Scalable and Flexible OpenFlow-based Distributed Rate Limiting

Taku Fukushima

Thesis submitted in partial fullfillment of
the requirements for the degree of

Master in Information and Computer Science

Student ID	5110B105
Submission Date	January 31th, 2012
Supervisor	Professor Tatsuo Nakajima

Department of Computer Science
Faculty of Science and Engineering
Waseda University

2011年度

修士論文

Scalable and Flexible OpenFlow-based Distributed Rate Limiting

早稲田大学基幹理工学研究科
情報理工学専攻

福島 拓

学籍番号	5110B105
提出	2012年1月31日
指導教員	中島達夫教授

Abstract

Explosive growth of the new style computers such as cellular phones, smartphones and tablets lead the strong demand for the large scale internet serivces, which are often taking an important role in the society. The massive scale datacenters which comprise tens or hundreds of thousands commodity computers achieve these services automating the dynamic management of the computational resources by the virtualization technologies and distributed systems. This cloud computing paradigm provides agility and cost effective solutions to users.

Although the operations on computational resources are highly automated, the network facilities are still managed manually since they do not provide flexible interfaces. In addition the limitation of existing network solutions prevent building the massive scale network distributed in regions in cheap price. To solve the problems network virtualization with software-defined network (SDN) is emerging. It provides programmable interfaces to the network facilities and enables dynamic deployment of the virtual machine (VM) instances.

On the other hand, distributed rate limiting (DRL) was proposed to control the traffic of cloud services distributed in racks, datacenters and regions. DRL paradigm enables cloud service providers to provision the traffic on the multi-tenant environment in the fair manner and introduces new billing system on the network loads. However the DRL requires the configurations and implementation of rate limiting and consensus agreement algorithms to switches, which need to be operated manually.

In this thesis we propose the DRL system on the virtualized network to achieve fully automated network operations. Our approach contributes to the realization of the more cost effective, scalable and flexible datacenter network and enhances the agility on the deployment of network facilities.

概要

携帯電話，スマートフォンそしてタブレットなどの新しい形のコンピュータの爆発的普及は大規模 Web サービスへの要求を強めており，それらの Web サービスはしばしば社会において重要な役割を担っている．それらのサービスを支える大規模データセンターは数万，数十万台のコモディティコンピュータから構成されており，仮想化技術や分散システムによって計算資源の動的な管理を自動化している．このクラウドコンピューティングというパラダイムはユーザに安価で即時的な計算資源を提供している．

しかし計算資源に関するオペレーションは高度に自動化されているものの，柔軟なインタフェースが提供されていないためにネットワーク設備の管理はは未だに人の手によって行なわれている．加えて既存のネットワーク機器の制限が分散した安価な大規模データセンターの実現を妨げている．これらの問題を解決するために Software-Defined Netowrk によるネットワーク仮想化が登場してきた．ネットワーク仮想化はネットワーク機器に対するプログラマブルなインタフェースを提供し Virtual Machine インスタンスの動的なデプロイメントを可能にする．

一方，複数の場所に分散するクラウドサービスのトラフィックを制御するために Distributed Rate Limiting (DRL) が提案された．DRL によってクラウドサービスプロバイダは公平な立場でマルチテナント環境のトラフィックのプロビジョニングを行なったり，ネットワークの使用量に応じた課金を行うことができる． しかし，DRL は帯域制御アルゴリズムやコンセンサスアルゴリズムの実装および設定を手動で行なう必要がある．

本論文で我々は仮想ネットワーク環境で完全に自動化された DRL システムを提案する．我々の提案手法は，より安価で柔軟な大規模データセンタおよびネットワーク機器のデプロイメントのアジャイル化に貢献することが期待される．

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Goals of This Thesis

The goals of this thesis are the following three points. First, we propose the design of the rate limiting methods on the virtualized network topology, which is scalable and flexible for the modern demands on the large scale datacenter. Second, we present the proof-of-concept implementation of the proposed rate limiting design. Through this implementation, we lead to clarify the design issues to build the rate limiting functionalities in the distributed environment. Finally, we evaluate it to show how the issues effect on the performance and what the best practice to manage the rate limiting on the real world is.

1.2 Contents of This Thesis

Following this introduction section, this thesis is organized in the six sections.

Chapter 2 presents the abstractions of the virtual resources, which decouple the virtualized network topologies from the physical ones, and the virtual bandwidths which are parts of the virtual resources which represent the bandwidth assigned to the tenant. Then we also depict the design of the rate limiting on those distributed and virtualized networks.

Chapter 3 shows the implementation of the formentioned rate limiting and the system overview.

Chapter 4 introduces the evaluation for the rate limiting and describes the issues which we experienced.

Chapter 5 gives the discussions based on the design, implementation and evaluation providing the possibility for future work.

Chapter 6 states the previous works related to this thesis.

Chapter 7 concludes this thesis with general descriptions.

1.3 Background

This section describes about the background of the distributed rate limiting (DRL) on the distributed and virtualized network topology, which is the main theme of this thesis. Cloud computing requires large scale datacenter networking infrastructures and they are suffed from lack of the scalability, inflexibility and the high cost operation. Software-Defined Network achieves programmable network at the lower cost and it enables virtualized network which is able to decouble the virtualized network topology from the physical one. Through virtualized network we can manage the network between virtual machine (VM) instances in a flexible manner. In that environment we need to manage VMs of the tenants distributed in the different network segments. For the cloud providers it is crucial to assign the fixed size bandwidth to the tenants based on the service level aggreement (SLA) in terms of the billing on them and carving the limited network resources up in the fair behavior.

1.3.1 Explosion of Internet Services

Due to increase of cellular phones, smartphones and tablets various people use Internet services and we experience the explosion of the traffic because the services serve hundreds of millions contents such as text messages, large data size images or movies in the realtime behavior and the number of their users is rapidly growing. These services such as web search, video sharing and social networking are required to be able to deal with millions queries a day or tens of petabytes data a day, and running 24/7.

1.3.2 Cloud Computing and Datacenters

Large scale cluster based datacenters which are able to provide the massive computation and storage resources are strongly demanded to achieve such rapidly growing Internet services. In the point of view from the small or medium companies, they have to prepare overprovisioned resources and to have their own datacenters take much cost. And it is difficult to predict the flush crowd or the epidemic of the users of the services.

Some companies provides their infrastructures as utilities to users not only use them by themselves [1, 2, 8, 6]. Users never care about the underlying infrastrucrues such as spaces for physical servers, network facilities and power management units because resources are abstracted and exposed to users via services or APIs. Such cloud service providers charge on the resources in the pay-as-you-go model and that frees users from overprovisioning for the unpredictable demands for their services.

In the context of the computer science datacenters are also referred as “warehouse-scale computers” because a datacenter can be considered as a single computer system [19]. Their workloads are mainly comprised collections of distributed systems and thousands of nodes, which are physical machines or VMs, dedicate to the computation intensive tasks. These distributed tasks are also heavily network intensive because nodes need to communicate with each other through remote procedure calls or the master-worker model.

1.3.3 Datacenter Networks

The conventional datacenters used to be equipped with tens of thousands servers connected with each other and they form the typical tree topology [9, 17]. Each rack has tens of servers and they are connected to the Top-of-Rack (ToR) switch. Each ToR switch is aggregated to the aggregation switches. And then each aggregation switch is converged to the datacenter switches connected with the Internet or other datacenters and highly over-subscribed with some aggregate switches. These over-subscribed switches have many 10GbE ports and tends to be expensive than the commodity 1GbE switches [9].

In such sort of typical network topology tenants hold VMs which share the network facility on the host machines. To keep tenants secure and seperate from other tenants virtualized isolation between tenants on top of the physical network is achieved by VLAN. However, VLAN experiences three major problems for the cloud providers.

First, VLANs have their limitations for the number of nodes which they can manage depends on the length of the 12 bit VLAN identifier, which is namely 4096 nodes. Those limitations are too small to assign for tens of thousands VM instances on the massive scale datacenter and manual configurations for the segments are required. To solve this problem of the scalability VXLAN [7] extending the VLAN with L2 tunneling over L3 and tenancy information was designed. But it is currently a RFC draft of the IETF and does not solve the following two problems completely.

Second, nodes owned by different tenants are not isolated completely because tenants can snoop the traffics on the same LAN where other tenants reside. Although communications in the same LAN can be encrypted, senders and receivers should be care about it, and that means cloud providers need to prepare instances which have the encryption mechanisms or tenants have to protect their traffic data by themselves. That would increase the cost of engineering for cloud providers and tenants consequently would have to pay for it.

