Per-Client Network Performance Isolation in VDE-based Cloud Computing Servers

JAESOO LEE[1], JONGHUN YOO[1,*] YONGSEOK PARK[2] AND SEONGSOO HONG[1, 3]
[1] *School of Electrical Engineering and Computer Science, Seoul National University, Republic of Korea*
{jslee, jhyoo, sshong}@redwood.snu.ac.kr
[2] *Digital Media & Communications R&D Center, Samsung Electronics, Co., Ltd*
yongseok.park@samsung.com
[3] *Department of Intelligent Convergence Systems,*
Graduate School of Convergence Science and Technology, Seoul National University, Republic of Korea

In a cloud server where multiple virtual machines owned by different clients are co-hosted, excessive traffic generated by a small group of clients may well jeopardize the quality of service of other clients. It is thus very important to provide per-client network performance isolation in a cloud computing environment. Unfortunately, the existing techniques are not effective enough for a huge cloud computing system since it is difficult to adopt them in a large scale and they often require non-trivial modification to the established network protocols. To overcome such difficulties, we propose per-client network performance isolation using VDE (Virtual Distributed Ethernet) as a base framework. Our approach begins with per-client weight specification and support client-aware fair share scheduling and packet dispatching for both incoming and outgoing traffic. It also provides hierarchical fairness between a client and its virtual machines. Our approach supports full virtualization of a guest OS, wide scale adoption, limited modification to the existing system, low run-time overhead and work-conserving servicing. Our experimental results show the effectiveness of the proposed approach. Every client received at least 99.4% of its bandwidth share as specified by its weight.

Keywords: Network performance isolation; cloud computing; Virtual Distributed Ethernet (VDE); proportionally fair resource allocation

1. INTRODUCTION

System virtualization draws a great deal of research interest these days as it is actively cited as a key enabling technology for recently emerging computing paradigms such as cloud computing and the future Internet. One of the critical requirements of such paradigms is in a physical computing node which is capable of providing versatile and possibly virtual computing platforms for diverse classes of clients only using a fixed set of already heavily invested hardware resources. This leads to difficult technical

This article is an extended version of the paper that appeared in *Proceedings of the 13th International Workshop on Future Trends of Distributed Computing Systems* [1].

The work reported in this paper is supported in part by the Technology Innovation Program (No. 10036495) funded by the Ministry of Knowledge Economy (MKE, Korea), by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (No. 2011-0001572 and No. 2010-0027809), by Digital Media & Communications R&D Center, Samsung Electronics, Co. Ltd (No. 0421-20110016), by Advanced Institutes of Convergence Technology (AICT), Institute for Green Smart Systems and by Automation and Systems Research Institute (ASRI).

* Corresponding author

challenges such as efficient hardware resource sharing, fair resource allocation, and reliable and secure service provisioning. System virtualization technology has been brought up to directly address many of such technical issues. System virtualization enables multiple existing operating systems, often referred to as a guest OS, to share a physical machine by providing them with an illusion of exclusive access to imaginary hardware called a virtual machine (VM). As a result, it has been successfully used in enterprise and desktop computing domains for the cost-effective sharing and maintenance of computing resources.

System virtualization is often subdivided into CPU, memory and I/O virtualization since different types of hardware resources require distinct virtualization techniques. Virtual Machine Monitors (VMMs) such as Xen and the VMware ESX server provide mechanisms for CPU, memory and disk virtualization. In case of network, virtual networks such as Virtual Distributed Ethernet (VDE) [2] provide multiple abstracted networks from the same physical network infrastructure. Particularly, VDE realizes the packet forwarding, switching and routing functionalities of the virtual network fabric.

Unfortunately, ensuring the network performance of cloud computing applications has not been of much concern until recently although network performance metrics such as throughput, delay and packet loss rate show large variations with the bandwidth usage pattern of other co-residing VMs [3]. For instance, excessive traffic generated by a small group VMs owned by certain clients may well jeopardize the quality of service of other clients. It is thus very important to provide per-client network performance isolation in a cloud computing environment.

The fundamental steps to achieve it are in the minimization of network performance interference and providing adequate network resources. While the latter necessarily involves hardware changes at a huge scale, the former can possibly be put into effect with software modifications. With the increasing scale and complexity of the network, the performance isolation has become essential for optimal and fair usage of the constrained and shared network resources.

In this paper, we address the network performance isolation problem in the context of system and network virtualization for cloud computing. The network performance interference in the cloud network occurs when clients do not get the required amount of bandwidth due to excessive network bandwidth usage by other clients. For instance, some users of the Amazon EC2 servers have reported of pronounced variations in delay and throughput due to “noisy neighbors” who run highly network intensive loads [4]. As the performance of cloud computing applications is highly affected by the network performance, it is vital to ensure a client’s network bandwidth requirement. The client can either specify the network requirement in the form of shares, or if not directly specified, it can be interpreted by the cloud provider from its computing requirements.

