C2QoS: Network QoS guarantee in vSwitch through CPU-cycle management

Ye Yang, Haiyang Jiang, Yulei Wu, Chunjing Han, Yilong Lv, Xing Li, Bowen Yang, Serge Fdida, Gaogang Xie

Version of Record: https://www.sciencedirect.com/science/article/pii/S1383762121001089
Manuscript_1c62824e32c2ea12158c51f39ffc1e33

C2QoS: Network QoS Guarantee in vSwitch through CPU-cycle Management

Ye Yang, Haiyang Jiang, Yulei Wu, Chunjing Han, Yilong Lv, Xing Li, Bowen Yang, Serge Fdida, Gaogang Xie

***Abstract*—Cyber-Physical-Social-System (CPSS) relies on cloud and edge computing for service deployment and acceleration. As an enabling technology of CPSS, network virtualization allows different services to be deployed on a single physical server and provides them with differentiated network performance based on service-level-agreement (SLA). Unfortunately, current virtual switches (vSwitches) cannot guarantee this functionality due to the common resource competition and unpredictable processing capacity. As a software on the server, vSwitch is allocated limited dedicated CPU cores to implement traffic forwarding for all services. But the QoS strategy inheriting from the hardware switch, does not consider the forwarding tasks' competition for CPU cores in terms of utilization and timing, thus cannot guarantee the SLA. To solve this critical issue, we propose a CPU-Cycle based QoS (C2QoS) strategy to realize the isolation and guarantee of service network performance by managing the usage and scheduling of the IO-dedicated CPU cores in vSwitch. C2QoS includes a CPU-Cycle based token bucket mechanism to strictly limit service's network bandwidth and a hierarchical batch scheduling mechanism to achieve hierarchical latency. Experimental results show that, compared with existing strategies, C2QoS can strictly guarantee the network bandwidth of the services and reduce the service latency by up to 80%.**

***Index Terms*—QoS, vSwitch, CPU resource competition, scheduling, cloud computing.**

I. Introduction

Cyber-Physical-Social-System (CPSS) relies on cloud computing and edge computing to accelerate service deployment for a series of emerging technologies, such as the Internet of Things (IoT), smart cities, smart healthcare etc [1]–[6]. As the foundation of cloud/edge computing, virtualization technology allows services to flexibly run on the edge clouds in the form of virtual machines (VMs) [7]. In the cloud virtualization environment, the deployment density of VMs is usually sufficiently high, and the VMs belonging to different tenants or different services will be deployed on the same physical server. These VMs share various physical resources of the host, including CPU, memory and network. Therefore, how to design a strategy for allocating these resources and providing differentiated services is the foremost issue for Cloud Service Providers (CSPs).

On the server of a cloud platform, VM network resources are provided by a software ***vSwitch***. The vSwitch connecting Network Interface Controller (NIC) to the VMs provides all VMs' network connectivity. All packets sent from the VMs need to go through steps such as classification and forwarding in vSwitch before they can be sent out via the NIC. Therefore, the upper limit of bandwidth and latency these VMs can achieve are determined by the processing capabilities of the vSwitch [8], [9]. But for a CSP, it is a common practice to increase the share of CPU resources with VMs, and thus very limited CPU resources are left for vSwitch's forwarding tasks, e.g., the Google cloud uses no more than two IO-dedicated physical cores to perform forwarding tasks [10]. As a result, all the VMs compete with each other for the limited processing capacity of the vSwitch, essentially for the CPU resources occupied by the vSwitch.

Meanwhile, the CPU resources that are needed to maintain particular network performance is hard to predict in vSwitch. Different from the hardware switch that has a constant forwarding capacity, the vSwitch's processing capacity with these IO-dedicated CPU cores is variable when being used to forward traffic with different characteristics. For example, at the same Bits-Per-Second (BPS) rate, compared to forwarding the traffic with 1518 byte packet size, forwarding the traffic with 64 byte packet size consumes 10 times more CPU cycles [11]. As a result, the vSwitch can hardly guarantee all tenants' network SLA performance in all situations.

Existing network Quality-of-Service (QoS) strategies in software vSwitch are inherited from the interface-based solutions of hardware switches, and do not consider the issues of CPU resources competition among tenants, as well as the variable vSwitch forwarding capacity. As a result, they cannot ensure VMs' SLA performance targets. Our experimental results on a multi-tenant cloud platform in Section II, show that the innocent VM bandwidth can be decreased by up to 20% due to the competition of IO-dedicated CPU usage. In terms of latency, all VM's latency increase hundreds of times, since all VMs' forwarding tasks compete for the limited CPU cores in terms of timing and are not processed in a differentiated manner. Some works [11], [12] have noticed the resources competition issue, and they added a module for CPU resources isolation before the QoS module. The effects of these works

Y. Yang and H. Jiang are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: yangye@ict.ac.cn; jianghaiyang@ict.ac.cn).

Y. Wu is with the College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, U.K. (e-mail: y.l.wu@exeter.ac.uk).

C. Han is with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail:hanchunjing@iie.ac.cn).

Y. Lv, X. Li and B. Yang are with Alibaba Group, Hangzhou 311121, China (e-mail: lvyilong.lyl@alibaba-inc.com; lixing.lix@alibaba-inc.com; bowen.ybw@alibaba-inc.com).

S. Fdida is with the LIP6 Laboratory, Sorbonne University, 75006 Paris, France. (e-mail: serge.fdida@sorbonne-universite.fr).

G. Xie is with the Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China (e-mail: xie@cnic.cn).

Y. Yang and G. Xie are also with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China.

were limited because they are still interface-based and the variable vSwitch forwarding capacity is still ignored.

Different from existing solutions, in this paper we propose a new CPU-Cycle based QoS strategy (C2QoS) to completely solve this issue. As the "virtual" network resources are realized by the IO-dedicated CPU cores, we guarantee VM's network SLA performance by directly apportioning these CPU cycles to VMs. The challenges of achieving this goal include: 1) How to establish a correspondence between VM's bandwidth and CPU usage. 2) How to assign CPU cycles to VM to strictly guarantee its bandwidth. 3) How to ensure the SLA latency, especially for the delay-sensitive applications (e.g., the web and video servers require lower response latency than file system servers).

To address these challenges, this paper makes the following main contributions:

- We propose a modeling methodology to build the correspondence between forwarding capacity and CPU resources in vSwitch. The model characterizes the effect of different working conditions over the vSwitch forwarding capacity. These conditions include the tenant traffic characteristics, as well as the deployment configurations.
- Based on the model, we propose the C2QoS strategy, containing a CPU-Cycle based Token Bucket (C2TB) mechanism for performing isolation enhanced rate limiting and a Hierarchical Batch Scheduling (HBS) mechanism for providing latency guarantee.
- We implement the C2QoS strategy on DPDK accelerated open vSwitch (OVS-DPDK) [13], [14] platform. The experiments on a multi-tenant cloud platform show that compared with existing strategies, the influence of CPU resource congestion on bandwidth is eliminated and that on latency is reduced by 80%.

The rest of this paper is organized as follows. Section II introduces the background and motivation. Section III presents the model between network performance and CPU usage. Section IV shows the design of C2QoS, and its implementation on OVS-DPDK is shown in Section V. Section VI carries out performance evaluation. Section VII discusses the issue to deploy C2QoS in real-world environment, and Section VIII concludes this paper.

II. Background and Motivation

A. Network QoS in vSwitch

Network QoS strategy is a well-studied topic in hardware switch, and a lot of works have been proposed. According to the implementation and function, the QoS strategies can be divided into two types: the token bucket mechanisms, and the fair queuing mechanisms. The token bucket mechanisms are used to limit the bandwidth sharing with little overhead, but it cannot ensure the latency [15]–[17]. In contrast, the fair queuing and traffic scheduling mechanisms represented by GPS, WFQ, and DDR, are proposed to guarantee SLA bandwidth and latency more finely, while bringing relatively high complexity [18]–[20].

When realizing these two kinds of QoS strategies in the hardware switch, the sufficient processing capacity inside the switch brings significant advantages. The main reasons are argued in [21]: the overhead of processing each packet is fixed; the token buckets and queues are implemented by hardware and they can complete the corresponding functions without compromising the performance; high-precision clock and hardware feedback support [22], [23].

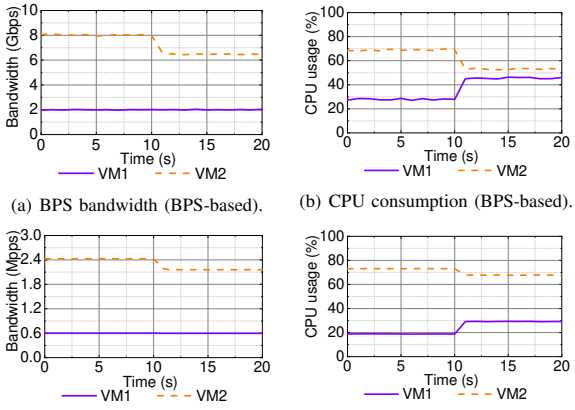
Unfortunately, none of the above advantages exists in software vSwitch. As mentioned in Section I, the CPU cores left for vSwitch are limited, and meanwhile their processing capacity is variable when being used to forward traffic with different characteristics. For example, in Google's experiments, forwarding a flow with 64-byte packets at a speed of 512 Mbps will consume more CPU cycles than forwarding a flow with 1518-byte packets at a speed of 2.4 Gbps [11]. On the other hand, as a software based process, the vSwitch has particular bottleneck and resource competition points, which are completely different from hardware switch. These differences make that the QoS strategies inherited from hardware switches cannot work well in the vSwitch. We will further demonstrate in this section that, existing QoS solutions can cause performance issues in both bandwidth and latency.

B. Bandwidth issue

The existing rate limiting methods in the vSwitch of cloud servers usually use the light-loaded token bucket for efficiency, e.g. the MBFQ rate limiting method used by Microsoft is implemented with a token bucket algorithm [21]. These existing token bucket mechanisms are all based on BPS or Packets-Per-Second (PPS), and directly limit the number or bits of packets that can be forwarded to guarantee tenants' SLA bandwidth. As the IO-dedicated CPU resources are limited and the processing capacity is variable when being used to forward different traffic, one tenant may legally squeeze the CPU resources and harm the bandwidth of others.

