네트워크 인터페이스 카드에 기반한 호스트 독립적인 네트워크 시스템의 설계 및 성능평가

(Design and Evaluation of a NIC-Driven Host-Independent Network System)

임 근 수 † 차 호 정 †† 고 건 †††
(Keun Soo Yim) (Hojung Cha) (Kern Koh)

요 약 전형적인 클라이언트-서버 컴퓨팅 모델에서 네트워크 서버 시스템들은 과중한 양의 계산과 통신 작업을 수행해야 한다. 하지만 현재 네트워크 서버 시스템에서 사용되는 통신규약 스택의 구조는 크게 세가지 성능상의 병목을 가지고 있다. 호스트 시스템의 통신규약 스택의 처리, 시스템 호출 처리, 그리고 네트워크 인터럽트 처리에 따른 성능상의 병목을 개선하기 위해서 본 논문에서는 네트워크 인터페이스 카드에 기반한 호스트 독립적인 네트워크 시스템을 설계하고 성능을 평가한다. 첫째, 통신규약 스택 처리를 호스트에서 네트워크 인터페이스 카드로 분산시킴으로써 호스트 시스템의 통신규약 스택 처리에 따른 계산량을 줄인다. 둘째, 이렇게 분산된 통신규약 스택과 사용자 수준의 라이브러리를 사용해 통신함으로써 통신규약 스택에 접근하기 위한 시스템 호출 비용을 제거한다. 셋째, 네트워크 인터페이스 카드에서 패킷이 아닌 세그먼트 단위로 인터럽트를 생성함으로써 호스트의 네트워크 인터럽트 처리 비용을 줄인다. 실험 결과 제안하는 네트워크 시스템을 사용할 경우 호스트의 통신규약 스택을 위한 계산량을 68-71% 감소시킴을 보인다. 이러한 특성으로 인해서 제안하는 시스템을 활용하면 호스트에 계산 및 통신 요구가 높은 경우에 통신 응답시간을 11-83% 가량 단축시킬 수 있음을 보인다.

키워드 : 고속 네트워킹, 통신규약 스택 구조, 네트워크 인터페이스 카드, 호스트 독립 구조

Abstract In a client-server model, network server systems suffer from both heavy communication and computational loads. While communication channels become increasingly speedy, the existing protocol stack architectures still include mainly three performance bottlenecks of protocol stack processing, system call, and network interrupt overheads. To address these obstacles, in this paper we present a host-independent network system where a network interface card (NIC) is utilized in an efficient manner. First, by offloading network-related portion to the NIC, the host can fully utilize its processing power for other useful purposes. Second, it eliminates the system call overhead, such as context-switching and memory copy operations, since the host communicates with the NIC through its user-level libraries. Third, it also reduces the network interrupt operation count as the host handles the interrupt in a segment instead of a packet. The experimental results show that the proposed network system reduces the host CPU overhead for communication system by 68-71%. It also shows that the proposed system improves the communication speed by 11-83% under heavy computational and communication load conditions.

Key words : High-speed networking, protocol stack architecture, network interface card, host independent

† 학생회원 : 서울대학교 컴퓨터공학부
ksyim@oslab.snu.ac.kr
†† 종신회원 : 연세대학교 컴퓨터산업공학부 교수
hjcha@yonsei.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
kernkoh@june.snu.ac.kr
논문접수 : 2003년 9월 1일
심사완료 : 2004년 7월 29일

1. 서 론

전형적인 클라이언트-서버 컴퓨팅 모델에서 네트워크 서버 시스템들은 클라이언트들의 요청을 빠르기 처리해야 하는데 반하여 과중한 양의 계산과 통신 작업에 대한 수요로 인해 이러한 요구를 효율적으로 만족시키지 못하고 있다. 일반적으로 이러한 서버 시스템의 성능을

개선하기 위해서는 병렬 또는 분산 처리 기술을 활용하고 있다. 하지만 이러한 방식은 서버의 계산 능력을 증강하는 것으로 근본적으로 통신 채널의 속도의 향상에 따라 네트워크 시스템의 새로운 병목으로 지목되고 있는 서버 시스템의 통신규약 스택의 처리시간 문제를 그대로 지니게 된다. 따라서 서버 시스템의 성능을 개선하기 위해서는 통신규약 스택의 병목을 효율적으로 개선해야할 필요가 있다.

전형적인 UNIX 계열의 운영체제에서 통신규약 스택은 커널 내부에 존재하기(*Kernel-Reside*) 때문에 그림 1에 제시된 것은 세가지 통신 성능상의 병목을 지닌다 [1]. 세부적으로 응용 프로그램은 시스템 호출 인터페이스를 사용하여 통신규약 스택에 접근하는데, 한번의 시스템 호출은 사용자 영역과 커널 영역사이의 두 번의 문맥교환과 한번의 메모리 복사 작업을 통해 이뤄진다 [2]. 또한 커널 내부 스택은 네트워크 패킷을 인터럽트 방식으로 처리하기 때문에 패킷 도착률이 높은 경우에 시스템의 성능이 크게 저하될 수 있다[3]. 뿐만 아니라, 호스트 CPU를 사용하여 TCP/IP와 같은 중량의 프토토콜 스택을 처리해야 하는 단점이 있다[4].

이러한 통신 성능상의 병목을 개선하기 위하여 본 논문에서는 네트워크 인터페이스 카드(Network Interface Card, NIC)에 기반한 호스트 독립적인 네트워크 시스템(*NIC-NET*)을 설계하고 성능을 평가한다. 제안하는 NIC-NET의 핵심적인 특성은 다음과 같다. 첫째, 통신규약 스택 처리를 호스트에서 NIC로 분산시킴으로써 호스트 시스템의 통신규약 스택 처리에 따른 계산비용을 줄인다. 둘째, 이렇게 분산된 통신규약 스택과 사용자 수준의 라이브러리를 사용해 통신함으로써 통신규약 스택에 접근하기 위한 시스템 호출 비용을 제거한다. 셋째, 네트워크 인터페이스 카드에서 패킷이 아닌 세그먼트 단위로 인터럽트를 생성함으로서 호스트의 네트워크 인터럽트 처리 비용을 줄인다.