Finally, when we need VMs to be migrated between the different segments we have to reconfigure network topology and reassign the IP address to the VMs. This degrades the agility of the deployment of the network facilities that raises the operational cost [9].

To deal with these problems of the datacenter networks the alternative network topology with the commodity switches were proposed [9, 18]. These designs offer advantages such as lower cost, lower latency, lower energy consumption, wasting fewer heat from switches, fault tolerance, scalable and flexible routing. In other words, by scaling out the network facilities on the datacenter network we can build the datacenter grade networks at the lower cost with commodity switches and multipath routing using load balancing mechanisms. We mention few solutions to realize such commodity switch-based datacenter network in the following 1.3.4 and 1.3.5.

1.3.4 Software-Defined Network

To solve the static and inflexible nature of the network falicicities the idea of the software-defined network (SDN) emerged. SDN enables us to program the behavior of the network. OpenFlow [22] is a one of the specification of the SDN and it seperates datapaths, where data go through, from control planes, which manage the actions bound to each flow. Because of this simple but powerful specification, we can achieve the following three contributions to the datacenter networks:

1. With OpenFlow we can manage the functions of the switches and extend the algorithms for the load balancing dynamic routings. Since they are not hard-coded in the firmware, they can be changed immediately in any forms when you wish.

2. It is able to transform the commodity servers into the commodity switches, and that doesn't take any costs without NICs and servers itself.

3. Some software switches based on the OpenFlow protocol which datapaths are the kernel module and balanced their flow processings by its IRQs among cores, which means they are multi-core available. In the future of multi-core era, this matches with the demands on the consolidating switches into the servers.

These properties drive us to realize the datacenter network aforementioned in Chapter 1.3.3. And it is able to change network policies depends on the network loads dynamically, which leads us to build more flexible networks than hardware based static networks.

1.3.5 Network Virtualization

SDN enables us to achieve the dynamic networking functionality based on the situations of the networks. Using SDN it is also available to provide the abstractions to decouple the interfaces from their underlying physical networks, which we call network virtualization. The pairs of the names and the locations are determined dynamically, and the communication between the different datacenters are multiplexed by the equal cost multi paths. The packets go through these paths are encapsulated and forwarded in the appropriate route without interfereing. From the perspective of the tenants VM instances in the different regions are connected on the same large LAN.

In addition we can manage firewalls, load balancers and VPNs as softwares built on the virtualized networks because they cloud be deployed automatically adapting the situation of the network loads or the demands of tenants.

On such virtualized network topologies we don't need to care about the physical topology among the different locations connected with the Internet. Because of this abstraction we can migrate VM instances from a datacenter to another one without changing of the IP addresses. That drives the agility of the deployment of the instances because they are automated and able to be manupluated from the outside of the datacenters through some interfaces such as browsers or Web APIs. It also enhance the high availability to avoid failures on the physical infrastructures caused by the hardware errors, management errors and disasters such as earthquakes or hurricane.

In this section we described about the network virtualization and its importance. In the virtualized network environment multi-tenancy is the crucial feature for the cloud providers. In the next section we state the one of the problem in such environment and show the solution for the problem which is the theme of this thesis.

1.3.6 Distributed Rate Limiting and QoS on the Virtualized Network

Rate limiting on the cloud infrastractures of datacenters is required in the industry because of the demand on performance isolation between tenants and protecting well-behaved tenants from malicious tenants [24, 26]. Current cloud providers do not provide the explicit charges on the network bandwidth, and tenants are not guaranteed with the network performance [27]. Therefore well-behaved tenants can be effected by the malicious tenants who dominate the network bandwidth and that is not fair for the well-behaved tenants. This unfairness can drop the reputation of the cloud providers.

On the virtualized network the network topology can be distributed in some racks, datacenters or regions and in that situation rate limiting also should be distributed. But rate limiting on the virtualized network is a challenging in technical as following three reasons:

1. DRL should deal with the joininig and withdrawing of nodes automatically and assign appropriate max rates for each node
2. The consistency of the allocated maximum rates among the instances assigned for a tenant should be kept.
3. The rate limiting should be work-conserving, which means if the traffic on the switch shared with some instances enough smaller than the bandwidth of the backplane instances should be allowed excessive bandwidth and "borrow" the bandwitdh from other tenatns on the same physical links.
4. The rate limiting systems should be fault torelant because if the rate limiting systems for a tenant fail down other tenants will be effected and that would cause the failures of the services.

To achieve the DRL on the virtualized network we propose the abstraction *virtual resources* which represents the network resrouces assigned to tenants in Chapter 2.2. Conducting with the cloud providers tenants will be given some virtual resources that stand for the summation of their allocated network resouces assigned to each instance. As a part of virtual resouces tenants will be guranteed the bandwidth they can use among their instances, which is referred as *virtual bandwidth* in the thesis. Based on these virtual bandwidths we collect data about the instances of tenants and calculate the appropriate maximum rates for each instance.

Chapter 2

Design

This chapter shows the design of our proposing rate limiting system on the virtualized network to meet the demand described in the previous chapter. First, we state about the system overview and then we will propose the virtual resources which are the abstraction for the network resources assigned to tenanats and the virtual bandwidth which are parts of the virtual resources.

2.1 System Assumptions and Requirements

The rate limiting has its constraint that the aggregated bandwidth of all instances managed by the rate limiter never exceeds the bandwidth assigned to the tenant, which is

$$\sum_{i \in \mathbb{I}} r_i \leq B_{virtual} \tag{2.1}$$

where $\mathbb{I}$ is the set of instances assigned to a tenant, r_i is the maximum rate assigned to the instance and $B_{virtual}$ is the virtual bandwidth allocated to the tenant. We describe about the detail of the virtual bandwidth in Chapter 2.2.

On the virtualized network environment the tenant can have instances distributed among racks, datacenters or regions, which are called sites and every instance reside on the site. DRL has its constraint that the sum of the all instances owned by the tenant on every site never excceds the virtual bandwidth assigned to the tenant, which is

$$\sum_{s \in \mathbb{S}} \sum_{i' \in \mathbb{I}'_s} r_{i'}(t) \leq B_{virtual} \tag{2.2}$$

where $\mathbb{S}$ is the set of the site, $\mathbb{I}'_s$ is the set of the instances on the site s and $r_{i'}(t)$ is the maximum rate assigned to the instance on the site s on time t and B_{tenant} is the virtual bandwidth allocated to the tenant.

On the other hand, the physical network environment where the virtualed networks are mapped forces the constraint that the sum of all instances on a physical host never exceed the maximum bandwidth which the host has, which is

$$\sum_{t \in \mathbb{T}} \sum_{i'' \in \mathbb{I}''_t} r_{i''}(t) \leq B_{local} \tag{2.3}$$

where $\mathbb{T}$ is the set of the tenants which a physical host serve, $\mathbb{I}''_t$ is the set of the instances which assigned to the the tenants t, $r_{i''}(t)$ is the bandwidth which the instance i consumes on time t and B_{local} is the maximum bandwidth available for the physical host serves the tenants.

We need to satisfy both of two equation (2.2) and (2.3) to rate limit appropriately. These constraints gurantee the fairness of allocated bandwidth and the predictable performance seperation among the tenants.

We specified the time t and loosened the restrictions on (2.2) and (2.3) because these constraints may not be satisfied at some points and we can not gurantee they are always holded.

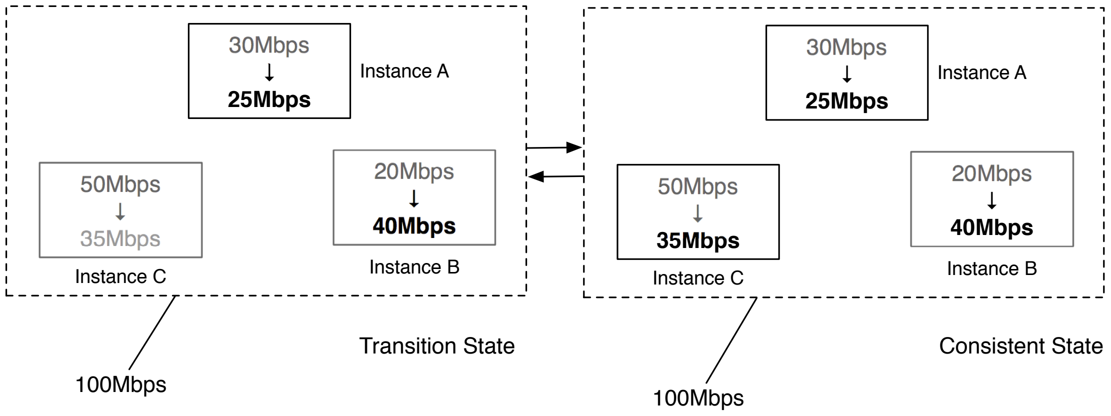

Figure 2.1: The consistency of the virtual bandwidth

2.2 Virtual Resource and Virtual Bandwidth

Virtual resources are the sets of resources assigned to the tenants. For instance, the tenant would contract for the services which are assigned network resources such as the maximum bandwidth allocated only for the tenant, which is refered as the virtul bandwidth.

