A Priority Based Virtual Network Bandwidth Guarantee Method in Software Defined Network

Qu Fu, Liao Qing, An Yingzhu, Fan Yamei
School of Information and Communication Engineering
Beijing University of Posts and Telecommunications
Beijing, China
{yikoudreams, liaoqing, await12an, fanyamei }@bupt.edu.cn

Abstract—**Virtual Network (VN) bandwidth guarantee is an important issue in scenarios such as enterprise network and multi-tenant datacenter. The Software Defined Network (SDN) is sophisticated technology for network control and management. In this paper we proposed a VN bandwidth guarantee method which use SDN controller and Open vSwitch to allocate the available bandwidth by VN's priority. According to simulation results, the proposed method can effectively guarantee the bandwidth for VNs while maximizing the bandwidth utilization.**

Keywords-virtual network; bandwidth guarantee; SDN; Open vSwitch

I. INTRODUCTION

Virtual networks have been greatly improved due to the large demand from enterprise network and multi-tenant datacenter. And concerned technologies are developed rapidly from VLAN to VXLAN (virtual extensible local area network) [1] and NVGRE (network virtualization using general route encapsulation) [2]. These technologies support the VNs to be logically insulated from each other which means more security and efficiency. However, the traffic form each VN still runs on the same physical devices, so they will compete for the bandwidth leaving the VN's bandwidth unguaranteed. SDN is the appropriate technology for this problem.

SDN is a network paradigm that provides for separation between the control and forwarding planes [3], [4]. A centralized SDN controller communicates with the forwarding equipment via a south-bound interface, which can be proprietary or public (e.g., OpenFlow [5]). And Open vSwitch is a kind of popular forwarding equipment, which supports various protocols on the south-bound interface and works well both on the virtual or real device. In this paper, we propose a novel approach which combines the SDN controller and the Open vSwitch to solve the VN's bandwidth guarantee problem.

The paper is organized in the following manner. Section 2 gives an introduction to the system model and the concerned components such as RYU (one of the most popular SDN controller), Open vSwitch and its HTB queuing technique. And it also describes a typical model about our topic. Section 3 presents our new VN's bandwidth guarantee method with concerning both the VN's priority and the total bandwidth utilization. In section 4, we briefly discuss the methodology for our experiment design, and the test result is given to show the effectiveness of our method. We finally conclude our discussion and show the future work.

II. SYSTEM MODEL AND COMPONENTS

A. System Model

Fig. 1 shows the typical SDN framework which contains both the control plane (RYU) and the data plane (Open vSwitch and computers). The largest bandwidth between br0 and br1 is constant, and if the VNs are all overloaded they will compete for the bandwidth which will make some key traffic without QoS.

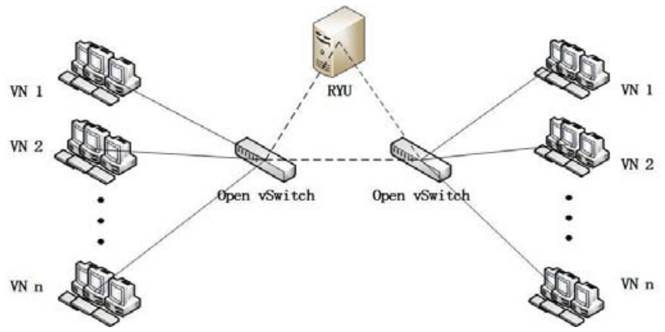

Fig. 1. System model.

B. RYU Controller

RYU is a powerful SDN controller developed by python which supports the OpenFlow protocol from version 1.0 to 1.4 [6]. One big advantage of RYU is the compendious software architecture. And it is very friendly to developers by providing many simple and useful APIs.

Our approach will use the OpenFlow protocol with version 1.3. The protocol's version 1.0 and version 1.3 are both widely supported and used. The version 1.3 was released in 2012. And it supports more features than version 1.0 such as vlans, logic ports, tunneling and per-flow traffic meters which means this version has the ability to handle both the VN and the QoS.

978-1-4799-8353-7 /15/$31.00 ©2015 IEEE

C. *Open vSwitch*

The Open vSwitch is an open source multilayer switch software which supports OpenFlow perfectly and works greatly in virtual environment [7]. The Open vSwitch contains three components: ovs-vswitchd, ovsdb-server and openvswitch_mod.ko as shown in Fig. 2. The ovs-vswitchd is a daemon used for querying the ovsdbserver to get the switch's configuration and communicate with both the controller and the kernel openvswitch_mod.ko module. The ovsdbserver is a database that contains the configuration of the switch. The openvswitch_mod.ko is the kernel module which supports datapath to transport packets. Open vSwitch supports many features that real switch does, such as VLAN trunking 802.1Q and Link Aggregation Control Protocol (LACP). Especially, Open vSwitch has QoS features—the Hierarchical Token Based (HTB) queuing technique.

