DOI: 10.3745/KIPSTC.2009.16-C.4.417

듀얼 프로세서 기반 DPI (Deep Packet Inspection) 엔진을 위한 효율적 패킷 프로세싱 방안 구현 및 성능 분석

양 준 호[†] · 한 승 재[††]

요 약

특화된 하드웨어의 도움 없이 범용 다중 프로세서 플랫폼에서 DPI(Deep Packet Inspection) 시스템을 구현하는 방법은 비용 측면에서 매력적이다. 문제는 성능인데, 일반적으로 다중 프로세서 시스템에서는 작업들을 여러 프로세서에 적절하게 배분하는 로드밸런싱 방법과 DPI 프로세싱 전용 개별 프로세서를 지정하여 시스템의 성능을 향상 시킨다. 그러나, 우리는 DPI 시스템의 경우 위와 같은 단순한 프로세서 통제 방안이 반드시 최선책이 아니라고 생각한다. 본 논문에서는 작업의 종류에 따라 정해진 프로세서에 할당한 후, 프로세서 상태에 따라 역할을 변경하는 방식을 제안한다. 우리는 제안하는 방식을 리눅스 기반 듀얼 프로세서 시스템에 구현하고 실험을 통해 그 성능을 기존의 로드밸런싱 방식과 비교하였다. 제안된 방식에서는 하나의 프로세서는 인터럽트 처리를 포함한 일반적 패킷 프로세싱 역할만을 담당토록 하고 다른 프로세서는 DPI엔진을 전담하도록 역할로 분리시켜 캐시접근실패 (cache miss) 과 스핀락(spin lock) 발생빈도를 낮추었으며, DPI 전담 프로세서가 처리 한계에 이르렀을 경우에는 두 프로세서 모두 DPI를 위해 자원을 사용토록 하여, 기존의 리눅스 로드 밸런싱 방식 DPI 시스템 대비 약 60%의 성능향상을 달성하였다.

키워드 : 네트워크 보안, DPI(Deep Packet Inspection), 듀얼 프로세서 시스템, 부하균형

Implementation and Performance Analysis of Efficient Packet Processing Method For DPI (Deep Packet Inspection) System using Dual-Processors

Joon-Ho Yang[†] · Seung-Jae Han[††]

ABSTRACT

Implementation of DPI(Deep Packet Inspection) system on a general purpose multiprocessor platform is an attractive option from the implementation cost point of view, since it does not require high-cost customized hardware. Load balancing has been considered as a primary means to achieve high performance in multi processor systems. We claim, however, that in case of DPI system design simply balancing the load of each processor does not necessarily yield the highest system performance. Instead, we propose a method in which tasks are allocated to processors based on their functions. We implemented the proposed method in dual processor Linux system and compare its performance with the existing load balancing methods. Under the proposed method, one processor is dedicated to deal with interrupt handling and generic packet processing, while another processor is dedicated to DPI processing. According to experimental results, the proposed scheme outperforms the existing schemes by 60%, mainly because of the reduction of cache miss and spin lock occurrences.

Keywords : Network Security, DPI(Deep Packet Inspection), Multi-processor system, Load balancing, Network security

1. 서 론

인터넷 서비스 증가로 네트워크 트래픽 양이 빠르게 증가하고 있는 동시에, 해커나 침입자들의 공격 역시 급증하고 있다. 이런 상황은 실시간 고속 패킷 처리가 가능한 효율적 DPI(Deep Packet Inspection) [1]시스템을 필요로 한다. IPS(Intrusion prevention system) 또는 UTM(Unified threat management)등의 DPI는 패킷의 출발지와 목적지와 같은 IP 헤더 정보를 포함하여 패킷 전체를 분석하여 웜, 해킹 등의 공격 여부를 식별하는데, 일반적으로 실시간 DPI는 많은 프로세싱 자원을 필요로 한다. 이를 위해, 고가의 네트워크 장비에서는 주문형 반도체(ASIC), 네트워크 프로세서, 하드

※ 이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2008-331-D00496).
† 정 회 원 : 한국지역정보개발원 시도 사이버침해대응지원센터 선임연구원
†† 정 회 원 : 연세대학교 컴퓨터과학과 부교수
논문접수 : 2009년 2월 17일
수 정 일 : 1차 2009년 5월 12일
심사완료 : 2009년 6월 1일

