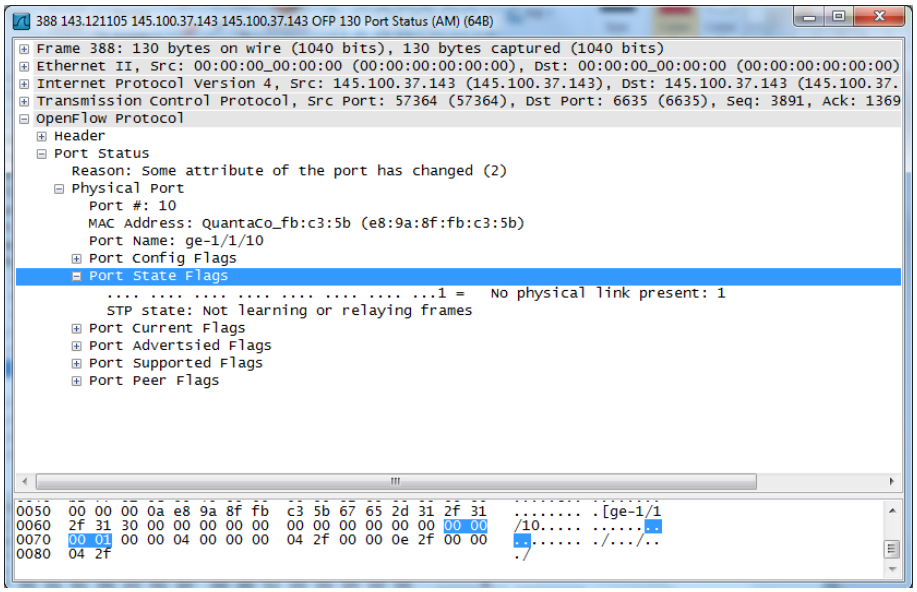

Figure 13: OpenFlow protocol port change packet on the wire

This experiment showed that switch events are recognised by FlowVisor and sent to the responsible OpenFlow controller.

However, debugging a big OpenFlow network can be hard, assuming multiple switches and controllers are present. Finding an error inside the network can then be a time intensive task, because all log files have to be inspected.

Flowvisor gives the possibility to grant READ permissions to slices. Which means to each FlowSpace a monitor slice can be added. So switch events can be found inside one single log file, which makes finding errors easier.

In experiment 6 a setup is tested which includes a monitor slice.

4.6 Experiment 6: MAC-address and application port based slice membership

This experiment is conducted to have traffic streams either based on a specific MAC-address that is sent to `port X` and traffic based on MAC-address and destination port 80 was sent to `port Y`. Furthermore, a slice for monitoring purposes was added. This experiment relates to the separation of switch resources and switch events.

4.6.1 Slices

For this experiment `slice1`, `slice2`, `slice3` and `slice4` were used. The FlowSpaces were created based on source MAC-address. An overview of the created FlowSpaces can be found in Table 11. Controller Floodlight 4 got read permissions for all the three slices.

Table 11: Network Slices for experiment 6

Slice	FlowSpace
Floodlight1	src-mac: 52:54:00:b2:0d:d6
Floodlight2	src-mac: 52:54:00:23:8b:87
Floodlight3	src-mac: 52:54:00:d4:4d:83

4.6.2 Flow Entries

Traffic will flow between VM01 and VM02, except for traffic on application port 80. This traffic will be sent from VM01 to VM03 and from VM02 to VM03 and vice versa. The traffic between VM01 and VM02 will be based on MAC-address and the web traffic matches on the source and destination application port 80. The created flow entries can be found in Table 12.

Table 12: Created Flows for experiment 6

Controller	Match	Action
Floodlight1	src-mac:52:54:00:b2:0d:d6	out_port 2
Floodlight1	src-mac:52:54:00:b2:0d:d6,ether-type:0x0800,protocol:6,dst-port:80	out_port 3
Floodlight2	src-mac:52:54:00:23:8b:87	out_port 1
Floodlight2	src-mac:52:54:00:23:8b:87,ether-type:0x0800,protocol:6,dst-port:80	out_port 3
Floodlight3	src-mac:52:54:00:d4:4d:83,dst-mac:52:54:00:b2:0d:d6,ether-type:0x0800,protocol:6,src-port:80	out_port 1
Floodlight3	src-mac:52:54:00:d4:4d:83,dst-mac:52:54:00:23:8b:87,ether-type:0x0800,protocol:6,src-port:80	out_port 2

Note that controller floodlight 4, which is attached to every FlowSpace as monitoring slice does not have any flow entries configured.