In this section, we adopt the best-performing three color marker (TCM) rate limiting algorithm [17], [24], [25] in OVS-DPDK platform to demonstrate the issue. On one server, we launch two VMs to connect to the OVS-DPDK as sender, and then use another directly connected server with the same hardware configurations as receiver. We use pkt-gen from netmap [26] inside the two VMs as packet generators. It should be noted that all experiments in this paper use the same platform configurations: Intel Xeon CPU E5-4603 v2 2.20GHz (32 logical cores on 4 NUMA nodes), 64GB DDR3 memory at 1333MHz, one Intel 82599ES 10-Gigabit Dual Port NICs and Ubuntu 16.04.1 (kernel 4.8.0) as operation system. The cloud platform is built on QEMU 2.10, DPDK 17.11.2 and OVS 2.9.2. Every VM is assigned with 2 GB memory and 1 logical CPU core.

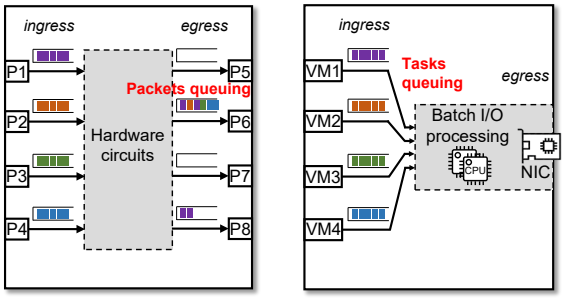
In the Fig. 1(a)-(b), we show how the BPS-based token bucket mechanism fails to ensure VM bandwidth. The VM1 and VM2 share one dedicated CPU core in OVS-DPDK for forwarding, and their bandwidths are limited to 2 Gbps and 8 Gbps respectively. Within the first 10 seconds, they send 512-byte packets and their BPS bandwidths are precisely limited. Starting from the 10th second, VM1 sends small packets (changing the packet size to 64-byte). In order to achieve the



(a) BPS bandwidth (BPS-based). (b) CPU consumption (BPS-based).

(c) PPS bandwidth (PPS-based). (d) CPU consumption (PPS-based).

Fig. 1. Bandwidth isolation issue in the existing token bucket algorithms. The results of BPS-based method are shown in (a) and (b), while the PPS-based method results are shown in (c) and (d).



(a) hardware switch. (b) software vSwitch.

Fig. 2. The difference of resource competition points in hardware switch and software vSwitch.

same BPS throughput (2 Gbps) as before, VM1's forwarding tasks in vSwitch consume 20% more CPU resources as shown in Fig. 1(b). This leads to a drop in the CPU consumption of VM2's forwarding tasks, which in turn reduces VM2's available bandwidth. Eventually, the innocent VM2 was affected by the tenant behavior inside VM1, resulting in approximately 20% decrease in VM2's bandwidth.

The same situation also occurs in the PPS-based token bucket mechanism. In the Fig. 1 (c)-(d), VM1 and VM2 also share one CPU core in OVS-DPDK for forwarding, and their bandwidths are limited to 0.6 Mpps and 2.4 Mpps respectively. Within the first 10 seconds, they behave well and both send a single flow. The VM1 starts to send multiple flows from the 10th second (change configurations in pkt-gen), which makes it require more CPU resources as the packet classification in OVS-DPDK gets slower. Similarly, in order to achieve the previous PPS bandwidth, VM1 preempts part of the CPU resources belonging to VM2 as shown in Fig. 1(d), which leads to a 16% decrease in VM2 bandwidth.

These two experiments demonstrate that, even the tenant behavior inside a small weight VM can influence the other innocent VMs' bandwidth. The reason is that the VM forwarding tasks compete for the IO-dedicated CPU cores in vSwitch, which is ignored in the existing rate limiting mechanisms. By exploiting this flaw, greedy tenants can obtain more CPU resources, or an attacker can construct specific traffic to harm the network performance of all tenants on the server. In either case, CSPs cannot provide the well-behaved tenants a stable network performance.

C. Latency issue

In addition to the bandwidth issue, the existing traffic scheduling mechanisms inherited from the hardware switch also cause latency issues in vSwitch. As processing engine and logic are very different between the hardware switch and the vSwitch, the resource competition occurs at different stages. Simply applying the previous strategy cannot solve the competition problem on a different platform.

As shown in Fig. 2, we present the abstract of packet processing logic in the hardware switch and the software vSwitch to analyze the different requirements of scheduling in the two architectures. In the hardware switch, the circuits with powerful processing capabilities make the resource contentions mainly occur at the *egress* stage, where the traffic of multiple in_ports is gathered for sending out on a particular port (see port P6 in Fig. 2(a)). The existing traffic scheduling mechanisms working at this stage [27]–[29] can queue up packets of different in_ports so as to guarantee differentiated latencies. However, in the software vSwitch, the packet processing capacity of CPU cores is far inferior to the hardware circuits that can achieve line speed. With the common practice in the cloud platform that very limited CPU cores are used for vSwitch, concurrent VMs compete for these CPU cores to execute the expensive batch I/O processing[1] in the *ingress* stage (see Fig. 2(b)). Due to the absence of task scheduling at the *ingress* stage, VMs indiscriminately queue up for batch I/O tasks to be completed, which causes mutual influence and high latency. In the worst case, on a server running n VMs, each VM will suffer from the additional latency caused by $n-1$ times of batch I/O processing.

We also use experiments to demonstrate this issue. To simulate the multi-tenant scenario on the cloud, we increase the number of VMs to 16 on one server and measure the TCP latency. One CPU core is used to forward traffic in the OVS-DPDK throughout. To measure TCP latency, we run qperf [30] as a client-side program in all VMs simultaneously, while the server-side program is run in another directly connected physical server. We measure 20 sets of data for each experiment to avoid accidents.

The results are shown in Fig. 3, where the latency in each case is shown in the form of a box diagram. It can be seen that when there is only 1 VM running on the server, its TCP latency is stable and maintained at 26 - 27 us. But with the number

[1]Batch I/O processing is the general running mode of existing vSwitches. To achieve efficiency, the CPU core always receives a batch of packets on a port for processing such as classification. After finally sending this batch of packets to the destination ports, the CPU will turn to receive the packets on the next port.

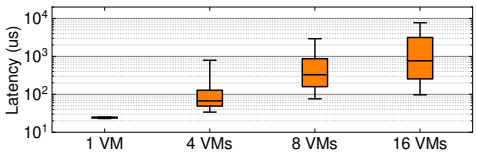


Fig. 3. Average TCP latency with different number of VMs deployed. With the number of VMs grows, the polling running mode in vSwitch causes all VMs suffering from the unstable high latency.

of VMs grows, the TCP latencies become unstable and all increase exponentially. When deploying up to 16 VMs on this server, all VMs suffer from hundreds of times higher latency indiscriminately due to waiting for the one IO-dedicated CPU core to sequentially process other VMs' batch I/O processing at the *ingress* stage.

D. Motivation

The reason that VM's bandwidth and latency cannot be guaranteed is that the existing QoS strategies ignore the IO-dedicated CPU resources competition inside the vSwitch. The lack of management and apportionment of CPU resources brings a series of flaws including bandwidth isolation and undifferentiated high latency. Some previous works have mentioned this issue, and some solutions have been proposed, e.g., Addanki et al. [12] considered separately apportioning IO-dedicated CPU resources and bandwidth on the software router, and Kumar et al. [11] proposed a method by using a CPU-based weighted fair queue to isolate CPU competition among VMs. But all of these works have limited effects because they only add a CPU isolation module before or after the existing interface-based QoS mechanisms, but fail to consider the variable vSwitch forwarding capacity and the different resource competition points in the software vSwitch process.

Essentially, the network forwarding capacity of vSwitches is not a kind of physical resources, but a kind of "virtual" resources that are provided by IO-dedicated CPU resources in the vSwitch. Starting from this point, the motivation of this work is to adopt the CPU resources apportionment, that reflects the network forwarding capacity more directly, in the VM network QoS solution. In order to do that, we first propose a modeling methodology to build the relationship between CPU resources and network forwarding capacity in vSwitch. Based on the vSwitch network performance model, we design and implement a new VM network QoS strategy.

This paper is based on a conference version [31], and the main improvements include: this paper contains more comprehensive investigation of the related works, and we strengthen the motivation through new experiments; this paper describes in more detail the modeling and design of C2QoS; we conduct more useful experiments to verify C2QoS's accuracy besides proving the guarantee of network QoS of tenants.

III. Bandwidth-CPU Model

To guide the design of QoS strategy, we first need to model the correspondence between forwarding capacity and CPU utilization in vSwitch. Our modeling of the vSwitch forwarding procedure is based on the OVS-DPDK platform, which is the state-of-the-art implementation and has been widely adopted by the industry. As the OVS-DPDK platform represents a lot of vSwitches in terms of packet processing logic, the modeling method can be easily applied to other vSwitch platforms.

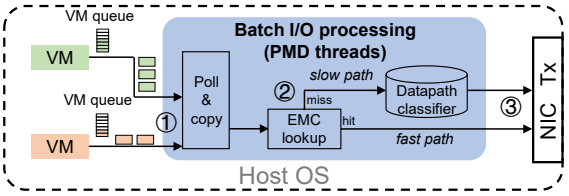


Fig. 4. Detailed processing logic in OVS. The batch I/O processing of sending packets from VM to NIC can be divided into three stages: ①*ingress*, ②*classification* and ③*egress*.

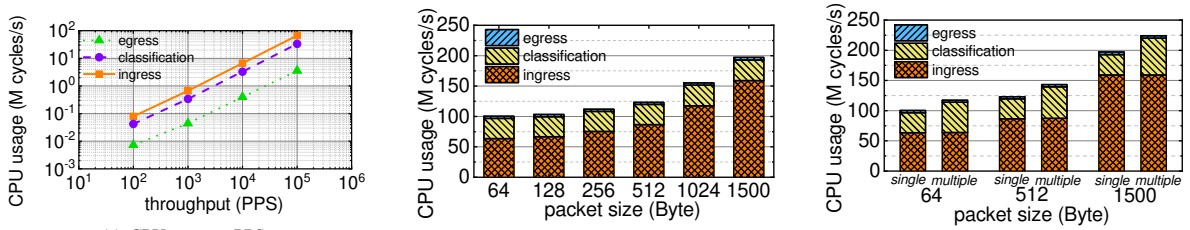
A. Packet forwarding procedure in vSwitch

In OVS-DPDK, several Polling Mode Driver (PMD) threads are launched and bound to the limited IO-dedicated CPU cores. For efficiency, these PMD threads use batch processing mode to process tasks. As shown in Fig. 4, the batch I/O processing procedure in the OVS-DPDK consists of three stages delivering packets from the VM to the external network. The first stage is *ingress*, the PMD thread copies a batch of packets from the VM memory to the vSwitch's buffer. Next, in the *classification* stage, the PMD thread looks up their destination port based on the five tuples. If the five tuple is found in the Exact Match Cache (EMC), we go to the next stage. But if it is missed, the PMD thread will use more CPU cycles to look up in the more comprehensive classifiers (the datapath classifier in Fig. 4) and then go to the next stage. Finally, it is in the *egress* stage that the PMD thread writes the packet descriptors to the NIC queue, and then the NIC can send packets out. According to these three stages, we also divide the CPU cycles consumed by the VM forwarding tasks into three parts as shown in the equation below:

$$C = C_{ingress} + C_{classification} + C_{egress} \quad (1)$$

where $C_{ingress}$ indicates the CPU cycles consumed in the *ingress* stage, $C_{classification}$ refers to that consumed in the *classification* stage and C_{egress} corresponds to the CPU cycles consumed in the *egress* stage.