웨어 가속기와 같은 특화된 하드웨어를 활용하는 DPI구현방법을 쓴다. 하드웨어 가속기나 주문형 반도체는 사용 용도에 최적화된 빠른 데이터 처리가 가능한 시스템을 구현할 수는 있는 반면, 구현 비용이 높고 구현 기간이 길다는 단점이 있다. 네트워크 프로세서 기반 구현은 주문형 반도체에 비해 변화 적응력에 강점이 있으나, 역시 설계와 개발과정이 복잡하고 비용이 높다는 단점이 있다. 한편, 범용 프로세서에 기반한 소프트웨어 구현방식은 높은 유연성을 제공하는 저비용 방식이다. 그러나, 실시간으로 패킷 시그니쳐 분석, traffic behavior 분석 등과 같은 복잡한 DPI기능을 단일 프로세서에 기반한 소프트웨어 방식으로 처리하는 데에는 트래픽 처리 용량이 제한되는 한계가 따른다. 이러한 문제를 해결하는 방안은 다중 프로세서를 활용하는 것인데, 다중 프로세서 환경에서DPI 엔진의 효율적 구현을 위해서는 DPI 작업의 특성에 맡는 자원관리 기법이 필수적이다. 일반적으로 다중 프로세서 시스템에서는 작업들을 여러 프로세서에 적절하게 배분하는 로드밸런싱 방법과 DPI 프로세싱 전용 개별 프로세서를 지정하여 시스템의 성능을 향상 시킨다. 그러나, 우리는 DPI 시스템의 경우 로드밸런싱에 의한 단순한 프로세서간 부하 균형이 반드시 최선책이 아니라고 생각한다. 본 논문에서는 듀얼 프로세서 환경에서 효율적 DPI 엔진 구현을 위해 인터럽트 처리를 포함한 일반적 패킷 프로세싱 역할만을 담당하는 프로세서와 DPI엔진만을 전담하는 프로세서로 역할을 분리한 후, DPI 를 전담하는 프로세서의 상태를 실시간으로 측정하여 그 부하가 정해진 임계치를 초과할 경우 두 프로세서 모두 DPI 처리를 수행토록 하는 새로운 방식을 제안한다.

우리는 제안하는 방식을 리눅스 기반 듀얼 프로세서 시스템에 구현하고 실험을 통해 그 성능을 기존의 로드밸런싱 방식과 비교하였다. 다중 프로세서 리눅스 시스템에서 패킷 처리 및 DPI 처리를 커널 스케쥴러에 맡기면 특정 프로세서로 작업이 집중되거나 프로세서간의 캐시 교환 및 동기화 비용의 증가로 인하여 그 성능이 저하될 우려가 있다. 본 논문에서는 듀얼 프로세서 기반에서 동작하는 리눅스 시스템을 환경에서 리눅스 기반, 로드밸런싱 방법으로 동작하는 kirqd[2] 방식과 irqbalance[3] 방식과 성능 비교를 하였으며, 제안하는 방식은 DPI를 수행하기 위한 작업들과 프로세서간의 바인딩하기 위하여 IRQ SMP Affinity [4] 기법을 이용한다. 아래에 이들 관련 기법들을 간략히 설명한다.

일반적으로 하드웨어 디바이스들은 시스템에 디바이스의 상태를 알리기 위해서 인터럽트를이용한다. 예를 들어 하나의 네트워크 패킷이 들어오면 특정 프로세서에게 해당 이벤트를 알리기 위해서 인터럽트를 보낸다. 인터럽트를 수신한 프로세서는 해당 이벤트를 처리하는 관련 커널 코드를 수행한다. 이때 특정 프로세서에게만 인터럽트가 집중되는 불균형을 피하기 위하여, 리눅스 배포판에서는 irqbalance 유틸리티를 제공한다. Irqbalance는 다중 프로세서 환경에서 프로세서들의 인터럽트를 분배하는 데몬이다. 즉 작업을 프로세서 사이에 스위칭하여 다중 프로세서 사이의 작업량을 조절하는 역할을 한다. 이를 위해 irqbalance는 이전에 어느 프로세서에 얼마나 많은 인터럽트가 발생했는지에 대한 통계정보를 유지하고 이 정보에 기반하여 성능 최적화와 파워 절약을 위한 정책을 설정한다. 다중 프로세서 기반 리눅스 시스템에서 IRQ의 분배는 커널에서 제공하는 kirqd 방식으로도 처리할 수 있다. kirqd는 매 실행 때마다 최소 작업을 수행한 프로세서를 휴리스틱 알고리즘을 통해 찾는다. 이전 작업 실행 이후부터 현재까지의 인터럽트의 발생 빈도를 통계적으로 분석하여 인터럽트 발생빈도가 가장 높은 프로세서가 "N"개의 IRQ 라인을 사용하고 있다면, 작업량이 낮은 프로세서로 "N/2 + 1"개의 IRQ라인을 이동시켜 모든 프로세서가 고르게인터럽트 처리를 수행하게 한다. 리눅스 커널에서 제공하는 IRQ SMP Affinity기능은 특정 하드웨어에서 발생한 IRQ신호를 지정된 프로세서로 전달할 수 있는 PROC 쉐도우 파일시스템 내 사용자 인터페이스 메모리 파일을 제공한다. 사용자는 하드웨어의 IRQ 번호를 확인하여 PROC Shadow 파일시스템 내 메모리 파일(/proc/irq/IRQ번호/smp_affinity)에 처리를 요구하는 프로세서 번호를 등록한다.

