Universiteit van Amsterdam
System and Network Engineering

Reseach Project 2

Network utilization with SDN in on-demand application-specific networks

Author: Ioannis Grafis
`Ioannis.Grafis@os3.nl`

Supervisor: Marc X. Makkes, Ph.D.c.
`M.X.Makkes@uva.nl`

Amsterdam, Netherlands

July 2015

Abstract

By using Software Defined Networking (SDN) instead of classical Internet routing protocols routing granularity can be increased. Moreover, routes can be manipulated even per flow. This research introduces a solution for network optimization by using OpenFlow[1] in on-demand application-specific overlay networks. The proposed algorithm manipulates the network metrics based on the network statistics gathered from the OpenFlow controller. Moreover, a proof of concept is built by extending the second implementation of Internet factories, which is called Compute factory. The extension has two parts. The first part is about enabling in the created networks to make use of OpenFlow instead of Open Shortest Path First (OSPF). The second part is about creating a second control loop in the Compute factory in order to monitor and control the routing flows. Consequently, the extended version of Compute factory enhances the performance of application-specific overlay networks.

Contents

1 Introduction

Nowadays, the use of the cloud is becoming a commodity. The end-users spending for public cloud services was 112 billion dollars in 2012 and it is estimated that it will be 293 billion dollars in 2018[2]. It is widely used by both commercial and scientific applications. Researchers make use of several test beds, such as GENI[3] and PlanetLab[4], in order to deploy and test new functionalities immediately. Both mentioned platforms make use of overlay networks. Overlay network is a virtual network that is built on top of one or more physical networks. From the user perspective each network is independent. Overlay networks are used in order to add extra functionality without modifying the underlying infrastructure. Moreover, this type of network can provide the needed infrastructure for Internet applications.

An approach, to create globally overlay networks on cloud platforms, is the Internet factories[5]. Internet factories can create, on-demand, IP-based overlay networks by using Infrastructure-as-a-Service (IaaS) clouds. The created networks consist of Virtual Machines (VMs) running in the cloud, which are connected via IP tunnels. Internet factories facilitates the task of creation, configuration and modification of the infrastructure. The modification is done with a control loop that monitors and adjusts the infrastructure.

SDN is an approach to network abstraction. It decouples the control plane from the data plane, by moving the control plane from the switches to the controller. A standard communication interface between the switches and the controller is OpenFlow. OpenFlow can provide us programmability over the network. Moreover, we can modify network metrics through the OpenFlow controller in order to achieve better routing granularity.

Elephant flows are long-lived flows which require a lot of bandwidth. On the other hand, mice flows are flows with small bandwidth demands. Studies in a variety of different network types[6–9] have been shown that only a small portion of flows is elephant flows, while the biggest part is mice flows.

The scope of this research is to evaluate the benefits that on-demand application-specific overlay networks can gain by using OpenFlow. For this purpose a proof of concept is created by extending the second implementation of Internet factories, called Compute factory. The extension consists of two parts. The first part is about using OpenFlow instead of OSPF in the created networks. The second part is about creating a second control loop that observes the network statistics and modifies the network metrics. For both the observance and the modification we are using the OpenFlow controller. Amazon IaaS cloud was used as a test bed in order to evaluate the created proof of concept.

The report has the following structure. The related work and a sufficient theoretical background to fully understand the proposed solution are presented in Sections 3 and 4. In section 5, the extension of the Compute factory is explained. Section 6 contains the experiments that we run in order to evaluate the created proof of concept. The conclusions of this research are discussed in section 7. The report ends in section 8 with the future work.

2 Problem statement

Compute factory creates on-demand application-specific overlay networks that use OSPF as a routing protocol. Based on of Dijkstra's algorithm OSPF computes the shortest path between all hosts in the OSPF network. OSPF is a mature protocol but it lacks in programmability and routing granularity that OpenFlow can provide us. Moreover, OSPF uses the same path for all the traffic between two specific end-hosts. Equal-Cost Multi-Path (ECMP) can be used in order to split the traffic into two (or more) paths with equal cost. ECMP does not consider what type of traffic it is load balancing between the different paths. Moreover, it is only works for paths with equal cost. Elephant flows are high bandwidth-consuming long-lived flows while mice flows are short-lived flows. Routing both elephant and mice flows from the same path can cause problems in the mice flows. Elephant flows consume almost all the resources of the network. They tend to fill the network buffers. As a result mice flows have increase packet drop and non-trivial queuing delay.

OpenFlow can provide routing granularity, even per flow level. Moreover, the network switches maintain statistics that can be collected through the OpenFlow controller. Based on the version 1.3.0 of OpenFlow Switch specification a wide variety of statistics can be maintained. The statistics can provide useful network statistics, for example the number of flows or the duration of each flow. Moreover, more complicated network statistics can be calculated, for example discover congested links based on the link utilization or characterize a flow as an elephant or mice. Based on the statistics and the programmability that OpenFlow provides, the flow entries can be modified to optimize the network, for example to decrease the total transfer time and the delay of the flows.

Therefore, the main research question is the following:

- To what degree can on-demand application-specific overlay networks benefit by making usage of OpenFlow?

and the following sub-question:

- How can we manipulate the flow entries in order to enhance the performance of both elephant and mice flows?

3 Related work

The programmability and the routing granularity are two main characteristics of OpenFlow. Based on these characteristics, researchers implemented automated procedures that enhance the network performance.

In [10], Sushant Jain et al., a SDN solution is proposed for deploying OpenFlow in Wide Area Network (WAN.) The paper proposes an algorithm to allocate bandwidth in the network. The algorithm takes care of sharing the available bandwidth between competitive applications. Moreover, it uses multiply paths between two end-hosts. This system increases the link utilization from 30-40% to almost 100%.