In our model the OpenFlow controller on each physical host manage virtual resources associated with the each tenant keeping the consistency of the virtual resources' information. In other words, OpenFlow controllers assigned to the tenant communicate each other and allocate some resources for the instances on the physical host from the virtual resources which the tenant owns as if the tenant has the allocated resources and cut out some pieces from the resources.

2.2.1 The Consistency of the Virtual Bandwidth

On the local environment it is available to keep the consistency among instances because they are on the same host and communication between one instance and other instances is enough fast and reliable.

However, on the virtualized network instances could be distributed among racks, datacenters or regions. In such situation we need to gurantee the consistency of the infomation of the virtual bandwidth among the controllers distributed in the sites as the equation (2.2). If the consistency is lost, the constraint that the tenants never use the bandwidth beyond the allocated range. This can cause the unfaireness among the tenants therefore it is critical to keep the consistency for DRL system.

In this situtation, we have two choices to keep the consistency between one controller and others:

1. **Store data of the virtual bandwidth on the centralized repository.** In this case we can manage data easily monitoring the repository and calculating the maximum rates on the repository. However, such centralized repository can be the single point of failure (SPoF) of the system, which effects the whole physical network reside on the same physical host and cause the inconsistency of the infomation of the virtual bandwidth.

2. **Store the data on each controller.** They comunicate each other to keep the consistency among controllers assigned to the tenant by the consensus algorithms such as gossip protocols [29] or Paxos [21]. Although the implementations of the consensus algorithms are required, this method extinguish the SPoF. However the constraint depend on the strong consistency that the sum of the bandwidths used by the instances owned by the tenant never exceed the allocated bandwidth as denoted in (2.2).

The main point to choose the method to keep the consistency is the performance. In our model statistics data collection and calculation of the maximum rates are done in a period of the fixed time interval and the controller will serve several tenants. In addition the controllers have to process OpenFlow packets except for the statistics data collections. Therefore we chose to use replicated and distributed directory service for storing data of the virtual resources. The directory service replicates the data among few replication databases to avoid being SPoF and performs in the scaling manner to deal with massive requests.

2.3 Service Overview

Our system consists of three components. The first one is a directory service which sotres the virtual resouces and the virtual bandwidth. The second one is rate limiter which collect statistics data and set max rates to the instances of the tenant. It repeats that processes in a period of the fixed time. And the last one is calculator which monitor the changes on the data of the virtual resources, calculates the appropriate max rates and notify them to the rate limiter.

Figure 2.2 depicts the system overview. At first the rate limiter collects statistics data of instances assigned to the tenant and report them to the directory service. The calculator monitoring the change of the data on the directory service then calculates the maximum bandwidth for all instances. Then the rate limiters are notified from the directory service and set the calculated maximum rates to the instances.

We decouple the calculation of the maximum rates base on the rate limiting algorithms and the actual rate limiting mechanisms into the rate limiters and the calculators to achieve decentralized and flexible rate limiting. Due to this decision we can change the calculation policy by replacing the calculator into another one without any modifications on the switches. Switches never care about the rate limiting algorithms but just set the maximum rates calculated by the calculator.

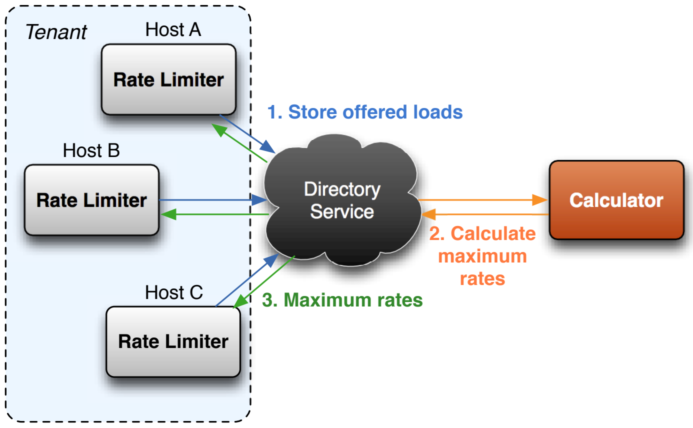

Figure 2.2: Service overview

2.4 Calculator

Calculators are the individual processes which monitor the information of the virtual resources for the tenants and calculate appropriate maximum rates for instances of each tenant. Calculators watch offered loads calculated base on statistics data from Rate Limiters and if they can find changes on the data they start calculation. The algorithms of the rate limiting are described in algorithm 1.

Algorithm 1 Max Rate Calculation Callback

```
Ensure: $t$ is the identifier of the tenant to which the calculator dedicated
 1: function UpdateRateLimitCallback
 2:     if $calculationLock$ is not acuired on the directory service then
 3:         acquire $calculationLock$
 4:         $o \leftarrow$ GetOfferedLoads(t)
 5:         $maxRates \leftarrow$ CalculateMaxRates(o)
 6:         store $maxRates$ to the directory service atomically
 7:         for all $p \in \mathbb{P}_t$ do
 8:             if rate limiter is active then
 9:                 SetUpdateCallback($p$, $maxRate$, UpdateRateLimiterCallback)
10:             else
11:                 SetUpdateCallback($p$, $maxRate$, Nil)
12:             end if
13:         end for
14:         release $calculationLock$
15:     end if
16: end function
```

In short, calculators will not calculate maximum rate but use offered loads directly as the maximum rates if the sum of the offered loads does not excceed the maximum bandwidth limit allocated for each tenant. Otherwise calculators normalize the sum of offered loads by the maximum limit and divided them based on some metrics, for example the ratio of the amount of each instance's traffic.

Algorithm 2 shows the ratio-based rate calculation.

Algorithm 2 Ratio-based Rate Calculation

```
Ensure: $t$ is the identifier of the tenant to which the calculator dedicated
Ensure: $O_t$ is the set of offered loads from which instances on this physical host
 1: procedure CalculateMaxRates($O_t$: OfferedLoads)
 2:     $demand \leftarrow \sum O_t$
 3:     if $demand > B_{virtual}$ then
 4:         for all $o \in O_t$ do
 5:             add $(oB_{virtual})/demand$ to $maxRates$
 6:         end for
 7:     else
 8:         $r \leftarrow (B_{virtual} - demand)/\text{Length}(O_t)$
 9:         add $o + r$ to $maxRates$
10:     end if
11:     return $maxRates$
12: end procedure
```

2.5 Rate Limiter

Rate limiters are the frontends of the OpenFlow controllers for DRL. They are the component of each OpenFlow controller on the physical host and able to be enabled or disabled by the controller. They behaves as following in sequential.

1. Collect statistics data for each instance
2. Report the data to other OpenFlow controllers which dedicate to the management of the virtual resources for the tenant, and the OpenFlow controllers store the data to the directory service
3. Wait for calculator to finish calculating the maximum rates for the all instances
4. Set the propagated maximum rates calculated by Calculator to the instances

The rate limiters repeat these processes on the fixed time interval. Hence all processes have to be done in the time interval.

Algorithm 3 presents the rate limiter's main loop.