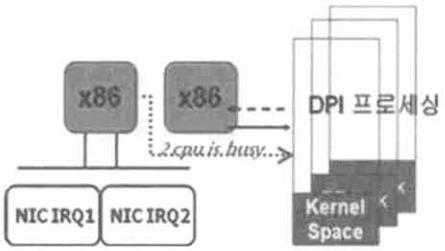
논문에서는 IRQ SMP Affinity를 이용하여 DPI 주요역할을 각 프로세서로 분리하여 처리토록 하였으며, DPI 전담 프로세서의 처리용량이 부족할 경우에는 타 프로세서에서도 DPI 프로세싱을 수행하는 방식을 제안한다. 제안된 방식에서는 기존의 로드밸런싱 방식에 비해 캐시접근실패율과 스핀락[5] 발생빈도를 크게 줄일 수 있다. 성능을 검증하기 위해서 리눅스 기반 듀얼 프로세서 환경에서 제안된 방식과 다른 방식들과 성능 비교를 수행하였다. 성능 비교 결과, 제안된 방식의 성능이 기존 방식에 비해 약 60% 이상 더 높게 나타났다.

논문은 다음과 같이 구성된다. 2 장에서는 제안하는 방식을 설명하고, 3 장에서의 성능평가 결과를 보인다. 마지막 4 장에서는 결론을 내린다.

2. IRQ처리와 DPI처리의 분리 구현방식

제안된 방식에서는 하나의 프로세서가 네트워크 디바이스 IRQ의 인터럽트 처리 등을 담당토록 하고, 다른 프로세서는 DPI 처리만을 담당하는 구조를 채택한다. 단 DPI 처리를 담당하는 프로세서가 처리 한계를 초과할 경우에는 두 프로세서 모두 함께 DPI 처리를 수행토록 한다. (그림 1)은 제안하는 방식을 듀얼 프로세서 환경에 적용하는 예를 보인다. 제안하는 방식은 단일 프로세서에 작업이 집중되는 현상을 막고 프로세서 사이의 작업을 명확히 분리하여 프로세서 사이의 동기화 비용 및 캐시접근실패율(L2 Cache Miss)을 현저하게 줄일 수 있을 뿐만 아니라 프로세서 상태에 따라 프로세서 역할을 재배치하여 프로세서 사용에 대한 효율성을 높일 수 있는 장점이 있다. 이러한 장점들은 프로토타입 구현과 실험을 통한 성능 측정으로 입증된다. 본 절에서는 제안된 방식을 듀얼 프로세서 리눅스 시스템에 구현하는 방법을 기술한다.

제안한 방안을 구현하기 위해서는 모든 IRQ라인을 첫 번째 프로세서에 바인딩하여 인터럽트가 해당 프로세서의인터럽트 라인으로 전달되게 하는 작업과 두 번째 프로세서에서 DPI만을 처리하도록 설정하는 작업이 필요하다. 패킷 처리 성능을 최대화하기 위하여 DPI시스템의 내부/외부 인터페이스를 브릿지 방식로 연결하여 외부에서 들어온 패킷을 내부



(그림 1) IRQ와 DPI 처리 담당 프로세서 구성

네트워크로 빠르게 전달토록 하였으며, DPI 엔진에 패킷을 전달하기 위해서 리눅스 넷필터(netfilter) [5] 기능을 이용하였다. 자세한 구현 방안은 아래와 같다.

2.1 DPI 엔진 시스템 환경 구성

두 개의 네트워크 인터페이스를 브릿지로 구성하였을 경우, 하나의 네트워크 디바이스를 통하여 들어온 모든 패킷은 브릿지로 구성된 반대편 네트워크 디바이스를 통해 투명하게 포워딩 된다. 브릿지 구성을 위하여 우리는 brctl 유틸리티 [6]를 사용한다. 각 네트워크 인터페이스로 전달된 패킷은 브릿지를 통하여 상대방 네트워크 인터페이스로 전달된다. 이때 모든 패킷은 넷필터를 지나게 되는데, 넷필터 FORWARD체인에 후킹 필터를 등록 함으로써 통과하는 모든 패킷들을 DPI엔진에 보내어서 검사할 수 있다. 이런 목적으로 ebtables 유틸리티[7] 또는 iptables 유틸리티[8] 등을 쓸 수 있는데, 우리는 L2 Layer (Data Link Layer) 패킷 디코딩이 가능한 ebtables을 사용했다. 후킹한 패킷을 DPI 엔진에 전달하기 위해서 ebtables에 타겟을 추가한다. 두 네트워크 디바이스에서 발생하는 인터럽트 모두를 첫 번째 프로세서로 향하도록 하기 위한 설정은 2장에서 설명한 IRQ SMP Affinity의 사용자 인터페이스를 통해 가능하다.