In the literature, there are some techniques for network performance isolation, such as Transmission Control Protocol (TCP) based techniques, Quantized Congestion Notification (QCN) [5] and Class of Service (CoS) technologies [6]. However, these are not quite effective for the cloud computing network as explained below. Also, the generic congestion control mechanism provided by TCP is inadequate for the cloud computing scenario since it ensures only max-min fairness among the flows [7] and does not take into account the client abstraction. It may lead to a situation where the client capable of opening a larger number of flows can hog more bandwidth than smaller-sized clients.

A TCP-based solution, Seawall [8], provides a network performance isolation solution based on end nodes. It introduces a thin software layer called Seawall's shim layer to perform rate control. It sets up TCP-like tunnels between each pair of communicating VMs or applications. The tunnels nest the transport layer protocol to control the flow rate so that each communicating entity gets its fair share of network bandwidth on every network link. In order to avoid the performance issues associated with nested TCP control loops such as deteriorated throughput and increased delay in lossy networks [9], it modifies the network stack to defer the congestion control to Seawall's shim layer. The need for changes to the network stack of the guest OS precludes this solution from working on full-virtualization systems.

The QCN protocol for congestion notification has been especially devised for cloud data center networks. This protocol also has some shortcomings. First, it works only within a layer-2 domain. Second, it necessitates hardware or software implementation on every network node, making it less flexible to changes and difficult to implement in a huge network.

The CoS technologies, namely, 802.1p Layer 2 tagging, Type of Service and Differentiated Services (DiffServ), provide quality of service among classes of traffic. The 802.1p Layer 2 tagging supports only eight classes per port due to hardware limitations, which are not sufficient for the scale of cloud data centers. If used for the cloud computing network, it would imply very fat groups in each class, rendering the isolation mechanism ineffective. The DiffServ approach specifies rules for the per hop treatment of different classes of traffic. The network providers have the independence to define the rules of treatment. Thus, end-to-end quality cannot be guaranteed unless standard rules are accepted by various providers. Since the cloud network inherently encompasses the World Wide Web, DiffServ cannot assure the performance isolation.

In this paper, we propose a software mechanism to provide performance isolation among clients in a cloud network. In order to suit the huge scale of a cloud data center and optimal resource utilization, our design principles include full virtualization support, ease in wide scale adoption, limited modification to the existing system, low run-time overhead and work-conserving servicing. Bearing in mind these principles, we propose VDE-based mechanisms for network performance isolation. They are a shaping mechanism for outgoing traffic and a policing mechanism for incoming traffic. These mechanisms are realized in the end nodes of a VDE-based cloud computing platform. We name the end result Fair Share-VDE (FS-VDE). It works for all network and transport layer protocols owing to the protocol agnostic nature of VDE. Also, it is scalable since it involves software modification only on the end nodes.

Our approach works as follows. For outgoing traffic, FS-VDE intercepts the traffic originating at the node, segregates it on a VM basis and sends it to the outgoing link in a work-conserving manner such that clients get fair share of network bandwidth. It uses the virtual-time round robin (VTRR) proportionally fair scheduling policy [10] to multiplex bandwidth among the clients. The overhead is minimal since VTRR has an $O(1)$ time-complexity. For incoming traffic, FS-VDE applies a penalty model to the packets violating the contractual agreement. To do so, it monitors incoming packets using the token bucket model [11] and checks them against the per-client contract and discards them if they do not conform to the contract.

For validation of our mechanisms, we have implemented them in the open source VDE switch and performed thorough experiments. The experimental results confirm that network performance isolation among the clients is achieved by the mechanisms. Every client has received at least 99.4% of its bandwidth share as specified by its weight. The performance overhead of the proposed mechanisms is only 5% which is small compared to the benefit.

This paper is organized as follows. Section 2 gives a background of the VDE to aid in understanding the solution mechanism and its implementation. Section 3 describes the target system architecture and gives the problem description. Section 4 provides the details of the solution mechanisms for both incoming and outgoing packets. Section 5 validates the mechanism through a set of experiments. Section 6 concludes the paper.

2. BACKGROUND

We briefly give an overview of the VDE and its architecture as a background to our work.

2.1. Overview of VDE

We have chosen to provide our network performance isolation mechanisms in VDE as it has been widely used by open source cloud computing platforms including Eucalyptus [12] to build private networks for a cluster of nodes. VDE is an open source layer-2 virtual distributed network originally developed at University of Bologna.