Algorithm 3 Local Rate Limiting Loop

```
Require: $\mathbb{P}_t$ is the set of the ports on this physical host
 1: procedure RateLimit
 2:     repeat
 3:         for all $p \in \mathbb{P}_t$ do
 4:             $s \leftarrow$ RetrieveStatisticsData(p)
 5:             add $s$ to the stats set
 6:         end for
 7:         $o \leftarrow$ CalculateOfferedLoads(stats set)
 8:         store $o$ to the directory service atomically
 9:         wait for $maxRates$ to be calculated
10:         for all $p \in \mathbb{P}_t$ do
11:             set GetMaxRate($maxRates$, $p$) to $p$
12:         end for
13:     until rate limiter is active on the fixed interval
14: end procedure
```

2.6 Rate Limiting Interval

In our model DRL should be done on the fiexed time interval, which gives the constraint

$$t_d + t_r + t_c \leq T_{interval} \tag{2.4}$$

where t_d is the total time involving in the directory service such as read from and write to it, t_r is the total time to the rate limiter, t_c is the total time to the calculator and $T_{interval}$ is the interval set to the OpenFlow controller to start the next rate limiting. If the constraint denoted in the equation (2.4) is broken, the rate limiting in that time interval will not be applied appropriately. Therefore the equation (2.4) have to be always holded.

2.7 Automatic Applying of DRL

On the virtualized network environment, the instances launched by the tenant can be placed in the different locations such as racks, datacenters and regions. In addition the instances can be migrated from one location to another one because of the economic, legal and disaster reasons.

For instance, if the expense of the electricity on a location is cheaper than another one the tenant could want to migrate to the location. Or the tenant's website is recognised as illegal on a location but other location. Otherwise an hurricane or an earthquake could strike the datacenter which hosts the instances of tenant.

Even though the tenants experience such situation and migrate their instances, the DRL should work and manage the traffic of instances appropriately. All information required for the DRL are stored on the directory service and the rate limiter should be able to treat the assigned instances in the state less manner.

Chapter 3

Implementation

In this chapter we presents the implementation of the system we described in the previouse chapter. As we stated, we propose the DRL on the virtualized network topology based on OpenFlow. We chose Open vSwitch [4] which is a software switch implementation of OpenFlow to build the virtualzed network.

3.1 OpenFlow Protocol and Open vSwitch

OpenFlow is the protocol through which we can retrieve the information of the OpenFlow switches and set the behavior to the specific flows. OpenFlow decouples the control planes and the datapaths.

Open vSwitch has the follwing major two interfaces for software switches:

1. **OpenFlow protocol interface:** We can retrieve the statics data for bridges, ports, QoSs, queues and so on through this interface. We construct the request packets, send them through this interface and get the response packets from OpenFlow switches.

2. **Open vSwitch Database (OVSDB) interface:** Open vSwitch stores the information of OpenFlow switches and their management data on the OVSDB. We communicate with OVSDB by JSON-RPC [3] through the TCP socket. OVSDB create, delete and update the data of the concrete OpenFlow switches, their ports, QoSs and queues keeping the integrity of the data. OVSDB synchronize with the OpenFlow switches by the periodically polling, however the consistency and the order of the operations are guranteed becaus it supports transactions.

3.1.1 Flow Rules and Actions

OpenFlow switches have the flow tables which are set the pair of the packet matching rules and the associated actions. Figure 3.1 presents the forwarding scheme of the OpenFlow switches. When OpenFlow switches get the packets they match the rules they have and if they can find the rules they will process the packets in the way the associated actions present. Otherwise the switches get the unknown packets and report them to the OpenFlow controller. The OpenFlow controller determine the actions to deal with the packets and advertise the pair of the match rules created based on the packets and action to the OpenFlow switches. The OpenFlow swithes notified the pair set the pair their flow tables and process the packets reported to the OpenFlow controller based on the action on the flow tables.

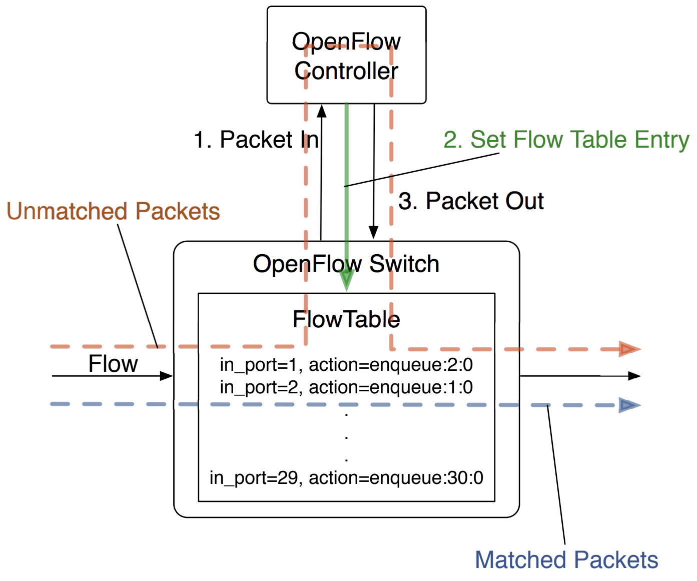

Figure 3.1: The forwarding scheme of the OpenFlow switches

3.1.2 QoSs and Queues on Open vSwitch

OpenFlow protocol 1.1.0 supports QoSs and queues providing the interface to retrieve the statistics data for them in its protocol [5]. It is also available to set the enqueue action in their flow rules. Table 3.2 shows the OpenFlow enqueue action packet. We can specify the queue to which the packets are enqueued and the port to which the packets will be forwarded after queueing.

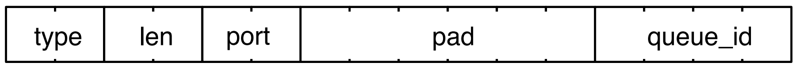

Figure 3.2: OpenFlow enqueue action packet

Figure 3.3 depicts the relationship between bridges, ports, QoSs and queues on Open vSwitch. Each port is bound to a VM instance and the port has a QoS which has one or more queues. We can configure the type of QoS, which is hierarchy token bucket (HTB) or hierarchical fair service curve (HFSC), and set the maximum rate and minimum rate for the queues. We control the bandwidth allocated for the tenant with these QoSs and queues changing the maximum rates based on the traffic demand from instances of each tenant.

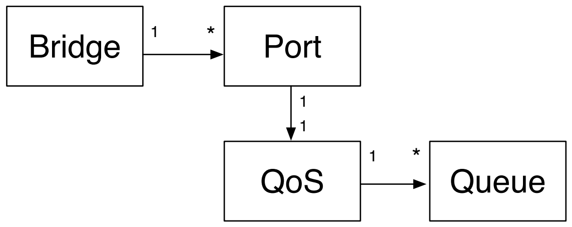

Figure 3.3: The relationship between bridges, ports, QoSs and queues

3.2 OpenFlow Controller

We use our own OpenFlow controller to manages the bridges, ports, QoSs and queues assigned to the physical host serves the OpenFlow controller. It processes the packets reported from the OpenFlow switches asynchronousely such as the creation, deletion and modification of bridges, ports, QoSs and queues.

Although the OpenFlow controller has the potential for programmable newtwork construction, it is beyond the range of this thesis to describe about the OpenFlow controller in details.

3.2.1 Connection Interface for Open vSwitch

Open vSwitch provides its control interface by JSON-RPC through the socket connections with the encription scheme as we described in Chapter 3.1. Though we can manage virtual network interfaces through this interface, the connection driver for Open vSwitch is required. We developed this interface with transactions and thread-safe manner to communicate with OVSDB. And OpenFlow protocol requires basically the natural number identifier as its packet data, i.e., bridge numbers, port numbers and queue numbers instead of bridge names, port names and queue UUID which are human readable and default identifiers on OVSDB. Hence we also added the utilities to convert identifiers and data on OVSDB into OpenFlow protocol formatted data. This interface used by the controllers to manage the virtual interfaces on the physical hosts.

3.3 Virtual Bandwidth and Directory Service

We selected ZooKeeper [20] as the forementioned directory service in Chapter 2.2.1. ZooKeeper can deal with from ten thousand to three thousand operations per second and replicate data among replicate servers [20]. ZooKeeper gurantees the order of operation and the consistency the data among replicas with ZooKeeper atomic broadcast. It also provides the file system abstraction for data, event driven style callback programming model, and atomic multiple operations, which gives fine grained data flow control among distributed OpenFlow controllers.

3.3.1 Virtual Bandwidth Data Model on ZooKeeper

ZooKeepr supports the file system abstraction for data management. Stored data are provided in the form of the tree topology to the client. Compared with the file system, all data are *nodes* which could have their children and every node can have its own data, which means all nodes are the directories and files at the same time in the point of view of the file system. Figure 3.4 draws the data structure related to the DRL on the virtualized network. We grouped the rate limiters and queues and let queues nodes are children of the rate limiter node as if the single rate limiter take care of the ports assigned to the VM instances directry. Each queue represents the allocated bandwidth for the associated instance and the rate limiter node hold the available bandwidth assigend to the tenant.