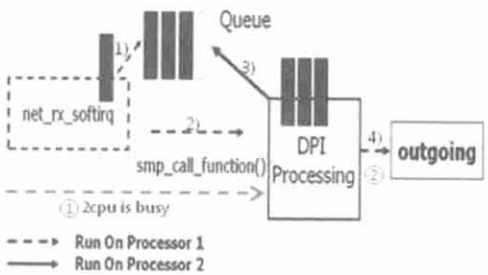
2.2 DPI 엔진을 위한 패킷 프로세싱 방안 구현

DPI 엔진은 시그니쳐 데이타베이스에 기반하여 트래픽을 분류하고 검사를 수행한 후 패킷이 시그니쳐 데이타베이스에 매칭되었을 경우에는 DROP하고, 매칭되지 않을 경우 브릿지를 통해 반대편 인터페이스로 하는 기능을 수행한다. 실제 패킷 처리는 넷필터에서 일어나는데, 넷필터는 DPI 엔진에서 리턴 된 값 (NF_DROP 또는 NF_ACCEPT)에 따라서 패킷을 실제로 DROP 또는 ACEEPT한다. DPI 엔진을 두 번째 프로세서에 바인딩 시키기 위하여 다음과 같은 과정을 ebtables DPI 타겟이 수행한다.

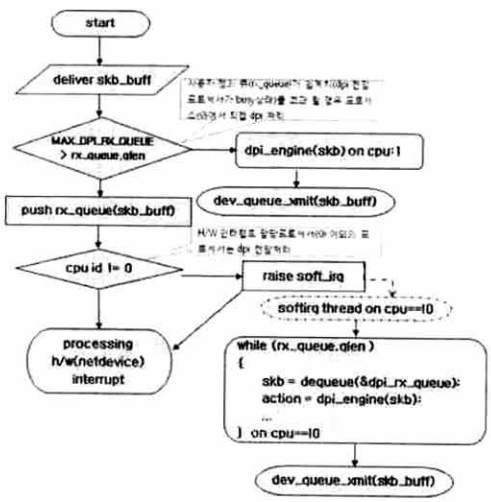
첫 번째 프로세서를 통해 패킷이 DPI 엔진에 도착하면 DPI 엔진은 패킷을 담을 수 있는 사용자 정의 INPUT 패킷 큐 상태를 측정한다. 측정 방안은 매 패킷이 입력 될 때마다 사용자 정의 INPUT 큐(dpi_rx_queue) 개수를 계산하여 DPI 전담 프로세서의 현재 상태를 측정한다.

만약 INPUT 큐가 사용자 정의 임계치(MAX_DPI_RX_QUEUE)를 초과했을 경우에는 DPI 전담 프로세서를 BUSY 상태로 판단한다. 이때 첫 번째 프로세서는 INPUT 큐의 패킷을 POP 하여 DPI 프로세싱을 직접 수행한다. 동시에 두 번째 프로세서(softirq 커널쓰레드)는 같은 INPUT 큐에서 패킷을 POP하여 DPI 프로세싱 작업을 INPUT 큐 개수가 사용자 정의 임계치 이하로 떨어질 때까지 진행한다. DPI 사용자 정의 패킷큐(dpi_rx_queue) 개수가 사용자 정의 INPUT 큐의 임계치(MAX_DPI_RX_QUEUE)를 초과하지 않았을 경우, 패킷을 DPI사용자 정의 패킷큐에 PUSH한 후, smp_call_function() 함수를 호출하여 사용자 정의 커널 쓰레드(softirq) 인 NET_DPI_RX_SORTIRQ를 발생시킨 후 종료한다. smp_call_function() 함수는 지정된 프로세서(두 번째 프로세서)에 softirq커널쓰레드를 명시적으로발생시킬 수 있다. smp_call_function() 함수를 통해 두 번째 프로세서에서 작업을 시작한 NET_DPI_RX_SOFTIRQ커널 쓰레드는 일정시간(DPI_JIFFTIME) 동안 INPUT 큐에서 패킷을 POP하여 dpi_engine(struct sk_buff *) 함수를 호출하여 DPI 패턴 매칭 모듈에 패킷을 전달한다. 패킷은 패턴 매칭 결과에 따라서 DROP되거나 dev_queue_xmit함수(ACCEPT)를 호출하여 외부로 전송된다.

(그림 2)에서 위 과정을 도식화 하였으며 (그림 3)에서 순서도를 표현하였다. DPI 전담 프로세서가 BUSY 상태로 판단된 경우 ①첫 번째 프로세서는 INPUT 큐의 패킷을 POP하여 DPI 프로세싱 모듈에 직접 전달, ②DPI 프로세싱 결과에 따라 패킷을 DROP 또는 ACCEPT 한다. 반대의 경우 인터럽트를 수신한 첫 번째 프로세서에서 동작하는 NET_RX_SOFTIRQ 커널 쓰레드가 1)네트워크 인터페이스



(그림 2) IRQ와 DPI 프로세스의 연결



(그림 3) 제안방식 구현 순서도

로부터 전달받은 패킷을DPI 사용자 정의(dpi_rx_queue)에 PUSH, 2) smp_call_function() 함수를 호출하여 두 번째 프로세서에서 동작하는 커널 쓰레드 생성, 3) 커널 쓰레드는 DPI 사용자 정의큐(dpi_rx_queue)로부터 패킷을 순차적으로 POP하여 DPI 프로세싱 모듈에 전달 4) DPI 프로세싱 결과에 따라 패킷을 DROP 또는 ACCEPT 한다.

