이동 서비스를 위한 터널링 기법의 구현과 성능분석

천정훈, 정진우, 강현국
고려대학교 전자 정보공학과

Implementation and Analysis of Tunneling Method for Mobile Service

Jung-Hoon Cheon, Jin-Woo Jung, Hyun-Kook Kahng
Department of Electronics and Information Eng. Korea University
grcurly@tiger.korea.ac.kr, jjw@hard.korea.ac.kr, kahng@tiger.korea.ac.kr

Abstract

In recent years, it is required that computing support mobile user with computer. The advantage of mobile computing is that users may access all their applications from any location, whether they are in another building or a different state. So, Internet combines with mobile computing technology to make new communication environment for supporting mobility. The research for solving the problem of mobility is actively in progress. This paper describes the implementation of tunneling method for flexible bypass between specific region. Tunneling method provide mobile service to mobile hosts. IP datagram's address transform method is IP-within-IP encapsulation by which an IP datagram may be encapsulated within an IP datagram. The developed IP-within-IP protocol can provide not only enhanced performance because it is implemented in kernel mode, but also convenience of usage to the application developers because it gives user interface as a dynamic link library. Verification of IP packet tunneling was text file transfer program.

I. 서론

최근 휴대용 컴퓨터와 노트북이 널리 보급되고, 인터넷과 웹에 연결된 사용자의 수가 급증함에 따라 네트워크 하부구조의 변화가 요구되고 있다. 이런 환경에서 이동 컴퓨터들은 네트워크의 접속점을 수시로 변경하고 있다. 따라서, 이동 디바이스가 연결의 끊김 없이 통신하기 위한 통신 구조가 필요하게 되었다. 즉, 이동 노드는 자신의 새로운 위치를 나타내는 다른 IP 주소로 재설정 되어야 한다. 하지만, IP 주소를 변경하게 되면, 이미 설정된 트랜스포트 계층의 연결을 잃어버리게 되는 일이 발생한다.

이러한 문제를 해결하기 위해서 IETF에서 제안한 네트워크 계층에서 이동성을 지원하는 Mobile IP 프로토콜은 홈 에이전트(Home Agent, HA)로부터 이동 노드(Mobile Node, MN)로 데이터그램을 전송하기 위해서 또는 이동 노드가 특정 목적지로 데이터를 전송하기 위해서 재수소화 방법을 사용하고 있다. 현재 사용되는 재주소화 방법은 몇 가지가 있으나 본 논문에서는 기존의 IP 헤더에 동일한 구조를 가지는 IP 헤더를 추가하는 방법을 채택하여 IP-within-IP 프로토콜을 커널 레벨로 구현하였다. 그리고, 텍스트 파일 전송 프로그램을 통해서 IP-within-IP 프로토콜이 성공적으로 IP 데이터그램을 캡슐화하여 목적지로 올바르게 전송하는 터널링 기능을 정상적으로 수행함을 확인 하였다.

II. 본론

1. IP-within-IP 프로토콜의 구현

본 연구에서는 IPv4 와 터널링 기능을 Windows NT 커널 내부에 프로토콜 형태로 구현하였다. 구현한 프로그램을 커널에 위치시켰다. 구현한 IP-within-IP 프로토콜은 NCPA(Network Control Panel Applet)를 이용하여 설치하며, 이를 위해 요구되는 스크립트 파일을 작성하였다. 또한 사용자의 선택에 따라 설치된 드라이버의 로딩과 언 로딩이 가능하도록 프로토콜 개발시 NT가 요구하는 루틴(WIN 32 API, Dispatches Routine, etc)들을 작성하였다.

1.1 IP-within-IP 프로토콜의 구현 구조

그림 1 은 IP-within-IP 프로토콜 구현 구조와 윈도우 NT 네트워크 드라이버 구조사이의 대략적인 연관성을 보여주고 있다. IP-within-IP 프로토콜은 상위 사용자의 명령(Command)에 의해서 호출된 내부 함수를 수행하여 사용자에게 서비스를 제공하며, 만약 하부 네트웍 카드의 서비스가 필요하다면 NDIS 를 통해서 네트웍 카드 드라이버를 구동시킨다. 디스패치 함수는 사용자의 호출이 있을 때 미리 지정된 동작을 수행하는 함수이다. Lower-Edge 함수는 NDIS 를 통해 네트워크 접속 카드로부터 패킷을 수신하였을 경우나, NDIS 에 요청한 송신의 비 동기적 완료를 통보해 주는 경우와 같이 NDIS 를 통해 프로토콜과 접속 카드

사이에 이벤트가 발생할 때 사용하게 된다. Windows NT의 DDK에서는 디스패치 함수와 같이 이를 위한 함수를 제공하므로 그 함수들을 이용하여 IP-within-IP 프로토콜을 위한 Lower-Edge 함수를 구현하였다.

