(2011 추계 우수논문)

# VDE 기반 클라우드 데이터 센터를 위한 클라이언트 단위 네트워크 성능 고립화: FS-VDE*

## (Per-Client Network Performance Isolation in VDE-based Cloud Data Centers: FS-VDE)

차 승 우† 유 종 훈‡ 홍 성 수§

(Seungwoo Cha) (Jonghun Yoo) (Seongsoo Hong)

**요 약** 클라우드 데이터 센터는 계층적 네트워크로 연결된 수천 대의 서버로 구성되고 각 서버는 서로 다른 클라이언트에게 속하는 복수의 가상 머신을 수행한다. 이들은 서버의 네트워크 링크를 공유하기 때문에 서비스 품질 보장을 위해서는 클라이언트 단위로 네트워크 성능을 고립시키는 것이 매우 중요하다. 그런데 기존의 네트워크 성능 고립화 기술은 클라이언트라는 개념 없이 TCP 흐름 단위의 고립만을 지원하기 때문에 클라우드 데이터 센터에 효과적이지 못하다. 뿐만 아니라 기존 네트워크 설비와 게스트 OS에 수정을 요구하여 대규모 도입에 어려움이 있다. 이 논문에서는 이 같은 문제를 해결하기 위해 각 클라이언트에게 가중치를 부여하고 이에 비례하여 네트워크 대역폭을 할당해 주는 FS-VDE를 제안한다. 제안된 기법은 송신 트래픽에 대해 쉐이핑 메커니즘을 수행하고 수신 트래픽에 대해 폴리싱 메커니즘을 사용한다. 이 기법은 기존 시스템에 최소한의 변경만을 요구하므로 대규모 도입에 효과적이다. 우리는 FS-VDE를 Linux 상에 구현하고 실험을 통해 성능을 측정하였다. 실험 결과 각 클라이언트는 가중치로부터 도출된 송신 및 수신 대역폭에 비해 각각 99.4%, 96.5% 이상을 보장 받았으며 이는 단 5%의 성능 저하로 달성될 수 있었다.

**키워드:** 네트워크 성능 고립화, 클라우드 컴퓨팅, VDE, proportional fair share resource allocation

***Abstract*** A cloud data center consists of thousands of servers connected by a hierarchical network. Each server runs multiple virtual machines owned by different clients. Since several virtual machines share network links, it is important to provide for per-client network performance isolation for quality of service guarantees. Unfortunately, existing network performance isolation techniques provide only per-flow isolation without the notion of clients and thus are not effective for a cloud data center. Moreover, these techniques lack scalability as they require non-trivial modification to legacy network equipment and the protocol stack of a guest OS. In this paper, we propose FS-VDE, a mechanism that assigns each client a weight and allocates it network bandwidth proportional to the weight. To do so, our approach performs outgoing traffic shaping and incoming traffic policing. FS-VDE can be widely adopted since it only requires limited modification to the host OS. We have implemented the proposed mechanism in Linux and measured the throughput of clients. Experimental results show that each client receives 99.4% and 96.5% of desired outgoing and incoming bandwidth, as specified by its weight, respectively. This result was achieved with only 5% of performance overhead.

**Key words:** Network performance isolation, cloud computing, VDE, proportional fair resource allocation

## 1. 서 론

클라우드 데이터 센터는 수천에서 수만 대의 서버로 구성되며 이들은 각기 다른 요구사항을 가진 클라이언트들에게 컴퓨팅 자원을 제공한다. 시스템 가상화는 클라우드 컴퓨팅의 핵심 기술로서 OS 에게 가상 머신이라 불리는 논리적 하드웨어를 제공하여 복수의 OS 가 동일한 하드웨어 자원을 공유할 수 있도록 하는 효과적인 기법이다. 이때 가상화된 OS 는 게스트 OS 라고 불린다.

클라우드 서버는 서로 다른 클라이언트에 의해 소유되는 복수의 가상 머신을 수행한다. 이 가상 머신은 물리 NIC(network interface card)를 공유하기 때문에

* 본 연구는 참고문헌 [1], [2] 의 연구결과에 바탕을 두고 이를 작성하였음.

† 학생회원 : 서울대학교 전기컴퓨터공학부
swcha@redwood.snu.ac.kr
‡ 비 회 원 : 서울대학교 전기컴퓨터공학부
jhyoo@redwood.snu.ac.kr
§ 종신회원 : 서울대학교 전기컴퓨터공학부 정교수
sshong@redwood.snu.ac.kr
(Corresponding author임)

논문접수 : 2011년 12월 2일
심사완료 : 2012년 2월 29일

과도한 트래픽을 발생시키는 일부 클라이언트에 의해 다른 클라이언트의 네트워크 성능이 급격히 저하될 수 있다 [3], [4]. 따라서 클라우드 서비스의 품질 보장을 위해 클라이언트 단위의 네트워크 성능의 고립이 필수적이다.

기존의 네트워크 성능 고립화를 위한 연구로 Quantized Congestion Notification (QCN) [5], Class of Service (CoS) [6], TCP 흐름 제어, Seawall [7] 등이 존재한다. 이들 기법은 다음과 같은 이유로 인해 대규모 클라우드 네트워크에 적용에 한계가 있다. 첫째, QCN 은 2 계층 도메인만으로 적용이 제한되며 네트워크 상의 모든 노드의 하드웨어와 소프트웨어에 수정을 요구한다. 둘째, CoS 는 각 포트마다 단 8 개의 서비스 클래스만을 정의할 수 있어 확장성이 제한된다. 셋째, TCP 흐름 제어는 TCP 흐름 단위의 고립만을 지원하여 클라이언트간 고립에 효과적이지 못하다. 마지막으로 Seawall 은 게스트 OS 의 네트워크 스택에 수정을 요구하여 임의의 게스트 OS 를 지원해야 하는 클라우드 서비스에 효과적이지 못하다.

본 논문에서는 기존 연구의 단점을 보완하고 네트워크 성능 고립화를 보장하기 위해 VDE (Virtual Distributed Ethernet) [8]를 기본 프레임워크로 사용하는 FS-VDE (Fair Share-VDE)를 제안한다. FS-VDE 는 각 클라이언트에 부여된 가중치에 비례하여 송수신 링크의 대역폭을 할당한다. 이를 위해 송신 트래픽에 대해 VTRR (Virtual-Time Round-Robin) [9] 알고리즘에 기반한 트래픽 쉐이핑을 수행하고, 수신 트래픽에 대해 토큰 버킷 [10] 알고리즘에 기반한 트래픽 폴리싱을 수행한다. VTRR 알고리즘은 O(1)의 계산 복잡도를 갖기 때문에 제안된 기법의 런타임 오버헤드를 최소화할 수 있으며, 토큰 버킷 알고리즘은 각 가상 머신에 최대 대역폭을 제한할 수 있을 뿐만 아니라 순간적인 트래픽 버스트에 효과적으로 대처 가능하다는 장점을 갖고 있다. 제안된 기법은 기존 네트워크 스택의 변경을 요구하지 않으며, 게스트 OS 의 전가상화를 지원하고 런타임 오버헤드가 낮다는 장점을 갖는다. 우리는 제안된 기법을 오픈 소스 VDE 스위치 상에 구현하고 실험을 통해 성능을 검증하였다. 실험 결과 각 클라이언트는 가중치로부터 도출된 대역폭의 99.4% 이상을 할당 받았으며 이때 네트워크 대역폭의 오버헤드는 5%에 불과하였다.

논문의 나머지 부분은 다음과 같이 구성된다. 2 장에서는 관련 연구를 소개하고 3 장에서는 VDE 의 구조와 동작 메커니즘을 설명한다. 이어서 4 장에서 대상 시스템과 FS-VDE 가 풀고자 하는 문제를 정의하고 5 장에서 제안된 FS-VDE 기법을 상세히 설명한다. 6 장에서 실험을 통한 제안된 기법의 검증을 보이고 마지막으로 6 장에서 논문의 결론을 맺는다.

## 2. 관련 연구

기존의 대표적인 네트워크 성능 고립화 기법에는 QCN, CoS, TCP 흐름 제어, Seawall 이 있다. QCN 은 클라우드 데이터센터를 위해 고안된 congestion notification 을 위한 프로토콜이다. QCN 프로토콜은 단지 2 계층 도메인에만 적용되기 때문에 네트워크 성능 고립화 적용 범위가 제한적이다. 또한 모든 네트워크 장비의 하드웨어와 소프트웨어에 변경이 필요하기 때문에 확장성이 제한적이다.

CoS 는 트래픽마다 클래스를 정하고 이에 따라 차별화된 QoS 를 제공한다. 이때 사용되는 802.1p Layer 2 tagging 은 최대 8 개의 클래스를 정의할 수 있다. 이는 수천에서 수만의 클라이언트가 공존하는 대규모 클라우드 데이터센터에 적용하기 충분하지 못하다.

TCP 흐름 제어 기법은 TCP 헤더의 window 필드를 사용하여 상호 간에 전송할 수 있는 버퍼의 크기를 알려줌으로써 TCP 흐름을 조절하는 기법이다. 이 기법은 TCP 흐름간 max-min fairness 를 보장하기 때문에 [11] TCP 흐름 단위의 네트워크 성능 고립만을 지원한다.

Seawall 은 TCP 와 유사한 터널을 네트워크 종단 사이에 제공하고 이를 통해 가상 머신 간 네트워크 대역폭을 제어한다. Seawall 은 이 논문에서 제안하는 FS-VDE 와 같이 클라이언트 단위의 네트워크 성능 고립을 지원한다. 그런데 Seawall 메커니즘은 nested TCP 제어 루프에 의한 성능 저하 문제를 방지하기 위해 게스트 OS 의 TCP 프로토콜에 수정을 요구한다. 따라서 게스트 OS 의 전가상화를 요구하는 환경에서 적용이 어렵다.

## 3. Virtual Distributed Ethernet (VDE)

우리는 제안하는 네트워크 성능 고립화 기법을 구현하기 위한 기본 프레임워크로 VDE 를 선택하였다. VDE 는 복수 노드로 구성된 클러스터를 논리적으로 연결하기 위한 소프트웨어로서 Eucalyptus 와 [12] 같은 오픈 소스 클라우드 컴퓨팅 플랫폼에 널리 채택되어

사용된다. 이 장의 나머지 절에서는 논문의 이해를 돕기 위해 VDE 의 아키텍처와 동작 방식에 대해 자세히 설명한다.

### 3.1. VDE 개관

VDE 는 Virtual Square 프로젝트의 [13] 일환으로 University of Bologna 에서 개발되었다. VDE 는 물리 머신, 가상 머신, 에뮬레이터, 응용 등의 연결을 위한 가상의 이더넷 (Ethernet) 호환 네트워크를 제공하는 오픈 소스 프레임워크이다. 또한 VDE 는 분산 노드에서 수행되는 VM 과 응용들을 하나의 가상 네트워크로 연결할 수 있다.

현재 VDE 는 클라우드 컴퓨팅, VPN (Virtual Private Network), 네트워크 교육 등 다양한 분야에서 널리 활용되고 있다. 이는 VDE 가 간단하고, 이더넷 기반의 네트워크 프로토콜이면 모두 지원할 수 있다는 특징 때문이다.

### 3.2. VDE 구조

VDE 는 VDE 스위치와 VDE 케이블의 두 가지 컴포넌트로 구성된다. 이들은 이더넷 기반 L2 스위치와 크로스 케이블을 각각 논리적으로 구현한다.

그림 1에 나타난 VDE 네트워크의 예에서 두 VDE 스위치가 하나의 VDE 케이블에 의해 연결되어 있고 스위치 1 은 가상 머신과 연결되어 있다. VDE 케이블은 VDE 플러그 (plug)와 inter-connection tool 두 개의 구성 요소로 이루어져 있다. VDE 플러그는 입출력 패킷 트래픽을 OS 의 표준 스트림 연결로 전환한다. 한편 inter-connection tool 은 두 플러그의 스트림을 양방향으로 전달하는 일을 하며 double pipe 기반 IPC, *ssh, rsh, netcat, cryptcab* 등의 tool 을 통해 구현된다. VDE 스위치와 가상 머신 사이의 연결은 *libvdeplug* 라 불리는 VDE 라이브러리가 제공하는 인터페이스를 통해 이뤄진다. 이 라이브러리가 지원하는 가상 머신에는 QEMU, Kernel-based Virtual Machine (KVM), User-Mode Linux (UML), 그리고 Virtual Box 가 있다. 또한 이 라이브러리는 응용과 VDE 스위치 사이의 연결 역시 지원한다.

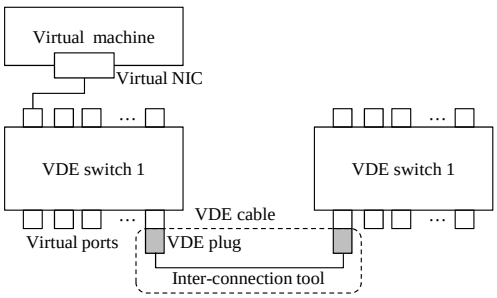



그림 1. VDE 네트워크 예시.

VDE 는 외부 네트워크와의 연결을 위해 TUN/TAP 가상 네트워크 인터페이스를 사용한다. 이 인터페이스는 VDE 스위치와 물리 NIC 를 연결함으로써 VDE 스위치에 연결된 가상 머신을 외부 네트워크에 연결한다. VDE 는 사용자 레벨 프로세스로 수행되며 TUN/TAP 사용을 위해 관리자 권한을 필요로 한다.

### 3.3. VDE 동작 방식

VDE 스위치의 포트는 파일 디스크립터 (file descriptor)를 통해 가상 머신과의 패킷 전송을 관리한다. VDE 는 관리 중인 모든 파일 디스크립터를 확인하여 새로운 패킷의 도착 유무를 판단한다. 만약 외부로부터 도착한 패킷이 있다면 VDE 스위치가 패킷의 목적지의 주소를 해쉬 테이블에서 찾는다. 만약 테이블에 해당 주소가 존재하고 이와 연결된 포트가 있다면 그 포트를 통해 패킷을 송신한다. VDE 스위치에는 스위치와 허브의 두 가지 모드가 존재한다. 스위치 모드에서는 목적지 포트로 패킷을 전달하고 허브 모드에서는 모든 포트를 통해 송신한다.

## 4. 시스템 모델 및 문제 정의

### 4.1. 대상 시스템 구조

클라우드 데이터 센터의 네트워크는 그림 2에 나타난 것과 같이 core, aggregation, access 의 세 계층으로 구성된다. 이는 클라우드 데이터센터에서 널리 사용하는 네트워크 구조로써 Cisco 에서 일반화하였다 [14]. Core 계층은 외부 인터넷과 연결된 하나의 border router 와 내부 access router 들과 연결된 core 스위치들로 구성되며 aggregation 계층은 다수의 access router, L3 스위치 그리고 각종 서비스 모듈로 구성된다. 서비스 모듈에는 로드 밸런서 (load balancer), 인트루젼 디텍터 (intrusion detector) 그리고 방화벽 등이 있다. 마지막으로 access 계층은 복수의 L2 도메인들로 구성되며 각 도메인마다 Top-Of-the-Rack (ToR) 스위치라고 불리는 하나의 L2 스위치와 물리 머신들이 연결되어 있다.

각 물리 머신은 호스트 OS 와 가상 머신 모니터를 수행하고 가상 머신 모니터는 복수의 가상 머신을 생성하고 관리한다. 또한 가상 머신 모니터는 그림 2와 같이 VDE 스위치와 VDE 케이블을 통해 가상 머신들을 외부 네트워크와 연결한다. 각 물리 노드 안에는 하나의 VDE 스위치가 있고 여기에 가상 머신들과 물리 NIC 가 연결된다.

### 4.2. 문제 정의

우리의 목표는 앞 절에서 기술된 클라우드 데이터 센터 위에서 클라이언트 단위의 네트워크 성능 고립화를 위한 메커니즘을 제공하는 것이다. 여기서 클라이언트란 클라우드 서버가 제공하는 서비스를 구매한 개인 또는 기관이다. 이들은 서버 내의 가상 머신을 통해 하드웨어 자원을 공유한다.

임의의 서버를 공유하는 클라이언트의 집합을 $\{c_1, c_2,\ldots, c_m\}$이라 하자. 그리고 클라이언트 $c_i$의 가중치를 $w_i$라 하고 $c_i$가 소유한 가상 머신의 집합을 $\{v_1^i, v_2^i,\ldots, v_n^i\}$이라 하자. 주어진 물리 네트워크의 링크 대역폭을 $B$ 라고 할 때 클라이언트 $c_i$는 다음 과 같이 가중치에 비례하여 대역폭 $B_i$를 할당 받아야 한다.

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

이때 클라이언트가 소유한 가상 머신들은 아래와 같이 $B_i$를 나눠 받는다. $b(v_j^i)$ 는 가상 머신 $v_j^i$의 대역폭이다.

$$B_i = b(v_1^i) + b(v_2^i) + \ldots + b(v_n^i)$$

제안된 기법은 위의 조건을 만족시키기 위해 (1) 클라이언트 간 공정한 대역폭을 할당하고 (2) 각 클라이언트가 소유한 가상 머신 간 공정한 대역폭을 할당한다.

## 5. 해결 방안

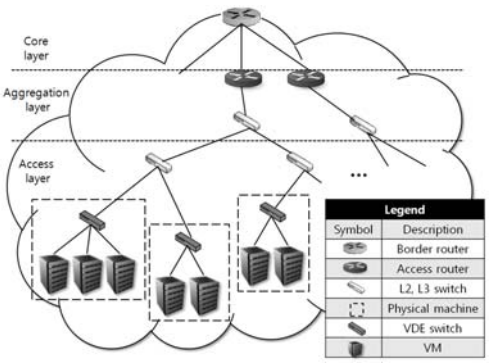



그림 2. 대상 클라우드 데이터센터 네트워크 구조.

이 논문에서 제안하는 FS-VDE (Fair share-VDE) 기법은 VDE 를 기본 프레임워크로 하여 앞 장에서 정의한 두 문제를 해결한다. FS-VDE 는 송신 트래픽에 대한 쉐이핑 (shaping) 메커니즘과 수신 트래픽에 대한 폴리싱 (policing) 메커니즘으로 구성된다.

### 5.1. 송신 트래픽 쉐이핑 메커니즘

FS-VDE 는 송신 트래픽의 클라이언트간 고립화를 위하여 2-level 링크 대역폭 스케줄링을 수행한다. 첫 번째 단계 스케줄링은 클라이언트 사이에서 수행된다. 이를 위해 O(1)의 낮은 계산 복잡도를 가지면서 높은 공정성을 보장하는 VTRR 스케줄링 알고리즘을 통해 클라이언트의 스케줄을 생성한다. 그리고 두 번째 단계에서는 동일 클라이언트 내의 가상 머신 사이의 스케줄링이 수행된다.

구체적으로, 우리는 기존 VDE 스위치 안에 (1) VTRR 스케줄러, (2) 송신 패킷 디스패처(dispatcher), (3) 클라이언트별 스케줄러, (4) 가상 머신별 패킷 큐의 총 네 가지 컴포넌트를 추가하였다. 그림 3은 추가된 컴포넌트가 포함된 VDE 스위치의 구조를 나타낸다. VTRR 은 물리 머신에 가상 머신이 추가되거나 제거될 때마다 새로운 스케줄 시퀀스를 생성한다. 그림 3의 클라이언트 $c_1$, $c_2$, $c_3$ 는 각각 가중치 5, 3, 1 을 갖는다. 가중치를 토대로 VTRR 에 의해 클라이언트의 스케줄 시퀀스 ($c_1$, $c_2$, $c_3$, $c_1$, $c_2$, $c_1$, $c_2$, $c_1$, $c_1$)이 생성된다. 송신 패킷 디스패처는 이 시퀀스에 따라 패킷을 송신할 클라이언트를 결정한다. 만약 선택된 클라이언트가 보낼 패킷이 없다면 다음 클라이언트가 선택된다. 이로 인해 송신 패킷 디스패처의 스케줄링은 작업 보장적 (work-conserving)인 특징을 갖는다. 이어서, 클라이언트별 스케줄러는 해당 클라이언트가 소유한 가상 머신 중 하나를 선택한다. 이때, 동일 클라이언트 내의 각 가상 머신이 동일한 가중치를 부여 받았을 경우 단순히 round-robin 스케줄링이 사용되며, 서로 다른 가중치를 부여 받았을 경우 VTRR 스케줄링이 사용된다. 마지막으로 선택된 가상 머신의 큐에서 패킷이 꺼내져 물리 NIC 를 통해 외부 네트워크로 전송된다.

### 5.2. 수신 트래픽 폴리싱 메커니즘

FS-VDE 는 수신 트래픽의 클라이언트간 고립화를 위하여 클라이언트마다 가중치에 의해 정해진 최대 속도 이상으로 유입되는 패킷을 드롭한다. 이를 위해 토큰 버킷 알고리즘이 사용된다. 이는 각 클라이언트의 네트워크 대역폭에 상한선을 설정하여 트래픽 과부하를 효과적으로 막을 수 있다.

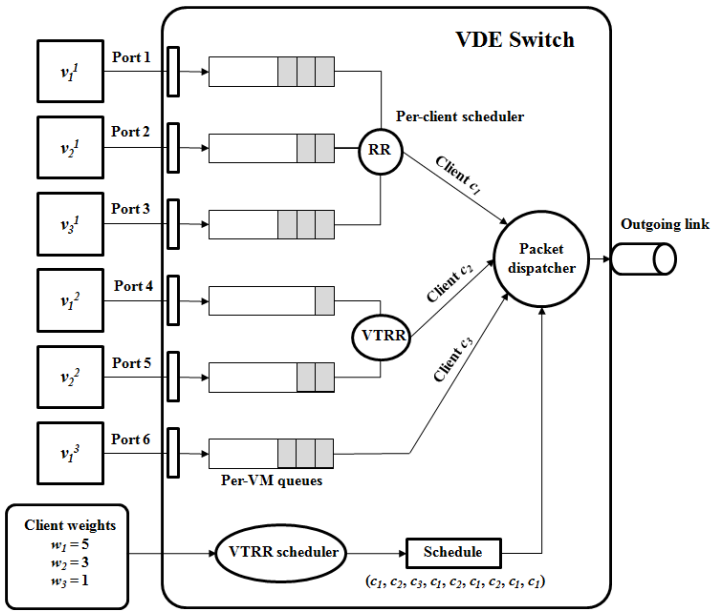



그림 3. 송신 트래픽 쉐이핑 메커니즘.

구체적으로, 우리는 기존의 VDE 스위치 안에 (1) 패킷 디스패처와 (2) 가상 머신별 토큰 버킷 두 컴포넌트를 추가 하였다. 토큰 버킷은 $(c, m, t_r)$의 3-tuple 로 정의되며 $c$ 는 현재 버킷 안에 든 토큰의 수, $m$ 은 해당 버킷이 수용할 수 있는 최대 토큰 수, $t_r$은 가장 최근에 토큰이 보충된 시각을 각각 나타낸다. 그림 4는 추가된 컴포넌트가 포함된 VDE 스위치의 구조를 나타낸다.

수신 패킷의 처리는 포워딩 (forwarding) 과정과 대역폭 할당의 두 과정을 거쳐 이루어진다. 먼저, 포워딩 과정은 다음과 같다. 수신 버퍼로부터 패킷을 전달받은 패킷 디스패처는 해당 패킷의 목적지 가상 머신의 토큰 버킷을 확인한다. 만약 토큰이 한 개 이상일 경우 해당 패킷을 가상 머신으로 전달하고 토큰을 하나 감소시킨다. 그리고 만약 토큰이 없다면 해당 가상 머신을 소유한 클라이언트에게 지정된 속도 이상으로 패킷이 전달되었음을 의미하므로 해당 패킷을 드롭한다. 이어서 대역폭 할당 과정에서는, 패킷 디스패처가 클라이언트 가중치로부터 도출된 보충 속도 (replenishment rate) 값인 $r$ 을 토대로 버킷에 토큰을 보충한다. 이때 $r$ 은 물리 머신에 가상 머신이 추가되거나 제거될 때마다 다시 계산된다. 패킷 디스패처가 NIC 로부터 패킷을 수신한 시간을 $t$ 라고 할 때 목적지 가상 머신의 토큰 버킷에 보충되는 토큰의 개수 $c$'은 다음과 같이 계산된다.

$$c' = \min(\lfloor (t - t_r) \cdot r \rfloor, m - c)$$

토큰이 보충된 후, 해당 버킷의 $c$ 와 $t_r$은 각각 $c$'와 $c'/r$ 만큼 증가된다.

## 6. 실험 평가

우리는 VDE 버전 2.2.3 상에 제안된 메커니즘을 구현하고 일련의 실험을 통해 네트워크 성능 고립 효과와 오버헤드를 검증하였다. 이를 위해 두 대의 동일한 물리 머신에 각각 가상 머신을 생성하고 FS-VDE 를 통해 이들을 연결하였다. 두 물리 머신의 하드웨어는 64-bit Intel Core2 Duo 프로세서와 2GB RAM 로 구성되며 호스트 OS 와 가상 머신 모니터로 각각 Linux 2.6.32 와 KVM 이 사용되었다.

우리는 가상 머신에 네트워크 트래픽을 발생시키기 위해 TCP flow burst 를 생성할 수 있는 *iperf* [15] 네트워크 벤치마크 툴을 사용하였다. 이를 통해 송신과 수신 두 가지 타입의 트래픽에 대해 각 클라이언트의 네트워크 성능을 측정하였다.

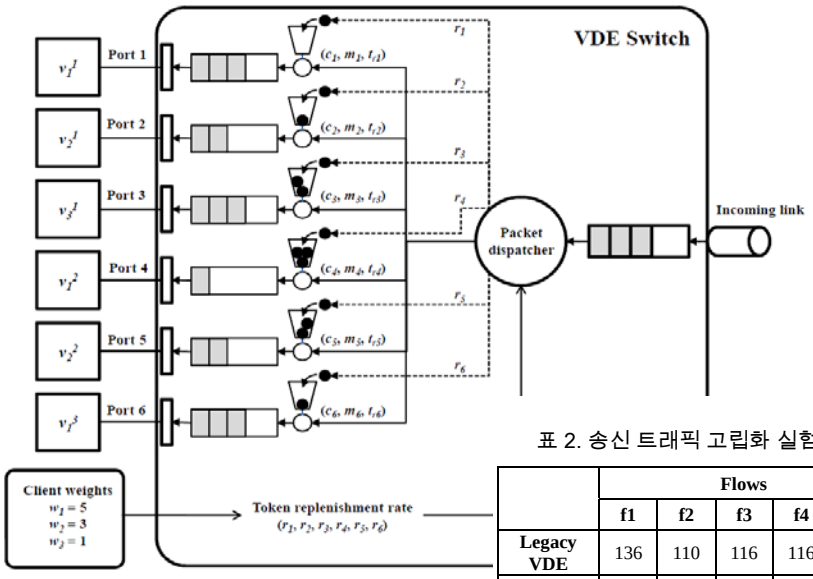



그림 4. 수신 트래픽 폴

표 2. 송신 트래픽 고립화 실험 결과. (Mb/sec)

| | Flows | | | | | Clients | |
|---|---|---|---|---|---|---|---|
| | f1 | f2 | f3 | f4 | f5 | $c_1$ | $c_2$ |
| Legacy VDE | 136 | 110 | 116 | 116 | 104 | **136** | **446** |
| FS-VDE | 183 | 107 | 93 | 75 | 93 | **183** | **369** |

표 3. 수신 트래픽 고립화 실험 설정.

| Flows | Clients | Source VM | Destination VM |
|---|---|---|---|
| f1 | $c_1$ | $v_1^1$ | $v_2^1$ |
| f2 | $c_1$ | $v_1^1$ | $v_3^1$ |
| f3 | $c_2$ | $v_1^2$ | $v_2^2$ |

그림 5. 송신 트래픽 실험 구성.

표 4. 수신 트래픽 고립화 실험 결과. (Mb/sec)

| | Flows | | | Clients | |
|---|---|---|---|---|---|
| | **f1** | **f2** | **f3** | **$c_1$** | **$c_2$** |
| **Legacy VDE** | 190 | 188 | 200 | **378** | **200** |
| **FS-VDE** | 94 | 90 | 355 | **184** | **355** |

그림 6. 수신 트래픽 실험 구성.

### 6.1. 송신 트래픽 고립화

송신 트래픽 고립화를 위한 실험 구성은 그림 5에 나타난 것과 같다. 총 6 개의 가상 머신은 클라이언트 1, 2 가 그림 5와 같이 소유하고 있으며 이들에게 부여된 가중치는 각각 1 과 2 이다. 표 1과 같이 총 여섯 개의 TCP 트래픽을 생성하고 각 흐름과 클라이언트 별 네트워크 대역폭을 측정하였다.

실험 결과는 표 2 에 나타난 것과 같다. 기존 VDE 는 TCP 흐름 제어에 따라 흐름 단위로 대역폭이 할당되기 때문에 흐름을 많이 생성하는 클라이언트가 더 많은 대역폭을 할당 받는다.

따라서 총 네 개의 흐름을 생성한 클라이언트 2 가 한 개의 흐름을 생성한 클라이언트 1 에 비해 약 네 배의 대역폭을 차지했다. 반면 FS-VDE 의 경우, 클라이언트 1 과 2 의 대역폭 비가 가중치의 비율인 1:2 에 근접한 1:2.01 로 나타났다. 이는 곧 각 클라이언트가 가중치로부터 도출된 대역폭의 최소 99.4% 이상을 할당 받았음을 의미한다. 따라서 FS-VDE 를 통해 클라이언트 간 송신 트래픽 네트워크 성능 고립이 효과적으로 이루어졌음을 알 수 있다. 한편 기존 VDE 의 대역폭 총합에 비해 FS-VDE 의 대역폭 총합은 95% 수준으로 나타나 송신 트래픽 쉐이핑 메커니즘의 네트워크 오버헤드가 5%에 불과한 것을 확인할 수 있었다.

표 1. 송신 트래픽 고립화 실험 설정.

| Flows | Clients | Source VM | Destination VM |
|---|---|---|---|
| f1 | $c_1$ | $v_1^1$ | $v_2^1$ |
| f2 | $c_2$ | $v_1^2$ | $v_3^2$ |
| f3 | $c_2$ | $v_1^2$ | $v_4^2$ |
| f4 | $c_2$ | $v_2^2$ | $v_3^2$ |
| f5 | $c_2$ | $v_2^2$ | $v_4^2$ |

### 6.2. 수신 트래픽 고립화

수신 트래픽 고립화를 위한 실험 구성은 그림 6에 나타난 것과 같다. 총 5 개의 가상 머신은 클라이언트 1, 2 가 그림 6와 같이 소유하고 있으며 이들에게 부여된 가중치는 각각 1 과 2 이다. 표 3 과 같이 총 세 개의 TCP 트래픽을 생성하고 각 흐름과 클라이언트 별 네트워크 대역폭을 측정하였다.

실험 결과는 표 4 에 나타난 것과 같다. 기존 VDE 의 경우 총 두 개의 흐름을 생성한 클라이언트 1 이 한 개의 흐름을 생성한 클라이언트 2 에 비해 약 두 배의 대역폭을 차지했다. 반면 FS-VDE 의 경우에는 클라이언트 1 과 2 의 대역폭 비가 가중치 비 1:2 에 근접한 1:1.93 으로 나타났다. 각 클라이언트가 가중치로부터 도출된 대역폭의 최소 96.5% 이상을 할당 받았음을 의미한다. 따라서 FS-VDE 를 통해 클라이언트 간 수신 트래픽 네트워크 성능 고립도 보장됨을 알 수 있다. 또한, 수신 트래픽 폴리싱 메커니즘의 네트워크 오버헤드는 기존 VDE 의 대역폭 총합에 비해 95% 수준이므로 송신 트래픽 쉐이핑 메커니즘의 오버헤드와 마찬가지로 5%에 불과하였다.

## 7. 결론

이 논문에서 우리는 VDE 기반 클라우드 컴퓨팅 플랫폼 상에서 클라이언트 단위의 네트워크 성능 고립화를 보장하기 위한 Fair Share-VDE 를 제안하였다. 구체적으로, 송신 트래픽을 위한 쉐이핑 메커니즘과 수신 트래픽을 위한 폴리싱 메커니즘을 제안하였다. 쉐이핑 메커니즘은 클라이언트와 가상 머신의 2-level 스케줄링을 통해 대역폭의 공정한 할당을 수행하며 이를 위해 $O(1)$ 의 계산 복잡도를 갖는 VTRR 스케줄링이

사용된다. 그리고 폴리싱 메커니즘은 수신 트래픽을 모니터링하며 각 가상 머신에 할당된 대역폭 이상의 트래픽이 유입되는 경우 패킷을 드롭시킨다. 이를 위해 각 가상 머신마다 토큰 버킷을 부여하고 가중치로부터 도출된 보충 속도에 따라 토큰을 보충해 준다.

실험 결과, FS-VDE 는 송신 트래픽의 경우 클라이언트가 기대하는 대역폭의 99.4%이상을 할당하였고, 수신 트래픽의 경우 96.5%이상을 할당하여 클라이언트 단위 네트워크 성능 고립화를 보장하였으며, 기존 VDE 에 비교하여 네트워크 오버헤드가 5%에 불과함을 입증했다.

## 참고문헌

[1] V. Rathore, J.H. Yoo, J.S. Lee, and S.S. Hong, “Providing Network Performance Isolation in VDE-based Cloud Computing Systems,” The 13th International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2011), pp. 721-725, Sep 2011.

[2] S.W. Cha, J.H. Yoo, S.S. Hong, “Per-Client Network Performance Isolation in VDE-based Cloud Computing Servers,” The 38th KIISE Fall Conference, Nov 2011.

[3] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, “Enforcing performance isolation across virtual machines in Xen,” Proc. of the ACM/IFIP/USENIX 7th International Middleware Conference, Melbourne, Australia, vol. 4290, no. 7, pp.342-362, Lecture Notes in Computer Science, Springer, November 2006.

[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-qcndescription.pdf, May 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,” Proc. of Internet Measurement Conference 2004, Sicily, Italy, pp. 135–148, October 2004.

[7] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, “Seawall: performance isolation for cloud datacenter networks,” 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10), 2010.

[8] R. Davoli, “VDE: Virtual Distributed Ethernet,” Proc. of IEEE/Create-Net Tridentcom 2005, Trento, Italy, pp. 213-220, May 2005.

[9] J. Nieh, C. Vaill, and H. Zhong, “Virtual-Time round-robin: An O(1) proportional share scheduler,” Proc. of USENIX Annual Technical Conference, Berkeley, CA, USA, pp. 245–260, June 2001.

[10] S. Shenker and J. Wroclawski, “General Characterization Parameters for Integrated Services Network Elements,” RFC2215, September 1997.

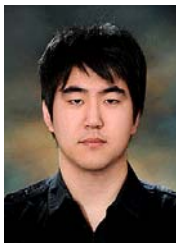
[11] J. Mo, and J. Walrand, “Fair end-to-end window-based congestion control,” IEEE/ACM Transactions on Networking, vol. 8, no. 5, pp. 556-567, October 2000.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus open-source cloud-computing system,” the 9th IEEE International Symposium on Cluster Computing and the Grid, 2009, (CCGRID '09), Shanghai, China, pp.124-131, May 2009.

[13] Virtual Square,http://www.virtualsquare.org/.

[14] Cisco Co., http://www.cisco.com/.

[15] 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.



차 승 우
2010년 서울대학교 전기공학부 (학사). 2010년~현재 서울대학교 전기컴퓨터공학부 석사과정. 관심분야는 내장형 시스템 소프트웨어, 실시간 운영체제

유 종 훈
2001년 한국과학기술원 전기및-전자공학과(학사). 2005년~현재 서울대학교 전기컴퓨터공학부 석박사 통합과정. 관심분야는 내장형 시스템 소프트웨어, 실시간 운영체제, 내장형 미들웨어

홍 성 수
1986년 서울대학교 컴퓨터공학과 (학사). 1988년 서울대학교 컴퓨터공학과(공학석사). 1994년 University of Maryland. Department of Computer Science(공학박사). 1996년~1997년 서울대학교 전기컴퓨터공학부 전임강사. 1997년~2001년 서울대학교 전기컴퓨터공학부 조교수. 2001년~2006년 서울대학교 전기컴퓨터공학부 부교수. 2004년~2006년 서울대학교 내장형시스템연구센터 센터장. 2006년~현재 서울대학교 융합과학기술대학원 지능형융합시스템학과 학과장. 2006년~현재 서울대학교 전기컴퓨터공학부 정교수. 관심분야는 내장형 실시간 시스템 설계, 실시간 운영체제, 내장형 미들웨어, 실시간 시스템 설계 방법론, 소프트웨어 공학, 컴포넌트 기반 소프트웨어 설계