3. 성능 평가

3.1 실험 환경 및 실험 계획

본 실험은 네트워크 성능 테스트 표준인 RFC2544 (Methodology for Network Interconnect Devices)[9]을 준수하는 SMB-6000[10] 계측기와 SmartFlow소프트웨어를 통해서 진행하였으며, Device Under Test구성은 Intel(R) Xeon(TM) CPU 2.80GHz 프로세서, 2MB L2 cache, 800MHz FSB이다. 리눅스 커널은 버전 2.6.13 커널을 사용하였으며 배포판은 Redhat사의 Fedora Core2이다.

성능평가는 두 가지 단계로 수행되었다. 첫 번째 단계로, DPI기능을 제거한 환경에서 리눅스 배포판에서 제공되는 방식인 irqbalance그리고 kirqd를 사용했을 경우와 본 논문에서 제안하는 방식의 패킷 전송량 을 측정 비교한다. 이 결과를 통해 리눅스 커널의 순수 패킷 처리 성능은 본 논문의 방식이 두 프로세서를 모두 활용한 irqbalance, kirqd 에 비해 성능저하가 없음을 증명한다.

성능 분석 대상은 전송량(throughput), 지연속도(latency), 프레임 손실(frame loss), 프로세서 인터럽트 발생 빈도, 프로세서 이용률 등이며, 커널 및 프로세서의 구체적 상태정보(캐시 접근 실패율 및 스핀락 발생빈도)를 얻기 위하여 OPROFILE [11] 도구를 활용한다.

3.2 DPI 기능 제거 시 성능 비교

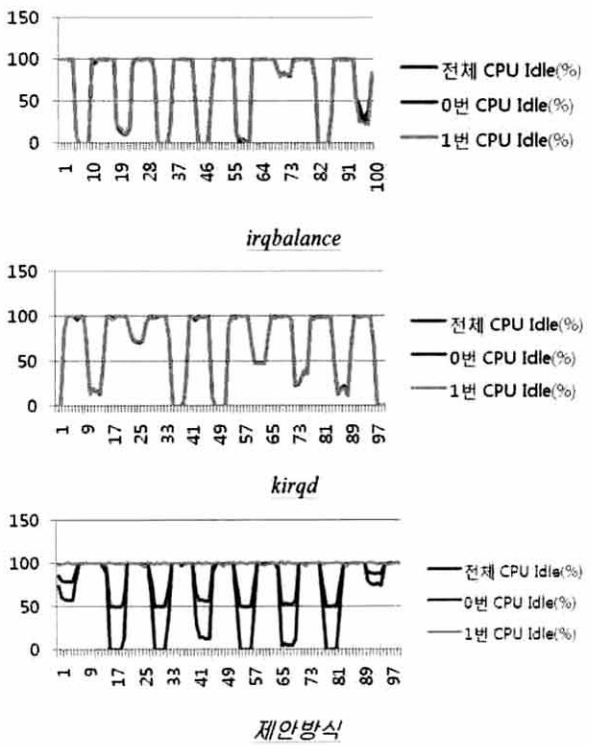
본 실험은 DPI 기능이 없을 경우에, 즉 일반적인 패킷 전달 과정에서 IRQ SMP Affinity 프로세서 제어 방식이 다른 방식들에 비해 성능저하가 없음을 증명하여 본 논문에서 제안한 IRQ SMP Affinity를 통한 DPI 엔진 추가 방안이 프로세서 활용도를 크게 높일 수 있음을 설명한다.

(그림 4, 5)는 아래 세 방식의 실험결과를 비교하였다. (그림 4)는 프로세서 이용률 (Y축), (그림 5)는 인터럽트 발생빈도이다. 우리는 irqbalance를 이용할 경우 실험 결과에 의하면, 브릿지로 구성된 두 인터페이스에 패킷량이 증가하면, 하나의 프로세서가 하나의 네트워크 인터페이스의 모든 IRQ를 처리하게 되는 1 대 1 매핑이 일어난다. 즉 첫 번째 프로세서는 첫 번째 네트워크 인터페이스의 트래픽을 처리하고, 두 번째 네트워크 인터페이스의 트래픽은 두 번째 프로세서에 각각 처리된다. (그림 4, 5)의 irqbalance실험 결과에 의하면, 두 개의 프로세서가 거의 같은 이용률 패턴을 가지면서 로드 밸런싱이 잘 이루어짐을 확인할 수 있다. 두 개의 프로세서로각각 전달되는 인터럽트의 발생 빈도 또한 비슷한 수치로 전달되는 것을 볼 수 있다.