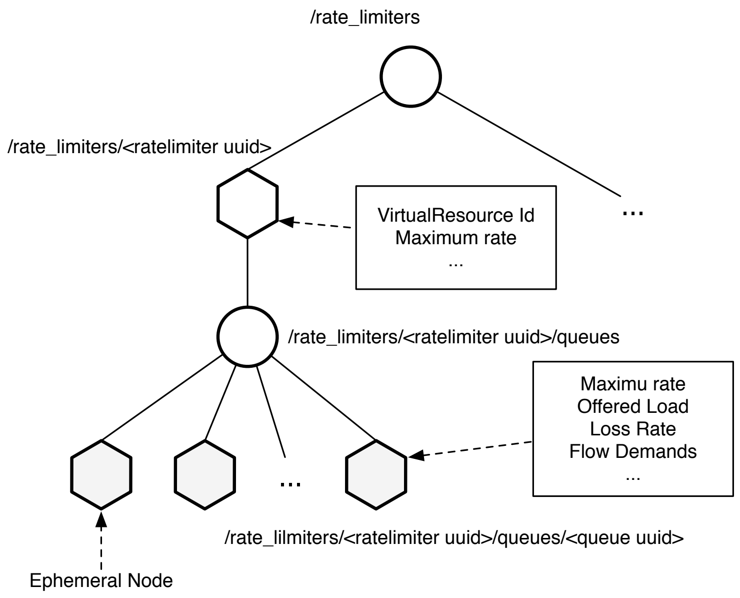
/rate_limiters
/rate_limiters/<ratelimiter uuid>
VirtualResource Id
Maximum rate
...
...
/rate_limiters/<ratelimiter uuid>/queues
Maximu rate
Offered Load
Loss Rate
Flow Demands
...
...
/rate_lilmiters/<ratelimiter uuid>/queues/<queue uuid>
Ephemeral Node

Figure 3.4: The data structure of the rate limiting information on ZooKeeper

3.3.2 Data Modify Notification on ZooKeeper

ZooKeeper advertises the event on the data such as creation, deletion, modification on nodes to the clients. As advertised the event, client will call the callback functions bound to the event.

We showed the callback style calculation in algorithm 1. The calculator starts to calculate the maximum rates when it receives the update notification from ZooKeeper. First it acquires the calculation lock and calculate the maximum rates based on the offered loads from the instances. Then it updates the maximum rate of each port setting itself as the callback function if the rate limiter is activated, otherwise it disables this recursive callback setting Nil as the callback function and stop the calculation loop.

3.4 Rate Limiting with QoSs and Queues on Virtualized Network

As described in Chapter 2.5, Rate Limiter needs to retrieve statistics data and set the maximum rates to the ports. Open vSwitch doesn't provide the interface for control maximum rates for the ports, however it provide the traffic control model with QoSs and queues.

3.4.1 The Architecture of DRL on the Virtualized Network

In Open vSwitch, QoS is the denomination to control on the traffic of the ports. We can specify the maximum traffic rate on the queues, however they slice the piece of available bandwidth from their associated QoS. Technically, QoS is a root class of the Linux HTB or HFSC and queues are the children of the root. Therefore we have to assign one QoS to each port. Each QoS can have at least one queue and we can set the enqueue action to enqueue the packets of the specific flows into the queues. Figure 3.5 depicts the detailed architecure of the DRL on the virtualized network.

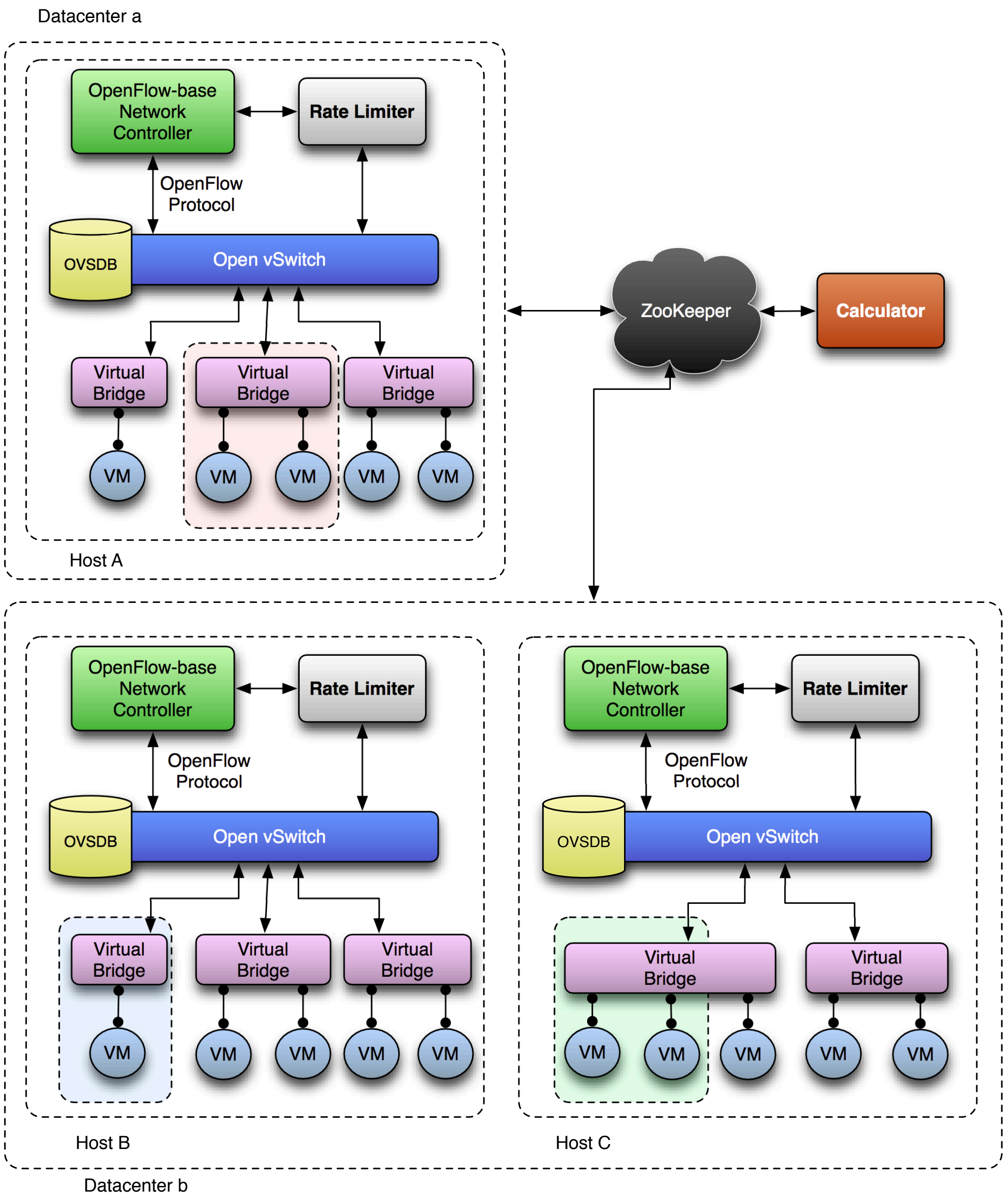

Figure 3.5: The detailed architecure of the DRL on the virtualized netowork

As presented in figure 3.6, the OpenFlow controllers are embedded to each physical host serving VM instances and ports assigned to the tenants. Each VM instance is bound a QoS with one or more queues. Packets of all flows go through the assigned queues if the demand is equal to or lower than the maximum rate, otherwise the packts will be enqueued into the queues limited by their maximum rate.

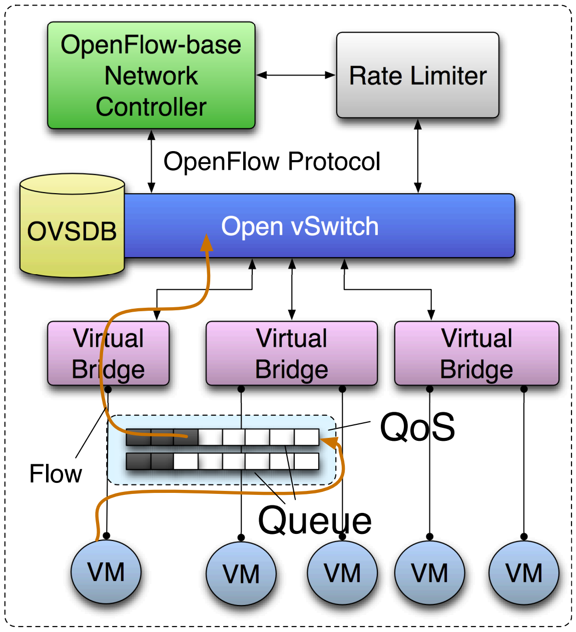

Figure 3.6: The local rate limiting implementation with QoSs and queues of Open vSwitch. Each VM is bound to a QoS and the QoS has one or more queues.