It belongs to the Virtual Square project [13] whose goal is to provide opportunities to leverage the power of virtualization. VDE provides an Ethernet-compliant virtual network to connect VMs, applications and real machines. VDE is a virtual network with its parts being completely built in software. It is distributed in the sense that parts of the same VDE network can run on different physical machines. It supports the Ethernet protocol and is able to forward, send and route plain Ethernet packets. It provides interfaces to connect with VMs, connectivity tools as well as virtual interfaces of real systems.

It is used in a variety of domains such as cloud computing, VPN, tunneling and education. Its power lies in its simplicity and ability to support any Ethernet-based network protocol. It provides seamless connection to real machines or VMs over the Internet. VDE permits the implementation of programs having the network stack embedded into it, thus enabling migration of programs with their IP addresses unchanged.

2.2. Architecture of VDE

VDE is component-based software. Its components have the same functionality as the hardware components in the modern Ethernet network. The two main components are the VDE switch and the VDE cable.

Fig. 1 shows an instance of the VDE network. There are two VDE switches connected via a VDE cable. VDE switch 1 is connected to a VM at one of its ports. The VDE switch is a virtual counterpart of the physical Ethernet switch. Similar to a real Ethernet switch, it has several virtual ports where several VMs, applications, connectivity

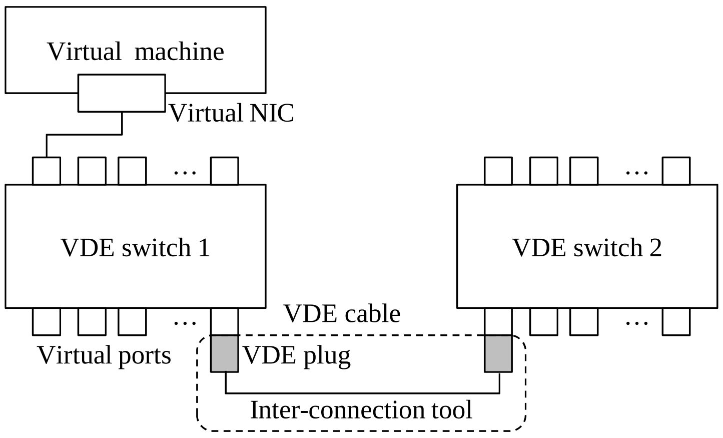

Fig. 1**.** Example of VDE-based network

tools, or even other VDE switches can be plugged in. Its main functionality is to switch Ethernet packets between ports. It manages dynamic association between a physical address and a port using a hash table. To keep up with topology changes, it implements an aging mechanism on the physical address to port mappings allowing graceful convergence to a new configuration.

The VDE cable is a virtual counterpart of a crossed cable and is used to interconnect two VDE switches. It consists of three software components: two VDE plugs, one at each end of the cable, and an interconnection tool. The VDE plug is a program which is connected to a switch to convert all the traffic to a standard stream connection. The interconnection tool bi-directionally connects the streams of the two plugs. In case the switches are located on the same physical machine, a double pipe is used to interconnect the plugs; otherwise, connectivity tools such as *ssh*, *rsh* and *netcat* are used. VDE also provides an encrypted connection tool called *cryptcab* which is connectionless in nature.

VDE supports connectivity to VMs such as QEMU, kernel-based virtual machine (KVM), User-Mode Linux (UML) and Virtual Box. This is achieved through the VDE library called *libvdeplug* which provides programming interfaces to connect the VMs to VDE. The library can also be used to connect applications to VDE. The interface of VDE towards the real network is realized using the TUN/TAP virtual network interfaces. The VMs or applications connected to the VDE get an illusion of being connected to a real Ethernet-based local area network. VDE runs as a user-level process and needs root access only when a TUN/TAP interface is required.

2.3. Operation of VDE

The VDE switch associates a file descriptor to each connected port. It polls the file descriptor for any packet arrival events. When a packet arrives from a connected VM or a process, the switch receives the packet and reads the destination physical address. It then looks up the egress port in the hash table and forwards the packet to the port.

