OpenFlow SDN testbed for Storage Area Network

O. Sadov, V. Grudinin, A. Shevel, D. Vlasov, S. Khoruzhnikov, V. Titov, A. Shkrebets, A. Kairkanov
ITMO University,
Russia
http://sdn.ifmo.ru

Abstract—The paper describes the testbed to determine the effectiveness of an approach to build network storage using Software-Defined networks (SDN) OpenFlow. It is assumed that main protocol to SAN is iSCSI over local area network. Prototyping tools for managing network resources and data flows on the basis of SDN and testing environments based on Free and Open Source software. We describe experiments with various modifications of OpenFlow controller NOX and set out the specifics for the use of various software and hardware OpenFlow switches. The main tests goals are Data Center SAN specific: implementation of QoS methods accordingly switchspecifics, topology changing, measuring of transmission parameters, simulating of large amount of requesting hosts (up to 100 thousands hosts).

Keywords—OpenFlow; SDN; SAN; network; NOX; QoS

I. INTRODUCTION

The aim of this work was to study the design principles and performance of Software-Defined Networks, as well as to develop prototypes of tools for managing network resources and data flows in SDN, the evaluation of the applicability of the SDN for data centers and distributed storage. For experiments were selected OpenFlow SDN and evaluated the effectiveness of their use for the management of iSCSI storage systems.

The requirements were specified for network resources management tools and Quality of Service (QoS) assurance.

II. TESTBED

A. *Software:*

- OpenFlow software switch based on CPqD/of12softswitch [1] and Open vSwitch [2];
- OpenFlow controllers based on CPqD/nox12oflib [3]and NOX [4];
- OpenFlow network emulator Mininet [5];
- VirtualBOX and KVM Virtual Machines with NauLinux 6.3/6.4 [6] distributions and Ubuntu 11.10 pre-configured CPqD OpenFlow-1.2 Virtual Machine [7].

B. *Hardware*

- OpenFlow switches – Pica8 3290 and HP 3500-24G-PoE yl.
- HP P4300 G2 7.2TB SAS Starter SAN BK716A was used as the iSCSI SAN.

III. SPECIALIZED SOFTWARE MODULES

For testing purposes was created a number of specialized Python modules and programs which used for changing of topology, QoS policies, starting/stopping of traffic generators and measuring of transmission characteristics. Developed prototypes were tailored for the hardware OpenFlow switches.

Specialized “switchqos” module was developed based on NOX module “switch” to manage network resources and data flows and to ensure QoS.

This module calculates routes for all packets in the testbed and generates flow tables for every OpenFlow switch. These calculations and flow tables modifications are performed after every topology change or data flow interruption.

The traffic classification for QoS control is based on TCP/UDP port numbers. Depending on switch type and capabilities, the different QoS control methods were used: OpenFlow queues, IP ToS, and VLAN PCP modifications.

Special software tools for QoS policy configuration of hardware switches were used to prioritize SAN traffic. The different switches (for example, Open vSwitch and HP ProCurve) had different QoS control mechanisms, which made the creation of a unified interface is quite a difficult task.

As the most important configurable parameters of QoS assurance, the bandwidth and the priority of the packet queues were selected.

The software prototypes for QoS control on HP 3500 and Pica8 in OVS mode were placed in the repository [8]. They can be easily extended to use different QoS settings.

Because different switches and controllers support variety versions of OpenFlow, several different modules were developed for NOX classic [8], NOX [4] and nox12oflib [3].

IV. NETWORK RESOURCES AND DATA FLOWS MANAGEMENT

As a system for network resources and data flows management, a set of software modules was developed for attaching and detaching links between switches.

In the emulation mode, this was carried out by means of Mininets Python module.

For hardware switches this was done via CLI commands over SSH connection, automated by a Python script.

978-1-4799-7595-2/14/$31.00 ©2014 IEEE

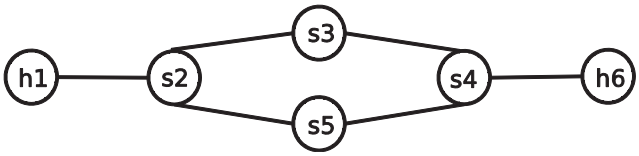

Fig. 1. Loop topology for experiments

A loop topology (Fig. 1) was selected for experiments, consisting of 4 switches (nodes s2, s3, s4 and s5), and two hosts for traffic generation and reception (nodes h1 and h6).