On the fixed time interval the rate limiters retrive the statistics data indicate the traffic demand for each instance and store the offered loads into ZooKeeper. The calculator watching the modification on the stored offered loads are notified the change on the data from ZooKeeper and calculate the maximum rates for each instances base on the offered loads. As the calculator store the calculated maximum rates back into ZooKeeper, ZooKeeper advertise the change on the maximum rates to the rate limiters and the rate limiters set the maxium rates to the instances. Repeating these process we achieve the DRL on the OpenFlow-based virtualized network.

3.4.2 Automaed QoS and Queue Assignement

We proposed the automatic applying of the DRL in Chapter 2.7. To achieve this automatic applying we need to know the join and leave of the instances to and from the OpenFlow switches managed by the OpenFlow controller. Upon the port is added, modified and deleted it suppose to send the OFPT_PORT_STATUS message to the controller in the OpenFlow protocol. Notified the event on the ports, the OpenFlow controller add the QoS and the queue, and bind them to the port which is assigned to the instances.

3.4.3 Enqueue Action On Packet Comming

We need to set enqueue actions to created QoSs and queues for rate limiting. The Enqueue action requires the identifier of the queue through which the packets go and the identifier of the queue where the packets are enqueued. OpenFlow switches notify the packet which does not match with the entries on its flow table to the controller by the OFPT_PACKET_IN message.

Upon the controler receive the OFPT_PACKET_IN message contains the unkown packet, the controller add new flow rule generated by the packet with enqueue aciton. The port number is looked up with the destination of the packet and the queue number can be determined by the source port entry on OVSDB.

3.4.4 OpenFlow Statistics Data Collection

We retrieve statistics data to observe the traffic loads of the ports on the OpenFlow switches. OpenFlow switches send back OFPT_STATS_RESPONSE messages if they receive OFPT_STATS_REQUEST messages asynchronously.

The OpenFlow controller sends the statistics request message creating the hash map which key is the identifier for the request and value is the blocking queue for the response in another thread. When the OpenFlow contoller receive the resposne the controller reacts to the response and push it to the queue. The thread waiting for the responsed by polling the blocking queue gets the response and the rate limiter take the traffic loads from the response.

3.4.5 The Maximum Rate Control on Open vSwitch

QoSs and queues on Open vSwitch supports maximum and minimum rates controled by the property of the entry on OVSDB. Formentioned in Chapter 3.4.1, a QoS is the unit to control the maximum bandwidth for the port. Queues share the maximum bandwitdh of the QoS and allocate their maximum bandwidth from the QoS. Therefore on Open vSwitch interface model we need to manage the maximum rates of both QoSs and queues.

3.5 Rate Limiter Implementation

Rate Limiter is implemented as a thread processed in the select loop shared with the controller. The rate limiting is scheduled on the fixed time interval and the procedure for settings of the maximum rates to the switches are called when the maximum rates on ZooKeeper are modified. The controller can manage the rate limiters with simple interface, *init*, *start*, *stop* and *delete*. We can initialize the rate limiting environemnt with *init* call creating QoSs and queues and attaching them to the ports. And then it is able to start and stop the rate limiting, although the calculation could be keeping by the calculator. Finally, we can delete QoSs and queues created in the initialization and completely abolish the rate limiting. Currently each rate limiter take care of the single tenant.

3.6 Calculator Implementation

Calculator is the totally individual process and watch the modification of the data on ZooKeeper. It calculates the maximum rates in the event driven style when the data on Zookeeper are modified. To react the data modifications it sets the callback function which retrieve the rate limiting information on ZooKeeper and claculate the maximum rates based on the rate limiting algorithms. Therefore it calculates the maximum rate unless the rate limiters stop performing rate limiting or it stop the calculation by itself. The calculator proviedes few interfaces to initialize and destroy the rate limiting information on ZooKeeper , stop the calculation and update the maximum rate assigned to the tenant.

Chapter 4

Evaluation

We evaluated our DRL system on the virtualized environment built with Open vSwitch. First, we tested our system on the single physical host with four VMs and obsesrve whether their traffic are rate limited in the manner we proposed in Chapter 2.4. The metrics to evaluate the DRL are the following elements:

1. **Performance:** This metrics shows whether the throughputs and latencies passed through the DRL is torolable for the rate limited instances or not. In the equation (2.2), the left hand side value should be approximated to the right hand side value as possible. In other words the sum of thethroughputs of the instances passed through the DRL are better if they are closer to the maximum bandwidth assigned to the tenant.

2. **Consistency:** All traffic loads should gurantee the equations (2.1) and (2.2). The sum of the allocated bandwidth to the instances assigned to the tenant by the end of the rate limiting interval should never exceeds the maximum bandwidth assigned to the tenant. Further, the sum of the allocated bandwidth to the instances on the physical host should never exceeds the available bandwidth of the physical host.

3. **Fairness:** The bandwidth assigned for the tenant should be allocated for each instances appropriately. If the sum of the demanded bandwidth by the instances does not exceed the maximum bandwidth assinged to the tenant, the maximum rate of each instance is determined based on the traffic loads of the physical host. Otherwise the maximum rates should be normalized by the maximum rate assigned to the tenant and allocated fairly following the rate limiting policy such that the rate limiter divide the maximum rates based on the ratio of traffic load of each instances. If the demand of the traffic loads the OpenFlow controller manages exceeds the the physical host the rate limiter should keep the equation (2.3).

Table 4.1 gives the evaluation environment where the the tests run. The test program launches the VM instances crating the QoSs and queues, and binds them to instances.

We use iperf to generate and evaluate the TCP and UDP traffic.

OS	Ubunutu Linux 11.04 x86_64
CPU	Quad core Intel Xeon E5620 2.4 GHz
RAM	16GB
NIC	1GbE
OpenFlow Switch	Open vSwitch 1.2.2-1
VMM	KVM with virtio
VM	Ubuntu Linux 11.04 x86_64

Table 4.1: Evaluation environment

4.1 DRL in the Single Site

First, we tested the DRL with four VMs on the same host. Figure 4.1 depicts the netowork topology of the test environment in the single site. Two VMs were data senders and other two VMs were data receivers. *Sender1* sends the packets to the *receiver1* and *sender2* sends the packets to the *receiver2*. We assigned 100Mbps to the tenant as the virtual bandwidth launching VM instances with KVM and attaching QoSs and queues to the VM instnces. Because 100Mbps traffic does not exceed the assigned maximum rates, the aggregated traffic of the instances should be allocated in the proportion of the demanded bandwidth to the instances.

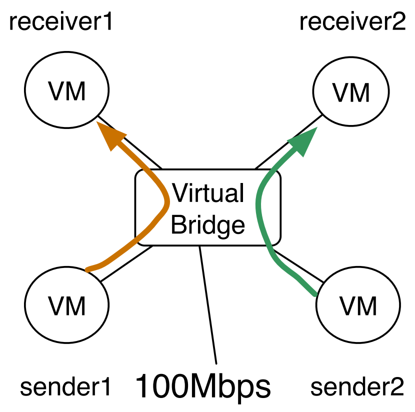

Figure 4.1: The network topology of the test environment in the single site

Figure 4.2 indicats the raw TCP traffic without the DRL. They tends to be bursty and exceeds the assigned virtual bandwidth.

4.1.1 DRL on the Same Physical Host in the Single Site

We tested our DRL system with the VM instances on the same physical host. First, sender1 and sender2 generate the UDP traffics of 300Mbps and 200Mbps respectively for thirty seconds, and then we changed the UDP bandwidth of the sender1 and sender2 to 400Mbps and 100Mbps and sends the packets for thrity seconds. We also generate the TCP traffic changing the number of flows at the same proportion and the period as the bandwidths of UDP traffics.