It is worth noting that in the *classification* stage, the EMC capacity is limited, e.g., it has only 8192 entries in OVS-DPDK, so it can only store the most recently searched five-tuples. The datapath classifier is the main body of the classifier algorithm like tuple-search-space (TSS) [32] and contains all the rules in the vSwitch. Each time a five-tuple search is hit in EMC, the $C_{classification}$ only contains lookup cost in EMC. But if a lookup is missed in EMC and hit in datapath classifier, the hit entry needs to be added to EMC [9]. Therefore, the $C_{classification}$ under this case contains lookup cost in EMC and datapath classifier, and the update cost in EMC. Obviously



(a) CPU usage - PPS. (b) CPU usage - packet size. (c) CPU usage - multiple flows.

Fig. 5. The impact of traffic characteristics on the relationship between CPU consumption and bandwidth. The "M cycles/s" in Figs means million cycles/s.

the latter is much larger than the former, and the specific value will depend on the number and complexity of the rules.

During the whole packet forwarding procedure in vSwitch, many factors can affect the CPU consumption in the three stages, e.g. throughput, the number of flows, the number of VMs, and so on. According to the main bodies that control them, we divide these factors into two types: traffic characteristics managed by tenants and VM deployment configurations managed by CSPs. We will experiment to study how the three parts of CPU consumption in Eq. (1) are affected by the these factors, and then use measurement-based methodology to model CPU consumption under different situations.

B. Impact of network traffic characteristics

The first factor we consider is the network traffic characteristics, which can be changed by the tenant behavior inside a VM: sending rate, packet size and the number of flows. We launch one VM on the OVS-DPDK platform and assign one CPU core as the IO-dedicated CPU resources on one server. The impacts of the three traffic characteristics on CPU usage for forwarding are shown in Fig. 5 (a)-(c). During each experiment, we vary one characteristic and record the results, while keeping the other two with a certain value.

Sending rate (PPS). We keep the packet size at 1500-byte and keep the number of flows at 1 during this experiment. In Fig. 5(a), we find the CPU cycles consumed in all the three stages are proportional to the PPS. So this is the basis and premise of all existing BPS/PPS-based rate limiting methods: with no other traffic characteristics changed, the CPU competition will not occur.

Packet size. In this experiment, we keep the PPS at 10^5 and keep the number of flows at 1. The results of different packet sizes are shown in Fig. 5(b), and it can be seen that increasing the packet size will only increase $C_{ingress}$ and have nothing to do with $C_{classification}$ and C_{egress}. The increase of $C_{ingress}$ is due to the fact that only the stage *ingress* contains packet copying, so the larger packet requires more time to copy. For example, the $C_{ingress}$ under the case of forwarding 1500-byte packets is more than twice that of forwarding 64-byte packets at the same PPS rate.

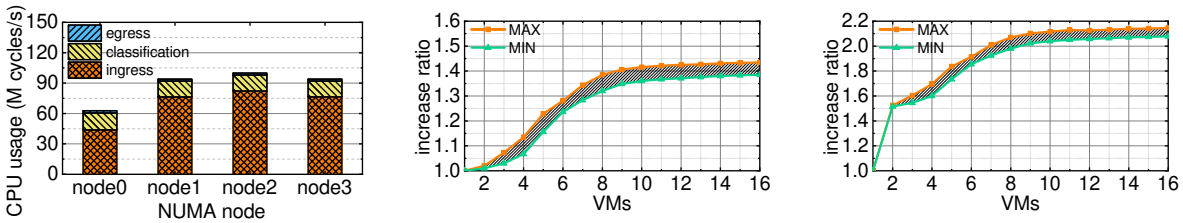
Number of flows. We keep the PPS at 10^5 and keep the packet size at 1500 byte during this experiment. The result of concurrent flows is shown in Fig. 5(c). Comparing with only sending one flow ("single" in the figure), sending a large number of concurrent flows ("multiple" in the figure, we range dst ip from 0.0.0.0 to 255.255.255.255 and at the same time randomize the port number) will cause the packet classification frequently misses in EMC lookup and the packet will enter the longer search path, and thus $C_{classification}$ is increased. In our experiment, the $C_{classification}$ in the worst case is 1.67 times more than that in the best case. But it should be noted that, the number and complexity of the rules in the flow table will affect this ratio.

The existing QoS strategies adopted by CSPs only consider one of the above three characteristics, i.e., PPS (BPS is considered as $PPS * average\ packet\ size$). Unfortunately, all existing strategies fail to guarantee the SLA as the other two characteristics can easily undermine it via affecting CPU consumption (see Fig. 5 (b)-(c)). From these experiments, we can see that a certain bandwidth-CPU relationship for a single VM can only be established with all the traffic characteristics are determined. Based on this relationship, we can allocate C_{single}, which indicates the CPU resources that a single VM requires to achieve the purchased bandwidth under specific traffic characteristics, to each VM. In this way, the IO-dedicated CPU utilization among VMs can be isolated, which will resolve the SLA issue.

It should be noted that besides the above three traffic characteristics, traffic sent by the tenant VMs also contains some other characteristics, e.g. flow duration. But these characteristics do not affect the CPU utilization per unit time, so we do not consider them in the modeling. But in practical scenarios, the forwarding procedure is often more complicated. For example, some packets need to be modified when executing the actions after the *classification* stage, which consumes more CPU resources. These additional operations are independent of the stages we are modeling, and the CPU consumption can be added separately.

C. Impact of deployment issues

When deploying multiple VMs on the same physical server, some deployment configurations will influence the CPU consumption in packet forwarding. These deployment issues in the case of multi-tenancy scenario include: VM memory location (on which NUMA nodes [33], [34] of the physical server), the number of VMs on the same server and the number of IO-dedicated CPU cores. As these factors are mainly independent with each other, the influence can be expressed as $\prod R_i * C_{single}$, where R_i represents the growth rate of CPU consumption under the influence of each factor.



(a) CPU usage - VM location.

(b) CPU usage increase ratios - one core.

(c) CPU usage increase ratios - two cores.

Fig. 6. The impact of different deployment situations on the relationship between CPU consumption and bandwidth. The "M cycles/s" in Figs means million cycles/s.

VM memory location. The influence of NUMA architecture on memory access widely exists in today's commercial servers, so we need to evaluate it. We have described the CPU information at the Section II-B, and there are 4 NUMA nodes on our servers. In the experiment setting, the vSwitch's IO-dedicated CPU cores and memory are located on NUMA node 0. So the memory on node 1, 2 and 3 requires the CPU cores in vSwitch to access memory across nodes, which is slower than accessing to the memory on node 0. At the same forwarding rate, the CPU cycles required by the forwarding task of VM deployed on each NUMA node are shown in Fig. 6(a). It can be seen the VMs on node 1, 2 and 3 need 40% more CPU cycles than the VM on node 0 to complete the forwarding task, and that is mainly due to the increase of $C_{ingress}$ by memory access across nodes. So for the VMs on node 1, 2 and 3, the coefficient R in this factor is 1.4.

Number of VMs. As the number of VMs grows, the competition on memory bus and cache will increase the CPU consumption of all VMs' three forwarding steps. As shown in Fig. 6(b), when the number of VMs is less than 8, the additional CPU consumption caused by competition for cache will greatly increase, while it will be almost the same after VM grows more than 10. The maximum and minimum curves in the figure show the CPU consumption increase ratio when deploying VMs on the same NUMA node (the worst case) and on different nodes (the best case). It is obvious that the former situation will lead to higher competition. The coefficient in this scenario is changeable and needs to be measured through actual experiments. For example, according to Fig. 6(b), if 4 VMs are deployed on the same NUMA node, the CPU resources consumed by each VM will be 1.14 times more than that under the single VM case. But if the 4 VMs are deployed on different NUMA nodes, the ratio changes to 1.07.

Number of CPU cores. Finally, we increase the number of IO-dedicated CPU cores to 2 in vSwitch and the result is shown in Fig. 6(c). Comparing Fig. 6(b) and Fig. 6(c), it can be found that using 2 logical cores for forwarding will consume about 1.47 times more CPU cycles than one logical core for forwarding in any case. It is mainly due to competition for locks in the code, e.g. the synchronization among multiple PMD threads.

Therefore, when considering the CPU cycles assigned to a particular VM in the practical environment, CSPs need to multiply the C_{single}, the necessary CPU cycles measured under the single-VM case, by all the increase coefficient recorded in the above experiment results. For example, if 4 VMs are deployed in NUMA node 1, and two IO-dedicated CPU cores are assigned to forward traffic for them in vSwitch, the coefficients to be multiplied under the above three deployment configurations are 1.4, 1.14 and 1.47, respectively, according to Figs.6 (a)-(c). Only when all these factors are considered, the CPU cycles allocated to each VM can really ensure its purchased bandwidth.

D. Modeling methodology

According to experiments and analysis, CSPs can build their own bandwidth-CPU models in their vSwitch platforms. When the tenant requirements and configuration information are given, the CPU resources required for the tenant's VM to achieve SLA network performance can be calculated. In the following, we will present the modeling procedure and the required information to guide CSPs to implement in the real environment.

Firstly, the CSPs need to perform measurements in advance to establish a bandwidth-CPU model as described in Section III-B and C. For the impact of traffic characteristics, some preset values can be selected for measurement and the results can be stored in a table. For example, the packet size can be $\{64, 128, 256, 512, 1024, 1500\}$, and the number of flows can be $\{$single, multiple$\}$. With this table, a particular input like (PPS = 10000, packet size = 1024, number of flow = single) will get a certain output C_{single}. Next, for the deployment configurations, CSPs can also use the experiments to get the corresponding configuration and its coefficient R_i, and store it in tables. As the example shown in Section III-B, given the input (the number of VMs = 4), we will get a coefficient R_i as 1.14.