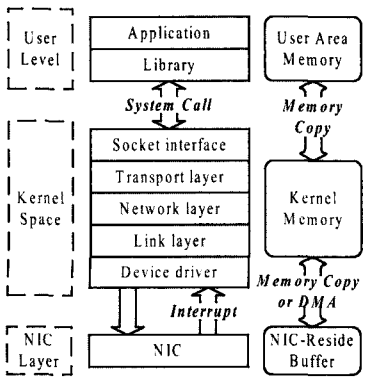


그림 1 커널 내부에 위치한 통신규약 스택의 구조

실험 결과 제안하는 NIC-NET을 사용할 경우 호스트의 통신규약 스택 처리에 사용되는 계산량이 68-71% 감소됨을 보인다. 이러한 특성으로 인해서 NIC-NET을 활용하면 호스트에 계산 및 통신 요구가 높은 경우에 통신 응답시간을 11-83% 가량 단축시킴을 보인다. 뿐만 아니라 실험을 통해 이와 같은 성능 향상은 NIC에 클럭이 50-200MHz인 범용 CPU를 사용함으로써 얻을 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 조사해 기록하고, 3장에서는 제안하는 NIC-NET의 전체 구조와 인터페이스에 대해 기술한다. 4장에서는 실험 방법을 설명하고, 5장에서는 실험 결과를 제시하고 이를 바탕으로 NIC-NET의 성능을 평가한다. 그리고 6장에서는 결론을 맺고 향후 연구과제를 제시한다.

2. 관련 연구

그동안 서버 시스템의 통신규약 스택의 성능을 개선하기 위하여 통신규약 스택과 관련 하드웨어를 최적화하는 방식의 다양한 기술들이 제안되었다. 이러한 기술에는 사용자 수준 통신규약 구조(User-Level)와[2-8] 가상 인터페이스 구조(Virtual Interface Architecture, VIA)가[9-13] 있다.

사용자 수준 스택은 그림 2에 제시된 것과 커널의 일부가 아닌 사용자 수준의 라이브러리로 구현된다. 이러한 특성으로 인해 사용자 수준 스택은 사용자 영역과 커널 영역사이의 시스템 호출 비용을 제거한다. 하지만 비록 일부 사용자 수준 스택의 경우 TCP/IP[14] 대신에 경량의 통신규약을 사용할 수 있으나 이 경우에도 통신규약 스택을 통해 패킷을 하나 처리하는데 수백 마이크로 초에서 수 밀리 초가 걸리게 된다[4]. 뿐만 아니라 패킷의 도착률이 높은 경우에 호스트 시스템은 많은 양의 네트워크 인터럽트를 처리해야 하기 때문에 성능이 크게 저하될 수 있다. 이러한 두가지 성능상의 문제점은 본 논문에서 제안하는 NIC-NET에서 통신규약 스택을 NIC로 분산시킴으로써 개선된다.

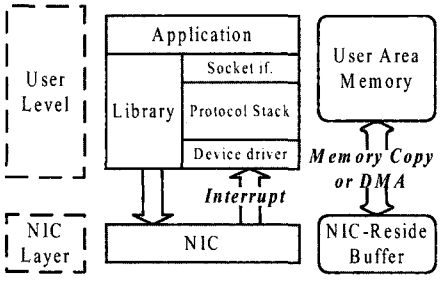


그림 2 사용자 수준 통신규약 스택의 구조

VIA 기술을 활용하면 사용자 수준 스택의 두가지 문제점을 NIC-NET과 유사한 방식으로 개선할 수 있다. 그러나 VIA 기술은 클러스터링 컴퓨터와 같이 작은 크기의 패킷을 빠르게 교환해야 하는 시스템을 위해서 개발되었기 때문에 특수한 경량 통신규약을 사용하며 Myrinet과[15] 같은 고속 통신 채널을 사용한다. 이러한 특성으로 인해 VIA 기술은 TCP/IP 통신 규약을 사용하는 네트워크 서버 시스템들에는 적합하지 않다.

최근에 무굴(Mogul)은 TCP를 NIC로 분산시키는 기술은 높은 성능 향상의 잠재력을 갖고 있으나, 기존에 존재하는 기술들이 몇가지 문제점을 갖고 있다고 주장한 바 있다[16]. 하지만 무굴이 지적한 문제점들은 3장에 제시된 것과 같이 본 논문에서 제안하는 NIC-NET에서 해결되거나 개선된다.

3. 네트워크 인터페이스 카드에 기반한 호스트 독립적인 네트워크 시스템

이 장에서는 제안하는 호스트 독립적인 네트워크 시스템(NIC-NET)의 전체 구조와 세부 작동 원리에 대해서 설명한다. 그리고 NIC-NET의 호스트 시스템과 NIC사이의 새로운 통신 인터페이스에 대해서 살펴본다.

3.1 전체 구조와 세부 작동 원리

제안하는 NIC-NET의 핵심 아이디어는 통신규약 스택을 NIC로 분산해 독립적으로 처리함으로서 호스트 시스템의 CPU를 사용하지 않고도 신뢰성있는 네트워킹 서비스를 응용 프로그램에 제공하는 것이다. 그림 3은 NIC-NET의 전체 구조를 나타낸 것으로 이러한 구조는 다음과 같은 네가지 특성을 갖는다.

첫째, NIC-NET에서는 통신규약 스택이 커널에서 NIC로 분산되기 때문에 호스트의 계산 능력을 다른 유용한 용도로 충분히 활용할 수 있다. 일반적으로 물리층에서 소켓 인터페이스까지의 통신규약 스택의 하나의 패킷에 대한 처리 시간은 CPU의 계산 능력에 따라 약 1-2 밀리 초가 걸린다[4]. 하지만 처리해야되는 패킷의 수가 증가함에 따라 이 짧은 처리 시간도 커널 내부 스택과 사용자 수준 스택에게는 높은 계산 비용이 될 수 있다. 제안하는 NIC-NET에서는 이러한 계산 비용을 NIC로 분산하기 때문에 NIC-NET의 성능이 CPU 계산 작업이 많은 경우에 최대화됨을 쉽게 예측할 수 있다.

둘째, NIC-NET은 NIC에 분산된 통신규약 스택과 사용자 수준 라이브러리를 사용해 통신하기 때문에 기존의 커널 내부 스택에 존재하던 문맥 교환과 메모리 복사 작업을 필요로 하는 시스템 호출 비용을 제거한다. 이때 NIC에 분산된 통신규약 스택과의 통신비용을 줄이기 위하여 3.2절에 제시된 것과 같은 세가지 새로운 통신 방법을 사용한다.

셋째, NIC-NET을 사용하면 호스트 시스템은 네트워크 인터럽트를 패킷이 아닌 세그먼트 단위로 처리하게 되어 네트워크 인터럽트 처리 비용을 크게 줄이게 된다. 세그먼트의 개수와 패킷의 개수를 비교하기 위해서 그림 4에 제시된 것과 같은 일반적인 TCP 통신 시나리오를 고려해 보자. 이 예제에서 호스트의 최대 세그먼트 크기(MSS)는 라우터의 그것보다 2배 크다고 가정하였기 때문에 모든 데이타 패킷은 라우터에서 2개의 패킷으로 분할된다. 그 결과 단지 7개의 세그먼트를 전송하는 과정에서 호스트 A와 B는 총 26개의 패킷을 수신하게 됨을 알 수 있다. 여기서 수신한 패킷의 개수는 커널 내부 스택과 사용자 수준 스택에서 네트워크 인터럽트의 발생 횟수를 의미하고, 전송한 세그먼트의 개수는 제안하는 NIC-NET에서의 인터럽트 발생 횟수를 의미한다. 따라서 이 예제에서는 NIC-NET을 사용하면 호스트 시스템의 네트워크 인터럽트 처리 횟수를 70% 이상 줄일 수 있다. 이러한 특성으로 인하여 NIC-NET의 성능은 유사하게 패킷 수신율이 높은 경우에 상대적으로 증가됨을 예측할 수 있다.

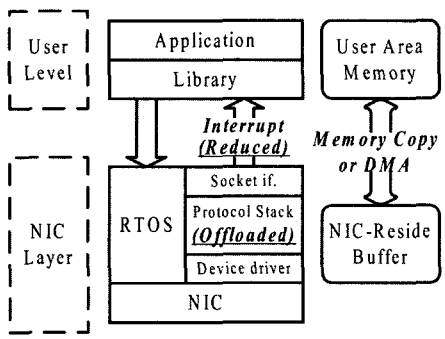


그림 3 NIC-NET의 전체 구조

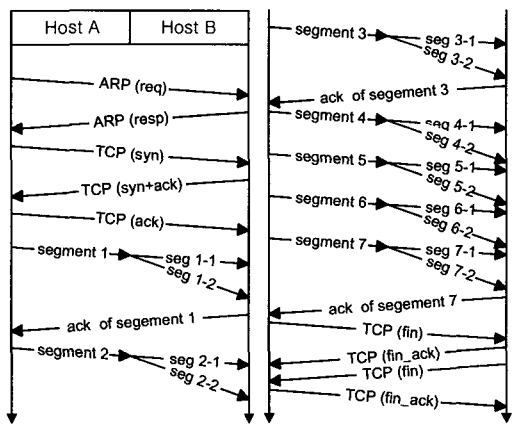


그림 4 일반적인 TCP 통신 시나리오

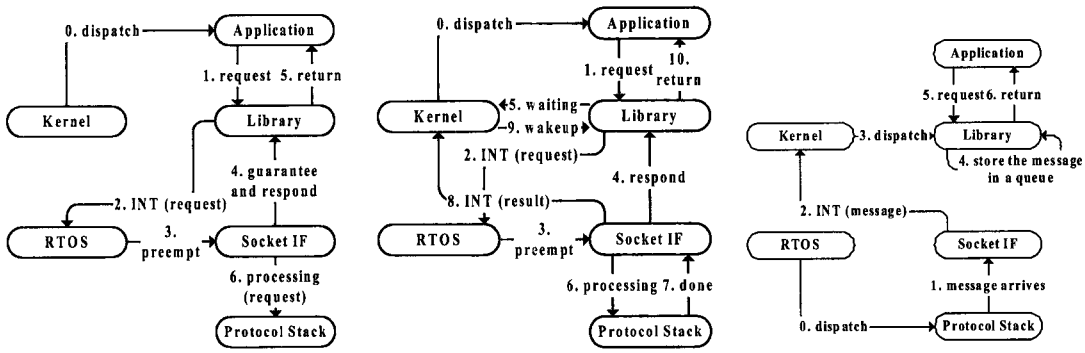


그림 5 호스트와 NIC사이의 통신 방법

넷째, NIC-NET을 사용하면 통신규약 스택이 커널 내부에 존재하지 않기 때문에 커널의 크기를 줄일 수 있다. 또한 NIC-NET은 현재 시스템의 라이브러리와 디바이스 드라이버만을 수정해 구현할 수 있다는 높은 이식성을 갖는다. 이때 NIC에 TCP/IP 또는 IPX 등과 같은 다양한 통신규약을 네트워크 디바이스 드라이버의 기능을 사용해 설치할 수 있다. 예를 들면 관리자가 통신규약 스택 업그레이드를 요청하면 네트워크 디바이스 드라이버는 새로운 프로토콜 이미지를 NIC로 전송함으로써 NIC가 자체적으로 통신규약 스택을 업그레이드하게 할 수 있다.

추가적으로 NIC-NET에 사용되는 NIC를 네트워크 전용 CPU가 아닌 범용 CPU를 사용해 개발하면 Moore의 법칙에 따른 이득을 취함과 동시에 개발시간을 단축시키는 장점을 갖게 된다. 최근들어 CPU와 메모리 소자의 성능은 크게 개선되면서 생산단가가 낮아짐에 따라서 이제 NIC-NET과 같이 커널 내부의 일부분을 이와 직접적으로 대응하는 주변기기로 분산시켜 처리하는 것은 보편적인 기술이 되고 있는 실정이다. 이러한 예로는 지능형 디스크 컨트롤러와 3차원 비디오 카드 등이 있다.

3.2 호스트와 NIC 사이의 인터페이스

NIC-NET을 사용하면 호스트 시스템은 NIC에 분산된 통신규약 스택과 다양한 소켓 작업을 처리하기 위하여 통신해야 한다. 각각의 소켓 작업은 서로 다른 목적과 작동 방식을 가지고 있기 때문에 성능 향상을 위해서는 통신 방법을 각각의 소켓 작업에 맞게 최적화할 필요가 있다. 따라서 본 논문에서는 그림 5에 제시된 것과 같은 세가지 새로운 통신 방법을 설계하고 이를 소켓 작업의 유형에 따라 알맞게 사용한다.

첫째, 그림 5(a)에 제시된 것과 같은 비동기 모드는 *sendto(), write(), select(), bind(), listen(),* 그리고 *create()*와 같은 비동기 작업을 위해 고안되었다. 비동기 모드에서는 호스트에서 요청한 작업의 올바른 수행을 NIC의 소켓 인터페이스에서 보증하고 가능한 빠르게 보증한 수행의 결과를 보냄으로써 호스트의 지연시간을 최소화한다. 세부적인 통신 과정은 다음과 같다. (단계 1-2) 호스트 시스템이 인터럽트를 사용해 NIC에게 작업 수행을 요청한다. (단계 3 4) 인터럽트를 받은 NIC는 호스트의 요청에 대한 보증된 수행 결과를 보냄으로써 빠르게 응답한다. (단계 5 6) 응답을 받은 호스트는 다음 작업을 수행하게 되고, 이때 NIC는 실제로 요청된 작업을 수행하게 된다.

둘째, 그림 5(b)에 묘사된 동기 모드는 *recvfrom(), read(), connect(), accept(),* 그리고 *close()*와 같은 동기 작업을 위해 설계되었다. 동기 모드의 세부 통신 과정은 다음과 같다. (단계 1-2) 호스트는 인터럽트를 사용해 NIC에게 작업을 요청한다. (단계 3-4) 인터럽트 신호를 받은 NIC는 호스트에게 요청한 작업을 수신하였음을 알린다. (단계 5) NIC의 응답을 받은 호스트는 현재 수행중인 프로세스를 블록시키고 대기중인 다른 프로세스를 스케줄한다. (단계 6) 호스트가 단계 5를 수행중인 동안에 NIC는 요청된 작업의 수행을 시작한다. (단계 7-8) 작업 수행이 완료되면 NIC는 인터럽트를 사용해 호스트에게 결과를 전달하고 이를 수신한 호스트는 블록된 프로세스를 다시 스케줄 큐에 넣는 방식으로 요청된 작업에 대한 처리를 종료하게 된다.

비록 이상의 두가지 모드만을 사용해서도 전체 소켓 인터페이스를 지원할 수 있지만, 본 논문에서는 고성능 서버 시스템을 위하여 한가지 모드를 추가적으로 설계한다. 서버 시스템의 경우 대용량의 네트워크 트레픽을 짧은 시간안에 처리해야 하기 때문에 NIC의 메모리가 부족해 넘칠 수 있다. 이러한 메모리 부족 문제를 완화하기 위해 NIC에 많은 양의 DRAM을 장착할 수 있으나 이는 NIC의 하드웨어 비용을 증가시키는 문제가 있다.

따라서 메모리 부족 문제를 비용 효율적인 형태로 개선하기 위하여 본 논문에서는 그림 5(c) 제시된 것과 같은 긴급 모드를 사용한다. 긴급 모드는 세부적으로 다음과 같이 동작한다. (단계 1-2) NIC의 사용중인 메모리 비율이 임계치 이상일 경우에 NIC는 호스트에 대한 인터럽트 신호를 생성하고, (단계 3-4) NIC 메모리상에 저장된 세그먼트들을 호스트의 주 메모리상에 위치한 큐에 저장될 수 있도록 전송한다. (단계 5-6) 이후 응용 프로그램이 큐에 저장된 세그먼트를 요청하면 호스트는 NIC와 통신할 필요없이 바로 큐에서 해당 세그먼트에 접근할 수 있다. 뿐만 아니라 긴급 모드는 긴급 메시지들을 NIC에서 버퍼링하지 않고 바로 호스트로 보내는데도 사용될 수 있다.

참고로 위의 세가지 모드는 커널 내부 및 사용자 수준 스택과의 형평성을 위하여 DMA 방식의 입출력을 사용하지 않는 일반적인 NIC를 기반으로 설계되었다.

4. 실험 방법

본 논문에서는 NIC-NET의 성능을 다양한 설계 옵션과 실행 환경에서 평가하기 위해서 모델링과 시뮬레이션 기법을 사용한다. 널리 사용하는 ns2와 같은 네트워크 시뮬레이터는 통신규약 스택의 구조를 커널 내부 스택으로 가정하고 있기 때문에 NIC-NET이나 사용자 수준 스택의 성능을 평가하는데 적절하지 못하다. 따라서 이 장에 기술된 것과 같은 커널 내부 스택, 사용자 수준 스택, 그리고 NIC-NET의 성능을 정확하게 평가할 수 있는 네트워크 시뮬레이터를 개발하여 사용하였다.

4.1 모델링

시뮬레이션 모델은 커널, 통신규약 스택, 벤치마크 프로그램, 입출력 버스, 그리고 통신 망 모델로 구성되어 있으며, 각각의 모델은 시간상의 특성을 중심으로 모델링 되었다. 전체적으로는 리눅스와[17] 일반 PC를 기본으로 설계되었으며, 설계된 모델에 벤치마크 데이타를 적용해 실제로 시뮬레이션 과정에서 사용되었다.

커널 모델은 200 밀리 초 단위로 라운드 로빈 스케줄링을 수행하는 비선점형 커널을 가정하였다. 이 커널 모델을 바탕으로 벤치마크 프로그램과 표준 이더넷 카드를 위한 네트워크 디바이스 드라이버 그리고 인터럽트 처리기를 모델링하였다. 그리고 추가적으로 메모리와 입출력 버스의 클럭 주기와 대역폭을 정의하였다.

통신규약 스택 모델은 리눅스의 TCP/IP 통신규약 모듈을 바탕으로 다양한 실행 경로를 반영해 설계되었다. 통신규약 스택 모델은 하향식 방식으로 설계되었다. 시스템 호출의 경우 문맥 교환과 메모리 복사 비용을 고려하였다. 전송층 통신규약의 경우 TCP 통신규약을 중점적으로 모델링하였다. TCP 통신규약 모델은 해더 처리, 연결 및 해제, 연결 상태 관리, 흐름 제어, 재전송 타이머, 그리고 지연된 ACK 응답 기능을 포함한다. TCP 모델에서 혼잡 제어가 빠진 이유는 하부 통신 망을 이더넷에 기반한 LAN으로 가정하여 시뮬레이터션시에 네트워크 혼잡이 발생하지 않기 때문이다. 네트워크층에서는 IP와 ICMP 통신규약을 멀티캐스팅 기능을 제외하고 모델링하였다. 그리고 연결층에서는 통신 초기의 성능을 정확하게 평가를 위하여 ARP 통신규약을 모델링하였다.

다음으로 통신 망 모델의 경우 이더넷에 기반한 LAN을 가정하였다. 통신 망 모델의 세부 구조는 수식 (1)-(3)과 같다. 여기서 $d_{trans}(l)$는 전송 지연시간, d_{prop}는 전파 지연시간, d_{queue}는 큐잉 지연시간, d_{proc}는 처리 지연시간, r_{avg}는 평균 전송속도, s_{avg}는 평균 전파속도, d_{tot}는 통신 선로의 총 길이, N은 중간 라우터의 개수, 그리고 DB는 지연시간의 변화 범위를 의미한다. 시뮬레이션시에 큐잉 지연시간과 처리 지연시간은 0으로 고려되었으며 DB는 난수를 발생시켜 동적으로 결정하였다.

$$D_{total}(l) = \left(D_{trans}(l) + D_{prop} + D_{queue} + D_{proc} \right) DB \quad (1)$$

$$D_{trans}(l) = \sum_{k=1}^{N+1} \frac{l}{r(k)} = \frac{l}{r_{avg}} \quad (2)$$

$$D_{prop} = \sum_{k=1}^{N+1} \frac{d(k)}{s(k)} \approx \sum_{k=1}^{N+1} \frac{d_{avg}}{s_{avg}} = (N+1)\frac{d_{avg}}{s_{avg}} = \frac{d_{tot}}{S_{avg}} \quad (3)$$

4.2 벤치마킹

이상에서 모델링한 모델의 실제 수행시간을 계산하기 위하여 Intel Celeron 450MHz에서 동작하는 리눅스 커널 2.2의 성능을 벤치마킹 하였다. 수행시간은 벤치마킹 루틴을 커널과 통신규약 스택에 추가하는 방식을 통해 측정되었다. 각각의 모델에 대한 수행시간은 10회 반복 측정 후에 평균치를 사용해 계산하였다.

표 1은 벤치마킹 결과를 정리한 것이다. 데이타의 송신시에는 *net/socket.c* 파일의 *sock_sendmsg()* 함수, *net/ipv4/tcp_ipv4.c* 파일의 *tcp_v4_sendmsg()* 함수, *net/ipv4/route.c* 파일의 *ip_route_output()* 함수, 그리고 *net/ipv4/ip_output.c* 파일의 *ip_queue_xmit()* 함수를 사용해 각각의 통신규약 계층의 수행시간을 측정하였다. 이와 유사하게 수신시에는 *net/ipv4/tcp_ipv4.c* 파일의 *tcp_v4_rcv()* 함수, *net/ipv4/ip_input.c* 파일의 *ip_rcv()* 함수, 그리고 *net/core/dev.c* 파일의 *net_bh()* 함수를 사용해 측정하였다. 위 실험은 루프백 장치를 사용해 하나의 컴퓨터에서 수행되었다. 추가적으로 일반적인 작업을 수행하는 경우의 시스템 호출, 소켓 인터페이

스, 그리고 네트워크 디바이스 드라이버의 수행시간을 측정하였다.

벤치마킹 결과 실험에 사용한 PC의 경우 하나의 송신 또는 수신 패킷을 처리하는데 700-800μs가 소요된다. 이 벤치마킹 결과는 실제 시뮬레이터의 입력으로 사용되어 대상 시스템의 CPU 사양에 따라 $C_{bench} : T_{bench} = C_{target} : T_{target}$ 수식을 사용하여 동적으로 재조정되었다. 이때 C_{bench}는 벤치마킹에 사용된 컴퓨터의 CPU 클럭, T_{bench}는 벤치마킹된 수행시간, C_{target}은 대상 시스템의 CPU 클럭, 그리고 T_{target}는 재조정된 대상 시스템의 수행시간을 의미한다.

표 1 벤치마킹 결과. (단위: μs)

유형	평균	유형	평균
송신: 소켓	128.5	수신: 소켓	102.2
송신: 전송층	302.1	수신: 전송층	346.9
송신: 네트층	222.8	수신: 네트층	155.8
송신: 연결층	148.0	수신: 연결층	126.6
전체 송신	798.0	전체 수신	371.5
전송 송수신	**1528.5**	실행: 소켓	**81.7**
실행: 시스템호출	**91.6**	실행: 물리층	**424.5**

4.3 시뮬레이터 설계 및 구현

이상의 모델과 벤치마킹 데이타를 바탕으로 커널 내부 스택, 사용자 수준 스택, 그리고 NIC-NET의 동작 방식을 반영한 네트워크 시스템 시뮬레이터를 개발하였다. 시뮬레이터는 다양한 설계 옵션과 실행 환경에서 성능을 측정할 수 있도록 개발되었다. 또한 다수의 호스트 CPU와 NIC CPU의 수행 특성을 정확하게 시뮬레이션하기 위하여 그림 6에 제시된 것과 같은 알고리즘을 사용하였다. 이 알고리즘을 사용하면 각각의 CPU를 하나의 쓰레드로 모델링하여 매 단위시간마다 한번씩 스케줄함으로써 다수의 CPU가 동시에 동작하는 것과 같이 시뮬레이션 가능하다. 또한 시뮬레이션을 자동으로 수행하기 위해 시뮬레이션 설정 언어를 정의해 사용하였으며, 손쉬운 제어를 위해 그래픽 유저 인터페이스를 사용하였다.

5. 성능 평가

이 장에서는 커널 내부 스택 및 사용자 수준 스택의 성능과 비교분석한 NIC-NET의 성능을 제시한다. NIC-NET에서 통신규약 스택은 NIC에서 동작하기 때문에 네트워크층의 ICMP, OSPF, 그리고 RIP와 같은 통신규약은 호스트에 계산 능력을 사용하지 않는다. 따라서 이 장에서는 TCP 통신규약을 사용한 경우의 종단간 메시지 응답시간을 중점적으로 분석한다.

5.1 CPU 사용률 분석

그림 7은 네트워크 서버 프로그램이 클라이언트로부터 TCP 통신규약을 통해 수신된 60개의 요청을 처리하는 경우에 호스트 CPU의 통신규약 스택의 처리 시간을 측정한 것이다. 그래프 상에서 심볼이 없이 직선으로 표시된 부분은 인터럽트 처리기를 사용해 NIC로부터 수신된 패킷을 읽어오는데 소요된 시간을 의미한다. 세그먼트의 개수와 크기가 패킷의 개수와 크기에 비하여 크게 작기 때문에 NIC-NET의 경우 인터럽트 처리 시간이 크게 줄어듬을 알 수 있다. 또한 NIC-NET의 경우 심볼을 사용해 표시된 통신규약 스택의 처리 시간도 NIC로 통신규약 스택의 인터페이스를 제외한 대부분의 기능이 위임되었기 때문에 크게 줄어듬을 알 수 있다.

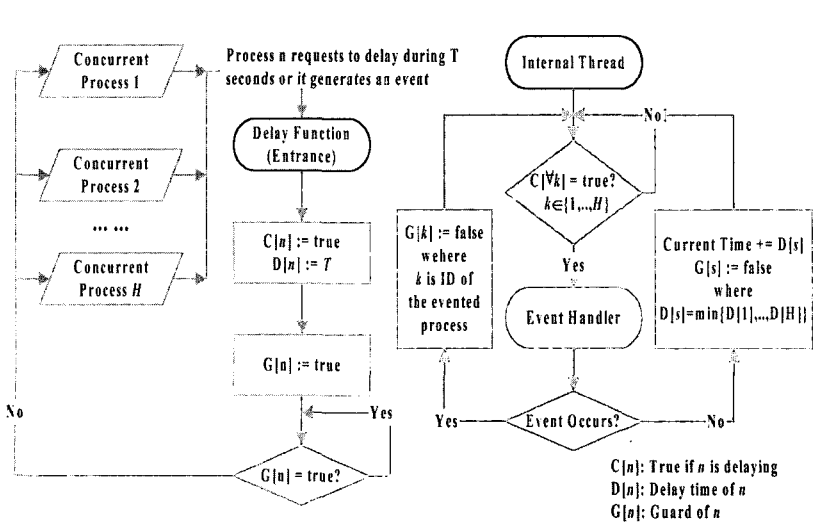


그림 6 다양한 프로세서들을 위한 시뮬레이션 알고리즘

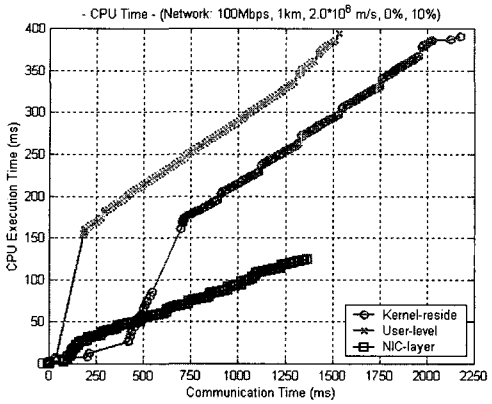


그림 7 TCP 통신 시간에 따른 CPU 수행시간

커널 내부 스택과 사용자 수준 스택에 대한 NIC-NET의 통신규약 스택에 대한 CPU 처리 시간의 감소 비율(rdec)은 수식 (4)와 같다. 수식 tkernel, tuser, 그리고 tNIC는 각각 커널 내부 스택, 사용자 수준 스택, 그리고 NIC-NET의 CPU 수행시간을 의미하며 결과는 약 68%의 감소이다.

$$r_{dec} = 1 - \frac{2t_{NIC}}{t_{kernel} + t_{user}} \quad (4)$$

이 결과는 에러가 없는 통신 망을 가정한 실험에서 얻어진 것으로 패킷 에러율(e)이 주어진 경우의 각각의 시스템의 성능은 다음과 같다. 수식 (5)는 패킷 에러율을 고려한 경우 커널 내부 스택과 사용자 수준 스택의 CPU 처리 시간을 의미한다. 하지만 NIC-NET의 CPU 처리 시간은 통신규약 스택을 NIC로 분산시킨 구조적 특성으로 인하여 패킷 에러율이 주어진 경우에도 에러율이 0인 경우와 동일하다 (TNIC=tNIC). 에러율이 주어진 경우의 NIC-NET의 CPU 수행시간의 감소비율(Rdec)은 수식 (4)와 유사하게 계산할 수 있다.

$$T_x = \sum_{k=0}^{\infty} t_x e^k = \frac{t_x}{1-e}, \; x \in \{kernel, user\} \quad (5)$$

표 2는 다양한 통신 채널을 사용해 1KB 크기의 패킷을 전송하는 경우에 NIC-NET의 CPU 수행시간의 감소비율을 계산한 것이다. 그 결과 NIC-NET을 사용하면 CPU 수행시간이 커널 내부 및 사용자 수준 스택과 비교하여 통신 채널의 에러율에 따라 68-71% 가량 감소됨을 알 수 있다.

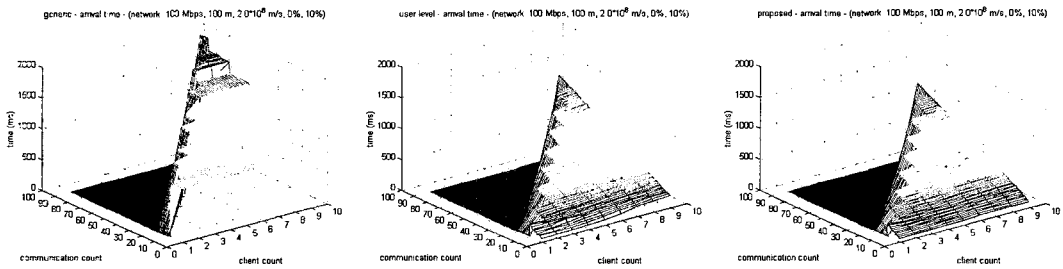
표 2 통신망의 에러율에 따른 CPU 수행시간 감소 비율

Type	Ideal	Wired	Wireless	
e_{bit}	0	$1.0*10^{-7}$	$1.0*10^{-6}$	$1.0*10^{-5}$
e	0	$8.0*10^{-4}$	$8.2*10^{-3}$	$7.9*10^{-2}$
R_{dec}	68%	68%	69%	71%

5.2 메시지 응답시간 분석

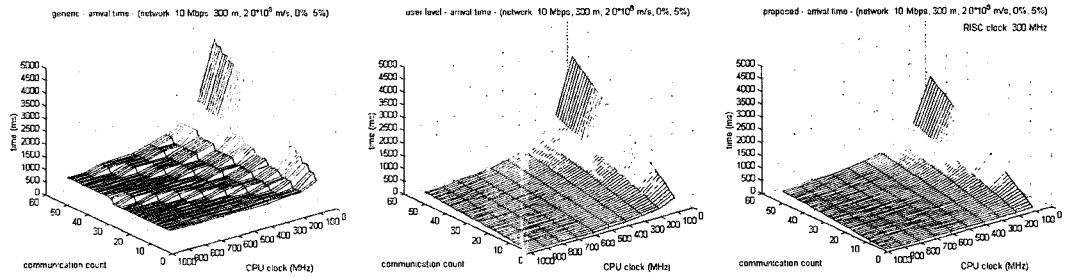
NIC-NET의 CPU 수행시간을 줄이는 특성은 NIC-NET이 고성능 네트워크 서버 시스템에 적용되었을때 해당 서버의 성능을 크게 개선할 수 있음을 의미한다. 이 실험에서는 고성능 서버 시스템에 NIC-NET을 적용한 경우 클라이언트의 서비스 요청에 따른 메시지 응답 시간을 분석한 것이다. 이때 많은 양의 통신 및 계산 작업을 수행해야 하는 서버 시스템의 환경을 실험에 반영하였다.

그림 8은 각각의 클라이언트가 매 10 밀리 초마다 요청을 보내는 경우에 클라이언트 수를 1대에서 9대로 증가시키며 메시지 응답 시간을 측정한 것이다. 그림 8(a)는 커널 내부 스택의 경우 클라이언트의 요청을 계단 형태로 일정한 간격을 두고 버퍼링하여 처리함을 보인다. 왜냐하면 커널 내부 스택의 경우 인터럽트 서비스 루틴에서의 수행시간을 줄이기 위하여 수신된 패킷을 연결층에 위치한 큐에 저장한 후 커널의 하반부 처리기를 사용해 처리하기 때문이다[1]. 그림 8(b)에서 관찰한 사용자 수준 스택의 가장 큰 특성은 클라이언트 수가 7대 이상으로 설정되어 네트워크 트레픽이 높은 경우에 도착된 패킷을 수신하느라 전혀 응답을 하지 못한다는 점이다[3]. 반면에 그림 8(c)를 통해서 NIC-NET은 이



(a) 커널 내부 스택 (b) 사용자 수준 스택 (c) NIC-NET

그림 8 네트워크 트레픽 vs. 메시지 응답시간



(a) 커널 내부 스택 (b) 사용자 수준 스택 (c) NIC-NET

그림 9 호스트 CPU 클럭 vs. 메시지 응답시간

러한 상황에서도 응답을 할 수 있음을 알 수 있는데 이는 NIC-NET의 경우 NIC에서 패킷을 수신하는 동안에 호스트 시스템에서는 이를 처리해 응답 메시지를 생성하기 때문이다. NIC-NET의 커널 내부 및 사용자 수준 스택과 비교한 응답시간의 감소비율은 각각 32-83%와 11-36%이다.

그림 9는 호스트 CPU의 클럭을 조정함으로써 많은 양의 계산 작업을 호스트에 가한 경우에 메시지 응답시간을 측정한 것이다. 비록 실험 환경은 그림 8의 실험과 다르지만 이 실험에서도 앞의 실험과 유사한 성능상의 특성을 관찰할 수 있다. 뿐만 아니라 벤치마크 프로그램의 스케줄링 비율을 통해 계산 작업의 양을 조정한 실험에서도 그림 9와 유사한 결과를 확인할 수 있었다. 그 결과 NIC-NET은 호스트의 계산 수요가 높은 경우에 커널 내부 및 사용자 수준 스택과 비교하여 메시지 응답시간을 각각 32-81%와 20-61% 단축시킴을 확인하였다.

다음으로 NIC-NET의 성능은 NIC에 위치한 CPU의 클럭에 크게 의존적일 수 있기 때문에 NIC-NET의 성능을 NIC와 호스트에 위치한 CPU들의 클럭을 변화시키며 분석하였다. 이때 클라이언트로부터 초당 900 개의 높은 서비스 요청이 도착하는 환경을 가정하였다. 그림 10은 이 실험 결과중의 하나로 호스트 CPU 클럭이 500MHz이고 NIC의 CPU 클럭이 50MHz에서 1GHz인 경우의 메시지 응답시간을 측정한 것이다. 그 결과 NIC-NET의 메시지 응답시간이 NIC의 CPU 클럭이 200MHz 이하인 경우에 저하될 수 있음을 확인하였으며 이러한 특성은 호스트의 CPU 클럭과는 무관함을 관찰할 수 있었다. 따라서 NIC의 CPU 클럭을 50-200 MHz로 설정하면 가장 비용 효율적인 형태로 NIC-NET을 구축할 수 있다.