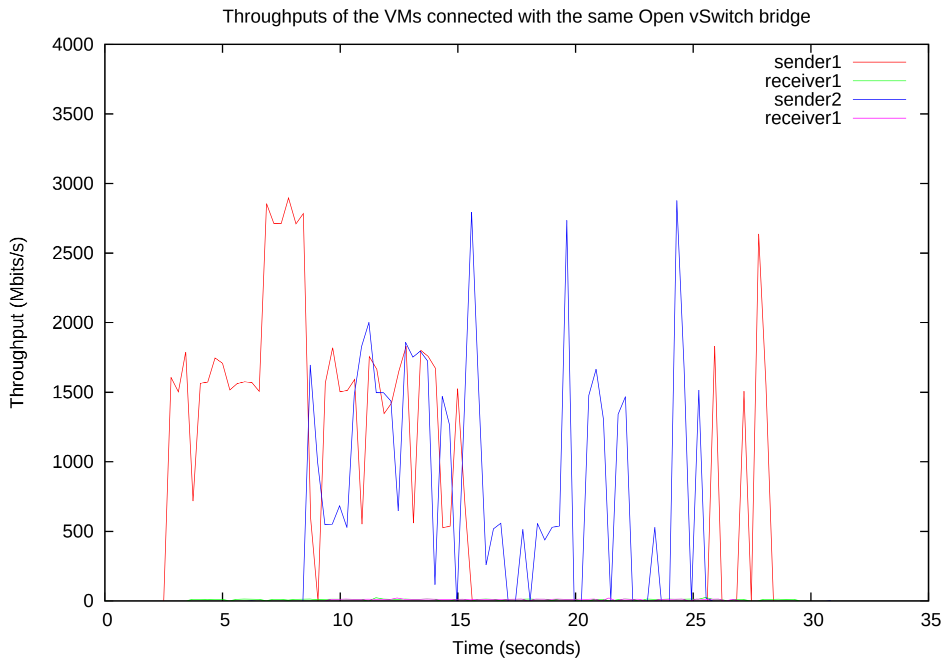

Figure 4.2: The raw traffic of the instances

Figure 4.3 depicts the result of the estimation of the throughputs for the receiving instances. It gives that the UDP traffics appear to be along with the ratio of the dmanded bandwidth of each instances, however TCP traffics show complete different proportion of the demanded bandwidth. In addition the throughput of the TCP traffics is much lower than we expected.

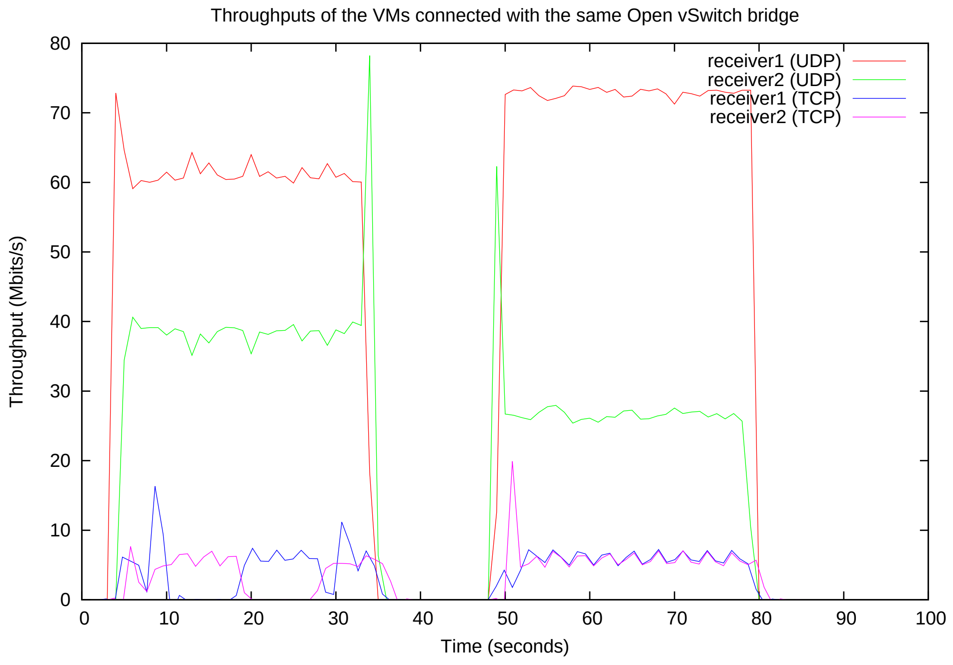

Figure 4.3: The traffic of the instances on the ratio-based rate limiting environment

Figure 4.4 depicts the aggregated bandwidth of the UDP and TCP traffics, which indicates that the assigned maximum rate is saturated with UDP traffics, however the TCP traffics show the about six times lower throughput than the maximum rate.

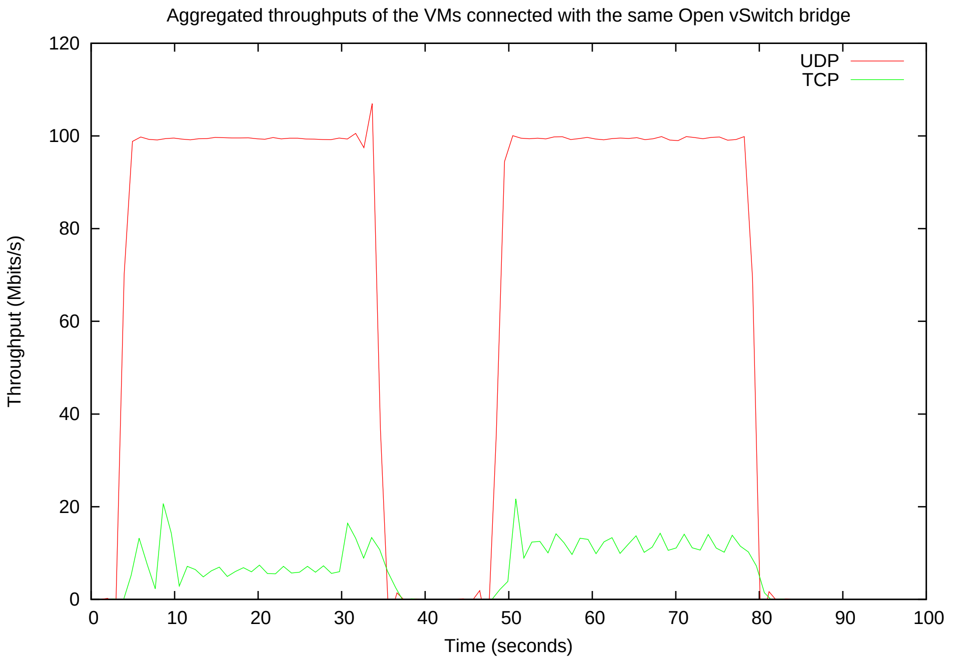

Figure 4.4: The aggregated traffic of the instances on the ratio-based rate limiting environment

Chapter 5

Discussion and Future Work

In this chapter we discuss about the design, implementation and evaluation results.

5.1 Performance Collapse of TCP Traffics

The result of the estimation in the figure 4.3 indicate that the TCP traffics do not follow the assumed behavior of the ratio-based rate limiting. The UDP traffic are rate limited in the well behavior, however the TCP traffic shows its low throughput suffered from unstable traffic loads.

We infer this is because the burstiness of the TCP traffics does not match with the drastically change of the maximum rates caused by our naive ratio-based algorithms. In that case the packets are dropped and retransmmitted again and again. To assert the correctness of this asssumption we need to implement C3P and D2R2 [28] algorithms which incease and decrease the maximum rate gradually based on the traffic loads.

5.2 Fault Torelance of DRL

Fault torelance is a strongly demanded feature by the industry because the cloud services are required to run avoiding large failures.

5.2.1 Fault Torelance of Directory Service

In our model the directory service must be fault torelant because it stores the critical information for not only the rate limiting but the virtualized networking.Altough we chose to use ZooKeeper as the centralized directory service, ZooKeeper nodes replicate the data among the replicated nodes and avoid to be the SPoF. All data operations are advertised to replicated nodes via ZooKeeper atomic broadcast protocol and keep the consistency and order of the data operations **??**.

5.2.2 Fault Torelance of Calculator

The calculator is a process and it can be corrupted due to the failure of the physical machine which serves it. As we described in 2.3, the calculator calculates the maximum rates assigned to the tenant in the state less manner. The calculator does not need to know the state of the rate limiters and it just calculate based on the information on ZooKeeper reported from the rate limiters.