After the measurement-based model is built, in the second step, the CSPs rely on two types of information to form the inputs of the model when deploying VMs: the traffic characteristic preference from the tenant's choice and VM deployment configurations. For the three traffic characteristics, they can be included in the SLA and tenant can choose them when purchasing the VM. As the iMIX traffic [35], [36] represents an average level of all tenants' traffic, and CSPs can also set its characteristics as the default values to meet most tenants' requirements. For deployment configurations, CSPs can easily detect them. But as they may change frequently

with the creation and deletion of VM instances, that requires CSPs to change the inputs in real time. Then according to the formula $\prod R_i * C_{single}$, the required CPU resources for each VM to achieve SLA network performance can be calculated based on the tables in the first step.

IV. C2QoS DESIGN

Based on the bandwidth-CPU modeling methodology developed in Section III, we are able to design C2QoS strategy. The premise of C2QoS is that the number of VMs to be deployed is in accordance with the resources on physical servers and there are no overprovision. As each VM's required IO-dedicated CPU resources can be calculated based on the model, we define that under the C2QoS strategy, the deployment of VMs should follow two rules: the sum of all VMs' purchased bandwidth should not be more than the NIC bandwidth; and the sum of CPU resources that we calculate for each VM according to the model should not exceed the IO-dedicated CPU cores. Without these rules, the resource shortage will occur all the time, and no strategy can work.

In C2QoS, we propose the C2TB mechanism and the HBS mechanism to provide isolation enhanced rate limiting and hierarchical latency respectively. In this section, we will illustrate in detail the design.

A. CPU-cycle based token bucket mechanism

To guarantee VM bandwidth through the CPU resources apportionment, C2TB needs two steps: allocating the IO-dedicated CPU resources to particular VMs; using the allocated CPU resources to strictly limit the forwarding rate.

1) CPU resources allocation: Firstly, we construct a new kind of token bucket for each VM. Different from the traditional token bucket algorithms that use the bits or number of packets as tokens, the tokens in C2TB represent the remaining usable IO-dedicated CPU cycles of each VM. The token generation rate of each VM is the IO-dedicated CPU cycles/s allocated to it. We use C_{alloc} to indicate the CPU cycles/s required by each VM to achieve purchased bandwidth. Using the modeling methodology in Section III, we can set the token generation rate to the fit value of C_{alloc}, and it can strictly ensure tenants' purchased bandwidth in practice.

Meanwhile, besides the C_{alloc}, the idle part of the IO-dedicated CPU resources also need to be entirely allocated to VMs for MIN-MAX bandwidth allocation policy [21], [37], which is widely used in industry. An example of the MIN-MAX bandwidth guarantee under C2TB is shown in Fig. 7. The MIN bandwidth (the purchased bandwidth) is ensured by only assigning basic C_{alloc} to the particular VM, while the MAX bandwidth is obtained by assigning the C_{alloc} plus C_{idle}, which means the idle CPU cycles of the IO-dedicated CPU cores. In the example, we assume the CPU resources required to achieve 1 Gbps and 2 Gbps bandwidth are 0.2G cycles/s and 0.4G cycles/s, respectively. After their purchased bandwidth are guaranteed, there are still 1G cycles/s left idle on the 2.2GHz CPU core and it can be fully assigned. For the 4 VMs, they can share the C_{idle} of 1G cycles/s according to their weights to achieve their MAX bandwidth.

C2TB	VM1	VM2	VM3	VM4
weight	1	1	2	2
purchased bandwidth	1 Gbps	1 Gbps	2 Gbps	2 Gbps
MIN(cycles/s)	0.2G	0.2G	0.4G	0.4G
MAX(cycles/s)	0.2G + 0.16G	0.2G + 0.16G	0.4G + 0.32G	0.4G + 0.32G

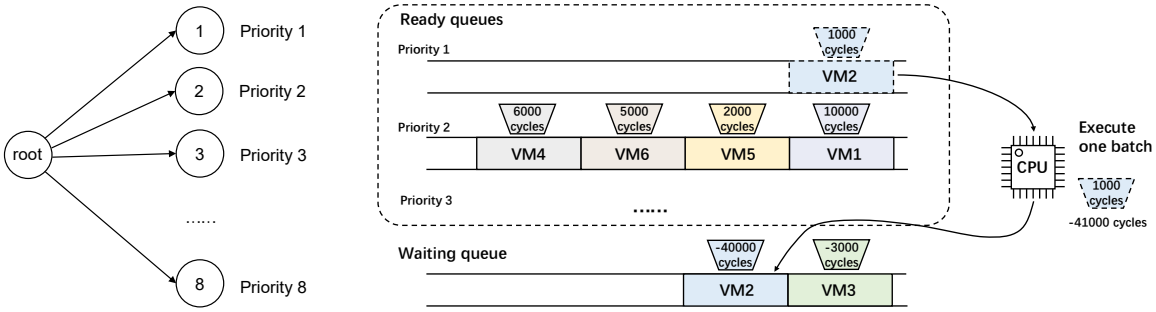
Fig. 7. C2TB bandwidth guarantee. C2TB guarantees the MIN and MAX bandwidth for VMs by assigning basic and all idle CPU cycles/s, respectively. In this example, 4 VMs share one 2.2Ghz CPU core for packet forwarding.

Therefore, the maximum CPU resources allocated to them are 0.36G cycles/s, 0.36G cycles/s, 0.72G cycles/s and 0.72G cycles/s, respectively.

After we entirely allocate the IO-dedicated CPU resources to VMs, another problem may occur that the allocated CPU resources may overflow from token bucket and be wasted, when VM's network load is light. In this condition, we also need to reallocate these unused CPU resources. For example, if one or two VMs in the Fig. 7 are sleeping and no traffic is generated, their tokens will always overflow and this part of the overflowed tokens can be redistributed to other VMs according to their weights. We present this reallocation logic in the token update function as shown in Algorithm 1. It will be called at regular intervals (its value is $invl$ in the function) to count the number of tokens in the token buckets for all VMs on this server. The variable $loop\_unused\_cc$ is used to collect all the overflowed CPU cycles in this loop and at last it will be stored in sum_unused_cc for reallocation next time to update tokens. In the loop, each VM will calculate the number of tokens generated at the rate of $C_{alloc} + C_{idle}$ within the $invl$ time. Then, each VM will get some overflowed CPU cycles based on its weight. We should point out that the $VM_i.weight$ in the algorithm is a ratio, calculated by dividing the VM's weight by the sum of all VMs' weights. So at last the tokens added into the token bucket contain two parts: the generated tokens and overflowed tokens. The last step in the loop is to check whether the number of tokens exceeds the bucket depth. If yes, the overflowed part needs to be taken out and saved for next token update.

Algorithm 1 C2TB token update function

```
1:  function C2TB_UPDATE_CALLBACK
2:      loop_unused_cc ← 0
3:      unused_cc ← 0
4:      for i = 0 → VM_cnt do
5:          gene_cc ← invl * (VM_i.C_alloc + VM_i.C_idle)
6:          unused_cc ← sum_unused_cc * VM_i.weight
7:          sum_unused_cc− = unused_cc
8:          VM_i.tokens += gene_cc + unused_cc
9:          if VM_i.tokens > bucket_depth then
10:             loop_unused_cc += VM_i.tokens − bucket_depth
11:             VM_i.tokens ← bucket_depth
12:         end if
13:     end for
14:     sum_unused_cc ← loop_unused_cc
15: end function
```



(a) Class-based architecture in HBS.

(b) The IO-dedicated CPU core processing logic under HBS.

Fig. 8. HBS design. The VMs are classified according to their priorities. Then the IO-dedicated CPU cores always pick up VM with the highest priority in the virtual "ready queues" to process batch I/O task. So the traditional undifferentiated execution in the polling running mode is replaced by the smart hierarchical batch I/O scheduling.

2) rate limiting: The last part of C2TB is to use the allocated CPU cycles in VMs' token buckets to limit their forwarding rates. In the traditional token bucket algorithm, the number or bits of packets are calculated during the batch I/O processing and the packets exceeding the available tokens will be dropped exactly. But in C2TB, since it does not know how many CPU cycles will be consumed, it is impossible to decide how many packets should be dropped. For efficiency, we adopt the following policy: we allow the number of tokens to be negative, and whether the tokens are greater than 0 determines whether this batch I/O processing task can be executed. For each VM, only if its tokens are greater than 0, it can send out a batch of packets, and the number of CPU cycles consumed is subtracted from its token bucket after the batch processing completes. As the CPU cycles used for each VM's packet forwarding tasks are reasonably assigned in C2TB, the VM bandwidth can be guaranteed with good isolation.

B. Hierarchical batch scheduling mechanism

As the existing scheduling mechanisms only work at the *egress* stage and cannot avoid the high latency of CPU resources contention in the other stages, we turn to think about scheduling the entire batch I/O processing procedure (including *ingress*, *classification* and *egress*) for VMs. In the field of CPU task scheduling, we find the batch task scheduling model in vSwitch is much closer to the works in [38], [39]. These works schedule tasks on CPU cores to ensure that tasks with light load will not be blocked too long by heavy load tasks. Though these works still fail to make it flexible to meet the hierarchical latency guarantee like HQoS [40], they inspire us to propose the HBS to break the undifferentiated execution in polling mode and schedule VMs' batch I/O tasks on the limited IO-dedicated CPU cores hierarchically.

The main goal of HBS is to achieve latency differentiation. In the field of task scheduling, we have chosen a useful scheduling mechanism — the priority queuing, which has been widely used in operation systems and traffic control in datacenter [41]–[46]. The batch processing task in our environment is to forward packets from VM to NIC, so using priority queue to reduce the waiting delay of high-priority tasks is equivalent to reducing the delay of packets inside them. Next we introduce the HBS design in detail.

1) class-based priority queues: The HBS design is shown in Fig. 8. To achieve hierarchical latency guarantee in HBS, as shown in Fig. 8(a), VMs are classified into 8 classes according to their priorities. The priority of each VM is determined by CSPs, e.g. the latency-sensitive services such as web or video can be set higher priority. These priorities will affect the order in which they are scheduled to perform batch I/O processing. On the data path shown in Fig. 8(b), all VMs' virtual structures are placed in virtual queues, and there are two kinds of queues: waiting queue and ready queue. As the C2TB allows CPU to skip VMs with tokens less than 0, we put these VMs that should be skipped into the waiting queue. The VMs with tokens greater than 0 are queued in the corresponding ready queues according to their classes. The IO-dedicated CPU cores will only poll and dequeue the VM structures in the ready queues and do batch I/O forwarding tasks.