In [11] is proposed a solution of load balancing mice flows in data center networks. Mice flows are more than 90% of all flows in a data center[12]. The paper proposes an algorithm in order to calculate the links utilization. Based on the outcome, the flow rules are manipulated in order to load balance the mice flows. Because of the big number of mice flows, flow aggregation is used to manipulate the mice flows.

Another solution for load balancing in data center network is described in [13]. Elephant flows in the data center are less than 10 % of the flows but they consume more than 80% of the bandwidth[12]. A mechanism of load balancing elephant flows between different paths is proposed. The mechanism calculates the link utilization. Based on links utilization, the mechanism allocates the elephant flows through multiple paths. With this mechanism the network utilization is increased from 70% for single path routing and from 84% for ECMP to 92%.

4 Theoretical background

In this section a sufficient theoretical background is presented in order to fully understand the proposed solution. Furthermore, more details will be explained about how Compute factory works.

4.1 Differences from previous studies

The three previous studies, that are mentioned in section 3, have been done either in WAN or in data center networks. Both WAN and data center are physical environments. The links between the nodes have a standard (fixed) bandwidth. This means that the throughput and the delay between two nodes can be changed based on 2 factors, the changes of the infrastructure and the usage of the network.

On the contrary, Compute factory uses IaaS cloud to create application-specific overlay networks on-demand. As we mentioned in the previous paragraph, the throughput and the delay can be changed based on 2 factors. In case of an application-specific overlay network, these 2 factors are both for the underlying and the virtual infrastructure. This means that instead of 2 factors, 4 factors need to be taken into account. From those 4 factors, the 2 factors of the underlying infrastructure are unknowns. The underlying cloud infrastructure might change, for example the network topology changes or VM migration might occur. These changes cannot be seen by the end-users, but they might affect the delay and throughput of the network links. Furthermore, the created network does not reserve paths in the underlying infrastructure. As a result, the links bandwidth, of the created network, might vary based on the other end-user usage of the cloud infrastructure.

4.2 Elephant and mice flows

Elephant detection is a well-researched field[14–16]. Many studies have been done and many solutions have been proposed. In these studies the detection of elephant flows is done either at the end-host, Mahout[14], or at the edge switch, DevoFlow[15] and Hedera[16]. Based on [14], the most efficient way of detecting an elephant flow is at the end-host by observing the socket buffers of the end-host. For this study the assumption that there is no access to network driver statistics at the end host is made. For this reason, Compute factory uses the statistics gathered from the switches to detect the elephant flows. The statistics can be accessed through the OpenFlow controller. Therefore, Compute factory collects the network statistics through the OpenFlow controller and based on these statistics it categorizes the flows to either elephant or mice.

4.3 Compute factory

Compute factory is the second implementation of Internet factories. It is written in Java. It can use three different cloud providers, Amazon[17], GoGrid[18] and Rackspace[19], to create application-specific networks on-demand. By specifying the number of the VMs with their characteristics, Compute factory creates the network topology. A VM is specified by 6 characteristics, the zone that the VM will be created, the VM Operating System (OS), the minimum number of virtual Central Processing Units (CPUs), the minimum Random-Access Memory (RAM) size, the minimum disk size and the VM hostname. After the creation of the VMs, Compute factory installs and starts the required software. Compute factory executor is also required in the VMs. Compute factory executor is used to configure, observe and adjust the VMs. The communication between the Compute factory and the Compute factory executors is done by using an Apache ActiveMQ[20] message broker.

Compute factory sends the required information to a Compute factory executor to create the Internet Protocol (IP) tunnels between the VMs. The IP tunnels are created based on the network topology that is specified on the Compute factory. Compute factory creates application-specific networks, therefore the network topology will vary based on its usage. The IP tunnels are created with VTun[21]. Compute factory executor creates a new network interface for each created tunnel. Compute factory takes care of assigning a unique IP address in every network interface. The IP addresses belongs to the 10.0.0.0/8 subnet. For each IP tunnel a /30 subnet is assigned in order to have 1 IP address for each one of the 2 network interface of the tunnel. Compute factory installs to the created VMs Quagga[22]. Quagga is a routing software suite, which provides routing protocols, such as OSPF and Routing Information Protocol (RIP). Compute factory starts the Quagga daemon in every VM to provide to the switches OSPF as routing protocol. The creation of the tunnels is part of the VM configuration.

When the configuration of the network is finished, the network topology is ready for use. Compute factory uses a control loop in order to observe and adjust the network. Through the network modification, Compute factory tries to adjust the network to match application needs. An example of network modification is the creation or deletion of a VM.

5 Extending Compute factory

In this section we analyze the parts of Compute factory that are changed or created during this research. The extension can be divided in two parts. The first part is about modifying Compute factory to create VMs that use OpenFlow. The second part is about adding a second control loop in Compute factory that observes the network statistics and controls the flows.

5.1 OpenFlow support

SDN decouples the control plane from the data plane. While the data plane is still in the switches, the control plane is moved to the controller. For this reason, a VM is created manually to be used as the OpenFlow controller. Floodlight[23] version 1.1 is used as OpenFlow controller. The IP address of the OpenFlow controller is given as a parameter in Compute factory to configure the created VMs. The VM that contains the OpenFlow controller can be placed anywhere. Placing the controller far from the network creates a high latency between the controller and the switches[24]. The high latency will introduce slow response from the controller. Therefore, controller location should be close to the network topology. Far and close are based on Round-Trip Time (RTT) between the controller and the switches.

As we mentioned in 4.3, Compute factory installs Quagga to the created VMs to use OSPF as routing protocol. In the switches, OSPF need to be replaced with OpenFlow. For that reason, Compute factory installs OpenvSwitch[25], instead of Quagga, in the created VMs. The latest stable version of OpenvSwitch is used, version 2.3.1. OpenvSwitch is installed during the installation of required software that we mentioned in Section 4.3. Compute factory starts and configures OpenvSwitch with the following 5 commands:

```
service openvswitch start
ovs-vsctl set-ssl /root/openvswitch/sc-privkey.pem /root/
    openvswitch/sc-cert.pem /root/openvswitch/cacert.pem
ovs-vsctl add-br br0
ovs-vsctl -- set bridge br0 protocols=OpenFlow13
```

```
ovs-vsctl set-controller br0 ssl:1.2.3.4:6653
```

The first command is to start the OpenvSwitch. The second command, which is not mandatory, is to set a private key and certificates that OpenvSwitch uses in order to communicate with OpenFlow controller by using SSL. The usage of SSL ensures integrity and confidentiality of the connections. The third command is used to create a bridge with name “br0” in the switch. The network interface, that VMs use for create the tunnels, will be added to this created bridge. The name of the bridge is added as a parameter to the Compute factory. The forth command is used to set the OpenFlow version to the created bridge. OpenFlow version 1.3 is used for this research. The default version of OpenFlow that OpenvSwitch uses is version 1.0. The last command sets the controller for bridge “br0”. In the command the “ssl” specifies that it is going to be used an SSL connection between the switches and the controller.

As we mentioned in Section 4.3, Compute factory uses VTun to create the IP tunnels between the VMs based on the topology that is given as an input to Compute factory. Each tunnel has two end-points. For each end-point a new network interface is created and added in the “br0” bridge. OSPF need to know the IP addresses of the network interfaces in order to create the routing table. Therefore, Compute factory needs to configure an IP address in every network interface. To use OpenFlow, this step is no longer required. For this reason, Compute factory is modified to create Ethernet tunnels instead of IP tunnels and it does not assign an IP address to the created network interfaces. As a result of this modification, each created VM has only 1 IP address, the IP address that is assigned to the VM by the cloud provider. Something that is worth mentioning is that the network interfaces should be added in the bridge after the tunnel has been established. Otherwise, in some cases, OpenvSwitch cannot recognize the network interfaces.

5.2 Second control loop

As we mentioned in Section 4.3 Compute factory already contains a control loop to monitor and adjust the infrastructure. During this research a second control loop was created. The second control loop is used to monitor the network statistics and based on them automatically control the flows of the network. The intention of this automated process is to enhance the network performance by redirecting the elephant flows to an alternative (longer) path in order to separate elephant and mice flows. The communication between the Compute factory and the OpenFlow controller is done by the REST API that the OpenFlow controller has. In Figure 1 we can see the algorithm of the flows control loop.

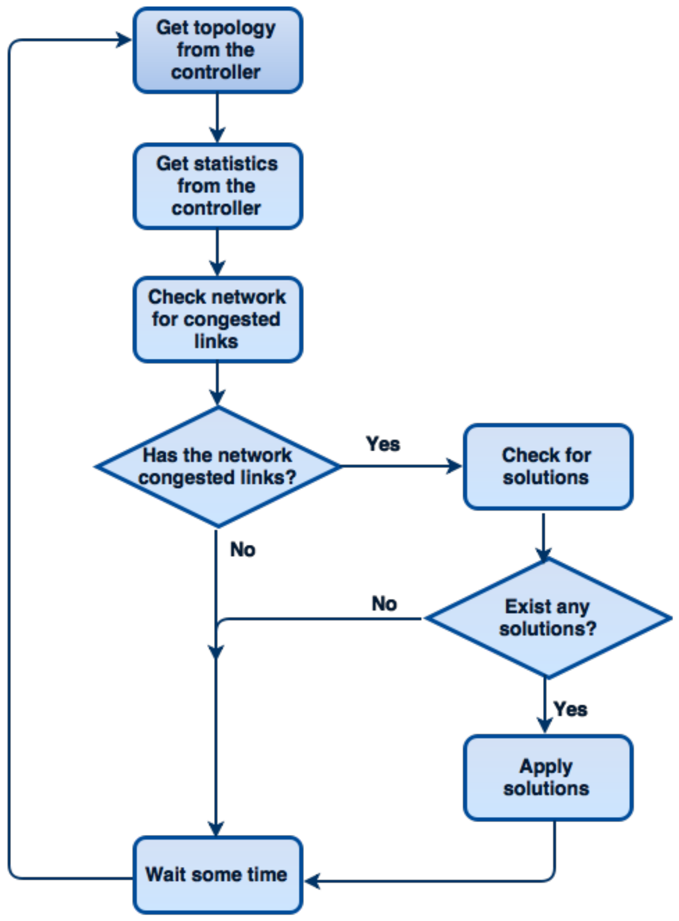

Get topology from the controller
Get statistics from the controller
Check network for congested links
Has the network congested links?
Yes
Check for solutions
No
No
Exist any solutions?
Yes
Apply solutions
Wait some time

Figure 1: Compute factory flows control loop algorithm

The control loop consists of 5 basic steps, get the network topology, get the network statistics, check for congested links, check for solutions and apply solutions. The first three steps are always executed, while the last two steps are executed only in specific cases. For this research project the execution frequency of the above loop is configured to be 1 second. By increasing the execution frequency, the reaction will be faster, but the load of the controller and the switches will be increased. On the other hand, opposite results will be achieved by decreasing the execution frequency. While there is no limit in lowering the frequency of the control loop, there is a limit in increasing the frequency. This limit is equal to the time that the control loop needs to be executed. The time that the control loop needs to be executed can vary based on the RTT between the Compute factory and the OpenFlow controller, the size of the network, the number of flows that exist on the network, etc.

5.2.1 Learning the topology

At the first step of the algorithm Compute factory learns the topology from the OpenFlow controller. Compute factory uses three different REST API calls of the Floodlight controller. With these REST API calls Compute factory collects the switches information, the links information and the switches ports. With the information gathered from the first two REST API calls, Compute factory creates a graph of the network topology. The last REST API call is used to learn which switches ports are used to connect with end-hosts. Except from the default (shortest) path that is used between two end-hosts, alternative paths should be used. For this reason, Compute factory calculates the K shortest paths between all hosts of the network. The K shortest paths are calculated based on the Bellman-Ford algorithm[26]. All the calculated paths stored for later usage. K needs to be equal with 2 or higher. After the first time that the loop is executed, in case that there is no change in the network topology, Compute factory does not recalculate the K shortest paths.

5.2.2 Getting the statistics

Compute factory needs to have the ability to check the link utilization and the type of flows. For these reasons Compute factory collects the network statistics through the OpenFlow controller. The statistics are gathered both per port and per flow.

The statistics per port are used in the third step. As each link is connected with two ports, Compute factories collects statistics for both ports. The created network is an overlay network in IaaS cloud. Therefore, a link in our network can consist of many physical links in the underneath infrastructure. For example, a virtual link uses a high-loaded switch which has a high packet drop rate. This means that the statistics gathered for the same link from the two switches might differ. For this reason Compute factory collects and stores statistics for both ports. The port statistic have different counters for transmitted and received data. Because the links of the network do not have separate channels for uplink and downlink, link statistics are stored as a sum of transmitted and received data.

The statistics per flow are used in forth step to categorize flows either as elephant or mice flow. Statistics for one flow are gathered from all the switch that this flow uses. For the same reason that we analyzed in the previous paragraph, Compute factory stores all the collected statistics. A flow is specified by 5 characteristics, the source IP address, the destination IP address, the source port, the destination port and the flow type. The flow type can be either elephant or mice.

5.2.3 Finding congested links

Based on the per port statistics that Compute factory collects, during the second step, at this step Compute factory checks the network for congested links. If the throughput of a link is bigger than a specified threshold, this link is characterized as congested. The threshold is specified as a parameter in Compute factory. For better accuracy the algorithm checks the throughput of a link as an average throughput of the last 5 seconds. A 5 second timeslot is used in order to not get affected by either positive or negative spikes. The disadvantage of using a bigger timeslot is that it increases the time that Compute factory needs to determine that a link is congested.

As we mentioned in Section 5.2.2, the link statistics is a sum of transmitted and received data. The reason for doing this is that the uplink and the downlink share the same tunnel. As a result, the throughput of the link is shared between the upload and the download. For example, a link with 100 Mbps throughput can have 100 Mbps upload and 0 Mbps download or 30 Mbps upload and 70 Mbps download. In addition, Compute factory stores two statistics, one for each port of the link. The algorithm calculates the average of the two ports statistics. The calculated average is the number that will be compared with the threshold in order to characterize a link as congested.

5.2.4 Finding solutions

In case that during the previous step Compute factory discovered at least one congested link, it will continue by executing this step. Otherwise, it will go to the first step. During this step Compute factory will try to find solutions for the congested links of the network. First, based on the flows statistics that Compute factory gathered during the second step it will categorize the flows as either elephant or mice flows. By default all flows are mice flows. In order to characterize a flow as an elephant flow, the flow should live for more than 10 seconds and consumes more than the link's bandwidth divided by the number of link's flows. For example, link's bandwidth is 10 Mbit/sec and 5 flows use this link, in order to characterize a flow as an elephant flow it should consume more than 2 Mbit/sec.

After finishing categorizing the flows, Compute factory checks for solutions for all the links of the network that have been marked as congested during the second step. In case that a link is used by more than one flow and these flows are both elephant and mice then Compute factory will search alternative paths to redirect the elephant flows. Compute factory checks the alternative paths if they have congested links or if they have mice flows. Compute factory searches for an alternative path that has no congested links and no mice flows. In case that Compute factory finds an alternative path with these characteristics, it continues to the next step in order to apply the solution. In case that there are more than one alternative paths Compute factory checks them all. In case there are more than one elephant flows and more than one alternative paths then Compute factory will load balance the elephant flows to all the paths. After applying this solution mice flows continue to use the shortest (default) path while the elephant flows use a longer (alternative) path.

Another solution would be to redirect the mice flows to a longer (alternative) and keep the elephant flows to the shortest (default) path. Due to the big number of mice flows in a network it might be expensive or even impossible to redirect them one by one. For this reason, general rules need to be applied for aggregating the mice flows. For the existing elephant flows the flow priority need to be increased to not follow the general rule that it is applied. Because the way that the proposed algorithm checks for elephant flows, through the controller by checking the network statistics, this solution is not possible. There are two reasons for that. The first reason is that after grouping a lot of mice flows together Compute factory will get the aggregate statistics as one flow. Therefore, there is a possibility that the group of mice flows will be recognized as an elephant flow. The second reason is that except from the already existing flows, new flows will also covered by the general rule. These new flows can be either elephant or mice flows. The problem is that the collected statistics will be for one flow which contains both elephant and mice flows. In order to apply this solution the check of the flows need to be done at the end-host, however for this research is considered that there is no access to the end-hosts.

5.2.5 Applying solutions

This step is executed in case that Compute factory found a solution during the previous step. Based on the output of the previous step, Compute factory generates and pushes the new flow entries, through the controller, in the switches of the new path. All the pushed flow entries have a higher priority than the default flow entries that Floodlight creates. The higher priority is needed to make the flows to use the pushed flow entries and not the default one. Moreover, all the flow entries that are pushed have an idle timeout of 5 seconds. By using an idle timeout the flow entries will be removed 5 seconds after the switches do not receive any packet that matches the flow entry. Therefore, Compute factory does not need to track the pushed flow entries to remove them. The new flow entries are created based on the information that the current flow provide us. Each flow entry is specified by the ID of the switch that is pushed and by the 5 flow characteristics that we already mentioned in Section 5.2.2. The flow entries are pushed sequential from the one edge of the path to the other. Moreover, flow entries are pushed for both directions of the flow.

6 Experimental results

In this section we are going to discuss the experiment that is used to evaluate the created proof of concept. This section is divided in two subsection. In the first subsection, the used test bed is explained. The second subsection presents the scenarios that are used in order to evaluate the performance of the created proof of concept. Moreover, the collected results are presented.

6.1 Testing environment

To deploy a topology Amazon cloud was used. Figure 2 shows the network topology that was used in order to run the testing scenarios. Two VMs of the network were created manually. In the first VM, the software of Compute factory was installed. The second VM was used as the OpenFlow controller. For this reason, the Floodlight software was installed and started in the second VM. The IP address of the OpenFlow controller was given as a parameter to the Compute factory. The switches were created by the Compute factory. The communication between the Compute factory with the OpenFlow controller and the OpenFlow controller with the switches was done by the public IP addresses that Amazon assigns to the created VMs. The first control loop of Compute factory, which monitors and adjusts the infrastructure, was not used during this experiment.

All the VMs use the t1.micro instance type, which is the smallest instance type in Amazon. All the VMs uses a CentOS 6.5 image from the community AMIs with number “ami-4d23507d”. For simulating end-host internal network interfaces were created at the switches. The assigned IP address of the internal network interfaces belong to the 172.16.0.0/24 subnet. The IP address is assigned from the Compute factory based on the hostname of each switch. Switch1 will get the 172.16.0.1 IP address, Switch2 will get the 172.16.0.2 IP address, and so for. All the tests have been done by using these assigned IP addresses. From the OpenFlow controller perspective the internal network interfaces seems as another node in the network.

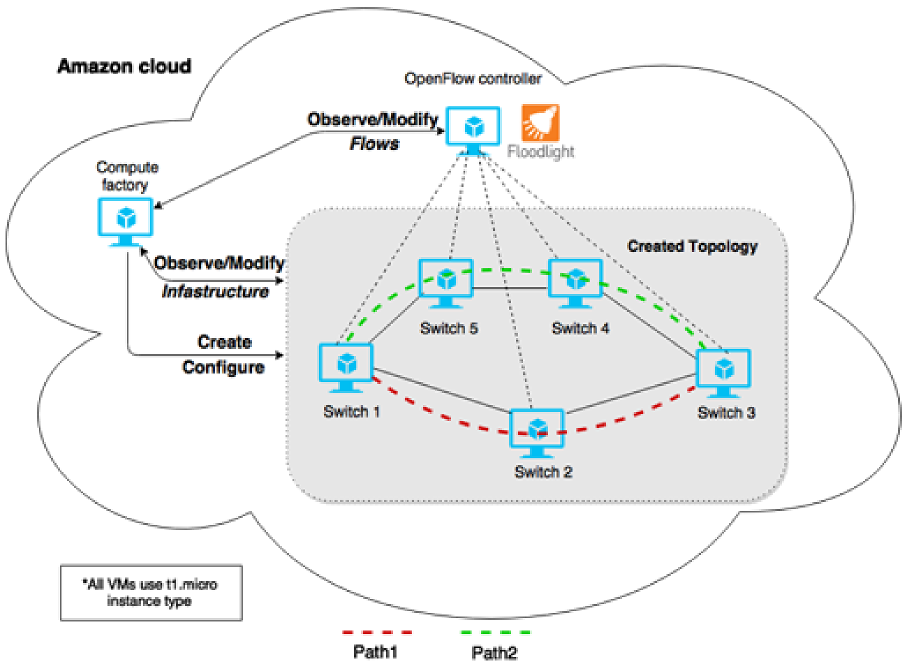


Figure 2: Testing environment topology

6.2 Testing scenarios and results

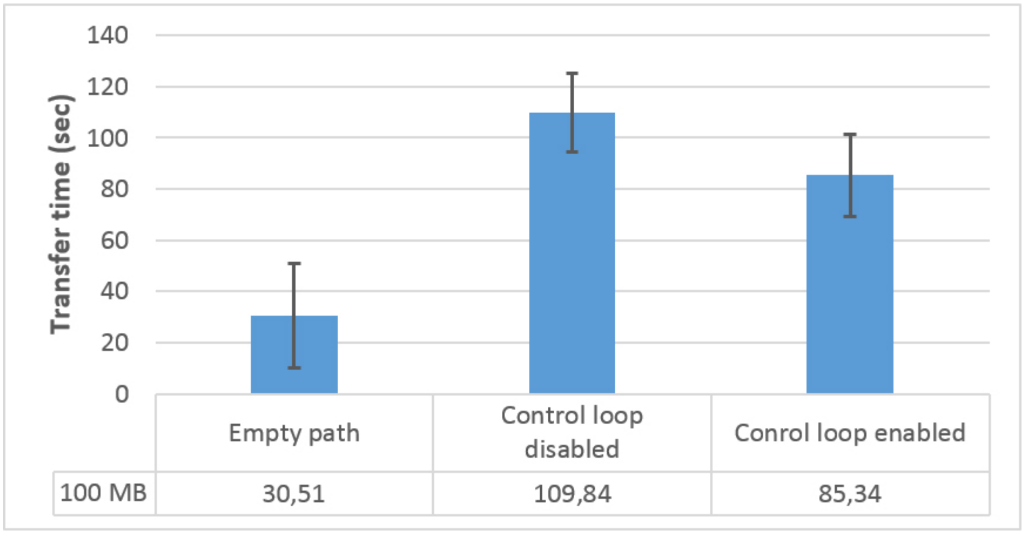
Three scenarios were created and used to evaluate the performance of the created proof of concept. In each scenarios both mice and elephant flows are used. The data are transferred from Switch1 to Switch3. SCP[27] is used to transfer data. The elephant flow is a transfer of a 500 MB file. In order to generate the mice flows, a bash script is created and used. The script can be found in Appendix. In every test, the total data that are transferred for mice flows is 100 MB which is consisted by 20 individual file transfers. Each file could have size from 1 till 10 MB, with a scale of 1 MB. A random interval is used between two sequential mice flows. The interval could be from 1 till 5 seconds. The interval is used between the start times of two sequential mice flows. Only the actual transfer time is measured and presented in the results. The total transfer time of mice flows is the sum of the transfer times of each mice flow. Therefore, the interval between the mice flows does not affect the total transfer time of mice flows. The 11 files that were used, one for the elephant flow and 10 for the mice flows, were created with the dd[28] utility. A script to generate these files can be found in the Appendix. In Figure 3 and Figure 4 we can see the results that we gathered by running the following three scenarios. All the measurements are taken 10 times for better accuracy.