The VDE switch has two modes of operation: switch mode and hub mode. The

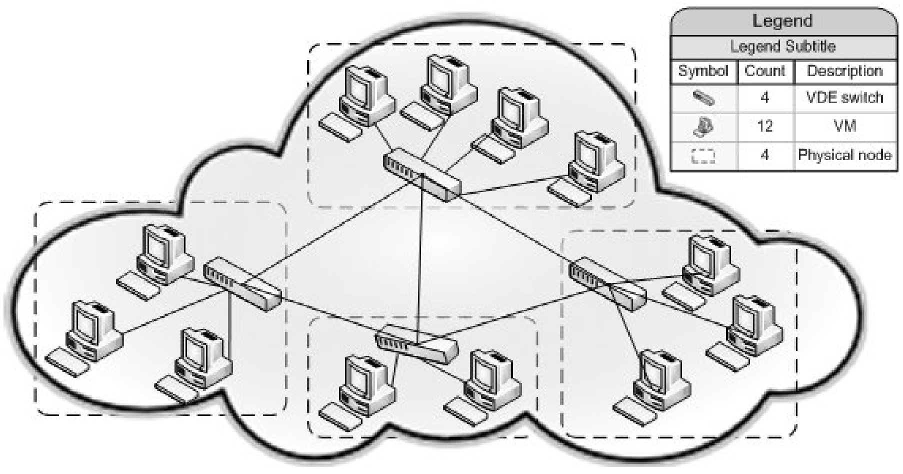

Fig. 2. Target cloud computing system architecture.

switch mode is used for switching packets to only specific ports while the hub mode is used to broadcast packets to all the connected ports. It also supports the retransmission of unsuccessfully sent packets by maintaining a separate queue of those packets. The user can enable or disable the retransmission as needed.

3. PROBLEM FORMULATION

In this section, we describe the target system architecture and formulate our problem.

3.1. Target System Architecture

The target cloud system comprises of a set of physical computing nodes. A physical node runs a host OS on which a VMM is executed. The VMM is used by the cloud provider to instantiate VMs according to clients' computing requirements. Hosted VMMs are needed in order to run the VDE switch and VDE cable processes on the host OS.

The cloud network is realized using VDE as depicted in Fig. 2. Each physical node has a VDE switch that interconnects the VMs on the node and provides connection with other physical nodes. The switches are connected via cables according to the needed topology.

3.2. Problem Definition

For the cloud computing system described above, our approach aims to provide network performance isolation among clients. Clients are individuals or organizations that buy computing, network and storage resources from the cloud providers. A contract is signed between a client and a cloud provider about the services received by the client.

A given physical node services a set of clients denoted by $\{c_1, c_2, \ldots, c_m\}$. Client c_i owns a set of VMs denoted by $\{v_1^i, v_2^i, \ldots, v_n^i\}$. On a particular node, client c_i's network requirement is expressed by its weight w_i. It expects a fraction of bandwidth proportional to this weight. For given link bandwidth B, the bandwidth entitled to client c_i is simply given by:

$$B_i = B \cdot \frac{w_i}{\sum_{j=i}^{m} w_j} \tag{1}$$

A client's share of bandwidth B_i is in turn divided among the VMs owned by it as:

$$B_i = b(v_1^i) + b(v_2^i) + \cdots + b(v_n^i) \tag{2}$$

where $b(v_j^i)$ is bandwidth used by VM v_j^i .

Our approach intends to (1) ensure proportionally fair allocation of bandwidth among the clients and (2) divide a client's share of bandwidth among the VMs owned by it in a fair way. A fairness criterion for VMs of a client is subject to the application. In most practical cases, VMs belonging to a client have similar network requirements and are given equal weights. In cases where the VMs have varied requirements, different weights are assigned. In that case, the VMs should be provided with proportionally fair shares of bandwidth. For the sake of presentation, we assume that packets transmitted and received by a VM have an equal size.

4. THE PROPOSED APPROACH

We propose a software approach to solve the problem of network performance isolation in the cloud network. Specifically, we provide a shaping mechanism for outgoing traffic and a policing mechanism for incoming traffic. We realize our mechanisms as an enhancement to the legacy VDE switch. We name it Fair Share-VDE (FS-VDE). The mechanism observes the design principles stated in Section 1. It is based on the VDE switch alone and does not need any modifications to the guest OS or its applications, thus supporting full-virtualization. The change to the VDE switch is incremental and can be easily adopted through a software patch. The mechanism does not perform static bandwidth reservations and multiplexes the bandwidth share among the VMs in a work conserving manner.

4.1. Shaping Mechanism for Outgoing Traffic

To provide network performance isolation for outgoing traffic, we adopt traffic shaping which can control the volume of traffic being sent into a network in a specified period. Our detailed mechanism is based on two-level link-bandwidth scheduling where the first level scheduling is done among clients and the second among VMs owned by