In the ready queues, to ensure that VMs in higher priority queues have lower latency, the higher priority queue has absolute execution privileges than the lower priority queue. For a queue with priority N, it will be executed dequeue operations only when there is no items in all $N-1$ queues with higher priorities. So in this case, the PMD threads poll each queue and perform batch execution according to the priority level. For example, in the case shown in Fig. 8(b), although the number of tokens in VM2 is the smallest among the VMs in the ready queues, VM2 will be dequeued and forwarded one batch of packets firstly because it has the highest priority. After the batch processing, VM2 is placed into the waiting queue for it has consumed 41000 tokens and its available tokens are negative. To ensure fairness that VMs in the same queue have similar latency, each virtual queue in the HBS follows first-in-first-out (FIFO) policy.

2) worst latency: With hierarchical execution privileges, the worst latency of VMs in each queue can be guaranteed and calculated. Although it is almost impossible to guarantee a specific value of the latency for each VM in software forwarding due to the uncertain hardware processing capacity, we still can guarantee the worst latency of VMs in each queue under HBS. We assume a case that the number of VMs in all 8 ready

queues is $\{N_1, N_2, N_3, \cdots, N_8\}$, respectively. The time used for one batch processing is c. So the worst-case latency of VMs in these queues is $\{N_1*c, (N_1*k_1+N_2)*c, (N_1*k_1+N_2*k_2+N_3)*c, \cdots, (N_1*k_1+N_2*k_2+\cdots+N_7*k_7+N_8)*c\}(k_i \geq 1)$. The variable k_i we use in this formula means that when a low-priority VM sends out one batch of packets, the higher-priority VMs may send out several batches of packets. So compared with original sequential execution that each VM equally suffers the worst $\sum N_i * c$ latency, HBS can provide hierarchical worst latency guarantee for VMs with different requirements. That helps CSPs formulate more flexible SLA policies based on the tenants' latency sensitivities.

3) starvation avoidance: The last but most common problem in the HBS is how to avoid the tasks starvation, which is the inherent problem in priority-based scheduling mechanisms. In HBS, the starvation will occur in two situations: 1) The first one is that the low-priority VMs cannot achieve their purchased bandwidths when we prefer to use many resources to forward traffic for VMs with higher priorities. But in fact, when CPU resources are strictly allocated and isolated in C2TB, there is no case that the bandwidth of low-priority VMs is squeezed by others. 2) The second one will happen when many high-priority VMs have no traffic, but the CPU will still give priority to them and consume the allocated cycles. In this case, the low-priority VMs can get better network performance, but they still need to wait for CPU cores wasting time on the idle VMs. To avoid this, we allow the HBS to hold a dynamic priority for each VM. When a VM has no traffic to send during several consecutive batch I/O processing loops, the priority of this VM will be gradually dropped. But once it is found that the VM sends traffic again, it will be directly adjusted to the original priority.

V. Implementation

According to the design in Section IV, we implement the C2QoS strategy in the OVS-DPDK platform. As shown in Fig. 9, We modified the PMD thread's main loop function and the original port ingress policy, which are implemented by Single-Rate-Three-Color-Marker (srTCM) in the *ovs-vswitchd* module [17]. For each PMD thread, it has an independent HBS module to manage several VMs that it needs to be responsible for packet forwarding. Different PMD threads will not have access to each other's priority queues, so there is no competition and lock issues. The original sequentially polling running mode in the main loop of PMD threads is replaced by HBS that finds VM with highest priority in the ready queues to execute batch I/O processing. The batch I/O processing logic of PMD threads has not changed, but before each batch processing, the rate limit strategies on VM ports are replaced with C2TB.

In order to make it easier for the CSPs to configure the VM's rate limiting and scheduling parameters on the command line, we also add two new commands to the OVS-DPDK. These commands realize our C2TB and HBS configurations through *ovs-vsctl* module as shown in Fig. 9. For example, we can use command "*ovs-vsctl set interface vhost-user-1 ingress_policing_cpucycles=10000*" to allocate VM1,

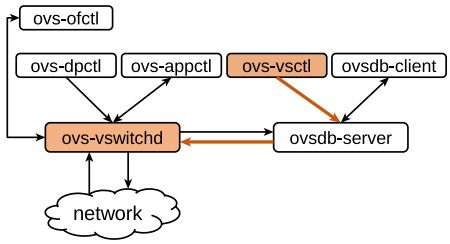


Fig. 9. Modifications to original OVS-DPDK. We only add the C2QoS strategy to the *ovs-vswitchd* module and use *ovs-vsctl* to control it. Less than 300 lines of code are added into *ovs-vswitchd* module.

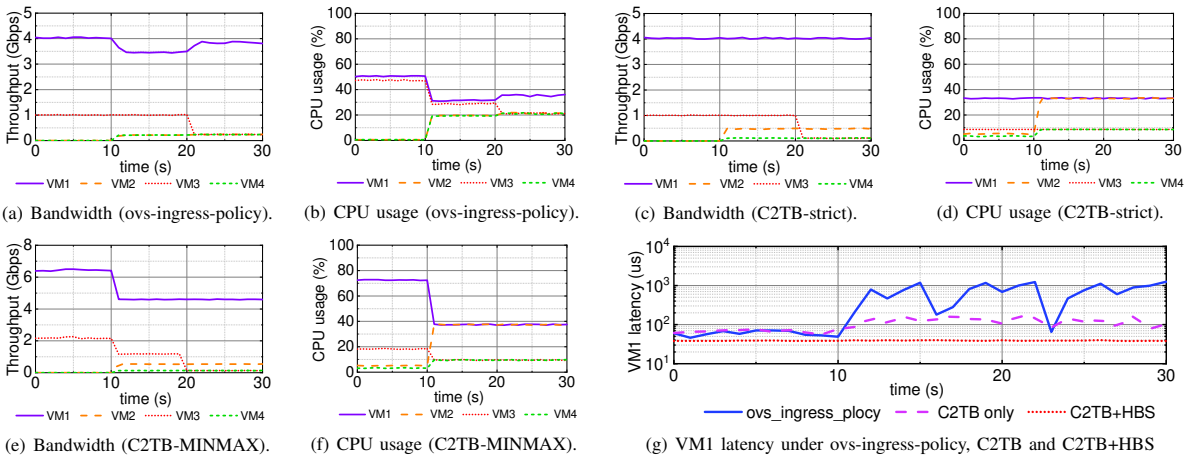
connected to "vhost-user-1" port, with 10000 CPU cycles per second for packet forwarding. The command "*ovs-vsctl set Interface vhost-user-1 options:priority=1*" can set VM1's priority as 1 in HBS scheduling. It should be noted that the C2QoS is a kind of general strategy, and in our implementation it can be used on all kinds of virtual ports such as "*vport*" and "*dpdk*" ports. All of these modifications require no more than 300 lines of code, which is easy to realize and meanwhile will not affect the original functions.

The biggest challenge in realizing C2TB is that it requires frequent measurement and calculation of CPU consumption in packet forwarding procedure, which will cause great impact on vSwitch's forwarding performance without lightweight implementation. We use the *rdtsc* instruction [47] to solve it. The instruction *rdtsc* is to get CPU cycles from booting by reading the value in registers, so it has almost no overhead and can be widely used in data path. Another overhead comes from maintaining queues in HBS, and is undertaken by another manager thread. If it runs in busy polling mode, our strategy will consume one more entitle CPU core than the native OVS-DPDK, which is unacceptable. So we set the manager thread to be woken up every 50us to update the token number of each VM. The wake-up interval is a kind of trade-off that can be set to meet different needs. For example, setting a longer interval can reduce the additional CPU usage but will face a decrease in scheduling accuracy, while a smaller interval will consume more CPU resources. In our implementation here, we set it as the average time used for one batch I/O processing task to achieve the trade-off between accuracy and CPU consumption. The effectiveness and overhead of C2QoS will be evaluated in Section VI.

VI. Evaluation

The main contribution of C2QoS is to ensure the bandwidth and latency of VMs on the physical server, so in this section, we evaluate the VM network QoS guarantee under C2QoS and the OVS-DPDK existing "ovs-ingress-policy" QoS strategy. Our experiments include the following aspects:

- Bandwidth and latency guarantee tests: Comparing the VM's TCP bandwidth and latency guarantee of C2QoS with that of ovs-ingress-policy.
- Accuracy: Measuring the accuracy of rate limiting in C2TB, and the latency levels of different priority queues in HBS.



(a) Bandwidth (ovs-ingress-policy). (b) CPU usage (ovs-ingress-policy). (c) Bandwidth (C2TB-strict). (d) CPU usage (C2TB-strict).

(e) Bandwidth (C2TB-MINMAX). (f) CPU usage (C2TB-MINMAX). (g) VM1 latency under ovs-ingress-policy, C2TB and C2TB+HBS

Fig. 10. TCP bandwidth and latency evaluation. (a)-(f) are the experimental results of TCP bandwidth tests, which are constructed by 4 VMs with 4 Gbps, 4 Gbps, 1 Gbps, and 1 Gbps purchased bandwidth respectively. (a)-(b) show TCP bandwidth and CPU usage under ovs-ingress-policy, while (c)-(d) depict that of under C2TB-strict and (e)-(f) exhibit C2TB-MINMAX results. (g) shows VM1 latency under the three strategies.

- Application experiments: Comparing the throughput of Ftp [48] server and response latency of Nginx [49] server under the C2QoS and ovs-ingress-policy.
- CPU overhead: Measuring the additional CPU overhead that C2QoS brings to OVS-DPDK.

The hardware and platform configurations are the same as described at the Section II-B.

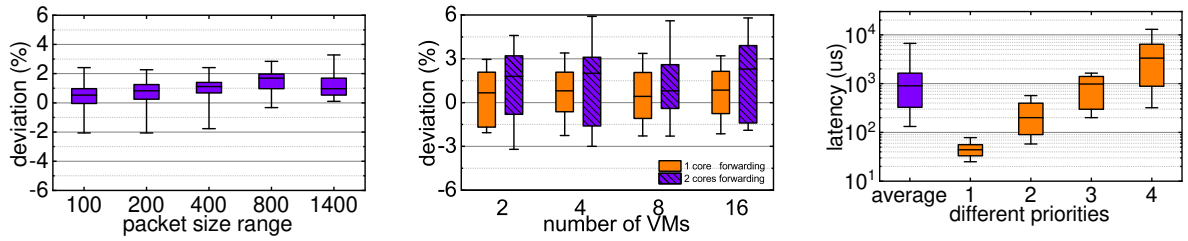
A. TCP bandwidth and latency

In this experiment, we use iperf [50] and qperf [30] tools to evaluate VMs' TCP bandwidth and latency. We launch 4 VMs with 4 Gbps, 4 Gbps, 1 Gbps, and 1 Gbps purchased bandwidth, respectively, and use one dedicated CPU core in OVS-DPDK for forwarding. The purchased bandwidths are guaranteed based on the SLA that sets the preferred packet size to 1024 byte and the number of flows to 1. According to our pre-measured model (as described in Section III), when deploying 4 VMs on NUMA node 1, the VMs with (4 Gbps, 1024-byte packet size, single flow) need 0.792G cycles/s (36%) CPU resources for packet forwarding, and the VMs with (1 Gbps, 1024-byte packet size, single flow) need 0.198G cycles/s (9%) CPU resources.