6.2.1 Empty path scenario

The first scenario is a best case scenario. During this test, data for elephant and mice flows are transferred sequential. The flows use the path Switch1 – Switch2 – Switch3 (Path1), because this is the shortest path between the end-hosts. This test is used to discover the best results that one can achieve. In Figure 3 and 4 the results of the first scenario are presented in the left column. In both figures the standard deviation is quite high, even that the transfers have been done in an empty path. A reason for that is that the links of the network do not use reserved paths on the cloud infrastructure. This means that based on the usage of the other users of the cloud the bandwidth between the switches can vary.

6.2.2 Control loop disabled

During this test, data for elephant and mice flows are transferred simultaneously. As in the first scenario, all the data transferred through Path1. As a result, that mice and elephant flows shared the same path, there is an increase in total transfer times of both elephant and mice flows. Both standard deviations of this scenario are in the same level as in the first scenario. In Figure 3 and 4 the results of the second scenario are presented in the middle column.



	Empty path	Control loop disabled	Conrol loop enabled
100 MB	30,51	109,84	85,34

Figure 3: Total transfer time of mice flows

6.2.3 Control loop enabled

In this scenario, data for elephant and mice flows are transferred simultaneously, as in the second scenario. The difference is that for this scenario the second control loop of the Compute factory is enabled. At the beginning, Elephant and mice flows use Path1. When Compute factory discovers that one of the two links of the path is congested it will check the flows statistics in order to categorize the flows. After Compute factory discovers that one of the flows is an elephant flow, it will redirect the elephant flow to the longer path, Switch1 – Switch5 – Switch4 – Switch3 (Path2). To be able to characterize the flow as elephant flow, the flow needs to live for more than 10 seconds. This means that in best case scenario Compute factory can redirect an elephant flow after 10 seconds. We can see that in both mice and elephant flows we have a decrease in total transfer time in compare with second scenario. For mice flows the decrease is 22.3% and for elephant flows the decrease is 12.7%.

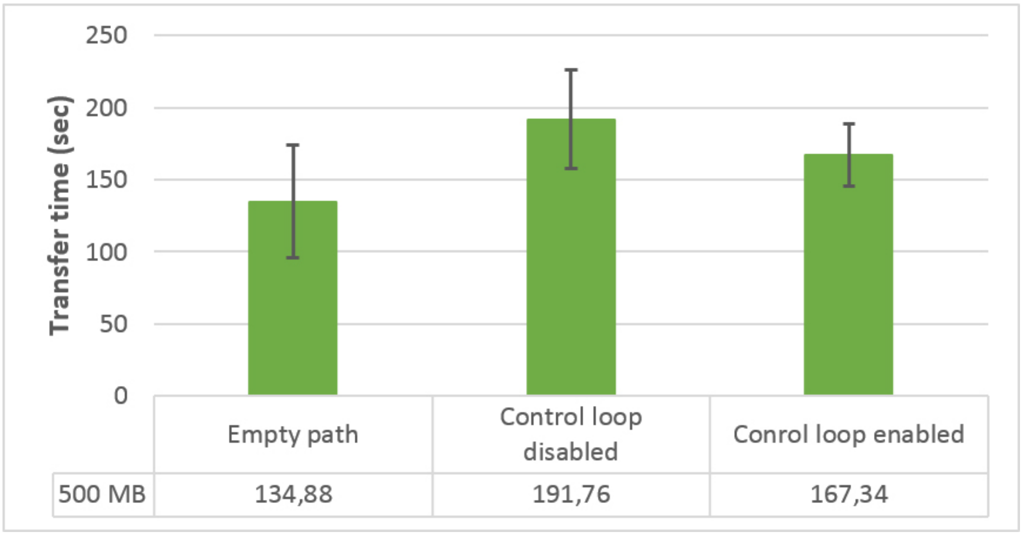


Figure 4: Total transfer time of elephant flow

Something else that can be noticed is that the standard deviation of mice flows is almost the same as with the other two scenarios, while there is a decrease for the standard deviation of elephant flows. In Figure 5 we can see the CPU utilization for the intermediate switches for both Path1 and Path2, Switch2, Switch4 and Switch5. This graph is automatically created by the Amazon website. The left part of the graph shows the average CPU utilization during the third scenario while the right part during the second scenario. During the third scenario the control loop of Compute factory was creating extra load for the switches by collecting the network statistics. Even with the extra load that Compute factory adds to the switches during the third scenario, by collecting the switch statistics, the CPU load of all switches during the third scenario are lower than the CPU load of Switch2 during the second scenario. A reason for that is that the traffic is balanced between the 2 paths. During all the tests, only the necessary software for these experiment was running on the VMs.

The high, close to 100%, CPU load of the Switch2 during the second scenario is undesirable. The high CPU load for long period creates 2 problems. First of all, it delays other task that need to be processed by the CPU, for example replay to the OpenFlow controller with the switch statistics. Second, it increases the processing delay, which can increase the queue delay. Increasing the queue delay can cause packet loss due to full buffers in the switch.

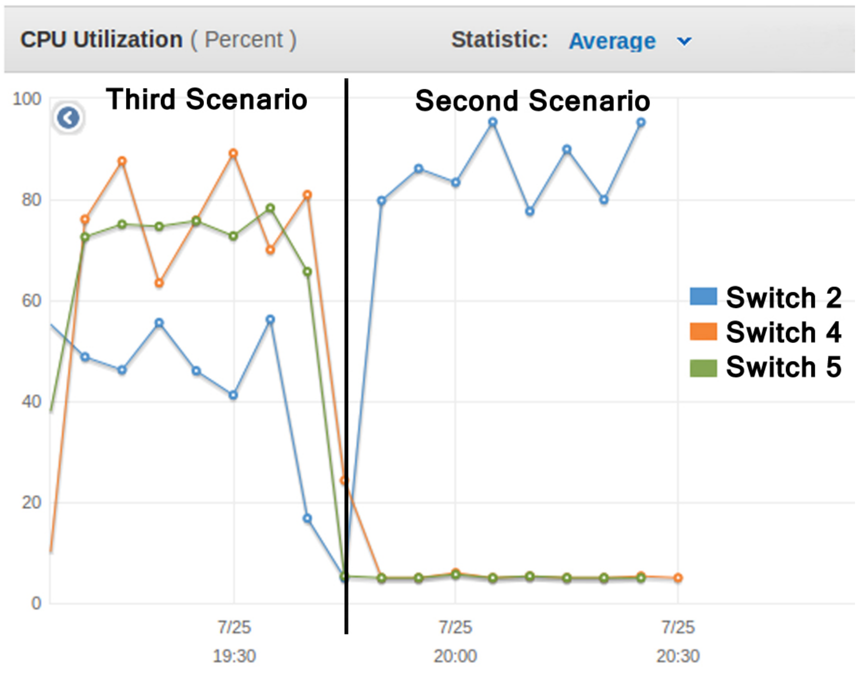
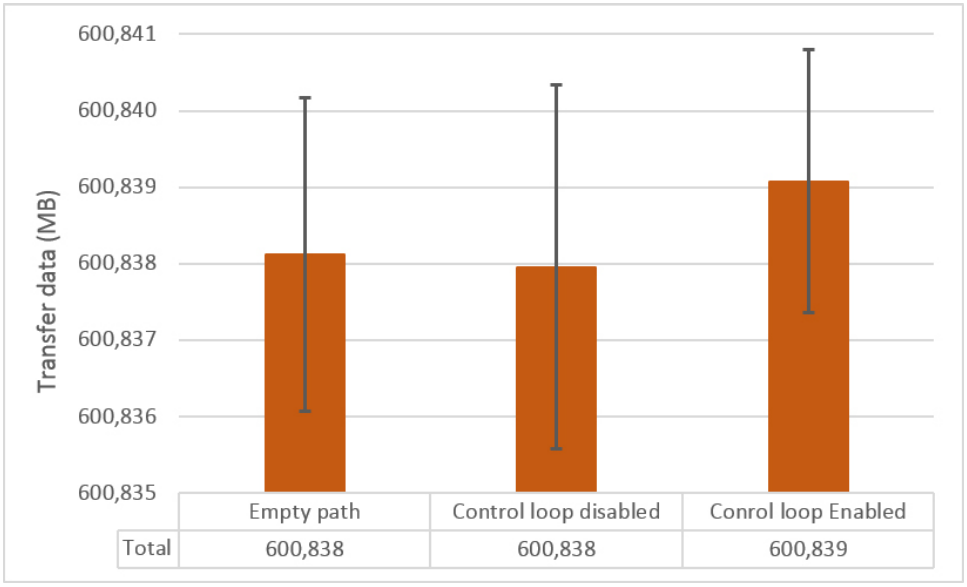


Figure 5: CPU utilization of intermediate switches

Another concern that might arise is the packet lose that might be created because of the flow redirection. Figure 6 shows the total transfer data, for both elephant and mice flows, for all three scenarios. The result is a sum of doth transmitted and received data based on the SCP output. In the case of the third scenario, there is an increase of 1 KB, which is less than 0.001% of the total transferred data. This means that the packet loss, due to the redirection of a flow, is negligible for an elephant flow, but it could be quite big for a mice flow.



	Empty path	Control loop disabled	Conrol loop Enabled
Total	600,838	600,838	600,839

Figure 6: Total data transfer

7 Conclusions

This research proves that the performance of on-demand application-specific networks can be enhanced with the usage of OpenFlow. With the created proof of concept the transfer time is decreased by 22.3% for mice flows and 12.7% for elephant flows. This is done by load balancing the flows based on the flow type. Through an automate procedure Compute factory can redirect the elephant flows to an alternative (longer) path and leave the mice flows to the default (shortest) path. By load balancing the flows we can decrease the CPU utilization on high load switches. By decreasing the CPU load a more deterministic environment is created.

8 Future work

For this research project an OpenFlow solution is developed to load balance elephant and mice flows. This solution might create concerns about scalability issues. Moreover, CPU utilization for all the components of the network should be investigating by increasing both the network size and the number of flows.

Compute factory can create on-demand application-specific network in global scale. During this research only one region of Amazon cloud is used in order to place the created network topology. By creating globally networks the big RTT between the OpenFlow controller and the switches or between the Compute factory and the OpenFlow controller can cause a big delay in the reaction of Compute factory. Therefore, solutions about how Compute factory can overcome these delay problems should be investigated.

References

[1] Mckeown N et al. “OpenFlow: Enabling Innovation in Campus Networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (Apr. 2008), pp. 69–74.

[2] URL: `http://www.ft.com/intl/cms/s/2/b3d40e7a-ceea-11e3-ac8d-00144feabdc0.html`.