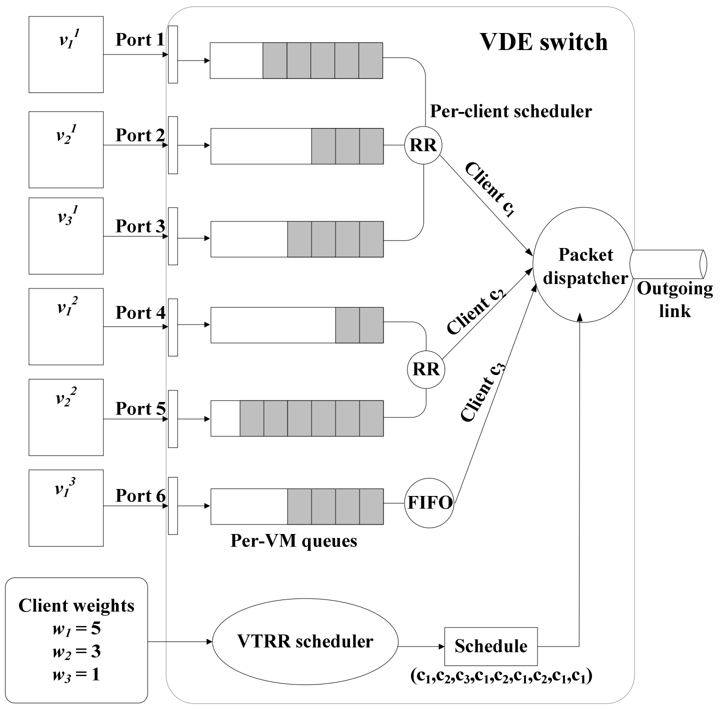

Fig. 3. Component structure of the proposed shaping mechanism for outgoing traffic.

each client. Using the specification of the clients and their VMs, our approach first generates a schedule of clients according to a proportionally fair scheduling algorithm. We particularly adopt the VTRR algorithm for its low time-complexity and improved fairness as compared to other algorithms. At the next level, our approach makes a schedule in terms of VMs. If VMs belonging to a client have equal weights, then a round-robin scheduler is used; otherwise, a proportionally fair scheduler is used in a nested fashion.

Specifically, as shown in Fig. 3, we extend the legacy VDE switch by adding four components to it: (1) a VTRR scheduler, (2) an outgoing packet dispatcher, (3) per-client schedulers and (4) per-VM packet queues. The VTRR scheduler computes a client schedule from the given set of clients and the associated weights. VTRR is a linear-time, proportional share scheduler that combines the round-robin and proportionally fair scheduling approaches. A client schedule is a sequence of clients such that the number of each client appearing in the sequence is proportional to its weight. The same sequence is repeated as long as the clients and their weights remain unchanged. Thus, the VTRR scheduler is invoked only once when such a change occurs.

For each client in the schedule, the packet dispatcher invokes the corresponding per-client scheduler to pick a packet and transmits it to the outgoing link. When a client has no packet to transmit, the packet dispatcher simply skips the client and continues to the

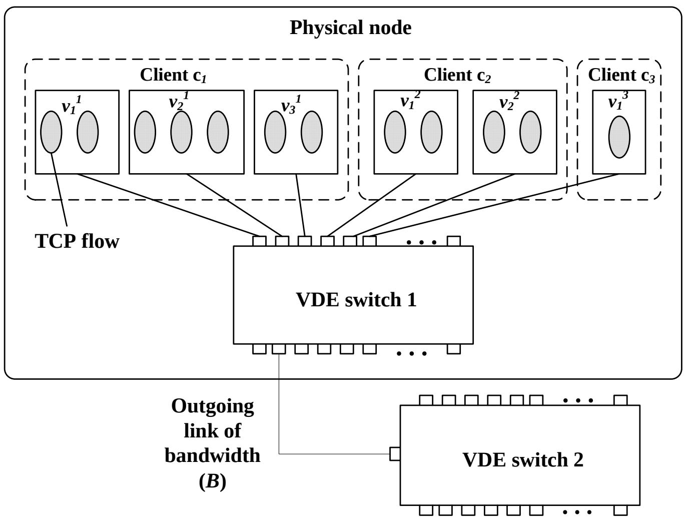

Fig. 4. Cloud computing system with VDE network.

next one in the schedule. This ensures the work-conserving property of our mechanism.

A per-client scheduler schedules multiple per-VM queues that belong to the same client. It is instantiated when a new client is allocated to the physical node. It is either a round-robin or proportionally fair scheduler depending on the weights of VMs in the client. When a request arrives from the packet dispatcher, it returns a packet from a queue according to its scheduling policy.

Finally, a per-VM packet queue is given to each VM. It is instantiated when a new VM is created on the physical node. Packets transmitted by a VM are inserted at the tail of the associated packet queue.

We illustrate our mechanism through an example shown in Fig. 4. A physical node services three clients c_1, c_2 and c_3. They own three, two and one VMs and their weights are given by 5, 3 and 1, respectively. All VMs owned by the same client is given the equal weights. The applications running on the VMs establish TCP flows such that all the flows pass through the outgoing link. In total, client c_1, c_2 and c_3 establish seven, four and one flows, respectively.