In our benchmark setting, VM1 acts as a well-behaved tenant and sends 1024-byte packets at the maximum rate all the time while the other 3 VMs behave as malicious users or noisy neighbors. In order to more clearly show the difference between C2TB and the original rate limiting method in OVS-DPDK, we design such a circumstance: 1) In the first 10 seconds, VM1 and VM3 send traffic with 1024-byte packets. The VM2 and VM4 are sleeping and generate no traffic. 2) From 10th second to 20th second, VM1 and VM3 keep sending traffic with 1024-byte packets, while VM2 and VM4 send traffic with 64-byte packets. 3) In the last 10 seconds, VM1 still sends traffic with 1024-byte packets, while VM2, VM3 and VM4 send traffic with 64-byte packets at the maximum speed. The TCP bandwidth and CPU consumption of the 4 VMs under three strategies are shown in Figs. 10 (a)-(f). In these figures, "C2TB-strict" means we only assign each VM the fixed C_{alloc} for packet forwarding, and the "C2TB-MINMAX" supports to entirely allocate C_{idle} and unused CPU resources to VMs for completing MIN-MAX bandwidth allocation.

Figs. 10 (a)-(b) show the results under default ovs-ingress-policy. It can be seen that VM1 and VM3 work well and keep their purchased bandwidth within the first 10 seconds. But from the 10th second, VM2 and VM4 start sending 64-byte packets, and they both compete and occupy 20% of the CPU resources. As shown in Fig. 10 (b), the behavior of VM2 and VM4 severely squeezes the CPU resources that originally used by VM1 and VM3. As a result, VM1 bandwidth drops by 12%. But for VM3, although the available CPU resources of VM3 have been squeezed, they are still enough to support VM3's purchased bandwidth. In the last 10 seconds, an interesting thing comes that the bandwidth and CPU usage of VM1 increase with the VM3 changes packet size from 1024 byte to 64 byte. But VM3 does not benefit from the change of traffic characteristics. We print all the log information and find this is caused by the running mode of OVS-DPDK. The sequential execution in the PMD thread makes it equal in the number of batch I/O processing loops performed for each VM per second. As VM3 reduces the packet size, the number of batch processing loops of each VM per second is increased. For the VM1, the increase in the number of batch processing loops per second means that more packets can be sent per second (before the bandwidth reaches the rate limiting threshold), which increases bandwidth and CPU consumption. So under OVS default BPS-based rate limiting strategy, the behavior of the tenants will cause unpredictable CPU allocation, and cannot guarantee VM bandwidth.

The bandwidth and CPU consumption of each VM under the C2TB-strict strategy are shown in Figs. 10 (c)-(d). According to the modeling results, the CPU resources that we should allocate to the 4 VMs are 36%, 36%, 9% and 9%, respectively.



(a) Mixed-size packets impacts. (b) Deployment impacts. (c) Hierarchical latency.

Fig. 11. Accuracy evaluation. (a)-(b) show the deviations of the rate limiting mechanism in C2QoS under different situations. (c) shows the different level of latency guarantee in HBS.

We analyze the bandwidth of each VM separately. For VM1, since its behavior keeps unchanged, its bandwidth is stable and keeps at 4 Gbps by using 36% of the CPU resources for forwarding all the time. For VM3, changing the packet size to 64 bytes in the last 10 seconds can only reduce its own bandwidth. For the two attackers (i.e. VM2 and VM4), in the event of sending 64-byte packets, the CPU resources allocated to them can only achieve very low bandwidth, and they cannot interfere with other VMs by competing for more CPU resources. It should be noted that in the last 10 seconds, VM3 and VM4 get extremely low bandwidth, and it seems they face starvation. But in fact, that is exactly what we want to achieve. The problem of bandwidth isolation is caused by these VMs using special traffic characteristics, rather than their preferences, to compete for more IO-dedicated CPU resources. The solution in C2TB is to let these "noisy" VMs only affect their own network performance through restricting CPU consumption for each VM. Compared with the ovs-ingress-policy, CBTB-strict can guarantee well-behaved VMs' bandwidth and eliminate the CPU resources competition.

While providing good isolation, the C2TB-strict still causes a waste of CPU resources in the vSwitch, as shown in Fig. 10(d), nearly 75% of CPU resources are wasted in the first 10 seconds. So the C2TB-MINMAX is used to solve this kind of waste. The main difference with C2TB-strict results happen in the first 10 seconds, in which VM1 and VM3 make full use of all idle CPU resources on the server and achieve higher bandwidth (6.3 Gbps and 2.1 Gbps) than the bandwidth they purchased (4 Gbps and 1 Gbps). So this rate limiting method also has good robustness while guaranteeing the network QoS of VMs.

On the aspect of TCP latency, we separately use qperf to measure the VM1 latency under the ovs-ingress-policy, "C2TB only", and C2TB+HBS strategies. The results are shown in Fig. 10(g). Under the ovs-ingress-policy, in the first 10 seconds, only VM3 competes with VM1 for the CPU core to do batch I/O processing tasks, which leads to a slight increase in VM1 TCP latency. In the following 20 seconds, the latency of VM1 becomes unstable and increases significantly (more than 1ms in the worst case) due to the competition of the other three VMs. Compared to ovs-ingress-policy, "C2TB only" can reduce part of the additional latency of VM1 by skipping ports with negative tokens. Another reason for the lower latency under C2TB is that it keeps the packets not being sent inside the VM, forming a "back-pressure"[2] to the senders and adjusting the sending rate of TCP protocol stack in VMs. But the latency under "C2TB only" is still unstable. With HBS, we set VM1 to be placed in Priority 1 queue which ensures VM1's forwarding tasks always to be executed firstly. The results show that the VM1 latency under C2TB+HBS is close to the native performance and is not affected by other VMs.

Therefore, with these experiments, C2QoS can provide good isolation from the CPU level. That enables C2QoS to provide tenants with good network SLA performance guarantees under the conditions of CPU resources competition and variable processing capacity in vSwitch.

B. Accuracy

In addition to the advantages on isolation, we also need to evaluate the accuracy of C2QoS. Based on the functions of C2TB and HBS, the accuracy is reflected in two aspects: the accuracy of rate limiting and the hierarchy of the worst latency.

We first evaluate the deviations of C2TB under mixed-size packets and its results are shown in Fig. 11(a). In this experiment, we use pkt-gen in VM to send packets with mixed sizes but keep a fixed average size. Then we set the average packet size parameter for C2TB according to the modeling methodology in Section III-D. We can see that the range of the packet size has little effect on the accuracy. Most of the results show that the deviation is greater than 0, which means that in most cases, we can guarantee that the VM's available bandwidth is greater than or equal to its purchased bandwidth. On average, the deviation of C2TB under mixed packet size is between (-2%, 3%).

In Fig. 11(b), we increase the number of VMs and deploy them on every NUMA node to compare their real bandwidth with their purchased bandwidth. The increase in the number of VMs did not have large impacts on the deviation. But it can be concluded that the more variables introduced in the modeling, the greater the deviations are. When using only one CPU core for forwarding in vSwitch, the deviation is between (-2%, 4%). But in the case of using two cores for forwarding,

[2]Back-pressure is a kind of congestion control. If the packets will be dropped in some points of the forwarding path, the sender should be informed to reduce the sending rate to avoid the waste of resources.

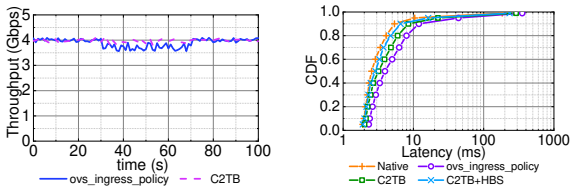
resource competition becomes even more unpredictable, so the deviation has almost doubled to (-3%, 6%). Although the accuracy of the C2TB is incomparable to the traditional precise rate limiting methods, the CSPs believe the rate limiting under software forwarding does not need to be so precise and occupy much resources [21]. So the deviations of C2TB are acceptable.

To evaluate what kind of latency levels can HBS provide, we run 16 VMs belonging to 4 priorities on one dedicated CPU core for forwarding, and evaluate their TCP latencies under the case that all VMs are sending traffic concurrently. The results are shown in Fig. 11(c). The "average" in this figure is the average latency of all 16 VMs under the C2TB mechanism only. In this case, since all VMs need to wait for batch I/O processing, their latencies are high and unstable. In HBS mechanism, we can see that although the latencies of VMs in different priority queues have intersections, the latency levels in most cases are obviously different. The latency of VMs with priority 1 and 2 is less than average latency, while the latency of VMs with priority 3 and 4 is much more worse than the average. Another fact is that the latency distribution of high-priority VMs is very concentrated. But as the VM priority decreases, the frequency of these VMs' batch I/O processing will be more uncertain, which contributes to the high discreteness. The different latency levels brought by HBS will be useful when providing differentiated services for tenants.

C. Application results

To make it more practical, we consider some common applications on the public cloud. For example, the latency-sensitive VMs (such as website and video services) compete with bandwidth-sensitive VMs (such as online disks) for vSwitch forwarding resources on the same physical server. So we evaluate the bandwidth of Ftp server and response latency of Nginx server in this experiment. We choose Nginx not only because it is a latency-sensitive service, but also because of its special traffic characteristics. The traffic of Nginx is usually composed of small packets, and the five-tuple segments of these packets are discrete in the case of high concurrency. That will cause the more CPU consumption in *classification* stage (as described in Section III-B). For test configurations, 2 VMs with 4 Gbps bandwidth are deployed as Ftp servers and 2 VMs with 1 Gbps are deployed as Nginx servers. The Ftp servers keep sending traffic while the Nginx servers bear pressure test during 30th-70th seconds using wrk [51] tool.