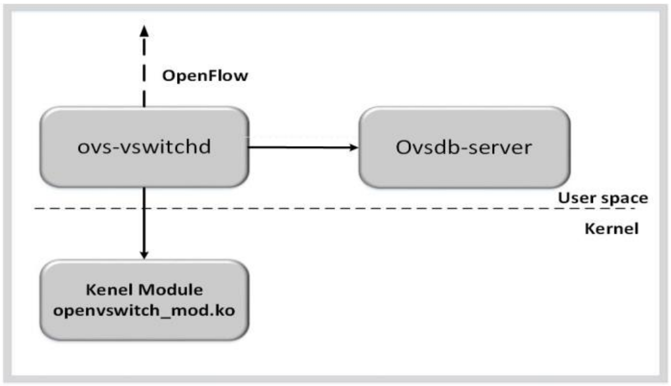

Fig. 2. The Components of Open vSwitch

HTB is developed as a replacement for the Class-Based Queuing (CBQ) technique [8]. It can be used to control the outbound traffic of a switch's port. When HTB is in use, every queue of the switch's port should be configured with a lower bound and a higher bound of available bandwidth. HTB allows us to allocate minimum bandwidth to every queue when the whole traffic is overloaded and to allocate the spare bandwidth to the VNs by its priority which means we can maximize the utilization of the total bandwidth.

III. THE PROPOSED BANDWIDTH-GUARANTEED METHOD

In this section we propose a new bandwidth guarantee method based on VN's priority meanwhile the method can maximize the bandwidth utilization in SDN. There are two steps. The first step is that the queue on the concerned port should be correctly set according to each VN's priority. The second step is that a corresponding app should be developed to handle the packets from different VN on the RYU controller.

A. *Set the Queue by the Priority*

The system model is shown in Fig. 1 already. And There are N VNs in the system VN 1, VN 2, ..., VN N. All the VNs are connected through the two Open vSwitch Br 0 and Br 1. The total available bandwidth between the Brs is B_ttl. The two Brs are also connected to the pc where the RYU controller is installed.

The priority of each VN is a number to indicate the amount of bandwidth that it may be allocated. And the larger the number is, the more bandwidth it will get. We use Pr1, Pr2, …, PrN to present each VN's priority. The summation of the priority parameter can be got from (1).

$$\Pr\_sum = \sum_{i=1}^{N} \Pr i \qquad (1)$$

B1, B2, …, BN denote the minimal bandwidth of each VN. From the summation of the priority parameter, each VN's priority parameter and the total available bandwidth, we can get Bi from (2).

$$Bi = B\_ttl * {\Pr i}/{\Pr\_sum} \qquad (2)$$

Then we configure the queue on the port with the lower bound of the bandwidth to be Bi and the higher bound of the bandwidth to be B_ttl. We assume that the switches use port eth0 to connect to each other. Then the configure command is shown as follows:

```
ovs-vsctl set port eth0 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=B_ttl queues=1=@q1,2=@q2,...,i=@Bi,...,N=@qN

-- --id=@q1 create queue other-config:min-rate=B1 other-config:max-rate=B_ttl

-- --id=@q2 create queue other-config:min-rate=B2 other-config:max-rate=B_ttl

……

-- --id=@qi create queue other-config:min-rate=Bi other-config:max-rate=B_ttl

……

-- --id=@qN create queue other-config:min-rate=BN other-config:max-rate=B_ttl
```

This command creates N queues on the port eth0. And each queue has a lower bound and a higher bound of available bandwidth.

B. *Control the Flows by the RYU*

First of all, the Brs should be configured to have the RYU controller as its remote controller with the command:

```
ovs-vsctl set-controller 'Br-name' tcp:'ip-addr':'port-num'
```

And then the RYU controller and the Open vSwitch will send hello message to each other, check the configuration and find out the latest OpenFlow protocol version that supported, as shown in Fig. 3. Then the RYU controller begin to take charge of the Open vSwitch's forwarding behavior by sending corresponding message such as packet-out message and flow-mod message.

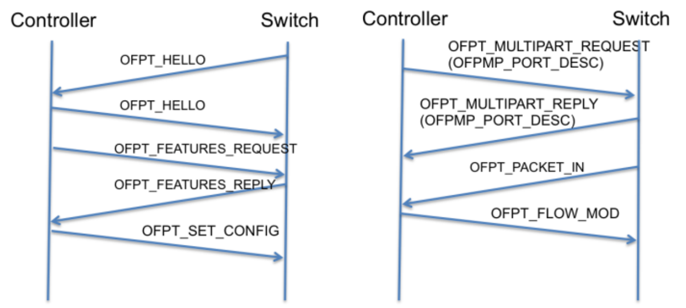

Fig. 3. The messages exchanged between the controller and the switch

The RYU controller can handle the first few messages in HANDSHAKE_DISPATCHER stage and DEAD_DISPATCHER stage by the build-in OFPHandler module. But the messages in CONFIG_DISPATCHER stage and MAIN_DISPATCHER stage should be handled by the app designed by the developers with their purpose. As a result, we designed our app with a class called VNBG which means 'virtual network bandwidth guarantee'.

In VNBG, we realized many functions such as the set_feature_handler function which configures the switch to send the packets to controller when they don't match any flow table, the _packet_in_handler function which judges the packet feature to take different movement and other functions to make sure the VN isolation, etc.

The _packet_in_handler is the most important function to support the bandwidth guarantee. In this function we have realized the logic as follows:

- Get the in_port (switch input port), eth_dst (ethernet destination address), eth_src (Ethernet source address) and vlan_vid (VLAN id) of the packet from the msg.match module of RYU.
- Learn the relationship of the eth_src and the in_port in VN's dictionary to avoid flood next time.
- If the eth_dst is not in dictionary then make a packet-out message to the switch to flood the packet in the VN.
- If the eth_dst is in dictionary and the port isn't eth0 then make a regular pop-vlan flow-mod message to the switch to install a flow table to avoid pack-in message next time.
- If the eth_dst is in dictionary and the port is eth0 then make a set_queue & push-vlan flow-mod message to the switch to install a flow table to forward the matched packets to the corresponding queue of eth0.

The last situation is when the bandwidth guarantee method works. The installed flow table has the instruction due to the parser's OFPInstructionAction function: inst=[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ ACTIONS, actions)] and the action set due to the parser's OFPActionSetQueue function: actions=[parser.OFPActionPushVlan(0x8100), parser.OFPActionSetField(field), parser.OFPActionSetQueue(queue),parser.OFPActionOutput(out_port)].

IV. Experiment Validations

The goal of this section is to implement and test our bandwidth guarantee method in section III. However, we have simplified the system architecture in Fig. 1.

As shown in Fig. 4, our experiment environment contains two servers, each of which has an Open vSwitch and 3 VMs (virtual machines) managed by KVM in CentOS 7, and one pc where our RYU controller is installed. In the system, three VNs are set with each of them has two VMs that on both Br0 and Br1.

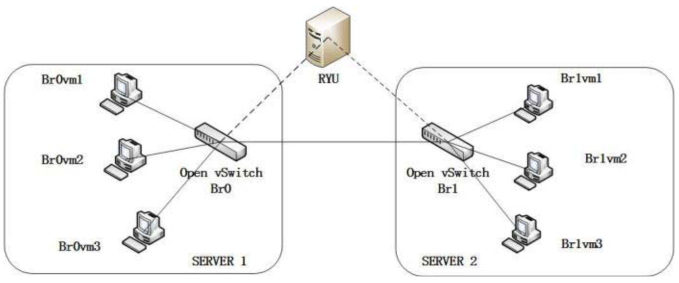

Fig. 4. The simplified system architecture

In our experiment we use VLAN to be the virtual network technology. The VLAN tag set on the Brs will do not work when there is a controller. The RYU will control the switch to push-vlan or pop-vlan according to the feature of the flow.

The total bandwidth B_ttl between the Brs is 100Mbits/sec.

The VNs' features is shown in Table 1.

TABLE I. THE VNS' FEATURS

VN	*Pri*	*Bi Mb/s*	*VM 1*	*VM 2*	*Tag*	*Que*
1	1	10	Br0vm1	Br1vm1	2	0
2	3	30	Br0vm2	Br1vm2	3	1
3	6	60	Br0vm3	Br1vm3	4	2

In our experiment, we use the Iperf [9] traffic generator to measure the usable bandwidth for each VN in every second. Iperf is installed in all six VMs. And we will start Iperf in VMs on Br1 acting as servers and Iperf in VMs on Br0 acting as clients. When the Iperf server and client are working, the bandwidth will printed out to the specified file for analysis.

We design a schedule that will fully show the changes of the network traffic and how the approach works. First, we generate a traffic on VN 1. After 15 seconds we generate a traffic on VN 2 to compete for the bandwidth. And after another 15 seconds we generate a traffic on VN 3 to make the network fully overload. All these traffics will persist for 45 seconds.

And the result is shown in Fig. 5.