4.6.3 Result

After configuring these flow entries host VM01 could ping host VM02 and vice versa. Also the web traffic on port 80 between host VM01 and host VM03 as well between host VM02 and host VM03 functioned as expected.

During the experiment the cable of port 2 was unplugged to see where the port modify massage are sent to when no FlowSpace based on port is configured. A log entry about the port was of course present in the log file of Flowvisor. Furthermore an entry was found in each Floodlight controller. This is caused due the fact that there is no FlowSpace attached to port 2.

The monitoring slice can read messages from all slices. The message of the modified port showed up only once. One could maybe expect that the message showed up tree times.

4.7 Experiment 7: Slices based on layer 2, 3 and 4 fields

This experiment is conducted to see how FlowVisor handles FlowSpaces based on different fields on layer 2, layer 3 and layer 4 when installing Flows that have this fields wildcarded. For example a FlowSpace based on destination MAC-address and the Flow is based destination IP-address.

There will be several tests for this experiment, in each section a short description will be given on which slices are used as well which flow entries are entered.

4.7.1 MAC Based

Slice `slice1` is used. FlowSpaces based on destination MAC-address: `aa:aa:aa:aa:aa:aa` and `bb:bb:bb:bb:bb:bb` were created.

Two flow entries based on source MAC-address are inserted to the OpenFlow controller as shown in Table 13

Table 13: Created Flows for experiment 7

Controller	Match	Action
Floodlight1	src-mac:aa:aa:aa:aa:aa:aa	out_port 2
Floodlight1	src-mac:bb:bb:bb:bb:bb:bb	out_port 1

After configuring the flow entries on the controller the following flow entries showed up at the switch:

```
root@XorPlus#ovs-ofctl dump-flows br0
dl_src=bb:bb:bb:bb:bb:bb,dl_dst=aa:aa:aa:aa:aa:aa actions=output:1
dl_src=bb:bb:bb:bb:bb:bb,dl_dst=bb:bb:bb:bb:bb:bb actions=output:1
dl_src=aa:aa:aa:aa:aa:aa,dl_dst=bb:bb:bb:bb:bb:bb actions=output:2
dl_src=aa:aa:aa:aa:aa:aa,dl_dst=aa:aa:aa:aa:aa:aa actions=output:2
```

As one can see four entries are present in the flow table. FlowVisor rewrites the flow entries to match them to the configured FlowSpace. Since the FlowSpace is based on destination MAC-address a flow entry with only a source MAC-address specified must match also the destination MAC-address, therefore both rules are added.

When running Flowvisor in DEBUG mode, one can see the modifications: (some output is omitted)

```
recv from controller:  OFFlowMod (...)  match=OFMatch[dl_src=aa:aa:aa:aa:aa:aa]
(...)  actions=OFActionOutput [type=OUTPUT];port=2

send to switch:OFFlowMod (...)  match=OFMatch[dl_dst=aa:aa:aa:aa:aa:aa,
dl_src=aa:aa:aa:aa:aa:aa] (...)  actions=OFActionOutput [type=OUTPUT];port=2

send to switch:OFFlowMod (...)  match=OFMatch[dl_dst=bb:bb:bb:bb:bb:bb,
dl_src=aa:aa:aa:aa:aa:aa] (...)  actions=OFActionOutput [type=OUTPUT];port=2

expanded fm 2 times
```

FlowVisor expanded the FlowMod 2 times.

Next, I added a flow entry based on destination IP-address `192.168.1.1` and would assume that FlowVisor acted the same and rewrites the rule.

The added rule looked like this in the flow table of the switch:

```
ip,nw_dst=192.168.1.1 actions=output:3
```

The flow entry was not modified to match the FlowSpace, but when looking into the log of FlowVisor the following is seen:

```
recv from controller:  OFFlowMod (...)  match=OFMatch[dl_type=0x800,
nw_dst=192.168.1.1] (...)  actions=OFActionOutput [type=OUTPUT];port=3

send to switch:OFFlowMod (...)  match=OFMatch[dl_type=0x800,nw_dst=192.168.1.1]
(...)  actions=OFActionOutput [type=OUTPUT];port=3

send to switch:OFFlowMod (...)  match=OFMatch[dl_type=0x800,nw_dst=192.168.1.1]
(...)  actions=OFActionOutput [type=OUTPUT];port=3

expanded fm 2 times
```

It seems that FlowVisor want to change something because it says it expanded the FlowMod 2 times, but actually the flow entry is the same as configured into the controller. This may be eventually a bug in the software.