[3] URL: `http://www.geni.net/`.

[4] URL: `https://www.planet-lab.org/`.

[5] Rudolf Strijkers et al. “Internet factories: Creating application-specific networks on-demand”. In: *Computer Networks* 68 (Feb. 2014), pp. 187–198. URL: `http://dx.doi.org/10.1016/j.comnet.2014.01.009`.

[6] Supratik Bhattacharyya et al. “Pop-Level and Access-Link-Level Traffic Dynamics in a Tier-l POP”. In: *ACM SIGCOMM Workshop on Internet Measurement* (2001), pp. 39–53.

[7] Wenjia Fang and Larry Peterson. “Inter-AS Traffic Patterns and Their Implications”. In: *Global Telecommunications Conference* 3 (1999), pp. 1859 –1868.

[8] Srikanth Kandula et al. “The Nature of Datacenter Traffic: Measurements & Analysis”. In: *ACM SIGCOMM conference on Internet measurement conference* (2004), pp. 202–208.

[9] Theophilus Benson, Aditya Akella, and David A. Maltz. “Network Traffic Characteristics of Data Centers in the Wild”. In: *ACM SIGCOMM conference on Internet measurement* (2010), pp. 267–280.

[10] Sushant Jain et al. “B4: Experience with a Globally-Deployed Software Defined WAN”. In: *ACM SIGCOMM* 43.4 (2013), pp. 3–14.

[11] Trestian R., Muntean G.-M., and Katrinis K. “MiceTrap: Scalable Traffic Engineering of Datacenter Mice Flows using OpenFlow”. In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)* (2013), pp. 904–907.

[12] Srikanth K. et al. “The Nature of Data Center Traffic: Measurements & Analysis”. In: *Proc. 9th ACM SIGCOMM on Internet Measurement Conference (IMC)* (2009), pp. 202–208.

[13] Jing Liu et al. “SDN Based Load Balancing Mechanism for Elephant Flow in Data Center Networks”. In: *2014 International Symposium on Wireless Personal Multimedia Communications (WPMC)* (2014), pp. 486–490.

[14] Andrew R. Curtis, Wonho Kim, and Praveen Yalagandula. “Mahout: Low-Overhead Datacenter Traffic Management using End-Host-Based Elephant Detection”. In: *IEEE INFOCOM* (2011), pp. 1629–1637.

[15] Mogul Jeffrey C. et al. “DevoFlow: cost-effective flow management for high performance enterprise networks”. In: *Hot Topics in Networks: Proceedings of the 9th ACM SIGCOMM Workshop* (2010), pp. 1–6.

[16] Mohammad Al-Fares et al. “Hedera: Dynamic Flow Scheduling for Data Center Networks”. In: *NSDI* (2010), pp. 281–296.

[17] URL: `http://aws.amazon.com/ec2/`.

[18] URL: `http://gogrid.com/`.

[19] URL: `http://www.rackspace.com/`.

[20] URL: `http://activemq.apache.org/`.

[21] URL: `http://vtun.sourceforge.net/`.

[22] URL: `http://www.nongnu.org/quagga/`.

[23] URL: `http://www.projectfloodlight.org/`.

[24] Heller B., Sherwood R., and Mckeown N. “The Controller Placement Problem”. In: *ACM SIGCOMM Computer Communication Review* 42.4 (2012), pp. 473–478.

[25] URL: `http://openvswitch.org/`.

[26] URL: `https://en.wikipedia.org/wiki/Bellman-Ford_algorithm`.

[27] URL: `https://en.wikipedia.org/wiki/Secure_copy`.

[28] URL: `https://en.wikipedia.org/wiki/Dd_(Unix)`.

[29] URL: https://www.tno.nl/nl/.

Appendix

A Compute factory

The Compute factory is an intellectual property of TNO[29]. Therefore, the source code of Compute factory cannot be published. The code of the second control loop that is created during this research can be found on Bitbucket at `https://bitbucket.org/igrafis/rp2`.

B Create files scipt

```
#!/bin/bash

`mkdir files`
for i in {1..10}
do
        `dd if=/dev/urandom of=files/$i bs=1024 count=0 seek=\$[1024*$i]`
done
`dd if=/dev/urandom of=files/500 bs=1024 count=0 seek=\$[1024*500]`
```

C Mice flows generator script

```
#!/bin/bash

files=()
total=0
size=100

for i in {0..19}
do
```

```
        let "file=($RANDOM%10)+1"
        files[$i]=$file
        total=$((total+file))
done

if [ $total -gt $size ]
then
        change=-1
elif [ $total -lt $size ]
then
        change=1
fi

if [ $total -ne $size ]
then
        while [ $total -ne $size ]
        do
                let "pos=($RANDOM%19)"
                if [[ $change -eq -1 && ${files[$pos]} -gt 1 ]]
                 ↪  || [[ $change -eq 1 && ${files[$pos]} -lt
                 ↪  10 ]]
                then
                        files[$pos]=$((files[$pos]+change))
                        total=$((total+change))
                fi
        done
fi

for i in {0..19}
do
        `scp -v files/${files[$i]} root@172.16.0.3:~/test/ &`
        let "sleep=($RANDOM%5)+1"
        sleep $sleep
```

```
done
```

D First scenario script

```
#!/bin/bash

for i in {1..10}
do
        `scp -v files/500 root@172.16.0.3:~/test/ >>
         ↪  results/results1_big 2>&1`
        sleep 5
        `./test_small.sh > results/results1_small_${i} 2>&1`

        sleep 60
done
```

E Second and third scenario script

```
#!/bin/bash

for i in {1..10}
do
        `scp -v files/500 root@172.16.0.3:~/test/ >>
         ↪  results2/results_big 2>&1 &`
        `./test_small.sh > results2/results_small_${i} 2>&1 &`

        wait
        sleep 60
done
```