고성능 IPv6-IPv4 프로토콜 변환기의 구현에 관한 연구

공인엽° 이중렬 이경렬 이정태
부산대학교 컴퓨터공학과
{leafgirl°, blueirix, idking96, jtlee }@pusan.ac.kr

Study on Implementation of High-performance IPv6-IPv4 Protocol Translator

InYeup Kong° JoongLyul Lee KyoungYeol Lee JungTae Lee
Dept. of Computer Engineering, Pusan National University

요 약

IPv6의 도입을 위한 기술 연구가 활발하게 이루어지고 있는 가운데, IPv4 네트워크와 IPv4 네트워크가 혼재하는 환경에서 네트워크 간의 통신을 가능하게 하는 IPv6-IPv4 프로토콜 변환기에 대한 관심과 수요가 증가되고 있다. 이에 IETF에서는 여러 기법을 제안하였고, 그 중 일부는 소프트웨어로 구현되었다. 그러나 기존에 구현된 IPv6-IPv4 프로토콜 변환기는 운영체제를 탑재한 호스트 시스템에서 동작하는 것으로서, 네트워크 간의 모든 트래픽에 대한 변환을 수행해야 하므로 고성능이 요구된다. 이에 본 논문에서는 기존의 소프트웨어 IPv6-IPv4 프로토콜 변환기의 성능을 측정하였고, 성능 저하의 요인을 분석하였다. 또한 성능을 개선할 수 있는 방안을 제시하고, 이를 적용한 하드웨어 기반의 IPv6-IPv4 프로토콜 변환기인 64Translator를 제안하였다.

1. 서 론

1990년대 초부터 IETF(Internet Engineering Task Force)에서 IPv6 프로토콜의 표준화를 진행함에 따라 IPv6에 관련된 연구가 여러 분야에서 활발히 진행되고 있다. IPv6는 128비트의 방대한 주소 공간을 사용하며, 주소 자동 설정, 품질 보장, 보안 기능 제공 및 이동성 지원 등의 향상된 기능을 제공하는 프로토콜이다. IPv6는 현 인터넷의 일부가 IPv6 호스트 및 네트워크로 교체 또는 신규 구축되면서 점진적으로 도입되므로 IPv6 네트워크와 IPv4 네트워크가 상당 기간 혼재할 것으로 예상된다. 이에 IETF에서는 IPv4에서 IPv6로의 전환 기법들을 제안하였고, 일부는 소프트웨어로 구현되었다[1][2].

그러나 기존의 IPv6-IPv4 프로토콜 변환기는 운영 체제에 내장된 형태로 구현되어 있어서, 이를 사용하기 위해서는 운영 체제 내에 구현된 TCP/IP 프로토콜 자체의 오버헤드와 비효율적인 메모리 엑세스 방식으로 인하여 처리 성능이 제한되는 문제점이 있다.

따라서 본 연구에서는 기존 IPv6-IPv4 프로토콜 변환기의 성능 측정 결과와 성능 인자를 분석하고, 이를 토대로 고성능 IPv6-IPv4 프로토콜 변환기로서 하드웨어 기반의 Standalone IPv6-IPv4 프로토콜 변환기(*64Translator*)를 제안하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기구현된 운영 체제 기반 IPv6-IPv4 프로토콜 변환기의 구조, 성능 측정결과 및 성능 인자를 분석하고, 3장에서는 IPv6-IPv4 프로토콜 변환기의 성능을 개선하기 위한 구현 방안을 제시한다. 그리고 4장에서는 이러한 성능 개선 기법을 적용한 하드웨어 기반의 64Translator를 제안하고, 마지막으로 5장에서는 결론과 향후 과제로 마무리한다.

2. 운영 체제 기반의 IPv6-IPv4 프로토콜 변환기

2.1 프로토콜 구조

운영 체제 기반의 IPv6-IPv4 프로토콜 변환기의 내부 구조는 그림 1에서 보는 바와 같다.

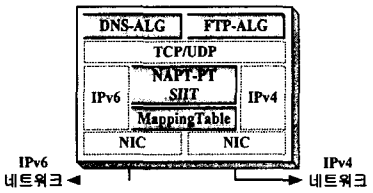

그림 1. IPv6-IPv4 프로토콜 변환기의 프로토콜 계층 구조

IPv6-IPv4 프로토콜 변환기는 IPv6 네트워크와 IPv4 네트워크의 사이에 존재하며, 상이한 IP 프로토콜의 변환 기능을 제공한다. IPv6-IPv4 프로토콜 변환기는 그림 1에서 보는 바와 같이 IPv6와 IPv4를 탑재한 Dual-Stack 호스트로서, NAPT-PT/SIIT, Mapping Table, DNS-ALG, FTP-ALG로 구성된다.

NAPT-PT/SIIT는 IPv6-IPv4 프로토콜 변환기의 핵심 변환 모듈로서 IPv6/ICMPv6 헤더와 IPv4/ICMPv4 헤더 간의 프로토콜 변환을 담당할 뿐만 아니라 IP 주소와 포트번호의 변환도 담당한다. Mapping Table은 그 구조는 다음과 같이 정의된다. DNS-ALG와 FTP-ALG는 각각 해당 응용 프로토콜 헤더에 포함된 IP 주소와 포트 번호에 대한 변환을 담당하는 응용 레벨 게이트웨이(Application Level Gateway) 모듈이다. 더불어 DNS-ALG는 IPv4(A 레코드 형식)와는 다르게 IPv6(AAAA 레코드 형식)를 위해 확장된 레코드 형식에 대한 변환을 수행하고, FTP-ALG는 IPv6와 IPv4에서 다르게 정의된 명령에 대한 변환을 수행한다.

2.2. 성능 측정

운영 체제 기반 IPv6-IPv4 프로토콜 변환기의 성능은 트래픽 생성기를 사용하여 지난 논문에서 제시하였다. 트래픽은 1440 바이트의 Payload를 가지는 UDP 패킷을 사용하여 Offered load를 생성하였다[3].

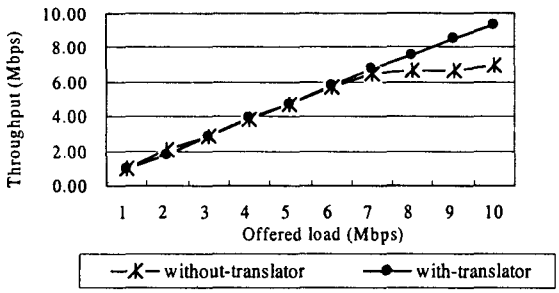

그림 2. 운영 체제 기반 IPv6-IPv4 프로토콜 변환기의 성능

그림 2에서 보는 바와 같이 변환기 없이 호스트 기반 IPv6 라우터로만 작동시켰을 경우(without-translator)의 처리율은 거의 트래픽 부하에 비례하는 성능을 보여준다. 이에 비해 운영 체제 기반의 IPv6-IPv4 프로토콜 변환기의 경우(with-translator), 7Mbps 이상의 트래픽 부하가 가해지면 처리 성능이 더 이상 증가되지 않으며, 부하가 커질수록 둘 간의 성능 차이는 커진다. 즉, 트래픽의 부하가 일정 수준 이상 증가하는 경우, IPv6-IPv4 프로토콜 변환기는 성능의 한계를 나타나게 된다.

2.3. 성능 인자 분석

기구현된 IPv6-IPv4 프로토콜 변환기는 운영 체제에 내장된 TCP/IP 프로토콜을 기반으로 동작한다[1][2][3]. 본 연구에서는 IPv6-IPv4 프로토콜 변환기의 성능 인자를 분석하기 위해서, 먼저 TCP/IP 자체의 성능 인자에 대한 기존 연구를 분석하였다[4].

표 1. 프로토콜 구현의 오버헤드

인자	처리 시간	단위
User-System Copy	200 μsec	바이트
TCP Checksum	185 μsec	
Network-Memory Copy	386 μsec	
Ethernet Driver	100 μsec	패킷
TCP+IP+ARP	100 μsec	
OS Overhead	240 μsec	

User-System Copy 는 사용자 버퍼와 커널 버퍼 간에 패킷을 복사하는데 소요되는 시간이다. TCP Checksum 은 TCP Checksum 을 바이트 단위로 계산하는데 소요되는 시간이다. Network-Memory Copy 는 커널 버퍼와 네트워크 카드의 버퍼 간에 패킷을 복사하는데 소요되는 시간이다. Ethernet Driver 는 Ethernet 네트워크 카드의 드라이버 프로그램에서 패킷을 제어하는데 소요되는 시간이다. TCP+IP+ARP 는 TCP, IP 및 ARP 프로토콜 헤더를 처리하는데 소요되는 시간이다. 마지막으로 OS Overhead 는 인터럽트를 제어하고, 운영체제 내부의 패킷 버퍼를 할당 및 해제하는데 소요되는 시간이다. 표를 살펴보면 이러한 오버헤드 중에서 User-System Copy 와 Network-Memory Copy 가 가장 많이 차지하는 것을 알 수 있는데, 이 두 가지는 메모리 접근에서 발생되는 오버헤드이다. 즉, TCP/IP 프로토콜 자체에서도 메모리 접근에 의한 오버헤드가 매우 크다.

다음으로 IPv6-IPv4 프로토콜 변환기가 프로토콜 변환을 위해 소요하는 수행 시간의 비율을 분석하기 위해 본 연구에서는 리눅스의 Profiling 기법인 'gprof' 유틸리티를 사용하여 소프트웨어 IPv6-IPv4 프로토콜 변환기의 주요 기능 모듈에 대해 측정한 결과는 표 2 와 같다.

표 2. S/W IPv6-IPv4 프로토콜 변환기의 수행 시간 및 호출 횟수

기능 모듈	수행시간(%)	호출 횟수(회)
DNS-ALG/FTP-ALG	28.95	14,407
하드웨어주소 획득	22.40	14,623
체크섬 계산	14.47	44,734
Main 함수	9.22	-
패킷 전송	9.22	29,662
주소/포트 변환	5.28	15,425
패킷 수신	3.96	17,990
IP/ICMP 변환	0.00 이하	30,330
기타	6.50	30,926

표 2에서 보는 바와 같이 IPv6-IPv4 프로토콜 변환기에서는 DNS-ALG/FTP-ALG, 하드웨어주소 획득, 체크섬 계산의 순서로 많은 수행 시간을 차지한다. 여기서 Main 함수, 패킷 전송, 패킷 수신은 프로토콜 변환기의 루틴이 아니라 TCP/IP 통신과 관련된 기본 루틴이다.

DNS-FTP/FTP-ALG의 경우, 메모리로부터 많은 Payload를 데이터를 읽어오고, 이를 순차적으로 분석하면서 변환을 하기 때문에 호출 횟수에 비해 수행 시간 비율이 높다. 하드웨어주소 획득은 목적지 호스트의 IP 주소에 대하여 ARP 테이블을 검색하기 위해 항상 메모리에 접근한다. 주소/포트 변환은 Mapping Table을 참조하게 되는데, 이를 위해서도 메모리에 항상 접근한다. 이러한 3가지의 연산은 메모리 참조로 인하여 수행 시간이 많이 요구된다. 반면 IP/ICMP 변환은 변환 규칙에 따라 고정적으로 필드 값만 정해지므로 호출 횟수에 비해 수행 시간이 극히 적다. 따라서 메모리 엑세스의 개선이 성능 향상을 위해 가장 중요한 문제임을 알 수 있다.

다음으로 체크섬 계산은 데이터를 메모리로부터 모두 읽어 온 후 계산을 수행하고, 다시 메모리에 쓰는 형태이므로 메모리 접근과 비효율적 계산 방식으로 인하여 수행 시간의 비율이 높다. 결론적으로 메모리 접근, 체크섬 계산에 대한 구현 방법을 개선하면 IPv6-IPv4 프로토콜 변환기의 성능을 향상시킬 수 있다.

3. IPv6-IPv4 프로토콜 변환기의 성능 개선 방안

3.1 데이터 엑세스의 개선

2절에서 언급한 메모리 접근 오버헤드를 줄이기 위해서는 메모리 접근 횟수를 줄여야 한다. 운영 체제 기반의 메모리 접근 과정은 그림 3에서 보는 바와 같이 NIC의 버퍼, 커널 메모리, 및 사용자 메모리로의 데이터이동이 필요하다.

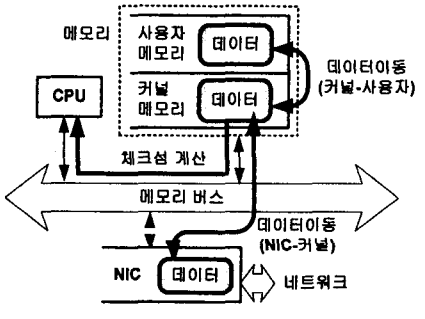

그림 3. 메모리 엑세스 과정 (운영 체제 기반)

그림 3에서 데이터를 송신하는 경우, 먼저 사용자 메모리에 저장된 데이터를 읽어서 커널 메모리에 저장한다. 그리고 CPU에서 커널 메모리로부터 데이터를 읽어서 체크섬을 계산한 후, 다시 커널 메모리의 데이터를 읽어서 DMA 방식으로 NIC 내의 버퍼로 전달한다. 이와 같이 기존의 운영 체제 기반 TCP/IP 프로토콜에서는 적어도 4번의 메모리 엑세스가 발생된다.

이러한 메모리 엑세스는 커널 메모리로 데이터를 복사하는 과정과 CPU가 체크섬 계산 및 검사를 위한 메모리 엑세스를 제거함으로써 개선될 수 있다. 그림 4은 하드웨어 기반 구현에서의 메모리 엑세스 과정을 보여 준다.

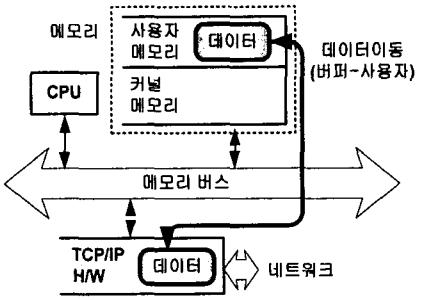

그림 4. 메모리 엑세스 과정 (하드웨어 기반)

하드웨어 기반의 메모리 접근 과정은 그림에서 보는 바와 같이 커널 메모리가 개입되지 않고 데이터가 사용자 메모리 영역으로 직접 전달되기 때문에 메모리 엑세스 오버헤드를 줄일 수 있다.

3.2 체크섬 오버헤드의 개선

본 연구에서는 그림 1에서 보는 바와 같이 포트 번호의 변환을 통해 IP 주소 효율을 높인 NAPT-PT 방식을 채택하였다. 이로 인하여 TCP와 UDP 헤더의 포트 번호 필드를 변환에 의해 체크섬 계산이 추가로 소요되는데, 특히 TCP에서는 TCP 헤더 뿐만 아니라 데이터에 대해서도 체크섬을 계산해야 하므로 오버헤드가 크다.

이를 해결하기 위하여 본 연구에서는 기존의 TCP/IP 하드웨어 구현에서 사용된 체크섬 계산 방식을 사용하였다[4]. 첫째, 체크섬 계산을 Combinational Logic 기반의 하드웨어로 구현하여 계산 속도를 높였다. 둘째, TCP의 헤더 필드 중 예측할 수 있는 필드와 데이터에 대해서는 체크섬을 미리 계산하는 방식을 사용하였다. 셋째, 체크섬 계산을 위해 메모리로부터 읽은 데이터를 FIFO에 저장함으로써 전송 시에는 메모리를 엑세스하지 않고 FIFO의 데이터를 전송하도록 하였고, 이와 동시에 메모리로부터 다음에 전송할 데이터를 읽어서 체크섬을 계산하는 방법을 사용하였다. 즉, 전송 과정과 체크섬 계산과정을 중첩한 Pipelining 방식을 사용하였다. 이 방식에서는 예측할 수 있는 TCP 헤더와 데이터에 대해 부분적으로 체크섬을 계산하고, 이렇게 계산된 체크섬과 송신 직전에 결정되는 Acknowledgement 번호와 Window 필드 값을 이용하여 최종 체크섬을 계산하도록 하였다.