SDN routing modules based on standard regular MAC-learning NOX "switch" modules.

During the experiment, test traffic (ping) was sent from the host h1 to host h6. In an initial state all nodes were connected accordingly Fig. 1. The controller was in an undefined state, it had no routing scheme, and the packets have not passed. After detaching one link by test framework, the route was constructed by NOX "switchqos" module, and the pass of the packets was established. After that, the restoring of the detached link (and loop) did not break the traffic flow. Detaching the active link led to an automatic topology rediscovery and redirection of the traffic to a different route.

V. QOS ASSURANCE METHODS

Data flows prioritization was carried out with the Python modules. These modules set bandwidth for OpenFlow queues or ToS/PCP bandwidth. The dpctl utility was used for the software switch control. Hardware switches were managed by CLI commands sent over SSH.

For the evaluation of a possible use of SDN in data center, a data center model (Fig. 2) was created. This model consisted of iSCSI SAN and few VMs. The first VM acted as an OpenFlow 1.2 switch while the second one generated iSCSI traffic; the others performed in generating and receiving the load traffic.

During the experiment, the data were read from iSCSI SAN with simultaneous load traffic generation.

IP diagnostic utility Iperf and VoIP test tool SIPp were used to generate the load traffic.

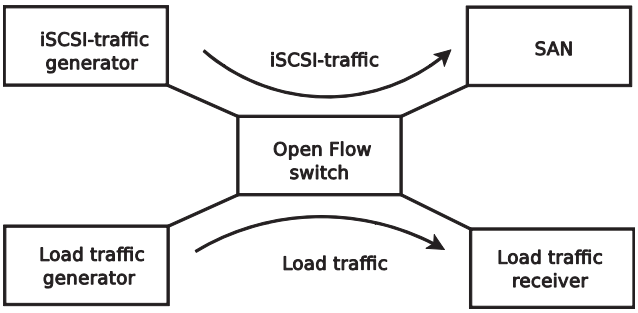

Fig. 2. SDN data center model

It was observed that under heavy load condition the iSCSI connectivity might be lost and later recovered. After iSCSI connectivity recovery the bandwidth is changing in arbitrary manner. To keep the same bandwidth after recovery we changed Linux Traffic Control dynamic bandwidth, which is defined by CpqD/of12softswitch, to static bandwidth setting. The modified module can be found in [1].

The utility dpctl sets the share of total bandwidth for selected QoS queues as percent of total bandwidth. The sum of shares is not necessary equal to 100.

Table I shows the influence of the presence of queuing on the resulting SAN I/O speed, but there is a little difference.

The experiment with the HP hardware switch has shown a correlation between the bandwidth share set and the resulting I/O speed (Fig. 2).

TABLE I. SAN I/O SPEED THROUGH SOFTWARE SWITCH DEPENDENCY ON QOS QUEUES BANDWIDTH SHARE

Bandwidth share, in % of the total		**Load traffic, Kb/s**
SAN I/O speed	***iSCSI traffic***	
100	0	35.1
100	0.1	31.6
100	100	8.3
0.1	100	5.4
0.1	0.1	9.2

TABLE II. SAN I/O SPEED THROUGH HARDWARE SWITCH DEPENDENCY ON QOS QUEUES BANDWIDTH SHARE

Bandwidth share, in % of total		**Load traffic, Mb/s**
SAN I/O speed	***iSCSI traffic***	
100	0	10.0
80	20	8.4
20	80	2.1
0	100	0

Not comparing the absolute transmission rate, it is possible, due to a priori restricted channel throughput, to specify the advantages of QoS control in hardware switches: a high degree of accuracy, an impossibility of setting a total bandwidth more than 100%.

VI. PROCESSING A LARGE NUMBER OF REQUESTS

In the test program (rd test) SCSI command "TEST UNIT READY" was sent to SAN in multithread mode via ioctl system call with SG IO code. The target characteristic was the number of completed requests for a selected period of time.

The developed "switchqos" module was optimized for speed of transmission of data passing through controlled switches. This optimization included a modification of the default NOX flow matching scheme. It was necessary because the used switches were unable to perform a flow match based on source and destination MAC addresses and VLAN PCP with the hardware acceleration. The software processing was limited to 10 000 packets per second.