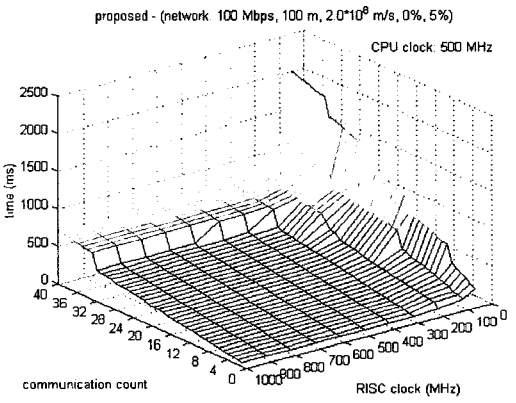


그림 10 NIC CPU 클럭 vs. 메시지 응답시간

6. 결 론

본 논문에서는 고성능 서버 시스템을 위하여 NIC에 기반한 호스트 독립적인 네트워크 시스템을 설계하고 성능을 평가하였다. 제안하는 시스템은 통신규약 스택을 호스트에서 NIC로 분산시킴으로써 호스트 시스템의 통신규약 스택 처리에 따른 계산량을 크게 줄이며, 이렇게 분산된 통신규약 스택과 사용자 수준의 라이브러리를 사용해 통신함으로써 통신규약 스택에 접근하기 위한 시스템 호출 비용을 제거한다. 또한 NIC에서는 패킷이 아닌 세그먼트 단위로 인터럽트를 생성함으로서 호스트의 네트워크 인터럽트 처리 비용을 크게 줄인다.

시뮬레이션 결과 제안하는 시스템의 CPU의 통신규약 스택의 처리 시간을 68-71% 단축시킴을 보였다. 이러한 특성으로 인하여 호스트에 계산과 통신 작업에 대한 수요가 높은 경우에 기존 시스템과 비교하여 메시지 응답시간을 11-83% 단축시킴을 보였다. 따라서 제안하는 시스템은 고성능 네트워크 서버나 그리드 컴퓨터와 같은 시스템에 적용될때 성능이 극대화될 수 있다.

추가적으로 모델링 과정에서 TCP의 혼잡 제어를 고려하지 않았는데, 연결 초기에 TCP 혼잡 제어 윈도우의 크기는 흐름 제어 윈도우의 크기보다 작기 때문에 실제 TCP 통신규약은 본 논문에서 사용한 TCP 모델

보다 동일한 크기의 세그먼트를 더 많은 수의 패킷으로 분할하게 된다. 이는 기존 시스템의 네트워크 인터럽트 처리 비용을 증가시키게 되어 궁극적으로는 제안하는 시스템의 상대적인 성능 향상폭을 더 증가시키는 요인으로 작용될 것이다. 뿐만 아니라 본 논문에서는 DMA 방식의 입출력을 활용하지 않았는데, DMA 입출력을 활용하면 호스트의 NIC로의 메모리 복사 비용을 완벽하게 제거(true zero-copy)할 수 있다. 따라서 향후 연구로 제안하는 시스템에 DMA 입출력을 적용하여 설계해 볼 수 있을 것이다.

참 고 문 헌

[1] U. Vahalia, *UNIX Internals: The New Frontiers*, pp. 31-33, 1996.

[2] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," *In Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pp. 40-53, 1995.

[3] J. Mogul and K. K. Ramakrishnan, "Eliminating Receive Livelock in an Interrupt-driven Kernel," *In Proceedings of the 1996 USENIX Annual Technical Conference*, 1996.

[4] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communication Magazine*, Vol. 27, No. 6, pp. 23-29, 1989.

[5] G. R. Ganger, D. R. Engler, M. F. Kaashoek, H. M. Briceno, R. Hunt, and T. Pinckney, "Fast and Flexible Application-Level Networking on Exokernel Systems," *ACM Transactions on Computer Systems*, Vol. 20, No. 1, pp. 49-83, 2002.

[6] M. Boosten, R. W. Dobinson, and P. D. V. Stok, "MESH: MEssaging and ScHeduling for Fine-Grain Parallel Processing on Commodity Platforms," *In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

[7] C. Csanady and P. Wyckoff, "Bobnet: High-Performance Message Passing for Commodity Networking Components," *In Proceedings of the International Conference on Parallel and Distributed Computing and Networks*, December 1998.

[8] G. Chiola and G. Ciaccio, "Operating System Support for Fast Communications in a Network of Workstations," *Technical Report DISI-TR-96-22*, 1996 (http://www.disi.unige.it/project/gamma).

[9] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke, and C. Dodd. "The Virtual Interface Architecture," *IEEE Micro*, Vol. 18, No. 2, pp. 66-76, 1998.

[10] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, Vol. 15, No. 1, pp. 29-36, 2002.

[11] S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet," *In Proceedings of the 1995 ACM International Conference on Supercomputing (SC)*, No. 55, 1995.

[12] B. N. Chun, A. M. Mainwaring, and D. E. Culler, "Virtual Network Transport Protocols for Myrinet," *IEEE Micro*, Vol. 18, No. 1, pp. 53-63, 1998.

[13] P. Shivam, P. Wyckoff, and D. Panda, "EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing," *In Proceedings of the 15th ACM International Conference on Supercomputing (SC)*, No. 57, 2001.

[14] D. E. Comer et al., *Internetworking with TCP/IP*, Vol. 1-2, 4th Ed., Prentice-Hall, 2000.

[15] Myricom Inc., http://www.myrinet.com/.

[16] J. C. Mogul, "TCP Offload is a Dumb Idea Whose Time Has Come," *In Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, pp. 25-30, May 2003.

[17] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, 2nd Ed., O'Reilly & Associates, 2002.

임 근 수

정보과학회논문지 : 시스템 및 이론
제 31 권 제 2 호 참조

차 호 정

1985년 서울대학교 컴퓨터공학 학사. 1987년 서울대학교 컴퓨터공학 석사. 1991년 Univ. of Manchester 전산학 박사. 현재 연세대학교 컴퓨터산업공학부 교수. 관심분야는 멀티미디어 시스템, 운영체제, 내장형 시스템임

고 건

정보과학회논문지 : 시스템 및 이론
제 31 권 제 2 호 참조