In the legacy VDE, there is no mechanism for allocating network bandwidth among clients. Thus, the outgoing link bandwidth is allocated to each flow by the TCP congestion control mechanism which maintains max-min fairness among the flows. Each client obtains a fraction of the bandwidth proportional to its number of flows. Specifically, client c_1 gets about a half of the bandwidth, c_2 gets a third and c_3 gets only a twelfth of the bandwidth instead of the due shares according to the weights.

In the proposed mechanism, the VTRR scheduler makes schedule (c_1, c_2, c_3, c_1, c_2, c_1, c_2, c_1, c_1). According to the schedule, the packet dispatcher invokes a per-client scheduler which in turn returns a packet from the associated queues in the round-robin manner. As a result, each client receives the outgoing link bandwidth in proportion to its weight.

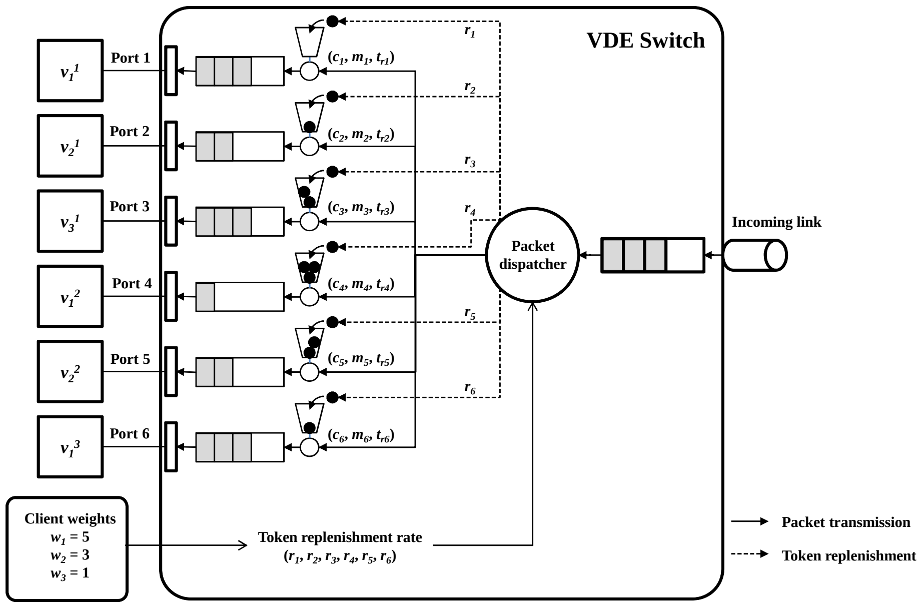

Fig. 5. Component structure of the proposed policing mechanism for incoming traffic.

4.2. Policing Mechanism for Incoming Traffic

To enforce the same contracts for the incoming traffic, we rely on traffic policing in which incoming packets are monitored and checked against the per-client contract and discarded if not conforming to the contract. As a meter to test the conformance, we adopt the token bucket algorithm [11] as it is known to effectively handle traffic burst while ensuring the conformance of network traffic to a specified bandwidth limit.

Fig. 5 shows the component structure of the proposed policing mechanism. As can be seen, we incorporate into the legacy VDE switch two components: (1) an incoming packet dispatcher and (2) per-VM token buckets. The per-VM token bucket is a data structure which is a three-tuple (c, m, t_r) where c represents the number of tokens in the bucket, m the number of the maximum allowable tokens in the bucket and t_r the last time when a token is replenished in the bucket. Among them, m for each bucket is a configurable parameter set by the operator and c and t_r represents the current state of the bucket.

The proposed mechanism works for an incoming packet in two steps: a forwarding step and a bandwidth allocation step. In the forwarding step, the packet dispatcher takes a packet from the incoming queue and decides whether to forward or drop it after referring to the state of the destination VM's token bucket. If the bucket contains more than one token, the dispatcher forwards the packet to the corresponding port and removes one token from the bucket. If the bucket is empty, the packet dispatcher discards the packet to ensure the fair share of the bandwidth. As such, a token in a bucket represents the right to receive one packet.

In the bandwidth allocation step, the packet dispatcher replenishes tokens to buckets at rates proportional to the bucket weights. Let *r* be the bandwidth of VM *v* in packets per second. In order to guarantee the receiving bandwidth, a token should be added to the *v*'s bucket every 1/*r* seconds unless it is fully occupied. A naïve approach to implementing token replenishment is associating each bucket with a high resolution periodic timer, which would incur a significant run-time overhead and not scalable to the number of VMs. Instead, in our architecture, tokens are added to a bucket only when a packet bound for the VM associated with the bucket arrives at the packet dispatcher. Our approach is valid in the sense that the exact number of tokens in a bucket is computed just before a forwarding step is performed.