4. 64Translator의 제안

3절에서 설명한 성능 개선 기법을 적용한 64Translator의 내부 구성은 그림 5와 같다. TCP/IPv6 Core, TCP/IPv4 Core 는 각각 해당 프로토콜에 대한 송수신, 헤더 분석/생성 기능을 구현한 하드웨어 모듈로서, 기구현한 모듈에 SIIT 기능 등을 추가 구현하여 사용하였다. ALGTrans 모듈은 ALG를 하드웨어적으로 처리하는 응용 프로토콜 변환 모듈이다[5]. TransCore 모듈은 IPv6 주소/포트 및 IPv4 주소/포트 정보를 Mapping Table로 관리하는 모듈이다[6].

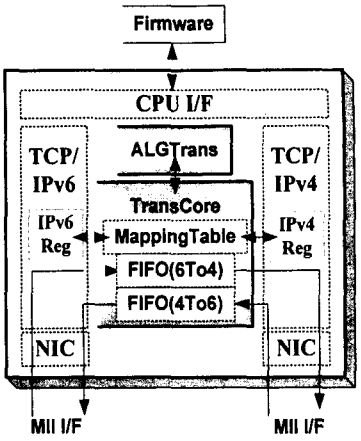

그림 5. 64Translator의 구조

TransCore 모듈에서 사용되는 프로토콜 헤더 정보는 메모리에 저장되지 않고 수신되는 즉시 상대편 TCP/IP Core로 전달되도록 함으로써 지연 시간 없이 바로 처리될 수 있도록 설계하였다. 그리고 Payload는 내부 메모리에 저장함으로써 메모리 접근 시간을 최소화하였다. Mapping Table을 내부 레지스터로 구현하여 메모리 엑세스를 개선하였다. 체크섬 연산은 TCP/IPv6 Core, TCP/IPv4 Core의 3.2절의 방식이 적용된 체크섬 모듈을 사용하여 개선하였다.

5. 결론

현재까지 IPv6 네트워크와 IPv4 네트워크를 연동하기 위한 IPv6-IPv4 프로토콜 변환기는 운영 체제를 기반으로 한 소프트웨어로 구현되었다. 이러한 프로토콜 변환기는 소프트웨어 TCP/IP 자체의 오버헤드 뿐만 아니라 비효율적인 메모리 접근 방식으로 인하여 성능이 저하되는 문제점이 있다. 이에 본 논문에서는 기존에 구현된 프로토콜 변환기의 성능 인자를 분석하여, 이를 개선할 수 있는 방안을 제시하였으며, 이를 적용한 하드웨어 기반의 64Translator를 제안하였다.

향후 과제로는 64Translator의 성능을 측정하여 비교 분석하고, 64Translator의 코드 최적화를 통해 성능을 더욱 개선하는 것이다.

참고문헌

[1] Marc E. Fiuczynski 외 2명, "The Design and Implementation of an IPv6/IPv4 Network Address and Protocol Translator", Proceedings of the USENIX Conference, 1998.

[2] IPv6 Forum Korea, "Linux-based Userspace NAT-PT", http://www.ipv6.or.kr/, 2001.

[3] 공인엽 외 2명, "Design and Implementation of IPv6-IPv4 Protocol Translation System using Dynamic IP address", 2nd HS@I Conference, LNCS2713 pp.496-506, 2003.

[4] 진교홍, "고속실시간 통신을 위한 TCP/IP 프로토콜의 하드웨어 설계 및 구현", 부산대학교 공학박사학위논문, 1997.

[5] 이중렬 외 3명, "IPv6-IPv4 프로토콜 변환기를 위한 ALG 모듈의 하드웨어 설계 및 구현", 한국정보과학회 2003년 추계학술대회 제출.

[6] 이경렬 외 3명, "IPv6-IPv4 프로토콜 변환기의 하드웨어 설계 및 구현", 한국정보과학회 2003년 추계학술대회 제출.