The Ftp bandwidth is shown in Fig. 12(a), the Nginx servers' traffic during 30th-70th seconds causes a bandwidth drop of about 11% on the Ftp servers under ovs-ingress-policy, while C2TB strictly guarantees the bandwidth of Ftp servers all the time. For the latency in the Nginx pressure test, we obtain the request response time distribution in Fig. 12(b). Under ovs-ingress-policy, the response time of Nginx requests is doubled compared to native performance. When only using C2TB, 50% additional latency is reduced by skipping ports with tokens less than 0. But with the C2QoS containing both C2TB and HBS, the additional latency is reduced by more



(a) Ftp bandwidth. (b) Nginx response time.

Fig. 12. Application performance results. Nginx latency and ftp bandwidth are evaluated. 2 VMs with 4 Gbps are deployed as the Ftp server, and 2 VMs with 1 Gbps are deployed as the Nginx server. We perform the pressure test on Nginx VMs at 30-70s.

than 80% and these Nginx servers achieve almost the native performance. Therefore, the C2QoS can ensure the network performance of both latency-sensitive and bandwidth-sensitive services while sharing the same physical resources in vSwitch.

D. Overhead

As we added a new module to vSwitch, it may bring new overhead. The overhead mainly reflects on two aspects: the performance decrease and the additional CPU overhead.

For the first concern, in the single-VM and multi-VM experiments, the OVS-DPDK using C2QoS strategy has no performance drop compared with the original version. This is because we have not made big changes to the data path, and the additional function added to PMD threads only contains CPU cycles counting. The CPU cycles counting function is composed of *rdtsc* instruction [47], which occupies only several cycles and has very little effect on forwarding performance.

For the additional CPU overhead, we tested the CPU consumption of C2QoS. On the IO-dedicated CPU cores, only 0.018% CPU usages are used for C2QoS. This part of CPU usages will not go up with the increase in the number of VMs because it is added in the batch I/O processing of all VMs' ports. Besides the overhead on IO-dedicated CPU cores, the manager thread's CPU overhead also needs to be considered. When deploying 28 VMs, 2.08% more CPU usages are used for tokens counting and queues managing. Moreover, the CPU usages of the manager thread can also be reduced by sacrificing accuracy and extending the wake-up interval. Although it is a trade-off, from our experimental results, the additional CPU resources consumed by the manager thread will not be too much (no more than 3%). So the additional CPU overhead in C2QoS is also acceptable for cloud platforms.

VII. Discussions

The evaluation shows that C2QoS achieves good results in ensuring the stability of tenant's network performance compared to the traditional strategy, but there are still some issues that need to be carefully addressed for real-world deployment. In this section, we first discuss the applicable scenarios of C2QoS, and under which scenarios it does not work well. Then, we provide our own solution.

The C2QoS strategy is designed for the vSwitch of the cloud platform. For cloud servers, if the forwarding capability of the vSwitch needs to reach the NIC bandwidth all the time,

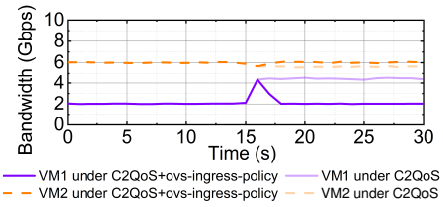


Fig. 13. The scenario where C2QoS fails and the effect of the handler. The light-colored lines show that when bottleneck comes to NIC, C2QoS fails to guarantee SLA. The dark-colored lines show that the traditional strategy can make up for the shortcomings of C2QoS in this case.

dozens of CPU cores shall be used by the vSwitch. Therefore, to maximize commercial benefits, the common practice is to allocate only several dedicated CPU cores to the vSwitch and ensure that the vSwitch's forwarding capability can reach the NIC bandwidth only in some usual cases. But on cloud, the tenant VMs are deployed with services that have diverse traffic characteristics. For example, many services like website and data transmission for IoT devices always send small packets and concurrent flows. That will consume most of the IO-dedicated CPU resources and drop the maximum vSwitch forwarding capacity. Thus the shortage and competition of the IO-dedicated CPU resources will become the norm. That is the problem that C2QoS can solve.

On the other hand, there is a potential extreme situation that the bottleneck may come to the NIC rather than the CPU cores. We assume that a VM is allocated IO-dedicated CPU resources according to the preferred traffic characteristics (512-byte packet size, multiple flows). But if it sends MTU (1518-byte) packets with a single flow, its throughput will rise and that may cause the vSwitch forwarding rate to exceed the upper limit of the NIC bandwidth. The C2QoS fails in such a condition, because it cannot solve the congestion and competition on the NIC. This situation is not common on the public cloud, so we can adopt a simple handler to deal with it. We use the original monitoring module that is inside the vSwitch to detect the forwarding rate of the vSwitch. If the throughput of the vSwitch reaches the upper limit of the NIC bandwidth, the vSwitch will be triggered to use the BPS/PPS-based rate limiting mechanism to assist C2QoS strategy for ensuring the performance of tenant networks.

We carry out an experiment to show this extreme situation. As shown in Fig. 13, the traffic characteristic preference of VM1 includes the factors of 2 Gbps, 128-bytes packet size, and the single flow, while the preference of VM2 contains the factors of 6 Gbps, 1024-byte packet size, and the single flow. Under C2QoS only, the 2 VMs work well for the first 15 seconds as they follow their preferences. But starting from the 15th second, VM1 starts to send 1500-byte packets. We can see that the throughput of VM1 rises sharply and competes with VM2 for the NIC bandwidth at the 16th second. This causes an 8% reduction in the bandwidth of VM2 (see the light-colored lines in Fig. 13). After adapting the handler, under the combined effect of C2QoS and the traditional rate limiting strategy, the bandwidth of the 2 VMs returns to normal after the 17th second (see the dark-colored lines in Fig. 13).

From this experiment, we can conclude that C2QoS is not the opposite of traditional QoS strategy. They can solve the performance isolation breakage problem in different scenarios. In practice, CSPs can flexibly choose the effective strategy depending on where the bottleneck or competition point is. In many cases, the two strategies can work together.

VIII. Conclusion

This paper focused on the VM network QoS on the cloud platform and addressed the key issue of IO-dedicated CPU resources management and apportionment in the vSwitch, which has been ignored by the existing QoS strategies. The competition among VMs for the IO-dedicated CPU resources in vSwitch seriously affects the network performance. In order to resolve the issue, this paper proposed C2QoS to apportion and schedule IO-dedicated CPU resources to VMs for network SLA guarantee. C2QoS consists of two mechanisms, C2TB and HBS. In C2TB, according to a measurement-driven bandwidth-CPU model, we limited VM's bandwidth by directly assigning CPU cycles to particular VM. To address the high additional latency issue brought by the undifferentiated execution, the HBS mechanism scheduled the VMs' entire batch I/O forwarding tasks on the IO-dedicated CPU cores, which provided hierarchical latencies for VMs according to sensitivities. The implementation on OVS-DPDK platform showed that compared with existing strategies, C2QoS eliminated the influence of CPU resource congestion on bandwidth and reduced the effect on latency by 80%.

References

[1] X. Wang, L. T. Yang, X. Chen, J. Han, and J. Feng, "A tensor computation and optimization model for cyber-physical-social big data," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 4, pp. 326–339, 2019.

[2] L. Ren, Y. Laili, X. Li, and X. Wang, "Coding-based large-scale task assignment for industrial edge intelligence," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2286–2297, 2020.

[3] B. Cao, X. Kang, J. Zhao, P. Yang, Z. Lv, and X. Liu, "Differential evolution-based 3-d directional wireless sensor network deployment optimization," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3594–3605, 2018.

[4] B. Cao, J. Zhao, P. Yang, P. Yang, X. Liu, and Y. Zhang, "3-d deployment optimization for heterogeneous wireless directional sensor networks on smart city," *IEEE Trans. Ind. Informatics*, vol. 15, no. 3, pp. 1798–1808, 2019.

[5] J. Qi, P. Yang, L. Newcombe, X. Peng, Y. Yang, and Z. Zhao, "An overview of data fusion techniques for internet of things enabled physical activity recognition and measure," *Inf. Fusion*, vol. 55, pp. 269–280, 2020.

[6] Z. Deng, P. Yang, Y. Zhao, X. Zhao, and F. Dong, "Life-logging data aggregation solution for interdisciplinary healthcare research and collaboration," in *15th IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, Liverpool, United Kingdom, October 26-28, 2015*, pp. 2315–2320, IEEE, 2015.

[7] A. K. S. Rajan, A. Feucht, L. Gamer, I. Smaili, and M. N. Devi, "Hypervisor for consolidating real-time automotive control units: Its procedure, implications and hidden pitfalls," *J. Syst. Archit.*, vol. 82, pp. 37–48, 2018.

[8] D. Firestone, "Vfp: A virtual switch platform for host sdn in the public cloud," in *Conf. on Networked Systems Design and Implementation*, 2017.

[9] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, *et al.*, "The design and implementation of open vswitch.," in *Conf. on Networked Systems Design and Implementation*, 2015.

[10] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, *et al.*, "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *Conf. on Networked Systems Design and Implementation*, 2018.

[11] P. Kumar, N. Dukkipati, N. Lewis, *et al.*, "Picnic: predictable virtualized nic," in *Conf. of the ACM Special Interest Group on Data Communication*, 2019.

[12] V. Addanki, L. Linguaglossa, J. Roberts, and D. Rossi, "Controlling software router resource sharing by fair packet dropping," in *IFIP Networking Conference and Workshops*, 2018.

[13] "Data plane development kit." https://www.dpdk.org.

[14] "Open vswitch." http://www.openvswitch.org/.

[15] P. P. Tang and T. . C. Tai, "Network traffic characterization using token bucket model," in *IEEE Conf. on Computer Communications*, 1999.

[16] "Hierarchy token bucket in linux kernel." https://www.man7.org/linux/man-pages/man8/tc-htb.8.html.

[17] "Dpdk traffic metering." http://doc.dpdk.org/guides/prog_guide/traffic_metering_and_policing.html.

[18] J. R. Davin and A. T. Heybey, "A simulation study of fair queueing and policy enforcement," in *Conf. of the ACM Special Interest Group on Data Communication*, 1990.

[19] J. C. R. Bennett and Hui Zhang, "Wf/sup 2/q: worst-case fair weighted fair queueing," in *IEEE Conf. on Computer Communications*, 1996.

[20] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Conf. of the ACM Special Interest Group on Data Communication*, 1995.

[21] K. To, D. Firestone, G. Varghese, and J. Padhye, "Measurement based fair queuing for allocating bandwidth to virtual machines," in *work. on Hot topics in Middleboxes and Network Function Virtualization*, 2016.