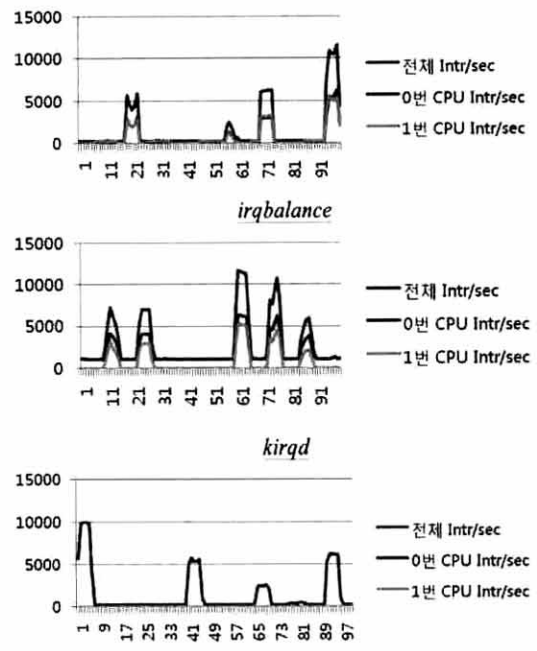
다음으로 kirqd를 활용할 경우의 실험결과를 설명하겠다. 리눅스 커널에 kirqd를 활성화 했을 경우, 커널에서 IRQ 라인을 각 프로세서로 이동시키며 프로세서간 인터럽트 발생빈도의 균형을 맞춘다. 즉 시스템의 성능향상을 목적으로 특정 프로세서로 워크로드(workload)가 집중되는 것을 막아 프로세서들의 IRQ 발생빈도의 균형을 맞추어 준다. irqbalance 방식과 같이 두 개의 프로세서가 거의 같은 이용률 패턴을 가지면서 로드 밸런싱이 잘 이루어짐을 확인할 수 있다. 그러나 두 개의 프로세서로 각각 전달되는 인터럽트의 발생 빈도는 irqbalance 방식에 비해 다소 차이가 생긴다. 이 차이가 성능 저하의 원인이 된다.

제안하는 방식에서는 두 개의 네트워크 인터페이스 카드의 인터럽트 처리를 하나의 프로세서에 바인딩 시킨다. 따라서, DPI기능을 제거한 경우 두 번째 프로세서는 항상 idle 상태에 있게 된다. (그림 4, 5)의 제안방식 실험 결과처럼 첫 번째 프로세서 ('0'번 CPU)만 동작하고 있으며 인터럽트 또한 첫 번째 프로세서에서만 발생하고 있다.

세가지 방식의 최종 전송량 비교 결과는 <표 1>에 정리되어 있다. 패킷 크기가 클 경우에는(512Byte이상) 인터럽트 처리 및 디코딩 비용이 낮기 때문에(Packet per second감소) 모든 방식이 최대 전송량(700Mbps)을 지원한다. 전반적으로 제안하는 방식과 irqbalance 방식은 비슷한 성능을 보이고 kirqd 방식이 상대적으로 낮은 성능을 보이는 것을 확인할 수 있다. 즉, 두 개의 네트워크 인터페이스를 브릿지로 묶어 패킷을 인바운드 인터페이스로부터 아웃바운드 인터페이스로 전송할 경우, 하나의 프로세서가 두 인터페이스에서 발생하는 인터럽트를 함께 처리하는 제안된 방식이 두 개의 프로세서에서 활용하여 각 프로세서가 각 인터페이스의 인터럽트를 처리하는 irqbalance방식에 비해 성능 저하가 없었다. 커널에서 IRQ 발생 빈도를 분석하여 실시간으로 인터럽



(그림 4) irqbalance, kirqd, 제안 방식의 CPU 이용률



(그림 5) irqbalance, kirqd, 제안 방식의 인터럽트 발생빈도

〈표 1〉 전송량 (Byte/Mbps) 비교

패킷 크기	64	128	256	512	1024
irqbalance 방식	158	267	454	700	700
kirqd 방식	139	228	444	700	700
제안 방식	158	287	513	700	700

트를 균형있게 두 프로세서에서 분배하는 kirqd방식은 다른 방식에 비해 나쁜 성능을 보이는데 그 이유는 <표 2>에 제시된 커널 프로파일링 결과 분석을 바탕으로 설명 가능하다. <표 2>의 symbolname컬럼은 실제적으로 호출 되어진 커널 함수를 의미하며, samples 컬럼은 symbolname 호출 횟수를 sampling 하여 수집한 횟수이다. (%)컬럼은 전체 샘플링되어 호출된 symbolname횟수 중 해당 symbolname의 sampling 횟수를 백분율로 구한 값이다. 인터럽트 처리를 단일 프로세서만을 사용하는 제안된 방안에 비해, kirqd방식은 하나의 프로세서에 두 개의 네트워크 인터페이스에 발생되는 인터럽트가 동시에 걸릴 수 있고 이는 스핀락 발생빈도가 상대적으로 높기 때문이다.