Another setting was in increasing the idle timeout. It was found during the experiments that HP 3500 switch had not refreshed the flow packet statistics frequently enough for the hardware processed flows. Usually, after 5 seconds of idle time (default for NOX "switch" module), the switch erroneously removed the record from the flow table. After increasing the idle time parameter in "switchqos" module to 20 seconds, this behavior was corrected and the necessity for repeatedly creating records of flow matching was eliminated. At the same time, an excessively large idle timeout value could degrade the performance due to an increased flow table size.

After these optimizations, the performance of the system increased significantly, and the value of 100 000 requests to SAN per second through OpenFlow switch was surpassed. The example of test program output is shown below.

```
# ./rd_test /dev/sdb 2 100
```

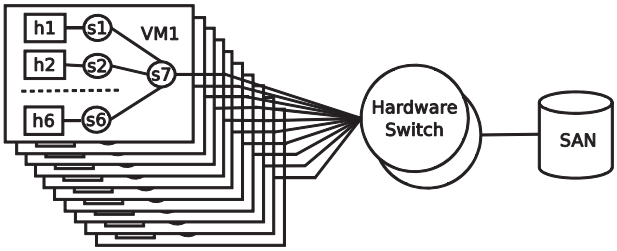

Fig. 3. Modeling the large number of requests to SAN in data center infrastructure

```
Result: 130124 requests/sec (260248/2)
```

VII. SAN RESPONSE TIME

Read operations were used to measure SAN response. SG IO ioctl was used to exclude the buffering influence, instead of the generic read.

The test program has measured the average latency and jitter performing SAN requests.

The results for 1000 packets and data block sizes 512 and 1024 bytes are as follows (the average latency and jitter are measured in seconds):

```
# ./rtt_iscsi_read /dev/raw/raw1 1000 \ 512 1024

Size=512 Packets=1000 Latency=0.000844 Jitter=0.000084

Size=1024 Packets=1000 Latency=0.000860 Jitter=0.000104
```

VIII. DATA CENTER MODELING

Our modeling of a data center involved a transmission of ICMP requests to SAN from different MAC addresses.

The test network consisted of SAN, 2 hardware OpenFlow switches from HP, VM with NauLinux 6.3 guest OS running NOX and 10 test nodes VMs running Ubuntu 11.10 and Mininet. Each test node launched 6 virtual hosts, 7 software switches Open vSwitch, and a local controller NOX (Fig. 3).

The test program, running on the main host, sent messages to the test nodes, starting local test programs, written as xinetd services. The local test programs on every virtual host pinged SAN from every MAC address in a specified range. Requests were forwarded to SAN through hardware switches, controllable by NOX launched in multithread mode (10 threads) on the main host. This controller instance has logged the number of different MAC addresses in the processed requests and the requests distribution in the running threads. After getting 100 000 different MAC addresses, test programs stopped.

IX. CONCLUSION

Described experiments have shown that developed OpenFlow testbed could be used for testing the dynamic (re)configurations of the network elements, (re)setting various data transfer parameters for different traffic types. It was shown the testbed is able to serve the requests from large number of hosts. Suggested inexpensive testbed might be used for detailed investigation of OpenFlow approach to the network architecture of data centers and distributed storage.

Software repositories [1], [3] and [4] contain developed software modules and tests. The controller applications are packaged in binary and source forms for NauLinux operating system distribution [6], binary compatible with RHEL/Oracle Linux/CentOS/Scientific Linux distributions.

REFERENCES

[1] ITMO OpenFlow 1.2 software switch repository, avail-able at https://github.com/itmo-infocom/of12softswitch

[2] Open vSwitch project, available at http://openvswitch.org

[3] ITMO OpenFlow 1.2 NOX repository, available at https://github.com/itmo-infocom/nox12oflib

[4] ITMO NOX repository, available at https://github.com/ itmo-infocom/nox

[5] Mininet, available at http://mininet.github.com

[6] NauLinux distribution, available at http://downloads. naulinux.ru/pub/NauLinux/

[7] CPqD OpenFlow-1.2-Tutorial, available at https:// github.com/CPqD/OpenFlow-1.2-Tutorial/wiki

[8] ITMO OpenFlow tests repository, available at https:// github.com/itmo-infocom/of-tests