Threfore the fault torelance of the calculator can be achieved by the a number of multiplexed calculators. The calculators try to connect to the ZooKeeper node and only one calculator can transit to the calculation state. If one of them succeed to connect to ZooKeepr it will acquire the lock on the calculation of the maximum rates until it complete the calculation. Otherwise the calculators failed to acuire the lock try to keep findng the different tasks.

5.2.3 Fault Torelance of Rate Limiter

The rate limiters can be broken because of the failures of the physical machines serve the rate limiters. In this case the instances are also collapsed and it is necessary to detect the failures of the instances and recover them. This can be achieved by the remove notification of the queue nodes in the rate limiter data hiererchy from ZooKeeper. Queue nodes are ephemeral nodes which are deleted automatically when the session of the nodes are expired. It is able to implement the monitoring services to decect the instance failures, however we have to note that the monitoring services should be also monitored. In addition the recover of instances is a technically challenge because we need to keep the state of the instances by live migrations or synchronization mechanisms. We pose this possibility as the future work.

Chapter 6

Related Work

We describe about the previous works on the virtualized network and distributed rate limitng comparing with our approach in this chapter.

6.1 Massive Scale Datacenters with Commodity Switches

Large scale datacenter network architectures such as Fat-tree [9], VL2 [18] and PortLand [23] propose the distributed network topology to achieve the massive scale and cost effective datacenters required by the cloud computing. They form a variant of the Clos network topology [13] with the massive commodity switches and the traffic spreading such as ECMP, VLB or their original routing mechanisms. They achieve lower cost, more utilization, lower energy consumption, wasting fewer heat and fault tolerance. In other words they proved that the network facilities are able to scale out with commodity switches as well as computing and storage resources in today. Our target deployment environemnt is such massive scale datacenters with commodity switches.

6.2 Software Defined Network

SDN technologies such as OpenFlow [22] emerged and led us to imagine about programming the networks. Enabling us to change the topology or properties of the networks dynamically along with the traffic loads on the network, SDN is one of the feasible solutions for the network virtualization. Onix **??** proposes the OpenFlow-based control stack for the large scale datacenters, which takes the similar approch with our OpenFlow controller. However, Onix focuses on the management of the network information base which is stored in the distributed Onix instances to track network stats prividing APIs, and it does not support any QoS and DRL functions.

6.3 Distributed Rate Limiting

On the other hand Distributed rate control on the cluster based infrastrucres have appeared because of the popularity of the cloud computing [24, 29, 25] and the demand for the controlling the network bandwidth of the instances assigned to the tenant spreded in different locations.

GRD and FPS are initially proposed solutions to apply DRL to the instances across different locations. They pretends to share the single token bucket with instances on the different locations even though instances are decentralized communicating with other rate limiters via Gossip algorithms.

In GRD each switches drop packets randomly if the aggregated throughput exceeds the maximum rate of the tenant. It simply emulate the centralized token bucket with the infomation collected from each deployed rate limiter. Although GRD works appropriately, the datacenter traffic is characterized as *"elephant and mice"*, which GRD is not always effective solution [24, 11]. of the distributed system such as MapReduce [14], GFS [16], BigTable[12] and Dynamo[15]. These distributed systems split large data into small chunks and apply comutation on them communicating with each other in the decentralized manner. On that workloads the traffic among nodes tends to be a large number, short live, lower bandwidth require flows and GRD equally drops packtes of big, long live, high bandwidth comsuming and relatively fault tolerable elephant flows and mice flows. This GDR's behavior introduces the unfairness on the datacenter environment.

In FPS rate limtiers determine the weighted maximum rate based on the flow proportion rather than the traffic loads to improve the unfairness to the traffic of the distributed systems.

C3P and D2R2 introduce the improved rate limiting deriving from GRD and FPS [28] They are well formed in mathematical way and providing the better fair share traffic loads predicted by the previoous traffic loads.

These algorithms were implemented on the simulation software or physical switches with the consensus algorithms. If we want to change the rate limiting algorithms we have to modify the configurations of the physical switches manually or the specific over-engineeered automation tools. Our approach is implemented on the OpenFlow switches without any modifications on the switches even if the rate limiting algorithms are changed because we decouple the rate limiting algorighms and actual rate limiting mechanisms. The rate limiters can limit the maximum rates of the switches in the stateless manner and never care about the actual calculation of the maximum rates. The calculators calculate the maximum rates as well without any consideration the physical locations where the instances assinged to the tenant.

6.4 Dynamic Bandwidth Allocation on Datacenter Network

Bandwitdh allocation for the datacenter network shared with tenants emerged to achieve the predictable performance and explicit charging on the network bandwidth.

Gatekeeper [26] and Seawall [27] introduce the bandwidth allocation on the physical host network shared with some instances. They gurantee the bandwidth assigned to the instances and provide work-conserving bandwidth sharing. If the aggregated bandwidth does not exceed the available bandwidth, the instances which demand more bandwidth assigned to them can borrow unused bandwidth form other instances. Hence they are focusing on keeping the constraints of (2.3) and different from our approach. Our approach empasizes the constraint (2.2) and introduces the possibility of slicing the bandwidth from the global available bandwidth assigning the virtual bandwidth to the tenant.

However, these bandwidth allocation approaches are the complement of ours and we need these solutions to gurantee the performance on the virtualized network. In this situation the location where the instances are mapped will have an significant effects to the performance because distributed systems indicate better performance if the nodes communicating with each other are on the same rack. Oktopus [10] proposes the dynamic assignment of the instances to the tenant based on the traffic loads and available bandwidth on the phyisical machine.

Chapter 7

Conclusion

We proposed the highly automated DRL system on the virtualized network. This approach can manage distributed instances migrating to the different sites fully automating the applying DRL on join and leave of the instances. We expect this contributes to enhance the agility of the network deployment on the cost effective and massive scale datacenter.

We showed the indirectly rate limiting by the QoSs and queues are achievable in the programmable behavior To prove our DRL sytem can deal with the massive scale datacenter network our system need to be deployed and tested on the real workloads environment.

Acknowledgement

I would like to thank my supervisor, Professor Tasuo Nakajima; Dan Mihai Dumitriu, Yoshi Tamura and Midokura collegues for their cooperation and suggestions on my research; my friends for fun time in life.

I would be grateful to my father, mother, little sister and family for their continual encouragement on hard time.

Bibliography

[1] Amazon web services. `http://aws.amazon.com/`.

[2] Google app engine. `http://code.google.com/appengine/`.

[3] Json-rpc. `http://json-rpc.org/`.

[4] Open vswitch: An open virtual switch. `http://openvswitch.org/`.

[5] Openflow v1.1.0 specification.

[6] The rackspace cloud. `http://www.rackspace.com/cloud/`.

[7] Vxlan: A framework for overlaying virtualized layer 2 networks over layer 3 networks. `http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-00`.

[8] Windows azure. `http://www.windowsazure.com/en-us/`.

[9] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[10] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference on SIGCOMM*, SIGCOMM '11, pages 242–253, New York, NY, USA, 2011. ACM.

[11] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40:92–99, January 2010.

[12] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[13] Charles Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[15] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[16] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[17] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39:68–73, December 2008.

[18] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[19] Urs Hoelzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.* Morgan and Claypool Publishers, 1st edition, 2009.

[20] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[21] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16:133–169, May 1998.

[22] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74, March 2008.

[23] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[24] Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex C. Snoeren. Cloud control with distributed rate limiting. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 337–348, New York, NY, USA, 2007. ACM.

[25] Vijay Shankar Rajanna, Smit Shah, Anand Jahagirdar, Christopher Lemoine, and Kartik Gopalan. Xco: explicit coordination to prevent network fabric congestion in cloud computing

cluster platforms. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 252–263, New York, NY, USA, 2010. ACM.

[26] Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival Guedes. Gatekeeper: supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proceedings of the 3rd conference on I/O virtualization*, WIOV'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.

[27] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 23–23, Berkeley, CA, USA, 2011. USENIX Association.

[28] Rade Stanojevi and Robert Shorten. Fully decentralized emulation of best-effort and processor sharing queues. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '08, pages 383–394, New York, NY, USA, 2008. ACM.

[29] R. Stanojevic and R. Shorten. Generalized distributed rate limiting. In *Quality of Service, 2009. IWQoS. 17th International Workshop on*, pages 1 –9, july 2009.