〈표 2〉 커널 프로파일링 결과(working time)

제안된 방식			kirqd 방식		
samples	%	symbol name	samples	%	symbol name
189752	12.82	eth_type_tran	320433	10.6	eth_type_trans
158711	10.72	dev_queue_xmit	315009	10.45	_spin_lock
153902	10.39	_spin_lock	259928	8.62	dev_queue_xmit

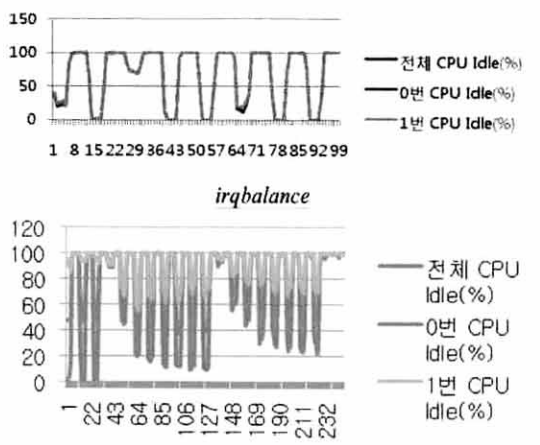
3.3 DPI 기능 활성화 시 성능 비교

위 결과를 통해 우리는 하나의 프로세서가 일반적인 패킷 처리를 담당토록 하여도, irqbalance, kirqd 등의 로드밸런싱 방식과 성능차이가 없음을 확인하였다. 여기서 우리는 다른 하나의 프로세서를 DPI 처리를 전담토록 하면 기존의 패킷 처리 성능을 유지하면서도, 다른 하나의 프로세서 활용도를 높일 수 있다고 판단하여 아래와 같이 DPI 엔진을 실제로 적용하여 테스트를 실시하였다.

kirqd방식은 DPI 기능을 제거했을 때 가장 나쁜 성능을 보였으므로 실험대상에서 제외하고 irqbalance방식과 제안된 방식의 성능 비교에 초점을 둔다. 먼저 irqbalance방식의 성능 분석 결과를 보자. DPI 기능을 활성화 한 채 irqbalance 방식을 쓸 경우 역시 DPI 기능을 제거한 경우와 마찬가지로, 패킷량이 증가하면 브릿지를 구성하는 각 네트워크 인터페이스의 IRQ와 프로세서들간에 1대1 매핑이 일어난다. irqbalance 방식에서 DPI 엔진을 활성화했을 경우의 시스템 전송량은 <표 3>에 보여진다. 패킷 크기가 64byte인 경우 전송량은 DPI 기능을 제거한 경우 보다 훨씬 낮은 약 99Mbps 측정되었다. (그림 6, 7)의 왼쪽 그림은 프로세서 이용률과 인터럽트 발생빈도를 보이고 있다. DPI 기능을 제거한 경우와 비슷한 경향을 보인다.

본 논문에서 제안하는 방식은 DPI 기능이 활성화 되었을 경우 irqbalance방식 보다 월등히 높은 전송량을 지원했다. 제안된 방식의 throughput 측정 결과는 <표 3>의 제안방식 컬럼에 보인다. 예를 들어 패킷 크기가 64byte인 경우 제안된 방식은 약 168Mbps의 전송량이 측정되었다. 이는 irqbalance 방식 대비 약 60% 이상의 성능향상을 의미한다. (그림 6, 7)의 제안방식의 CPU 이용률 실험결과에 의하면 두 개의 프로세서가 모두 활용되고 있지만 그들의 이용률 패턴이 irqbalance방식의 경우와 판이 하게 다르고 두 프로세서 간에도 큰 차이를 보인다. 또한 인터럽트 역시 첫 번째 프로세서에서만 나타나는 것을 볼 수 있다. (그림 6, 7) 의 인터럽트 발생빈도을 통해 그 차이를 비교할 수 있다.

본 논문에서 제안한 방식의 높은 성능은 두 개의 프로세서에서 동작하는 작업을 분리하여 얻어지는 상대적으로 낮은 스핀락 발생빈도(표 4 참조)와 캐시접근실패율(표 5 참조)에서 그 원인을 찾을 수 있다.



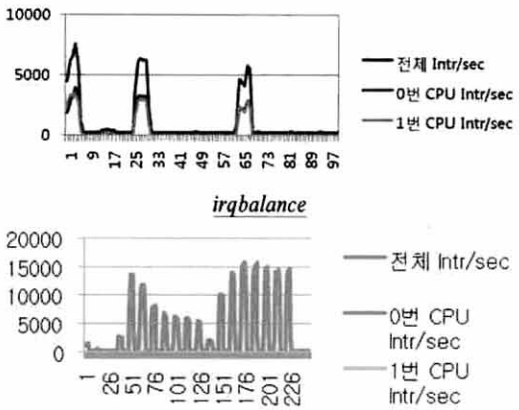
(그림 6) irqbalance, 제안방식의 CPU 이용률

〈표 3〉 전송량 (Byte/Mbps) 비교

Name/Framesize	64	128	256	512	1024
irqbalance 방식	99	178	345	631	700
제안 방식	168	286	532	700	700