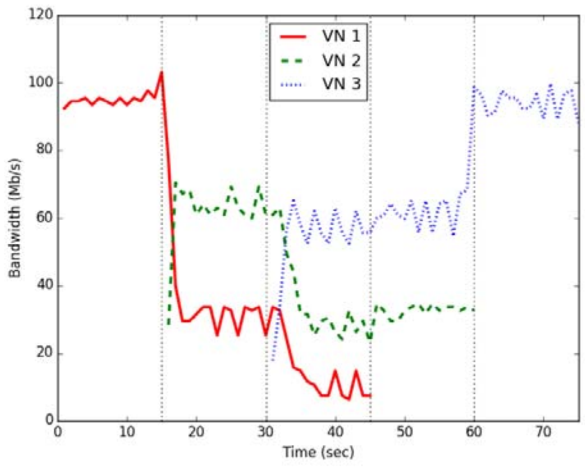

Fig. 5. The experiment result

Now, we analyze the result by 5 stages as follows:

- The 1st stage: The traffic of VN 1 is the only traffic through eth0, so it will get a bandwidth of nearly 100Mb/s (however, it can't really get the whole bandwidth because there are OpenFlow, SSH and other protocols in the background, and other measured number below will all a little smaller than its theoretical value) .
- The 2nd stage: The traffic of VN 2 is started, so the two traffic will get its bandwidth by their priority which means that the VN 1 will get a bandwidth of 25Mb/s and the VN 2 will get a bandwidth of 75Mb/s.
- The 3rd stage: The traffic of VN 3 is started. Likely, the VN 1 will get a bandwidth of 10Mb/s, the VN 2 will get a bandwidth of 30Mb/s and the VN 3 will get a bandwidth of 60Mb/s.
- The 4th stage: The traffic of VN 1 is stopped. Likely, the VN 2 will get a bandwidth of 33.3Mb/s and the VN 3 will get a bandwidth of 66.7Mb/s.
- The 5th stage: The traffic of VN 2 is also stopped. So the VN 3 will get a bandwidth of 100Mb/s.

From Fig. 5 and analysis above, we can firmly believe that our virtual network bandwidth guarantee method works effectively.

In Fig. 6, we show the summation of all the measured bandwidth of the different VN's traffic on every second. We can find out that the summation is always nearly the total available bandwidth.

The three unusual point around 15s, 30s and 60s is because that the alignment of the measured data is not quite accurate resulting from the time between when the traffic of VN i is started is not exactly 15 seconds. However, it still illustrate that our method can always maximize the utilization of the total bandwidth.

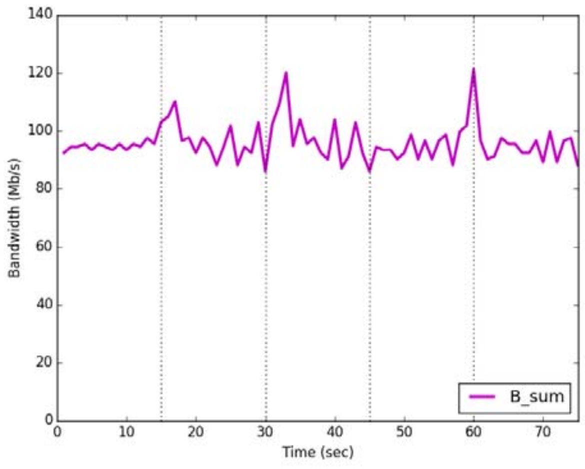

Fig. 6. The summation of the bandwidth of traffics

V. Conclusion & Future Work

In this paper we explore a new virtual network bandwidth guarantee method. In the method, we introduce the approach to configure the QoS queues on the concerned port of the Open vSwitch and the way to develop a VNBG app on the RYU controller. The experiment results demonstrate that our virtual network bandwidth guarantee method shows good performance for the VNs with different priorities, and it can maximize the utilization of the total bandwidth. In our future study, we will use our method in a more complicate system with the VXLAN and develop the app that support it. On the other hand, we will explore the bandwidth guarantee method with the use of meter tables of the OpenFlow protocol 1.3.

References

[1] MAHALINGAM M, DUTT D, DUDA K, et al. VXLAN: a framework for overlaying virtualized layer 2 networks over layer 3 neworks[EB/OL]. http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-02.

[2] MAHALINGAM M, DUTT D, DUDA K, et al. NVGRE: network virtualization using generic routing encapsulation[EB/OL]. http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-01.

[3] F. Durr, "Towards cloud-assisted software-defined networking,"" Technical Report 2012/04, Institute of Parallel and Distributed Systems, Universitat Stuttgart, Tech. Rep., 2012.

[4] M. Mendonca, B. N. Astuto, X. N. Nguyen, K. Obraczka, T. Turletti et al., "A survey of software-defined networking: Past, present, and future of programmable networks," 2013.

[5] N. Mckeown et. al., OpenFlow: Enabiling Innovation in Campus Network, ACM SIGCOMM, 2008.

[6] "RYU" [Online] Available: https://github.com/osrg/ryu/.

[7] "Open vSwitch" [Online] Available: http://openvswitch.org/

[8] M. Devera, "HTB Linux queuing discipline manual - user guide." [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.html.

[9] "Iperf" [Online] Available: http://iperf.sourceforge.net/.