Actually this can create a security problem since now all traffic to IP-address `192.168.1.1` is allowed, and not as the FlowSpace says only traffic with destination MAC-address: `aa:aa:aa:aa:aa:aa` and `bb:bb:bb:bb:bb:bb`.

4.7.2 IP Based

`Slice1` is used. FlowSpaces based on destination IP-address `172.16.17.18`, priority 100 and `192.168.1.1`, priority 200 are configured.

A flow is inserted with a match on destination IP-address `192.168.1.1` and output port 2.

The flow table of the switch shows the entry:

```
ip,nw_dst=192.168.1.1 actions=output:2
```

This is an entry which one can expect, because it matches the FlowSpace.

Now a flow entry was added that matches the source IP-address 10.10.10.10. Since this IP-address is not part of any flow space, one would expect that the destination IP-addresses `172.16.17.18` and `192.168.1.1` were added to the flow entry. Looking in to the flow table of the switch shows us that this is not the case.

```
ip,nw_src=10.10.10.10,nw_dst=172.16.17.18 actions=output:3
```

Only destination IP-address `172.16.17.18` was added to the flow even though

it has a lower priority. Thus it seems that priority is ignored.

4.7.3 Application port Based

For this test `slice1` is used. FlowSpaces based on destination `port 80` and `443` are used.

When adding a flow based on destination port 80 and with action output port 1 the result is as expected. Since destination port 80 matches the configured FlowSpace:

```
root@XorPlus#ovs-ofctl dump-flows br0
tcp,tp_dst=80 actions=output:1
```

After adding a flow based on destination IP-address `192.168.1.1` and action output port 2. One would expect that both application ports 80 and 443 are added by FlowVisor to match the FlowSpace. But this is unfortunately not the case, as can seen in the flow entry on the switch:

```
root@XorPlus#ovs-ofctl dump-flows br0
tcp,nw_dst=192.168.1.1 actions=output:2
```

This behaviour is the same as with the first MAC based test, FlowVisor shows also that the flow entry is not edited.

Trying again to add a flow entry but now with source MAC-address `bb:bb:bb:bb:bb:bb` and output port 1. The output on the switch looks the same, only the MAC-address is present in the flow table.

```
root@XorPlus#ovs-ofctl dump-flows br0
dl_src=bb:bb:bb:bb:bb:bb actions=output:1
```

But when looking into the FlowVisor output, one can see that FlowVisor changes the flow entry to match both destination ports 80 and 443:

```
recv from controller:  OFFlowMod (...)  match=OFMatch[dl_src=bb:bb:bb:bb:bb:bb]
(...)  actions=OFActionOutput [type=OUTPUT];port=1

send to switch:OFFlowMod (...)  match=OFMatch[dl_src=bb:bb:bb:bb:bb:bb,tp_dst=443]
(...)  actions=OFActionOutput [type=OUTPUT];port=1

send to switch:OFFlowMod (...)  match=OFMatch[dl_src=bb:bb:bb:bb:bb:bb,tp_dst=80]
(...)  actions=OFActionOutput [type=OUTPUT];port=1

expanded fm 2 times:  OFFlowMod
```

However, this change is for an unknown reason not seen and received by the switch and therefore not applied to the flow table.

4.7.4 Results

Based on the observed output during the different tests it seems that not everything is working properly as one would assume after reading the documentation.

When using wildcards it seems that not every time a match for the FlowSpace is created, which opens the whole network for unwanted traffic.

5 Summary of results

In this chapter the results of the conducted experiments from the previous chapter will discussed in relation to the research questions.

Experiment 1: Switch-port based slice membership
Experiment 1 showed that an OpenFlow controller only can insert flows for the slice where the controller belongs to when trying to insert flows for other slices FlowVisor gives a permission error message. FlowVisor takes care of the separation of the switch resources when talking about hardware. This experiment answered the question: In which way are the switch resources separated?

Experiment 2: VLAN based slice membership
This experiment showed the working of a topology using VLANs with a shared trunk port. Traffic was assigned to different network slices based on the VLAN. For a short time a bug caused a disconnection between FlowVisor and the OpenFlow controller. This experiment did answer two questions: "In which way are the switch resources separated?" and "Which kind of bugs are known and still present?"

Experiment 3: Application-port based slice membership
One saw in this experiment that creating slices based on application ports are a good way to separate traffic streams between several servers. When using VLANs a similar setup was possible, but a router would be needed to allow traffic between the VLANs. With OpenFlow and FlowVisor this can be done with a switch, which reduces hardware costs. This experiment answered the question: In which way are the switch resources separated?

Experiment 4: MAC-address based slice membership
This experiment showed that a switch like configuration could be setup using OpenFlow. By defining flow entries based on a match on destination MAC-address and as action the port where the host is attached, the network acts in a way as a normal switch does. The flow entry for the ARP traffic was necessary to let the hosts find the MAC-address of the host which it want to reach. This experiment answered the question: In which way are the switch resources separated?

Experiment 5: Switch events
This experiment showed how switch events are handled. Basically the switch shows only port up and port down messages that could be seen in the logs of FlowVisor and the OpenFlow controller where the port belongs to. However this messages inside the logs originate from the OpenFlow packets that are sent on the wire. This experiment answered the question: How are switch events handled?

Experiment 6: MAC-address and application port based slice membership
This experiment was combination of experiment 3, 4 and 5. Several traffic streams based on different fields of a packet worked as expected. The traffic streams were separated. In experiment 6 the switch messages were sent to all

OpenFlow controllers since there was not a single slice attached to one port. This experiment did answer two questions: "In which way are the switch resources separated?" and "How are switch events handled?"

Experiment 7: Slices based on layer 2, 3 and 4 fields
The last experiment showed that when having FlowSpaces and flow entries based on different values of a packet the rewriting of this flow entries doesn't always work.

Using only MAC-address based flow entries as well FlowSpaces worked well, but as soon as an other layer is involved something went wrong. One would expect the same behaviour using different layers, for example MAC-address and IP-address. This experiment answered the question: In which way are the switch resources separated?

6 Conclusion

In this chapter a conclusion is given based on the results of the experiments that are conducted during this research project.

The main research question of the project was:

> *"Is the current FlowVisor implementation version 0.8.5 suitable to create stable virtual networks in production environments?"*

To answer this research question several sub-research-questions were created, which were answered by conducting experiments.

- **How stable is an FlowVisor environment?** Overall there were no issues regarding the stability of FlowVisor, once running FlowVisor did its job. However, bugs could cause a connection disconnect between FlowVisor and the OpenFlow controller.

  Although I used a VM during the research to run FlowVisor and several OpenFlow controllers on it, that did not give a bad performance at all. Sometimes it took some seconds to insert a command using the `fvctl`-tool, but this was a reasonable amount of time.

- **Is FlowVisor user friendly?** This is surely not an easy question to answer, it will heavily depend on the user who is using FlowVisor. However, based on the user-experience I would call FlowVisor not that user friendly. Having to enter the password every time when issuing a command using the `fvctl`-tool is a bit annoying. Although a password file can be specified when using the `fvctl`-tool this introduces some kind of security issues, storing such a powerfull password in clear text on the server. Even when the password is stored encrypted in a file, an attacker could use this to bring down the network or create slices for his own purposes.

  An interface called FlowVisor CLI is in development which offers a console to the user. In this console the FlowVisor commands can be entered, without password. This will improve the user experience and makes FlowVisor more user friendly.

- **In which way are the switch resources separated?** The resources of the switch are separated according to the created slices. A slice should not be able to enter flow entries for a slice it does not belong to.

  Separating switch resources can be done based on application port or based on VLAN membership. FlowVisor ensures that only the responsible OpenFlow controller can add flow entries.

  However, when using wildcarded flow entries like in experiment 7 it seems that the separation of the switch resources not always works as one would expect. FlowVisor does not in every case rewrite the flow entry to match the FlowSpace.

  Other resource separating features like slice resource limits in the meaning of a limit in how many flow entries a slice can use, will be implemented in future releases of FlowVisor, see section 7.

- **How are switch events handled?** Switch events are handled by FlowVisor in a way that they are sent to the appropriate OpenFlow controller that is responsible for the slice where the switch event happens. If there is no OpenFlow controller, which is only connected to a switch port the message regarding the switch event is sent to all possible Openflow controllers. To have only one place to look for switch events one can implement monitor slices, which get all switch events.

  The switch events are sent through the OpenFlow protocol and interpreted by FlowVisor as well the OpenFlow controller.

  One could develop a module that gets the OpenFlow messages from FlowVisor and/or the OpenFlow controller. To receive all messages at a single point. It could be also maybe an extension to current monitoring tools like Nagios [32] or Cacti [33].

- **What kind of bugs are known and still present?** During experiment 2: VLAN based slice membership I did encounter the problem that the connection between FlowVisor and the OpenFlow controller was closed when using the stip-VLAN option in a flow entry.

  Furthermore during experiment 7 some difficulties showed up when using wildcards in flow entries to match fields in FlowSpaces, which created some holes in the network so that unwanted traffic could be inserted from an other slice.

  FlowVisor is not a finished product and bugs are present. Even when FlowVisor is finished there may be bugs. However, it may that some bugs are present but do not influence the working of FlowVisor in a specific setup, because the feature that has the bug is not used.

Based on the answers to the sub-research questions I want to answer my main research question

> *"Is the current FlowVisor implementation version 0.8.5 suitable to create stable virtual networks in production environments?"*

as follows:

The current implementation of FlowVisor is not suitable to run in production environments. During the research several setups are used and they mostly turned out to work well. But some advances setups introduced some security issues. It may depend on the need of the user how complex a FlowVisor topology should be. Some users may be happy with a simple setup with only port based slices, to get some hands on experience with OpenFlow. Others may really need more advanced setups, and when using these one should really know what he is doing. Not only how to configure this kind of setup, but more important how to handle the setup in case of an unexpected error.

7 Future Research

FlowVisor is not a finished product yet, new releases are published every few months. Therefore future research in the field is needed. One thing that should be tested is the new released version 0.8.6 [26] and 0.10.0rc1 [34] that were released recently. They give some improvements to version 0.8.5, which is used during the research [26]. Some of these improvements are:

- Hard slice resource limits of the TCAM space a slice can use.
- Path and naming mismatches between FlowVisor package and documentation.
- FlowVisor 0.6.x package improvements.
- Missing instructions for package install.

They all relate to some bugs, missing documentation or feature requests from users. There are more improvements, which can be found in the references that are stated above.

Furthermore a more advanced/bigger OpenFlow setup should be tested, with multiple OpenFlow switches. FlowVisor requires a lot of hardware resources according to the system requirements. During the research I worked with a very basic OpenFlow topology that consists of only one switch, this didn't add that much load to the VM where I ran Flowvisor as well the OpenFlow controllers. Although sometimes it took some seconds to insert a FlowSpace but this did not interrupt the use of FlowVisor.

Also the ignored priorities of the FlowSpace should be further investigated since this is an essential feature of FlowVisor to be able to classify traffic based on this priority, where more specific FlowSpaces get a higher priority.

One can have also a look into slicing a slice. Which means to use a Flowvisor on top of an other FlowVisor instance. Creating the right matching FlowSpaces may become harder in this situations.

A List of Acronyms

API Application Programming Interface
ARP Address Resolution Protocol
DPID Datapath Identifier
IPv4 Internet Protocol version 4
IPv6 Internet Protocol version 6
JSON JavaScript Object Notation
MAC-address Media Access Control address
REST REpresentational State Transfer
SDN Software Defined Networking
SSL Secure Sockets Layer
TCP Transmission Control Protocol
TLS Transport Layer Security
VLAN Virtual Local Area Network
VM Virtual Machine
XML-RPC Extensible Markup Language - Remote Procedure Call

B Configurations

B.1 Default Configuration

The default configuration of the setup:

Switch
Pronto 3290 with Open vSwitch installed provided by SARA
Br0 created including the OpenFlow interfaces
Switch DPID: `00:00:e8:9a:8f:fb:c3:5b`

Virtual Machines
Some pre-installed VMs were provided by SARA with running Ubuntu 12.04 LTS.
Each VM has two interfaces:
eth0: is used to configure the VMs from outside using SSH, and the OpenFlow controllers are connected through this interface.
eth1: is connected to the Pronto switch.

IP configuration of the VMs

VM01:		VM02:	
`eth0:`	`145.100.37.140`	`eth0:`	`145.100.37.142`
`eth1:`	`192.168.1.1`	`eth1:`	`192.168.1.2`
VM03:		VM04:	
`eth0:`	`145.100.37.143`	`eth0:`	`145.100.37.144`
`eth1:`	`192.168.1.3`	`eth1:`	`192.168.1.4`

FlowVisor was cloned from repository and compiled from source.

Configuration of FlowVisor on VM03

FlowVisor:	
`Port:`	`3366`
`RPC-Port:`	`8080`

The floodlight controllers were cloned from the repository, some fields in the file `floodlightdefault.properties` from the `src`-directory were edited according to the table below. Thereafter floodlight was compiled.

Configurations of the Floodlight controllers on VM3

Floodlight 1:		Floodlight 2:	
`port:`	`6634`	`port:`	`6635`
`RPC-port:`	`8081`	`RPC-port:`	`8082`
Floodlight 3:		Floodlight 4:	
`port:`	`6636`	`port:`	`6637`
`RPC-port:`	`8083`	`RPC-port:`	`8084`

B.2 Configuration of experiment 1

Slices:

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
fvctl createSlice slice2 tcp:145.100.37.143:6635 slice2@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=1 Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=2 Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=3 Slice:slice2=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=4 Slice:slice2=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow-1-2",
"cookie":"0","ingress-port":"1","active":"true", "actions":"output=2"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow-3-4",
"cookie":"0","ingress-port":"3","active":"true", "actions":"output=4"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"wrong-flow-2-4",
"cookie":"0","ingress-port":"2","active":"true", "actions":"output=4"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json
```

B.3 Configuration of experiment 2

Slices:

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
fvctl createSlice slice2 tcp:145.100.37.143:6635 slice2@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=1 Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=10 Slice:slice2=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=11,dl_vlan=60
Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=11,dl_vlan=50
Slice:slice2=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow1",
"cookie":"0","ingress-port":"11","vlan-id":"50","active":"true",
"actions":"output=10"' http://127.0.0.1:8082/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow2",
"cookie":"0","ingress-port":"11","vlan-id":"60","active":"true",
"actions":"output=1"' http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow3",
"cookie":"0","ingress-port":"1","active":"true","actions":"set-vlan-id=60,
output=11"' http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow4",
"cookie":"0","ingress-port":"10","active":"true","actions":"set-vlan-id=50,
output=11"' http://127.0.0.1:8082/wm/staticflowentrypusher/json
```

B.4 Configuration of experiment 3

Slices:

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
fvctl createSlice slice2 tcp:145.100.37.143:6635 slice2@example.com
fvctl createSlice slice3 tcp:145.100.37.143:6634 slice3@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=3,tp_dst=80
Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 101 in_port=1,tp_src=80
Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 102 in_port=3,tp_dst=8080
Slice:slice3=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 103 in_port=2,tp_src=8080
Slice:slice3=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 103 in_port=1,tp_dst=3306
Slice:slice2=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 103 in_port=2,tp_src=3306
Slice:slice2=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"fromwebserver",
"cookie":"0","ether-type":"0x0800","protocol":"6","src-port":"80",
"ingress-port":"1","active":"true", "actions":"output=3"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"towebserver",
"cookie":"0","ether-type":"0x0800","protocol":"6","dst-port":"80",
"ingress-port":"3","active":"true", "actions":"output=1"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"mysqlroclient",
"cookie":"0","ether-type":"0x0800","protocol":"6","src-port":"3306",
"active":"true","actions":"output=1"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"mysqltoserver",
"cookie":"0","ingress-port":"2","ether-type":"0x0800","protocol":"6",
""dst-port":"3306",active":"true","actions":"output=1"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"fromwebserver2",
"cookie":"0","ether-type":"0x0800","protocol":"6","src-port":"8080",
"active":"true","actions":"output=3"'
http://127.0.0.1:8083/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"towebserver2",
"cookie":"0","ingress-port":"3","ether-type":"0x0800","protocol":"6","dst-port":"8080"
,"active":"true","actions":"output=2"'
```

```
http://127.0.0.1:8083/wm/staticflowentrypusher/json
```

B.5 Configuration of experiment 4

Slices:

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_dst=52:54:00:b2:0d:d6
Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_dst=52:54:00:23:8b:87
Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_type=0x0806 Slice:slice1=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"macvm01",
"cookie":"0","dst-mac":"52:54:00:b2:0d:d6","active":"true", "actions":"output=1"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"macvm02",
"cookie":"0","dst-mac":"52:54:00:23:8b:87","active":"true", "actions":"output=2"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"arp",
"cookie":"0","ether-type":"0x0806","active":"true", "actions":"output=all"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json
```

B.6 Configuration of experiment 5

Slices:

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
fvctl createSlice slice2 tcp:145.100.37.143:6635 slice2@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=1 Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=2 Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=3 Slice:slice2=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 in_port=4 Slice:slice2=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow-1-2",
"cookie":"0","ingress-port":"1","active":"true", "actions":"output=2"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow-2-1",
"cookie":"0","ingress-port":"2","active":"true", "actions":"output=1"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow-3-4",
"cookie":"0","ingress-port":"3","active":"true", "actions":"output=4"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow-4-3",
"cookie":"0","ingress-port":"4","active":"true", "actions":"output=3"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json
```

B.7 Configuration of experiment 6

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
fvctl createSlice slice2 tcp:145.100.37.143:6635 slice2@example.com
fvctl createSlice slice3 tcp:145.100.37.143:6634 slice3@example.com
fvctl createSlice slice4 tcp:145.100.37.143:6635 slice4@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_src=52:54:00:b2:0d:d6
Slice:slice1=4,Slice:slice4=2
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_src=52:54:00:23:8b:87
Slice:slice2=4,Slice:slice4=2
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_src=52:54:00:d4:4d:83
Slice:slice3=4,Slice:slice4=2
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"vm01-vm02",
"cookie":"0","src-mac":"52:54:00:b2:0d:d6","active":"true", "actions":"output=2"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"vm01-web",
"cookie":"0","src-mac":"52:54:00:b2:0d:d6","ether-type":"0x0800","protocol":"6",
"dst-port":"80","active":"true", "actions":"output=3"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"vm02-vm01",
"cookie":"0","src-mac":"52:54:00:23:8b:87","active":"true", "actions":"output=1"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"vm02-web",
"cookie":"0","src-mac":"52:54:00:23:8b:87","ether-type":"0x0800","protocol":"6",
"dst-port":"80","active":"true", "actions":"output=3"'
http://127.0.0.1:8082/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"web-to-vm01",
"cookie":"0","src-mac":"52:54:00:d4:4d:83","dst-mac":"52:54:00:b2:0d:d6",
"ether-type":"0x0800","protocol":"6",src-port":"80","active":"true",
"actions":"output=1"' http://127.0.0.1:8083/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"web-to-vm02",
"cookie":"0","src-mac":"52:54:00:d4:4d:83","dst-mac":"52:54:00:23:8b:87",
""ether-type":"0x0800","protocol":"6",src-port":"80","active":"true",
"actions":"output=2"' http://127.0.0.1:8083/wm/staticflowentrypusher/json
```

B.8 Configuration of experiment 7

B.8.1 MAC based

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_dst=aa:aa:aa:aa:aa:aa
Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 dl_dst=bb:bb:bb:bb:bb:bb
Slice:slice1=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flowa",
"cookie":"0","src-mac":"aa:aa:aa:aa:aa:aa","active":"true",
"actions":"output=2"' http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flowb",
"cookie":"0","src-mac":"bb:bb:bb:bb:bb:bb","active":"true",
"actions":"output=1"' http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flow100",
"cookie":"0","ether-type":"0x0800","dst-ip":"192.168.1.1","active":"true",
"actions":"output=3"' http://127.0.0.1:8081/wm/staticflowentrypusher/json
```

B.8.2 IP-based

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 nw_dst=172.16.17.18
Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 200 nw_dst=192.168.1.1 Slice:slice1=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flowa",
"cookie":"0","ether-type":"0x0800","dst-ip":"192.168.1.1",
"active":"true", "actions":"output=2"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flowb",
"cookie":"0","ether-type":"0x0800","src-ip":"10.10.10.10",
"active":"true", "actions":"output=3"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json
```

B.8.3 Application port Based

```
fvctl createSlice slice1 tcp:145.100.37.143:6634 slice1@example.com
```

FlowSpaces:

```
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 100 tp_dst=80 Slice:slice1=4
fvctl addFlowSpace 00:00:e8:9a:8f:fb:c3:5b 200 tp_dst=443 Slice:slice1=4
```

Flowentries:

```
curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flowc",
"cookie":"0","ether-type":"0x0800","dst-port":"80","protocol":"6",
"active":"true", "actions":"output=1"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flowa",
"cookie":"0","ether-type":"0x0800","protocol":"6","dst-ip":"192.168.1.1",
"active":"true", "actions":"output=2"'
http://127.0.0.1:8081/wm/staticflowentrypusher/json

curl -d '"switch":  "00:00:e8:9a:8f:fb:c3:5b", "name":"flowb",
cookie":"0","src-mac":"bb:bb:bb:bb:bb:bb","active":"true",
"actions":"output=1"' http://127.0.0.1:8081/wm/staticflowentrypusher/json
```

References

[1] Software-defined networking: The new norm for networks. White Paper, April 2012.

[2] Openflow. Website. available at http://www.openflow.org/; on 04th September 2012.

[3] Openflow @ google. Presentation. available at http://opennetsummit.org/talks/ONS2012/hoelzle-tue-openflow.pdf; on 05th October 2012.

[4] Network development and deployment initiative (nddi). Website. available at http://www.internet2.edu/network/ose/; on 27th September 2012.

[5] Global environment for network innovations (geni). Website. available at http://www.geni.net/; on 27th September 2012.

[6] Japan gigabit network extreme (jgn-x). Website. available at http://www.jgn.nict.go.jp/english/info/technologies/openflow.html; on 27th September 2012.

[7] Openflow in europe: Linking infrastructure and applications. Website. available at http://www.fp7-ofelia.eu/; on 27th September 2012.

[8] Flowvisor. Website. available at https://openflow.stanford.edu/display/DOCS/Flowvisor; on 04th September 2012.

[9] Rob Sherwood, Glen Gibby, Kok-Kiong Yapy, Guido Appenzeller, Martin Casado, Nick McKeowny, and Guru Parulkar. Can the production network be the testbed? Website. available at http://www.deutsche-telekom-laboratories.de/~robert/flowvisor-osdi10.pdf; on 27th September 2012.

[10] Rob Sherwood, Glen Gibby, Kok-Kiong Yapy, Guido Appenzeller, Martin Casado, Nick McKeowny, and Guru Parulkar. Flowvisor: A network virtualization layer. Website. available at www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf; on 27th September 2012.

[11] Sara. Website. available at https://www.sara.nl/; on 5th October 2012.

[12] Surfnet. Website. available at http://www.surfnet.nl/; on 5th October 2012.

[13] Ronald van der Pol. D1.2 openflow. Website. available at https://noc.sara.nl/nrg/publications/RoN-2011-D1.2.pdf; on 27th September 2012.

[14] Openflow tutorial. Website. available at http://www.openflow.org/wk/index.php/OpenFlow_Tutorial; on 03th October 2012.

[15] Pica8 3290. Website. available at http://www.pica8.org/products/p3290.php; on 5th October 2012.

[16] Open vswitch. Website. available at http://openvswitch.org/; on 5th October 2012.

[17] Openflow switch specification. Website. available at http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf; on 3th October 2012.

[18] Openflow 1.2. Website. available at https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.2.pdf; on 3th October 2012.

[19] Beacon openflow controller. Website. available at https://openflow.stanford.edu/display/Beacon/Home; on 3th October 2012.

[20] Floodlight. Website. available at http://floodlight.openflowhub.org/; on 04th September 2012.

[21] Maestro. Website. available at http://code.google.com/p/maestro-platform/; on 3th October 2012.

[22] Nox/pox. Website. available at http://www.noxrepo.org/; on 3th October 2012.

[23] Trema - full-stack openflow framework in ruby and c. Website. available at http://trema.github.com/trema/; on 14th October 2012.

[24] Static flow pusher api. Website. available at http://www.openflowhub.org/display/floodlightcontroller/Static+Flow+Pusher+API; on 3th October 2012.

[25] The floodlight rest api. Website. available at http://www.openflowhub.org/display/floodlightcontroller/REST+API; on 3th October 2012.

[26] Flowvisor 0.8.6 released. Website. available at https://mailman.stanford.edu/pipermail/openflow-discuss/2012-October/003705.html; on 11th October 2012.

[27] Ubuntu 12.04.1 lts (precise pangolin). Website. available at http://releases.ubuntu.com/12.04/; on 11th October 2012.

[28] Lighthouse - joint research lab. Website. available at https://www.sara.nl/project/lighthouse; on 19th September 2012.

[29] Strip vlan id action causes connection reset. Website. available at https://mailman.stanford.edu/pipermail/openflow-discuss/2011-January/001808.html; on 22nd October 2012.

[30] Apache http server. Website. available at http://httpd.apache.org/; on 10th October 2012.

[31] Mysql. Website. available at http://www.mysql.com/; on 10th October 2012.

[32] Nagios homepage. Website. available at http://www.nagios.org/; on 2nd November 2012.

[33] Cacti. Website. available at http://www.cacti.net/; on 2nd November 2012.

[34] Flowvisor 0.10.0rc1 released. Website. available at https://mailman.stanford.edu/pipermail/openflow-discuss/2012-October/003706.html; on 11th October 2012.