Let t be the time at which the packet dispatcher receives a packet from the incoming link. The dispatcher replenishes tokens that account for $(t_r, t]$ which is the time interval between the last replenishment time and the current time. The number of tokens replenished for the interval is computed as:

$$c' = \min(\lfloor (t - t_r) \cdot r \rfloor, m - c) \ . \tag{3}$$

After computing c', c and t_r are increased by c' and c'/r, respectively.

5. VALIDATION

We have implemented the proposed mechanisms in the open source VDE switch version 2.2.3. The incoming and outgoing packet dispatchers run as separate threads whereas other components run in the main thread of a VDE switch.

We have performed experiments to compare the performance isolation capability of the legacy VDE and FS-VDE. The experimental setup is shown in Fig. 6. We used two identical physical machines whose hardware consists of a 64-bit Intel Core2 Duo processor and 2GB of RAM. On each physical machine, the Linux 2.6.32 kernel was run as a host OS and KVM was used as a VMM. On each host OS, three VMs were instantiated and connected to either a legacy VDE or a FS-VDE. The switches were connected though a couple of VDE plugs and a connection tool. Six VMs were owned by two clients, c_1 and c_2 as shown in Fig. 6. Their weights were given by 1 and 2,

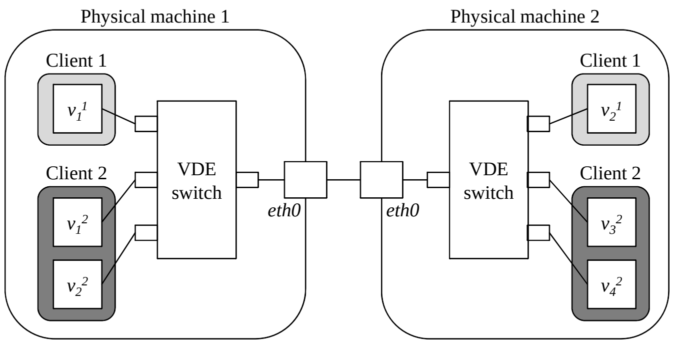

Fig. 6. Experimental setup.

TABLE I. FLOWS ESTABLISHED FOR THE EXPERIMENT

Flows	*Client*	*Source VM*	*Destination VM*
f1	c_1	v_1^1	v_1^{1}'
f2	c_2	v_1^2	v_1^{2}'
f3	c_2	v_1^2	v_2^{2}'
f4	c_2	v_2^2	v_1^{2}'
f5	c_2	v_2^2	v_2^{2}'

TABLE II. ACHIEVED THROUGHPUT (MB/S)

	Flows					***Clients***	
	f1	***f2***	***f3***	***f4***	***f5***	c_1	c_2
VDE	136.0	110.4	116.0	116.0	104.0	136.0	446.4
FS-VDE	183.2	107.2	93.6	75.3	93.6	183.2	369.7

respectively. Thus, client c_2 expects to get twice the bandwidth used by client c_1. That is, $B_2 / B_1 = 2$.

In order to generate network traffic between VMs, we ran *Iperf* version 2.0.4 which is a network performance benchmark tool [14]. It generated a burst of TCP traffic from a source VM to a destination VM. The source-to-destination relationship between VMs is shown in Table I.

TABLE II shows the experimental results. We can observe that FS-VDE allocates the link bandwidth to clients according to their weights whereas the legacy VDE gives nearly an equal amount of bandwidth to each flow. In FS-VDE, the total bandwidth was 552.9 Mbps and the expected share of bandwidth for c_1 and c_2 were 184.3 and 368.6 Mbps, respectively. The actual share of bandwidth received by c_1 and c_2 were 99.4% and 100.3% of the expected ones, respectively. B_2 / B_1 was 2.01 in FS-VDE whereas it was 3.28 in the legacy VDE. We observe that FS-VDE provides each VM owned by a client with equal share of bandwidth in a fair way. We can also observe that the run-time performance overhead of the proposed mechanism is very small. Total throughput obtained by FS-VDE was 552.9 Mbps which is 95% of the throughput obtained by the legacy VDE.

6. CONCLUSION

In this paper, we have presented a VDE extension named Fair Share-VDE (FS-VDE) to provide network performance isolation at end nodes in a VDE-based cloud computing platform. Specifically, we have proposed a shaping mechanism for outgoing traffic and a policing mechanism for incoming traffic. FS-VDE performs the shaping operation by intercepting all outgoing packets, segregating it on a VM basis and multiplexing it to the outgoing link such that the clients and their VMs get the fair share of the network bandwidth. In doing so, it makes use of the VTRR proportionally fair scheduling policy which has an $O(1)$ time-complexity. In order to enforce the contractual agreement imposed on incoming traffic, FS-VDE applies a penalty model to packets violating the agreement. It monitors incoming packets using a token bucket model which is maintained for each VM and drops excessive packets. Like outgoing packet handling, it employs the VTRR scheduler to fairly allocate the incoming bandwidth among the clients.

We have implemented the proposed mechanisms in the open source VDE switch. The change to the original VDE switch was incremental and the implementation did not necessitate any other modification to the guest OS or its applications. Thus, the proposed mechanisms can be easily adopted through a software patch. To validate our mechanisms, we have performed thorough experiments. The experimental results showed the effectiveness of the proposed mechanisms. Every client received at least 99.4% of its bandwidth share as specified by its weight. The performance overhead of the proposed mechanisms is only 5% which is small compared to the benefit.

REFERENCES

1. V. Rathore, J. Yoo, J. Lee and S. Hong, "Providing network performance isolation in VDE-based cloud computing systems," in *Proceedings of the 13th International Workshop on Future Trends of Distributed Computing Systems*, 2011, pp. 721-725.
2. R. Davoli, "VDE: Virtual Distributed Ethernet," in *Proceedings of IEEE/Create-Net Tridentcom*, 2005, pp. 213-220.
3. D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in Xen," in Proceedings of the ACM/IFIP/USENIX 7th International Conference on Middleware, 2006, pp. 342-362.
4. http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm.
5. R. Pan, B. Prabhakar, and A. Laxmikantha, "QCN: Quantized Congestion Notification," http://www.ieee802.org/1/files/public/docs2007/au-prabhakar-qcn-description.pdf, 2007.
6. M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," in Proceedings of Internet Measurement Conference, 2004, pp. 135–148.
7. J. Mo, and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, 2000, pp. 556-567.
8. A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: performance isolation for cloud datacenter networks," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010.
9. C. Kiraly, G. Bianchi, and R. L. Cigno, "Solving performance issues in anonymization overlays with a L3 approach," *University of Trento Information Engineering and Computer Science Department Technical Report DISI-08-041, Ver. 1.1*, 2008.
10. J. Nieh, C. Vaill, and H. Zhong, "Virtual-time round-robin: An O(1) proportional share scheduler," in *Proceedings of USENIX Annual Technical Conference*, 2011, pp. 245–260, 2001.
11. S. Shenker and J. Wroclawski, "General characterization parameters for integrated services network elements," *RFC2215*, 1997.

12. D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus open-source cloud-computing system,” in *Proceedings of the 9th IEEE International Symposium on Cluster Computing and the Grid*, 2009, pp.124-131.
13. Virtual Square,http://www.virtualsquare.org/.
14. A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, “Iperf 1.7.0 - The TCP/UDP bandwidth measurement tool,” http://dast.nlanr.net/projects/iperf, 2004.

Jaesoo Lee earned his BS degree in Electrical Engineering from Seoul National University, Korea, in 2001. He received his PhD degree in Electrical Engineering and Computer Science from Seoul National University, Korea, in 2009. He is currently a senior engineer at Samsung Electronics Co. His research interests include real-time embedded systems, flash memory-based storage systems, and cloud computing systems.

Jonghun Yoo earned his BS degree in Electrical Engineering from the Korea Advanced Institute of Science and Technology, Korea, in 2001. He is currently a PhD candidate at the School of Electrical Engineering and Computer Science of Seoul National University. He is also a member of Real-Time Operating Systems Laboratory at Seoul National University. His current research interests include software platforms for embedded systems, real-time scheduling for multi-core systems and virtual machines. He is a student member of the IEEE.

Yongseok Park earned his BS and MS degrees in Electronics Engineering from Seoul National University, Korea, in 1986 and 1988, respectively. He received his PhD degree in Electrical and Computer Engineering from Purdue University, in 1996. He is currently with the Digital Media Communications R&D Center of Samsung Electronics. His current research interests include network systems, system software, and cloud computing.

Seongsoo Hong earned his BS and MS degrees in Computer Engineering from Seoul National University, Korea, in 1986 and 1988, respectively. He received his PhD degree in Computer Science from the University of Maryland, College Park, in 1994. He is currently a professor with the School of Electrical Engineering and Computer Science at Seoul National University and an affiliated department head of the Department of Intelligent Convergence

Systems, Graduate School of Convergence Science and Technology at Seoul National University. His current research interests include embedded and real-time systems design, real-time operating systems, embedded middleware, and software tools and environments for embedded real-time systems. He served as a general co-chair of IEEE RTCSA 2006 and CASES 2006 and as a program committee co-chair of IEEE RTAS 2005, RTCSA 2003, IEEE ISORC 2002 and ACM LCTES 2001. He has served on numerous program committees, including IEEE RTSS and ACM OOPSLA. He is currently a senior member of the IEEE and a senior member of the ACM.