[22] F. Checconi, L. Rizzo, and P. Valente, "Qfq: Efficient packet scheduling with tight guarantees," *IEEE/ACM Transactions on Networking*, vol. 21, no. 3, 2012.

[23] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, *et al.*, "Programmable packet scheduling at line rate," in *Conf. of the ACM Special Interest Group on Data Communication*, 2016.

[24] "Rfc2697(srtcm)." https://www.rfc-editor.org/rfc/rfc2697.html.

[25] "Rfc2698(trtcm)." https://www.rfc-editor.org/rfc/rfc2698.html.

[26] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *USENIX Annual Technical Conference*, 2012.

[27] "qdisc." https://lwn.net/Articles/564978/.

[28] A. Saeed, N. Dukkipati, V. Valancius, *et al.*, "Carousel: Scalable traffic shaping at end hosts," in *Conf. of the ACM Special Interest Group on Data Communication*, 2017.

[29] A. Saeed, Y. Zhao, N. Dukkipati, E. Zegura, *et al.*, "Eiffel: efficient and flexible software packet scheduling," in *Conf. on Networked Systems Design and Implementation*, 2019.

[30] "qperf." https://linux.die.net/man/1/qperf.

[31] Y. Ye, J. Haiyang, W. Yulei, L. Yilong, L. Xing, and X. Gaogang, "C2qos: Cpu-cycle based network qos strategyin vswitch of public cloud," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021.

[32] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Conf. of the ACM Special Interest Group on Data Communication*, 1999.

[33] "What is numa." https://www.kernel.org/doc/html/latest/vm/numa.html.

[34] "Numa locality." https://www.kernel.org/doc/html/latest/admin-guide/mm/numaperf.html.

[35] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM SIGCOMM Conference on Internet Measurement*, 2010.

[36] "Imix genome: Specification of variable packet sizes for additional testing." https://tools.ietf.org/html/rfc6985.

[37] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, "A flexible model for resource management in virtual private networks," in *Conf. of the ACM Special Interest Group on Data Communication*, 1999.

[38] W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, and T. Wood, "Flurries: Countless fine-grained nfs for flexible per-flow customization," in *Conf. on emerging Networking EXperiments and Technologies*, 2016.

[39] J. Mace, P. Bodik, M. Musuvathi, *et al.*, "2dfq: Two-dimensional fair queuing for multi-tenant cloud services," in *Conf. of the ACM Special Interest Group on Data Communication*, 2016.

[40] I. Stoica, H. Zhang, and T. S. E. Ng, "A hierarchical fair service curve algorithm for link-sharing, real-time and priority services," in *Conf. of the ACM Special Interest Group on Data Communication*, 2016.

[41] T. K. Wignall, "Priority queuing systems with and without feedback," *Operations Research*, vol. 21, no. 3, pp. 764–776, 1973.

[42] A. Derbala, "Priority queuing in an operating system," *Computers & Operations Research*, vol. 32, pp. 229–238, 2005.

[43] P. Chuprikov, S. Nikolenko, and K. Kogan, "Priority queueing with multiple packet characteristics," in *IEEE Conf. on Computer Communications*, 2015.

[44] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Conf. on Networked Systems Design and Implementation*, 2015.

[45] J. Martinez, D. Dasari, A. Hamann, I. Sañudo, and M. Bertogna, "Exact response time analysis of fixed priority systems based on sporadic servers," *J. Syst. Archit.*, vol. 110, p. 101836, 2020.

[46] N. Guan, W. Yi, Q. Deng, Z. Gu, and G. Yu, "Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling," *J. Syst. Archit.*, vol. 57, no. 5, pp. 536–546, 2011.

[47] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual*, 2019.

[48] "vsftpd." https://security.appspot.com/vsftpd.html.

[49] "Nginx." http://nginx.org/.

[50] "iperf." https://iperf.fr/.

[51] "wrk." https://github.com/wg/wrk.

Ye Yang is currently a Ph.D. student of Network Technology Research Center at Institute of Computing Technology, Chinese Academy of Sciences since 2016. His main research topics are virtualized network and high-performance packet processing.

Haiyang Jiang received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences (ICT, CAS), in 2014. He is currently an Associate Researcher at ICT, CAS. His research interests include NFV/SDN, high-performance packet processing and network security.

Yulei Wu (Senior Member, IEEE) is a Senior Lecturer with the Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, United Kingdom. He received the B.Sc. degree (First Class Honours) in Computer Science and the Ph.D. degree in Computing and Mathematics from the University of Bradford, United Kingdom, in 2006 and 2010, respectively. His expertise is on intelligent networking, and his main research interests include computer networks, networked systems, software defined networks and systems, network management, and network security and privacy. He is an Editor of IEEE Transactions on Network and Service Management, IEEE Transactions on Network Science and Engineering, IEEE Access, and Computer Networks (Elsevier) and. He is a Senior Member of the IEEE, and a Fellow of the HEA (Higher Education Academy).

Chunjing Han received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2018. She is a Senior Engineer at Institute of Information Engineering, Chinese academy of sciences. Her main research interests are network measurement and behavior analysis, cyberspace security and network big data.

Yilong Lv received the M.S. degree in computer science and technology from Wuhan University, in 2015. He is currently a Research and Development (R&D) Expert in Alibaba Cloud. His research area is concentrated on cloud network, vSwitch and network virtualization.

Xing Li received the M.S. degree in computer science and technology from Zhejiang University, in 2012. He is currently the Leader of Research and Development (R&D) team in Virtual Private Cloud (VPC) department of Alibaba Cloud. He has been deeply involved in the cloud computing industry for years, and focusing on network virtualization technologies in data centers. He is now actively seeking cooperation with academia to solve the complex problems encountered in practical and production environments.

Bowen Yang received the M.S. degree in communication and information systems from the University of Electronic Science and Technology, in 2015. He is currently a Research and Development (R&D) expert in Alibaba Cloud. His research area includes computer network, cloud network and virtualization technologies.

Serge Fdida is a Professor with Sorbonne University since 1995. He has been leading

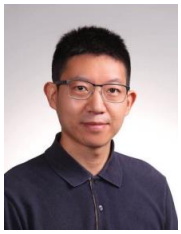
many research projects in High Performance Networking in France and Europe, notably pioneering the European activity on federated Internet testbeds. Currently, he is coordinating the French National Research Instrument FIT and the OneLab facility, two large test platforms on Future Internet technologies. He is a Distinguished ACM Member and an IEEE Senior member. He was one of the founders of the ACM Conext conference, general chair of ACM Mobicom 2015 and IEEE Infocom 2019. Serge Fdida has also developed a strong experience related to innovation and industry transfer, - he was the co-founder of the Qosmos company, - one of the active contributors to the creation of the Cap Digital cluster in Paris, and the President of the EIT Health French community.

Gaogang Xie (Member, IEEE) received the B.S. degree in physics and the M.S. and Ph.D. degrees in computer science from Hunan University in 1996, 1999, and 2002, respectively. He is currently a Professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences (UCAS), and the Vice President of CNIC. His research interests include Internet architecture, packet processing and forwarding, and Internet measurement.

Ye Yang

Haiyang Jiang

Yulei Wu



Chunjing Han

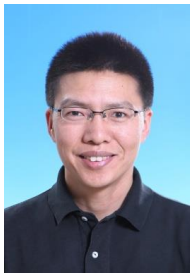
Yilong Lv

Xing Li

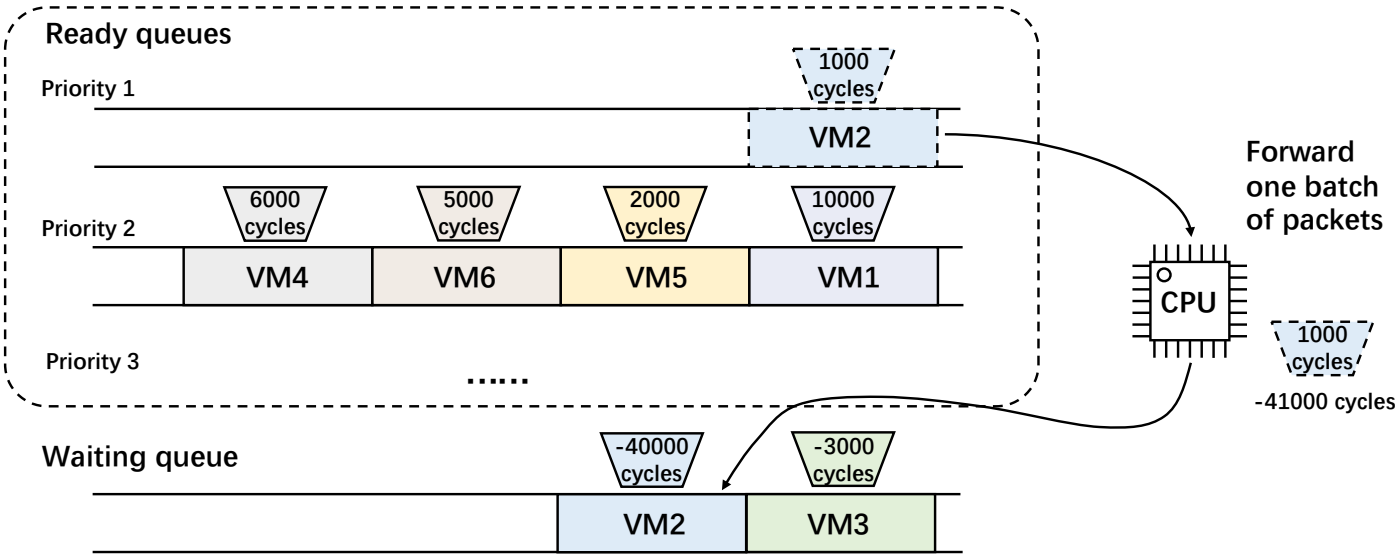
Bowen Yang

Serge Fdida

Gaogang Xie



- Tenant’s network performance is unpredictable due to resource competition in vSwitch.
- Building the relationship between network performance and resources.
- Apportioning resources to tenants for isolated and stable network performance.
- Providing stable network quality for tenants with only few additional resources used.


Ready queues
Priority 1
1000 cycles
VM2
Priority 2
6000 cycles
5000 cycles
2000 cycles
10000 cycles
VM4
VM6
VM5
VM1
Priority 3
.....
Forward one batch of packets
CPU
1000 cycles
-41000 cycles
Waiting queue
-40000 cycles
-3000 cycles
VM2
VM3

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: