공학석사학위논문

다중 큐 가상 네트워크 어댑터를 위한 동적 패킷 로드 밸런싱 메커니즘

Dynamic Packet Load Balancing Mechanism for Multi-queue Virtual Network Adapter

2013년 2월

서울대학교 대학원
지능형융합시스템학과
한 장 원

다중 큐 가상 네트워크 어댑터를 위한 동적 패킷 로드 밸런싱 메커니즘

Dynamic Packet Load Balancing Mechanism for Multi-queue Virtual Network Adapter

지도교수 홍 성 수

이 논문을 공학석사학위논문으로 제출함

2013년 2월

서울대학교 대학원

지능형융합시스템학과

한 장 원

한장원의 석사학위논문을 인준함

2013년 2월

위 원 장 _____ 박 재 흥 _____ (인)

부 위 원 장 _____ 홍 성 수 _____ (인)

위 원 _____ 안 정 호 _____ (인)

초 록

다중 큐 가상 네트워크 어댑터는 가상 머신으로 전송되는 패킷들을 병렬적으로 처리하도록 하여 기존의 단일 큐 가상 네트워크 어댑터보다 더 뛰어난 네트워크 성능을 보인다. 다중 큐 가상 네트워크 어댑터는 하나 이상의 큐를 포함하기 때문에 패킷이 어떠한 큐에서 처리되어야 하는지 결정하는 작업이 추가적으로 필요하게 된다. 이와 같은 일을 수행하는 메커니즘을 패킷 로드 밸런싱 메커니즘이라고 한다. 현존하는 다중 큐 가상 네트워크 어댑터는 오직 flow 단위의 정적 패킷 로드 밸런싱 메커니즘에 기반을 둔다. 구체적으로 이는 flow 식별자를 (e.g. 패킷 헤더에 담긴 송수신자의 IP 및 port 번호와 통신 프로토콜 번호) 큐의 개수로 나눈 나머지 값을 기준으로 flow를 큐 간에 분배하는 방식을 취한다. 이러한 패킷 로드 밸런싱 메커니즘은 flow가 많이 발생되는 대형 시스템에서 유용하게 사용 될 수 있지만 flow 개수가 상대적으로 적게 발생되는 소형 시스템에서는 flow들이 특정 큐에 집중되는 경우가 발생해 병렬 처리성이 떨어져 네트워크 성능 저하의 문제를 가져올 수 있다.

본 논문에서는 flow들이 특정 큐에 집중되어 네트워크 성능이 저하되는 문제를 해결하기 위해 동적 패킷 로드 밸런싱 메커니즘을 제안한다. 제안하는 동적 패킷 로드 밸런싱 메커니즘은 일정 주기마다 각 큐의 분배 된 flow들을 큐 로드에 비례하여 고르게 분배시키기 때문에 특정 큐

에 패킷이 집중되는 문제를 완화한다. 제안한 패킷 로드 밸런싱 메커니즘은 KVM 가상 머신 환경에서 구현되었으며 실험 결과 주어진 시나리오 상에서 기존의 정적 패킷 로드 밸런싱 메커니즘보다 약 8%정도 높은 네트워크 성능을 보여 그 효용성을 입증하였다.

핵심용어 : KVM, 네트워크 I/O 가상화, 로드 밸런싱

학 번 : 2010-23960

목 차

표 목 차

그 림 목 차

제 1 장 서 론

1.1 연구 배경

최근 가상화 기술을 이용한 다양한 서비스들이 등장함에 따라 가상 머신의 성능을 향상시키려는 연구가 매우 활발히 이루어지고 있다. 이러한 연구들의 일환으로 가상 머신으로 전송되는 패킷을 최대 가상 CPU의 개수만큼 병렬 처리 하도록 하여 가상 머신의 네트워크 성능을 향상 시킬 수 있는 기술이 개발되었다. 이는 다중 큐 가상 네트워크 어댑터라고 불리는 기술이다.[1] 다중 큐 가상 네트워크 어댑터는 패킷 처리를 위한 큐를 가상 CPU 개수만큼 갖고 각각의 큐가 가상 CPU와 1:1로 대응 되어 동작하는 구조를 갖는다.

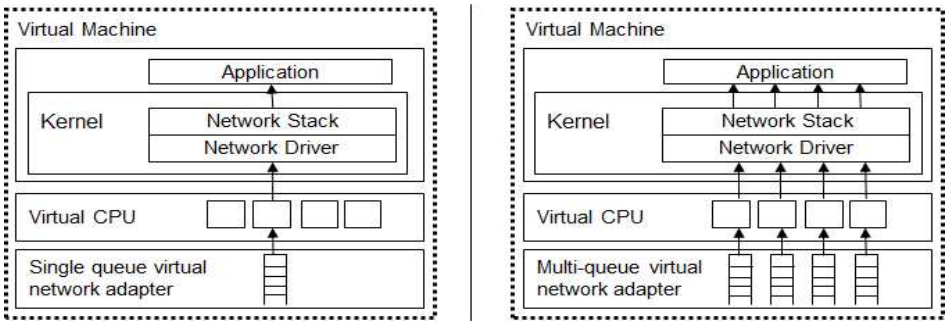

그림 1.1. 단일 및 다중 큐 가상 네트워크 어댑터의 구조

다중 큐 가상 네트워크 어댑터는 이러한 구조로 인해 패킷들의 병렬 처리가 가능하여 단일 큐 가상 네트워크 어댑터보다 더 향상 된 네트워크 성능을 제공한다. 하지만 가상 네트워크 어댑터가 다수의 큐를 갖는 구조로 확장되면서 큐 간에 패킷을 분배시켜주는 메커니즘이 필요하게 된다. 이와 같은 메커니즘을 패킷 로드 밸런싱 메커니즘이라고 하며 이에 대한 개념은 그림 1.2와 같다.

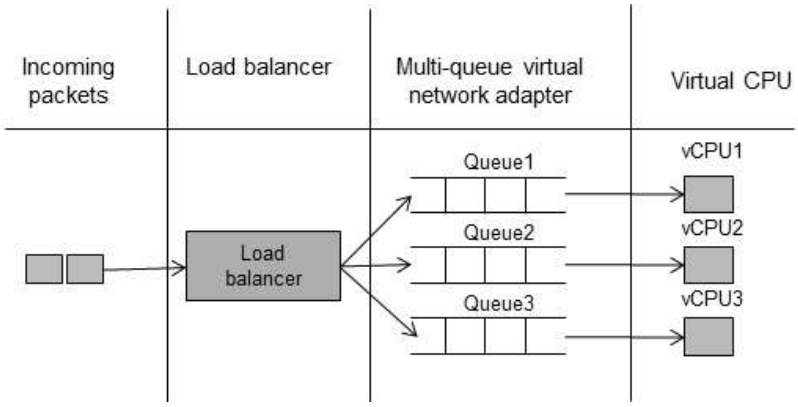

그림 1.2. 패킷 로드 밸런싱 메커니즘의 개념.

패킷 로드 밸런싱 메커니즘은 패킷을 처리유닛 간에 적절히 분배하여 네트워크 성능을 최대화 시키는 것을 목적으로 한다. 네트워크 성능을 최대화하기 위해선 다음의 두 가지를 고려하여 패킷 로드 밸런싱을 수행해야 한다. 첫째는 패킷들이 다수의 처리유닛 상에서 최대한 병렬적으로 처리 될 수 있도록 해야만 한다. 이는 패킷들을 큐간에 고르게 분배시킴

으로써 달성 될 수 있다. 둘째는 패킷이 처리유닛에서 순차적으로 처리 될 수 있도록 해야 한다. 다양한 네트워크 프로토콜이 패킷이 순차적으로 전송되는 것을 가정하기 때문에 패킷 순서 보존을 지켜주는 것이 필요하다. 만약 이가 지켜지지 않는다면 네트워크 성능의 저하 문제를 일으킬 수 있게 된다. 예로 TCP/IP 패킷의 경우 패킷의 처리가 순차적이지 않을 때 패킷 재정렬 현상이 일어나 해당 패킷의 처리가 지연되거나 폐기되는 문제를 겪을 수 있다.[2] 이를 해결하기 위해선 동일한 flow에 대해선 동일한 처리유닛에서 처리될 수 있도록 보장하는 것이 필요하다. 여기서 flow란 동일한 송수신지를 갖는 패킷들의 집합을 일컫는다. 이는 일반적으로 5개의 튜플인 송수신지의 IP 및 포트번호, 그리고 통신 프로토콜 번호로 식별 될 수 있다.

1.2 관련 연구

Jon Bennett 외 2인은[2] 패킷을 병렬적으로 처리하는 경우 발생되는 패킷 재정렬 현상과 그로 인한 네트워크 성능 저하 문제에 대해 분석 하였다. 그들은 이 문제의 해결책으로 flow 단위로 패킷을 처리하는 정적 패킷 로드 밸런싱 메커니즘을 제시하였다. Laor 와 Gendel는[3] 패킷 재정렬 현상으로 인한 네트워크 성능 저하 문제를 완화시키는 SACK과 D-SACK 메커니즘이 사용되는 환경에서 패킷 재정렬로 인한 네트워크 성능 저하를 문제를 분석 하였다. 그들은 SACK과 D-SACK 메커니즘이 사용되는 환경에서도 flow 단위로 패킷을 처리하는 것이 높은 네트워크 성능을 얻는 방법이라고 밝혔다.

G. Dittmann 와 A. Herkersdorf은[4] flow 단위 정적 패킷 로드 밸런싱 기법을 사용할 경우 패킷이 처리유닛 간에 고르게 분배 되지 못하여 병렬처리성이 떨어지고 이로 인해 네트워크 성능이 저하된다는 점을 지적하였다. 그리고 이를 해결 하기위해 처리 유닛의 상태를 고려하여 동적으로 flow를 처리 유닛 간에 재분배시키는 동적 패킷 로드 밸런싱 메커니즘을 제안하였다.

Ju-Yeon Jo외 3인은[5] 동적 패킷 로드 밸런싱 메커니즘을 이용할 때에도 패킷 재정렬 현상이 발생 될 수 있다는 점과 이로 인해 야기되는 네트워크 성능 저하 현상에 주목하였다. 그들은 처리유닛 간에 flow를 재분배 할 때 패킷 재정렬 현상이 일어 날 수 있다는 점을 밝혔고 flow 재분배를 최소화하는 것이 재정렬 현상을 줄이고 네트워크 성능을 향상

시킬 수 있는 중요한 요인이라고 분석하였다. 그들은 flow 재분배 횟수를 최소화하기 위한 해결책으로 flow들의 용량을 고려하여 flow의 재분배하는 동적 패킷 로드 밸런싱 메커니즘을 제시하였다.

이 외에도 flow 단위 동적 패킷 로드 밸런싱 메커니즘은 라우터 도메인이나 분산 네트워크 도메인 등 다양한 도메인에서 연구된 사례가 많지만[6][7] 현재까지 가상 네트워크 어댑터 도메인에서 연구된 사례는 아직 존재하지 않는다.

1.3 연구 내용 및 논문의 구성

현존하는 다중 큐 가상 네트워크 어댑터는 오직 flow 단위의 정적 패킷 로드 밸런싱 메커니즘에 기반을 둔다. 하지만 정적 패킷 로드 밸런싱 메커니즘은 여러 flow들이 특정 큐에 집중되어 처리 될 수 있기 때문에 병렬 처리성이 떨어져 네트워크 성능이 저하되는 문제를 내포한다. 본 논문에서는 다중 큐 가상 네트워크 어댑터 도메인에서 flow가 특정 큐에 집중되어 네트워크 성능이 저하되는 문제를 해결하는 것을 목표로 한다. 이를 위해 다중 큐 가상 네트워크 어댑터를 위한 동적 패킷 로드 밸런싱 메커니즘을 제안한다. 제안하는 로드 밸런싱 메커니즘은 큐 간 로드를 고려하여 flow를 동적으로 재분배시키기 때문에 flow들이 특정 큐에 집중되는 현상을 완화하여 네트워크 성능이 저하되는 문제를 해결 할 수 있다.

논문의 구성은 다음과 같다. 먼저 2장에서는 대상 시스템 모델링 및 패킷 처리 과정을 분석하고 문제점을 제시한다. 3장에서는 제시한 문제점을 해결 할 수 있는 동적 로드 밸런싱 메커니즘을 제안한다. 4장에서는 제안한 패킷 로드 밸런싱 메커니즘의 효용성을 검증하기 위한 실험을 하고 그에 대한 결과를 평가한다. 그리고 마지막으로 5장에서는 본 논문의 결론을 내린다.

제 2 장 문제 제시

이번 장에서는 대상 시스템의 정의와 해당 시스템에서의 패킷 처리 과정을 분석 한다. 그리고 대상 시스템에서 발생될 수 있는 네트워크 성능 저하 문제에 대해서 살펴본다.

2.1 대상 시스템

본 연구에서 대상으로 하는 다중 큐 가상 네트워크 어댑터는 Redhat이 개발한 다중 큐 virtio 가상 네트워크 어댑터이다.[1] 이를 대상으로 하는 이유는 이것이 현존하는 상용 다중 큐 가상 네트워크 어댑터로써는 유일하기 때문이다. 다중 큐 virtio 가상 네트워크 어댑터의 소스코드는 [10][11]에서 찾아 볼 수 있다. 현재 다중 큐 virtio 가상 네트워크 어댑터는 KVM 가상 머신에서만 지원되고 있다.[1] KVM이 수행되는 시스템은 VMM이 OS안에서 수행되는 구조를 갖는데 이와 같은 VMM을 hosted VMM이라고 한다. 본 논문에서는 hosted VMM이 수행되는 시스템을 대상 시스템으로 하며 이러한 시스템의 구조는 그림 3.1과 같다.

이러한 시스템에서는 크게 2가지 종류의 OS가 수행된다. 첫째는 물리 머신 바로 위에서 수행되는 호스트 OS, 둘째는 가상 머신 위에서 수행되는 게스트 OS이다. 그리고 두 종류의 OS위에서 수행되는 응용을 각각 호스트 응용, 게스트 응용이라고 한다.

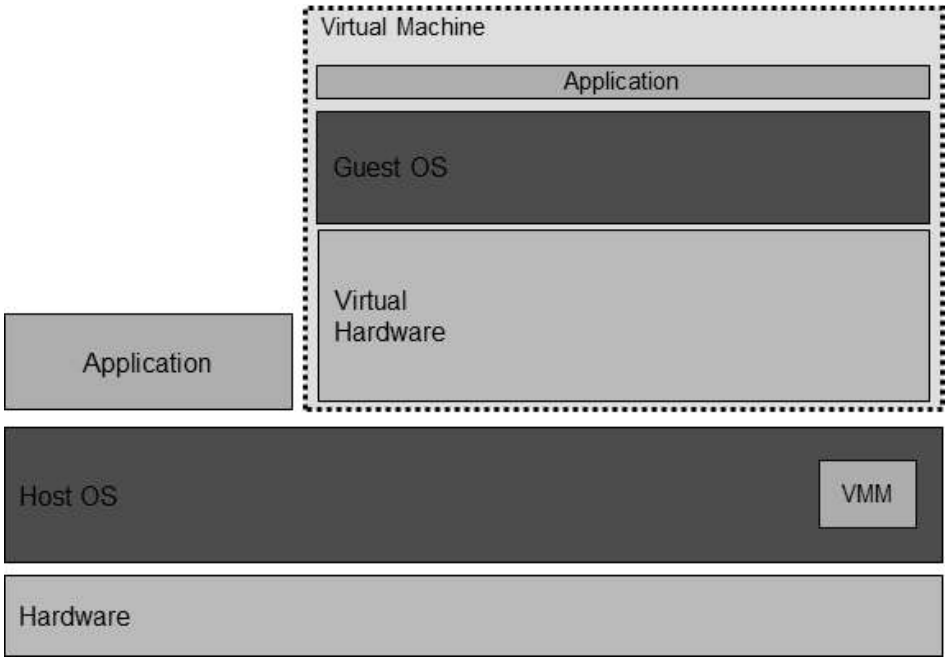

그림 2.1 대상 시스템의 구조.

2.2 패킷 처리 과정

그림 2.2은 호스트 응용으로부터 게스트 응용으로 패킷이 전송되는 과정을 보인다.

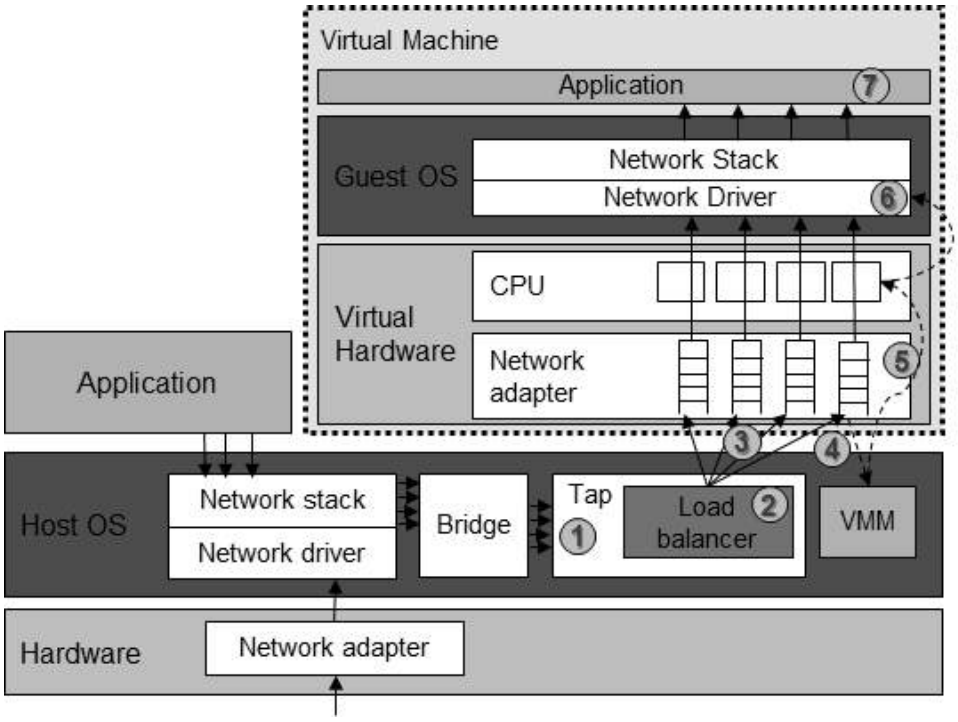

그림 2.2 패킷 처리 과정.

게스트 응용으로 전송되는 패킷은 우선 호스트 OS에서 동작하는 bridge를 거쳐 tap으로 전송된다. 패킷을 수신한 tap은 내부에 구현된 로드 밸런싱 메커니즘을 통해 패킷이 전송 될 큐를 선택한다. 로드 밸런싱

메커니즘에 의해 패킷이 전송될 큐가 선택되었다면 tap은 해당 큐로 패킷을 전송하게 된다. 패킷이 큐에 전송된 후에 가상 네트워크 어댑터는 해당 사실을 VMM에게 신호를 보내 알린다. 신호를 수신한 VMM은 패킷이 삽입된 큐와 1:1로 대응하는 가상 CPU에게 인터럽트를 보낸다. 인터럽트를 수신한 가상 CPU는 게스트 OS에 존재하는 네트워크 드라이버의 인터럽트 서비스 루틴을 수행하여 큐에 저장된 패킷을 네트워크 스택을 거친 뒤 게스트 응용으로 전달하게 된다.

2.3 기존 시스템에서의 네트워크 성능

대상 다중 큐 가상 네트워크 어댑터는 패킷의 병렬처리를 위해 하나 이상의 큐를 갖고 각각이 가상 CPU와 1:1로 대응되어 동작하는 구조를 갖는다. 이러한 구조로 인해 flow가 큐 간에 얼마나 고르게 분배 되느냐에 따라 패킷 처리를 위해 사용되는 가상 CPU의 개수가 달라지므로 네트워크 성능이 차이 나게 된다. 이러한 사실을 명확히 보이기 위해 표2.1와 같은 환경에서 flow 분배 정도에 따른 네트워크 성능을 측정하였다.

표 2.1 실험 환경 I.

Machine	Item	Specification
Physical Machine	CPU	i7@3.20GHz x 4
	Memory	8GB
	OS	Linux kernel 3.7.0
	VMM	KVM
	Emulator	QEMU-KVM 1.1.50
Virtual Machine	CPU	i7@3.20GHz x 4
	Memory	1GB
	OS	Linux kernel 3.7.0
	Network adapter	Multi-queue virtio virtual network adapter

실험은 호스트 응용에서 게스트 응용으로 패킷을 전송하는 4개의 flow가 존재할 때 flow 분배 정도에 따른 네트워크 성능을 측정하는 것을 목적으로 한다. 네트워크 성능을 측정하기 위해 호스트 응용에서 게스트 응용으로 패킷을 전송하는 netperf[9] 인스턴스를 4개 수행하였고 flow들이 1개의 큐에서 처리되는 경우와 2, 3, 4개의 큐에서 분배되어 처리되는 경우를 나누어서 네트워크 성능을 측정하였다. 패킷이 어떠한 큐에서 처리되는 것을 분석하기 위해 리눅스의 /proc/interrupt 파일을 참조하였다. /proc/interrupt 파일은 시스템 내에 존재하는 CPU들에 대해서 각 CPU에 요청된 인터럽트 횟수를 저장하는 파일이다. 이를 살펴보면 어떠한 가상 CPU에서 인터럽트 서비스 루틴이 수행되어 패킷을 처리했는지 알 수 있으므로 flow들이 어떠한 큐에서 분배되어 처리되는지 확인 할 수 있다. 그림 2.3는 해당 실험의 결과를 나타낸다.

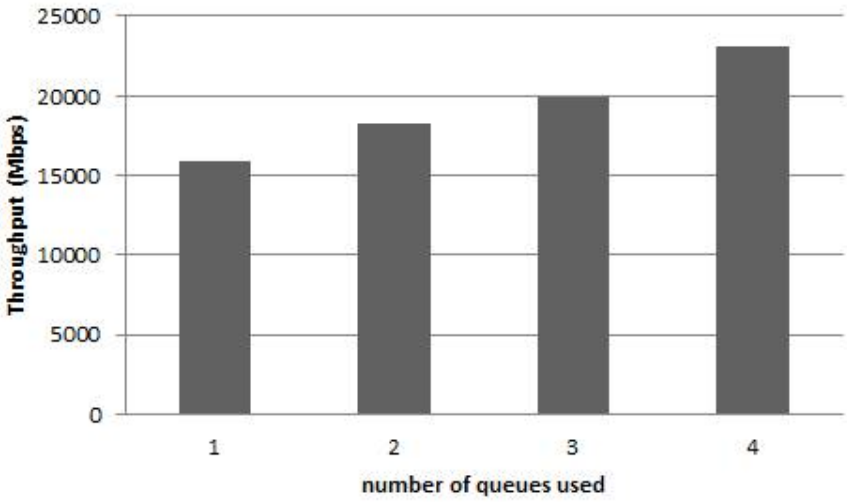

그림 2.3 flow 분배 정도에 따른 네트워크 성능.

그래프에서 볼 수 있듯이 flow들이 큐에 분배되는 정도에 따라 다중 큐 가상 네트워크 어댑터의 네트워크 성능이 상당히 차이가 남을 알 수 있다. 4개의 flow가 오직 하나의 큐에서만 분배되어 처리될 경우 네트워크 성능은 15Gbps 정도만을 보이는 반면 4개의 큐에 고르게 분배된 경우 네트워크 성능은 23Gbps 정도의 성능을 보인다.

2.4 기존 패킷 로드 밸런싱 메커니즘과 문제점

2.3절에서 살펴본 바와 같이 다중 큐 가상 네트워크 어댑터는 flow들이 큐에 분배되는 정도에 따라 네트워크 성능이 차이 나게 된다. 때문에 높은 네트워크 성능을 얻기 위해선 flow들을 큐간에 고르게 분배시키는 패킷 로드 밸런싱 메커니즘을 사용하는 것이 중요하다. 본 절에서는 기존 시스템에서 사용되던 패킷 로드 밸런싱 메커니즘을 분석하여 기존 패킷 로드 밸런싱 메커니즘의 문제점을 도출한다.

기존 시스템은 Tom Herbert가 개발한 Receive Packet Steering[8] 이라는 패킷 로드 밸런싱 메커니즘에 기반을 둔다. Receive Packet Steering은 flow 단위의 정적 패킷 로드 밸런싱 메커니즘의 하나로 수식 2.1과 같이 flow_id를 다중 큐 가상 네트워크 어댑터가 갖는 큐의 개수로 나누는 연산을 통해 flow가 처리될 큐를 선택하게 된다. 여기서 flow_id란 패킷 송수신지의 IP 및 포트번호, 그리고 네트워크 프로토콜 번호를 32-bit 값으로 해시한 값을 일컫는다.

$$\text{selected queue} = \text{flow\_id} \ \% \ N \tag{2.1}$$

위의 메커니즘은 로드 밸런싱을 위한 연산이 매우 단순하기 때문에 사용되는 CPU 자원이 매우 적게 드는 장점이 있다. 하지만 연산 결과가 동일한 flow들에 대해선 항상 동일한 큐에 flow를 분배하기 때문에 특정

큐에 flow들이 집중되어 패킷의 병렬처리성이 떨어지는 문제를 갖고 있다.

이러한 패킷 로드 밸런싱 메커니즘은 flow의 개수가 많이 발생되는 대형 시스템에서는 큰수의 법칙에 의해 결과적으로 flow들이 큐간에 균일하게 분배되므로 유용하게 사용 될 수 있다. 하지만 flow의 개수가 적게 발생되는 소형 시스템에서는 특정 flow가 특정 큐에 집중되는 현상이 빈번히 발생 될 수 있으므로 적합하지 않다. 결국 데스크톱이나 차량용 인포테인먼트와 같은 소형 시스템에서는 동적 패킷 로드 밸런싱 메커니즘을 이용하는 것이 더 유리하다고 볼 수 있다.

제 3 장 동적 패킷 로드 밸런싱 메커니즘 설계

2장에서 언급한 것처럼 flow 단위의 정적 패킷 로드 밸런싱 메커니즘은 flow들이 특정 큐에 집중되어 네트워크 성능이 저하되는 문제를 갖는다. 본 장에서는 이러한 문제를 해결하기 위한 동적 패킷 로드 밸런싱 메커니즘을 제안한다. 제안하는 동적 패킷 로드 밸런싱 메커니즘의 구성은 다음과 같다.

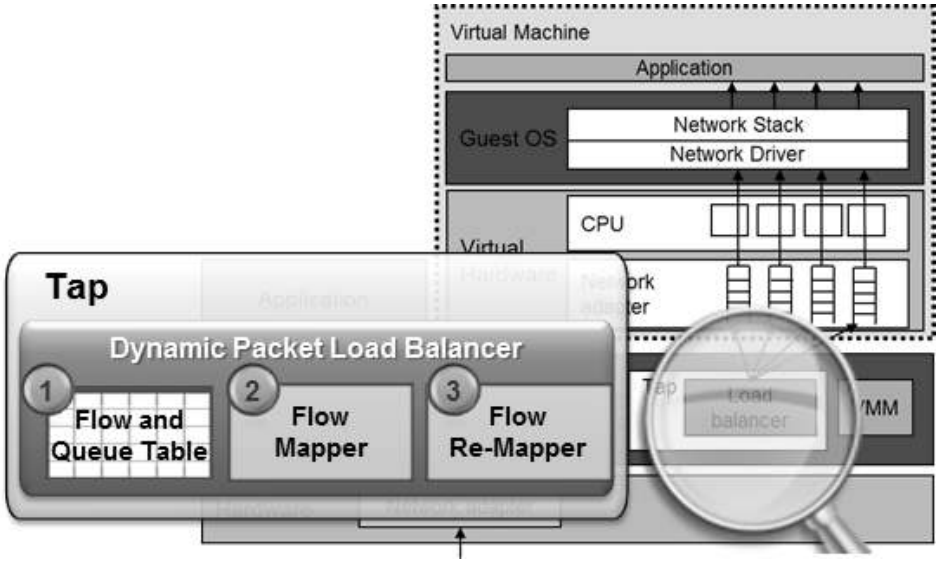

그림 3.1 제안하는 로드 밸런싱 메커니즘의 구성

(1) flow와 큐의 상태 정보를 저장하는 flow & queue table,

(2) tap 디바이스에 패킷이 도달할 때마다 패킷을 큐 간에 분배하는 flow mapper,

(3) 일정 주기마다 큐 간 로드 밸런싱을 하는 flow re-mapper.

3.1 Flow & Queue Table

Flow & Queue Table은 tap에 유입되는 flow들의 정보를 저장하는 FLOW_TABLE과 가상 네트워크 어댑터가 가지는 큐들에 대한 정보를 저장하는 QUEUE_TABLE이 저장되는 공유 메모리 공간이다.

FLOW_TABLE은 flow_id, queue_index, packet_count라는 변수를 저장하는데 flow_id는 flow를 식별을 할 수 있는 flow 식별자 값을 저장하는 변수이다. flow 식별자는 송수신지의 IP 및 포트번호, 그리고 통신 프로토콜 번호를 32-bit CRC로 해시한 값이다. queue_index는 해당 flow가 분배 된 큐의 번호를 저장하는 변수이고 packet_count는 일정 주기 동안 해당 flow에서 유입된 패킷 수를 저장하는 변수이다.

QUEUE_TABLE은 queue_index와 queue_load라는 변수를 저장하는데 queue_index는 가상 네트워크 어댑터의 큐를 식별 할 수 있는 숫자를 저장하는 변수이며 queue_load는 각 큐의 부하가 저장되는 변수이다. 큐의 부하는 큐에 저장되어 처리를 기다리는 패킷들의 개수로 정의된다.

3.2 Flow Mapper

flow mapper는 패킷이 어떠한 큐에서 처리 되어야 하는지 결정하기 위해 존재한다. flow mapper는 동일한 flow에 해당하는 패킷은 동일한 가상 CPU에서 처리 될 수 있도록 보장하는 것이 특징이다. flow mapper의 구체적인 동작 과정은 아래와 같다.

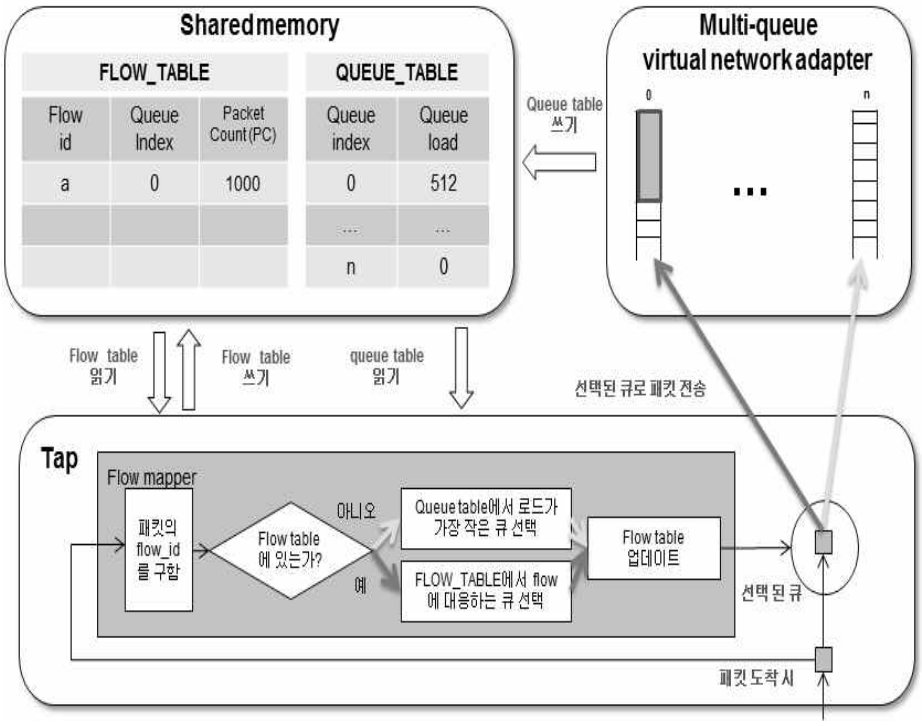

그림 4.2 flow mapper의 동작 과정.

우선 패킷이 tap에 도착하면 해당 패킷이 새로운 flow 인지 아닌지의 여부를 확인한다. 이는 해당 패킷에 대한 flow 식별자를 구한 뒤 이를 flow 테이블에서 검색하는 방식으로 이루어진다. 만약 flow 테이블에 해당 flow 식별자가 등록되지 않았을 경우엔 새로운 flow가 유입되었다고 판단한다. 새로운 flow가 유입되었다고 판단된다면 큐 테이블을 참고해 큐 로드가 가장 작은 큐로 패킷이 전송되며 이렇게 결정된 사항은 flow 테이블에 저장된다. 그리고 만약 flow가 기존에 존재하던 것이었을 경우엔 flow 테이블에 등록 된 flow에 대응하는 큐에 패킷이 전송되도록 한다. 그림 3.3은 flow mapper가 수행하는 의사코드를 나타낸다.

```
Algorithm: Flow mapping

// triggered at every packet arrival
// returns Q: Target virtual network queue
Function: flow_mapping(PACKET, FLOW_TABLE, QUEUE_TABLE)
1: F <- calculate flow identifier of the PACKET
2: Q <- find queue corresponds to F in FLOW_TABLE
3: if(Q == NULL) then
4:     Q <- find queue which has lowest load in QUEUE_TABLE
5:     update Q which corresponds to F in FLOW_TABLE
6: endif
7: return ( Q )
```

그림 3.3 flow mapping 알고리즘의 상세

3.3 Flow Remapper

flow remapper 메커니즘은 큐 간의 로드를 고르게 하기 위해 사용되는 메커니즘이다. 이는 일정 주기마다 큐 간에 로드를 비교하여 flow를 재분배 시키는 메커니즘을 수행하여 이루어진다. 또한 flow 재분배 횟수를 줄이기 위해 flow들의 용량을 고려하여 flow를 재분배 시키도록 한다. flow 재 매핑 메커니즘의 동작 과정은 아래와 같다.

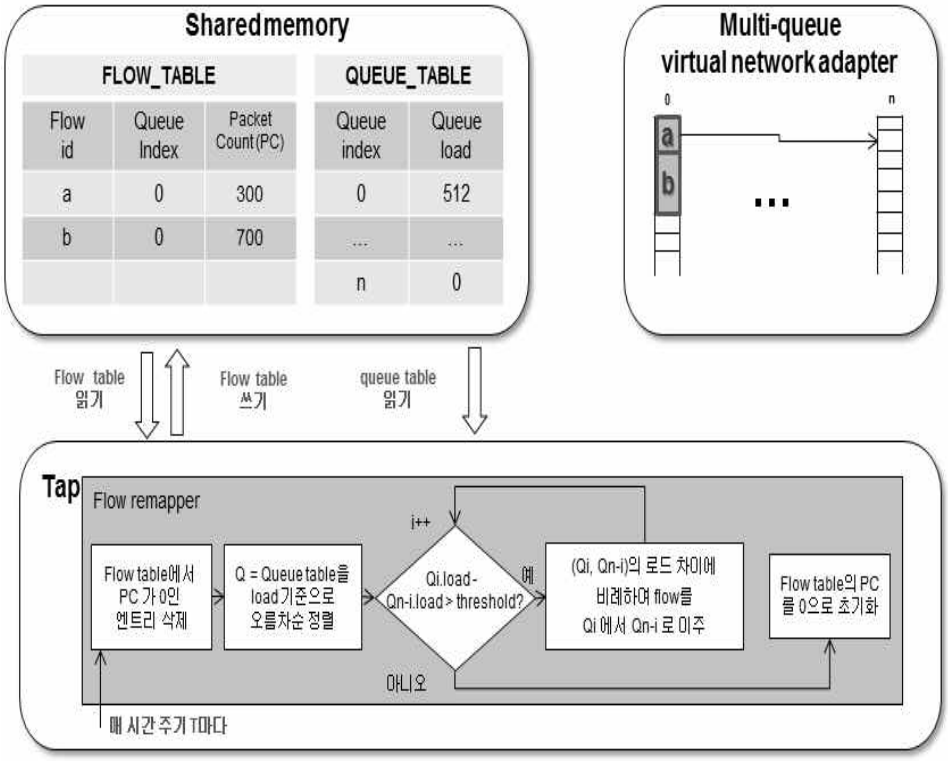

그림 3.4 flow mapper의 동작 과정.

우선 flow remapper는 QUEUE_TABLE을 로드를 기준으로 오름차순으로 정렬 된 리스트를 생성한다. 그리고 해당 리스트에서 i번째 엔트리와 n-i번째 엔트리에 해당하는 큐의 로드를 비교하여 두 큐간 로드의 불균형 상태를 검사하게 된다.(i=0, 1,...,n/2) 불균형 상태는 두 큐의 로드차가 threshold를 넘을 경우 불균형 상태로 정의한다. 두 큐가 불균형 상태로 판단된 경우 i번째 큐에 분배 된 flow들을 n-i번째 큐로 재분배시킴으로써 큐간 로드가 균일화 되도록 동작한다. 이때 재분배되는 flow는 두 큐의 로드차에 비례하도록 한다. 그림 3.5는 flow mapper가 수행하는 의사코드를 나타낸다.

```
Algorithm: Flow remapping

// triggered at every time interval T
// load balance between queues by migrating flows
Function: flow_remapping(FLOW_TABLE, QUEUE_TABLE)
1:  Delete entries which has 0 packet size in FLOW_TABLE
2:  Q_LIST <- sort QUEUE_TABLE as ascending order by queue load
3:  i <- 0
4:  n <- number of entries in Q_LIST
5:  while(i < n/2)
6:  then
7:    if (Q_LIST(i).load - Q_LIST(n-i).load > threshold)
8:    then
9:      F_LIST1 <- list of entries corresponds to Q_LIST(i).index in FLOW_TABLE
10:     F_LIST2 <- list of entries corresponds to Q_LIST(n-i).index in FLOW_TABLE
11:     R <- ((Q_LIST(i).load - Q_LIST(n-i).load)/Q_LIST(i).load)/2
12:     call migrate_flows(F_LIST1, F_LIST2, R)
13:  endif
14:  i <- i+1
15: end
16: set packet count to 0 for all entries in FLOW_TABLE
17: return

//migrate flows
Function: migrate_flows (F_LIST1, F_LIST2, R)
1:  SUM1 <- sum packet count of all entries in F_LIST1
2:  j <- 0
3:  m <- number of entries in F_LIST1
4:  M <- R * SUM
5:  while(j < m)
6:  then
7:    if (F_LIST1(i).packet_cnt < M)
8:    then
9:      M <- M - F_LIST(i).packet_cnt
10:      F_LIST1(i).queue_index <- F_LIST2(i).queue_index
11:    endif
12:  i <- i+1
13: end
14: return
```

그림 3.5 flow mapping 알고리즘의 상세

제 4 장 실험 및 평가

4.1 실험 환경

실험에서 사용된 물리 머신과 가상 머신의 구성은 표 8.1에 나타난 것과 같다. 물리 머신은 3.20GHz로 동작하는 Intel i7 CPU와 8GB의 메인 메모리를 사용한다. 운영체제로는 Linux kernel 3.7.0이 사용 되며 VMM으로 KVM 가상 머신 모니터가 사용 되었다.

표 5.1 실험 환경 II.

Machine	Item	Specification
Physical Machine	CPU	i7@3.20GHz x 4
	Memory	8GB
	OS	Linux kernel 3.7.0
	VMM	KVM
	Emulator	QEMU-KVM 1.1.50
Virtual Machine	CPU	i7@3.20GHz x 4
	Memory	1GB
	OS	Linux kernel 3.7.0
	Network adapter	Multi-queue virtio virtual network adapter

가상 머신은 4개의 가상 CPU와 1GB의 메인 메모리를 사용하며 운영 체제로는 Linux kernel 3.7.0이 사용 되었다. 가상 네트워크 어댑터로는 다중 큐 virtio 가상 네트워크 어댑터가 사용되었다.

4.2 실험 설계

제안한 패킷 로드 밸런싱 메커니즘의 검증은 기존 패킷 로드 밸런싱 메커니즘이 적용된 시스템과 제안한 패킷 로드 밸런싱 메커니즘이 적용된 시스템에서 네트워크 성능을 측정하여 비교하는 방식으로 이루어진다. 네트워크 성능은 초당 수신된 패킷의 양으로 정의되며 이를 수식으로 표현하면 식 (5.1)과 같다.

$$Throughput = \frac{size\,of\,received\,packets}{second} \tag{5.1}$$

검증을 위한 실험 시나리오의 구성은 그림 4.2과 같다.

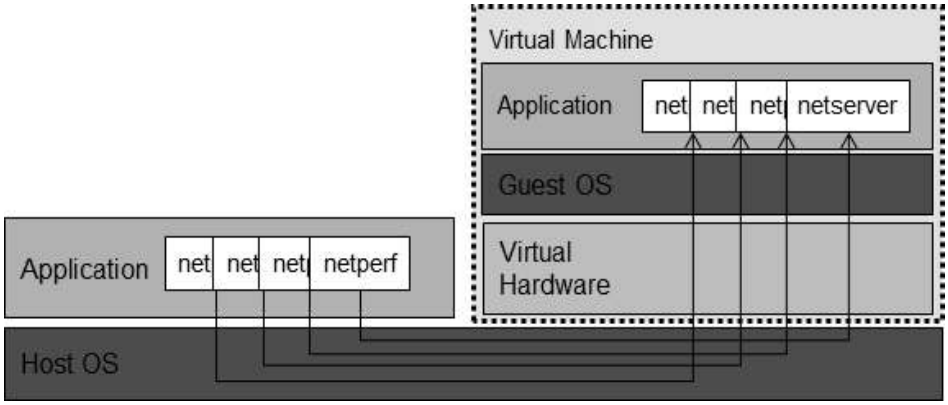

그림 4.2 실험 시나리오의 구성.

실험 시나리오는 호스트 응용에서 게스트 응용으로 TCP/IP 패킷을 전송하는 것이다. 이때 네트워크 대역폭을 최대한 사용하기 위해 netperf 툴을 이용한다.[9] netperf는 네트워크 성능을 측정하는 벤치마킹 툴로 클라이언트인 netperf와 서버인 netserver로 나뉘어 동작한다. 호스트 응용에서 게스트 응용으로 패킷을 전송하기 위해 netperf는 호스트에서 수행하며 netserver는 게스트에서 수행한다. 또한 가상 머신이 갖는 가상 CPU의 사용을 최대화하기 위해 flow를 4개 생성하였으며 이는 1회당 10초씩 총 20번 수행하였다. flow remapper의 로드 밸런싱 주기는 0.2초로 설정하였으며 threshold 값은 최대 큐 길이의 25%에 해당하는 256으로 설정하였다.

4.3 실험 결과

실험 결과는 다음과 같다. 그림 4.3은 기존 시스템과 제안한 시스템 간의 네트워크 성능을 시행 횟수별로 비교한 그래프이다. 그래프에서 보이듯이 기존 패킷 로드 밸런싱 메커니즘이 적용된 시스템에서는 네트워크 성능이 저하되는 경우가 종종 발생한다.

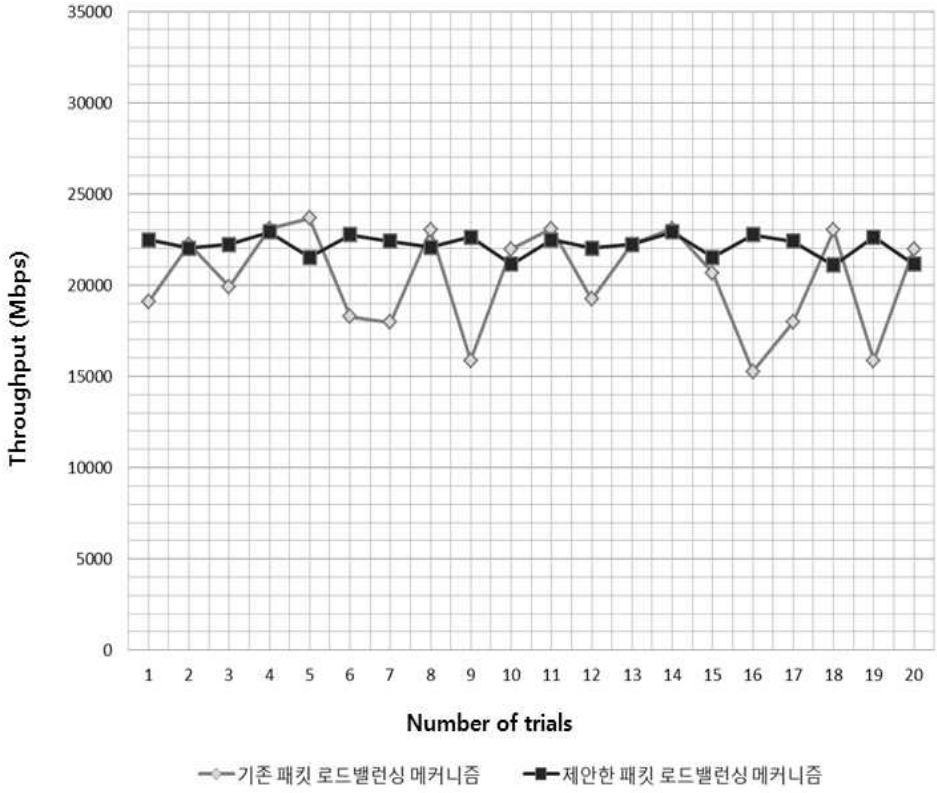

그림 4.3 시행 횟수별 네트워크 성능 비교.

이는 2장에서 언급한 것처럼 flow가 특정 큐로 집중되어 생기는 현상이다. 반면에 제안한 패킷 로드 밸런싱 메커니즘이 적용된 시스템에서는 네트워크 성능이 저하되는 경우가 발생하지 않는다. 이는 제안한 동적 패킷 로드 밸런싱 메커니즘이 큐 간 로드를 균일화하는 메커니즘을 수행하여 flow가 큐 간에 고르게 분배되기 때문이다. 그림 4.4는 두 시스템에서의 평균 네트워크 성능을 비교한 그래프이다.

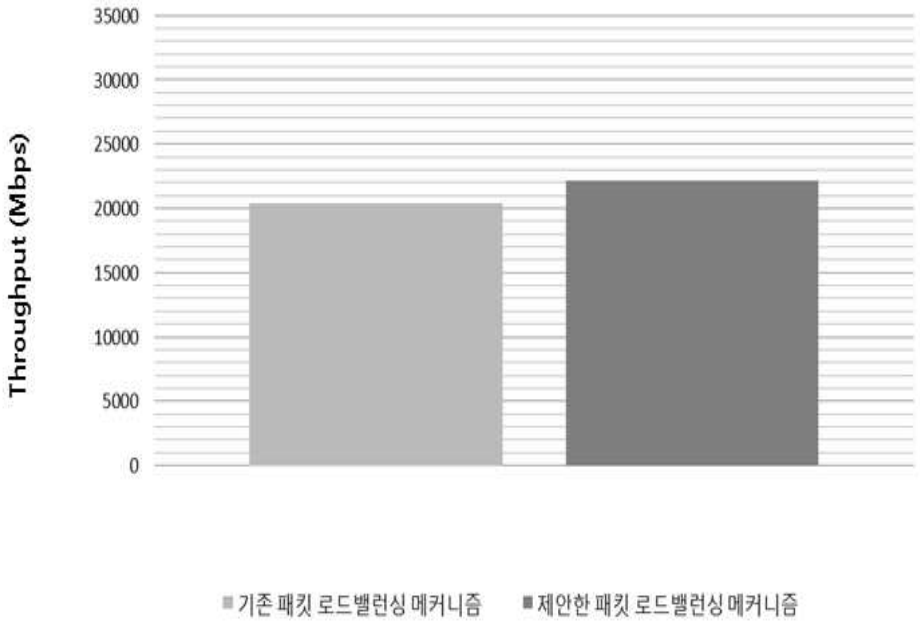

그림 4.4 평균 네트워크 성능 비교.

그래프에서 알 수 있듯이 제안한 패킷 로드 밸런싱 메커니즘이 적용된 시스템의 경우 기존 패킷 로드 밸런싱 메커니즘이 적용된 시스템보다 약 8%정도의 네트워크 성능이 향상되었다.

제 5 장 결 론

다중 큐 가상 네트워크 어댑터는 가상 CPU 개수만큼 큐를 가지고 큐 각각이 가상 CPU에 1:1로 대응되어 동작하는 구조를 갖는다. 이러한 구조로 인해 flow들이 큐에 얼마나 고르게 분배되느냐에 따라 패킷 처리를 위해 사용되는 가상 CPU의 개수가 달라져 네트워크 성능의 차이가 나타나게 된다. 이 사실을 확인하기 위해 주어진 환경에서 flow 분배 정도에 따른 네트워크 성능을 실험해본 결과 flow 분배 정도에 따라 최대 약 30% 정도 네트워크 성능의 차이가 나는 것을 알 수 있었다.

현존하는 다중 큐 가상 네트워크 어댑터는 Tom Herbert가 개발한 Receive Packet Steering[8] 이라는 패킷 로드 밸런싱 메커니즘에 기반을 두고 있다. 이는 flow 단위의 정적 패킷 로드 밸런싱 메커니즘으로 flow들이 특정 큐에 집중되어 처리되는 현상이 종종 발생 된다. 이러한 현상은 곧 다중 큐 가상 네트워크 어댑터의 네트워크 성능의 저하 문제로 이어지게 된다.

본 논문에서는 이러한 문제를 해결하기 위해 다중 큐 가상 네트워크 어댑터를 위한 flow 단위 동적 패킷 로드 밸런싱 메커니즘을 제안했다. 제안한 동적 패킷 로드 밸런싱 메커니즘은 큐 로드를 고려하여 flow들을 큐 간에 재분배시키기 때문에 특정 큐에 flow가 집중되는 현상을 완화하여 네트워크 성능이 저하되는 문제를 해결한다.

제안한 패킷 로드 밸런싱 메커니즘의 효용성을 검증하기 위해 기존의 패킷 로드 밸런싱 메커니즘과 제안한 로드 밸런싱 메커니즘이 적용된 시스템에서 네트워크 성능을 실험을 통해 비교하였다. 실험은 호스트 응용에서 게스트 응용으로 패킷을 전송할 때 네트워크 성능을 측정하는 방법을 통해 이루어 졌다. 실험 결과 본 논문에서 제안한 패킷 로드 밸런싱 메커니즘이 주어진 시나리오 상에서 기존 패킷 분배 메커니즘보다 평균 약 8%정도 더 높은 네트워크 성능을 보임을 확인할 수 있었다.

참 고 문 헌

[1] Multi-queue virtio virtual network adapter, http://www.linux-kvm.org/page/Multiqueue.

[2] Jon C. R. Bennett, Craig Partridge, and Nicholas Shectman, "Packet Reordering is Not Pathological Network Behavior“, IEEE/ACM Transactions on Networking, December 1999.

[3] M. Laor and L. Gendel, "The Effect of Packet Reordering in a Backbone Link on Application Throughput", IEEE Network, 2002.

[4] G. Dittmann and A. Herkersdorf, “Network Processor Load Balancing for High Speed Links,” Proc. of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems, 2002.

[5] Ju-Yeon Jo, Yoohwan Kim, H. Jonathan Chao, Frank Merat, “Internet Traffic Load Balancing Using Dynamic Hashing with Flow Volume,” The Smithsonian/NASA Astrophysics Data

System, 2002.

[6] Yaxuan Qi, Bo Xu, Fei He, Baohua Yang, Jianming Yu, and Jun Li, "Towards High Performance Flow Level Packet Processing on Multi-core Network Processors," Architecture for Networking and Communications Systems, 2007.

[7] I Khazali, A Agarwal, “Load balancing with Minimal Flow Remapping for Network Processors,” Computers and Communications, 2012.

[8] Receive Packet Steering, http://lwn.net/Articles/361440.

[9] Netperf Benchmark Tool. http://www.netperf.org.

[10] Multi-queue Virtio Virtual Network Adapter: kernel source code, git://github.com/jasowang/kernel-mq.git.

[11] Multi-queue Virtio Virtual Network Adapter: emulator source code, git://github.com/jasowang/qemu-kvm-mq.git.

Abstract

Multi-queue virtual network adapter has an ability to process packets in parallel which are to be sent to a virtual machine. Since it have multiple queues to process packets, a mechanism that distributes packets among the queues is needed. This kind of mechanism is called packet load balancing mechanism. Existing multi-queue virtual network adapter only depends on flow based static packet load balancing mechanism. The way it selects a queue to process a packet is done by using the result of moduloing flow identifier value (e.g. IP and port number of sender and receiver, and protocol type number) by the number of queues in virtual network adapter. This is appropriate on large scale systems which manages a huge number of flows. However in small scale systems which only manages a small number of flows, may suffer from performance degradation when flows are concentrated in a certain queue.

To solve this problem we propose a dynamic packet load balancing mechanism for multi-queue virtual network adapter. Proposed mechanism balance load among queues by periodically redistributing flows among the queues regarding the load of the queues. For

validation we have implemented our mechanism in KVM virtual environment. Experimental result showed that our load balancing mechanism improved 8% of network performance compared to existing packet load balancing mechanism.

Keywords : KVM, Network I/O virtulization, Load balancing

Student number : 2010-23960