〈표 4〉 커널 프로파일링 결과(working time)

제안된 방식			irqbalance 방식		
Samples	%	symbol name	samples	%	symbol name
1140667	15.32	eth_type_type	852937	9.92	eth_type_t.ype
388815	5.22	Kfree	851833	9.90	_spin_lock
382021	5.13	_spin_lock	380408	4.45	Kfree



(그림 7) irqbalance, 제안방식의 인터럽트 발생빈도

〈표 5〉 커널프로파일링 결과(cache miss)

제안된 방식			irqbalance 방식		
samples	%	symbol name	samples	%	symbol name
165449	6.01	dev_queue_xmit	114577	6.07	_spin_locak
147885	5.37	netif_receive_skb	85336	4.52	_reqd_lock_bh
136885	4.97	_read_unlock_bh	82269	4.36	eth_type_trans
123843	4.50	eth_type_tran	65570	3.48	_spin_tryloack

4. 결 론

본 논문은 특화된 하드웨어의 도움 없이 범용 다중 프로세서 플랫폼에서 효율적인 DPI(Deep Packet Inspection) 엔진을 구현하는 이슈를 다루고 있다. 일반적으로 다중 프로세서를 이용한 병렬처리는동일한 작업을 여러 프로세서에 어떻게 적절하게 배분하여 로드 밸런싱을 이룰 것인지에 초점을 맞추는 반면, 본 논문에서는 DPI 시스템의 경우 작업을 기능별로 구분하여 종류에 따라 정해진 프로세서에 할당하는 방식이 비록 프로세서들의 부하가 다소 불균형을 이루더라도 시스템 전체의 성능을 향상시킬 수 있음을 보여주고 있다. 제안된 방식에서는 하나의 프로세서를 하드웨어 인터럽트를 처리하는 역할만을 담당토록 하고 다른 프로세서는 DPI엔진을 전담하는 역할로 분리시켜 성능향상을 이루었다. 제안된 방식을 리눅스 기반 듀얼 프로세서 시스템에 구현한 후 기존의 방법들과 성능 비교 결과, 기존의 리눅스 다중 로드 밸런싱 방식에 비해 약 60%의 성능향상을 달성하였다. 제안한 방식은 두 개 이상의 프로세서(쿼드 프로세서)를 활용 했을 경우에도 단일 프로세서에서 패킷의 일반적인 처리만을 담당하고 나머지 세 개의 프로세서에서 DPI를 전담토록 하여 패킷 처리 요구량이 더 높은 시스템을 쉽게 만들 수 있다(각 프로세서가 고유의 역할만을 수행하므로 캐시 접근 실패율 및 스핀락 발생빈도 감소).

참 고 문 헌

[1] Ido Dubrawsky, http://www.securityfocus.com/infocus/1716, SecurityFocus, July, 2003.
[2] G.Grill, "IRQ DISTRIBUTION IN MULTIPROCESSOR SYSTEMS", Linux Kernel Hacking Free Course 3rd edition, April, 2006.
[3] http://irqbalance.org/documentation.php
[4] Barry Arndt, "Linux Networking Scalability on High-Performance Scalable Servers", IBM Systems and Technology Group, April, 2007.
[5] Robert Love, "Kernel Locking Techniques", Aug, 2002
[6] Rusty Russell, Harald Welte, "Linux netfilter Hacking HOWTO", July, 2002.
[7] Lennert Buytenhek, http://linux.die.net/man/8/brctl
[8] Nick Fedchik, Grzegorz Borowiak, http://ebtables.sourceforge.net/
[9] netfilter core team, http://www.netfilter.org/projects/iptables/index.html
[10] S. Bradner, http://www.faqs.org/rfcs/rfc2544.html, Network Working Group, March, 1999.
[11] http://nsslabs.com/grouptests/ips/edition2/appendix_b/appendix_b.htm
[12] John Levon, http://oprofile.sourceforge.net/doc/internals/index.html, 2003.

양 준 호
e-mail : jonooyang@yonsei.ac.kr
2003년 한국외국어대학교 산업공학과(학사)
2008년 연세대학교 컴퓨터공학과(공학석사)
2003년~2008년 안철수연구소㈜ Network Unit 선임연구원
2008년~현 재 한국지역정보개발원 시도 사이버침해대응지원센터 선임연구원
관심분야 : 네트워크 보안, 정보보안, CERT(Computer Emergency Response Team) 등

한 승 재
e-mail : sjhan@cs.yonsei.ac.kr
1991년 서울대학교 컴퓨터공학과(석사)
1998년 Univ. of Michigan, Ann Arbor, CSE (Ph.D.)
1999년~2005년 BellLaboratories, Wireless Research Lab 연구원
2005년~현 재 연세대학교 컴퓨터과학과 부교수
관심분야 : 차세대이동통신망, 내장형 시스템, 멀티미디어네트워킹, 센서네트워크 등