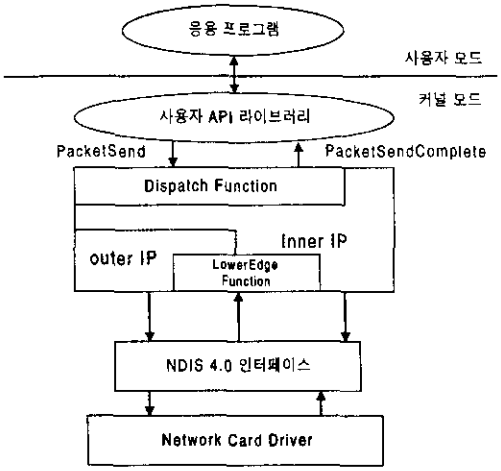

그림 1. IP-within-IP 프로토콜의 구현구조

본 논문에서 구현한 IP-within-IP 프로토콜은 이동 노드가 에이전트의 위치를 검색하고, 등록하는 과정을 통해 상태 정보 리스트(Home List, Visitor List)를 생성한 것으로 가정하고, Mobile IP 프로토콜의 핵심인 라우팅 메커니즘만을 구현하였다. 구현 범위를 나타내면, 그림 2의 회색 부분과 같다.

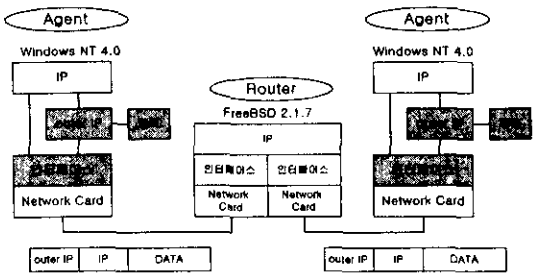

그림 2. IP-within-IP 의 구현 범위

1.2 IP-within-IP 프로토콜 동작

구현된 프로토콜은 세 가지 모듈로 나뉘어 동작하게 된다. 하나는 Inner IP를 처리하는 모듈과 다른 하나는 Outer IP를 처리하는 모듈, 그리고 하부 통신망 접속 카드의 이벤트를 처리하는 모듈로 나뉘어 진다.

윈도우의 한 객체가 데이터를 전송하고자 할 때, 각각의 모듈에서 처리되는 과정을 살펴보면 그림 3과 같다.

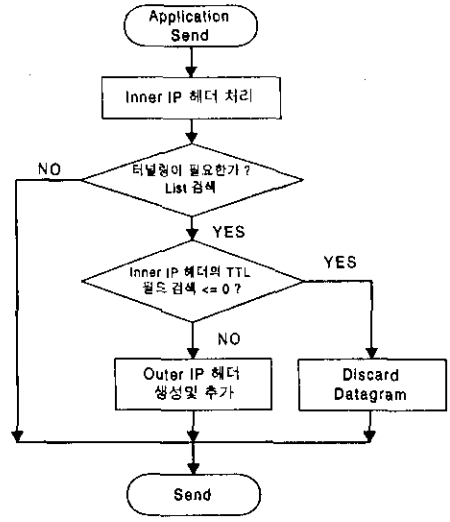

그림 3. IP-within-IP 데이터그램 전송과정

(1) 기존의 IP 헤더를 처리하는 부분은 송신할 데이터에 Inner IP 헤더를 추가하고 필요한 IP 헤더 필드를 처리한다. 만약 송신될 메시지가 네트워크 카드의 최대 전송 단위보다 크다면, 메시지를 분할하여 최대 전송 단위보다 작은 데이터그램 형태로 만든다.

(2) Inner IP 헤더 처리가 끝난 후 이동 노드의 상태정보를 유지하는 리스트(Home list)를 검색하여 목적지 노드가 이동하였다면 캡슐화하기 위해서 Outer IP 헤더 처리 모듈을 호출하게 된다.

(3) Outer IP 헤더 처리 모듈은 캡슐화 방법으로 IP-within-IP 인캡슐레이션을 사용한다. 따라서 Inner IP 데이터처리 모듈에서 넘겨진 패킷에 홈 리스트(Home List)의 외부 에이전트(FA) 주소를 참조하여 새로운 Outer IP 헤더를 생성하고 패킷에 추가한다.

(4) 완전한 데이터그램이 생성되면 프로토콜 드라이버는 네트워크를 통해 전송하기 위해 하부 통신망 이벤트 처리 모듈을 호출하여 전송한다.

하위의 네트워크 카드가 데이터그램을 수신했을 때 각각의 처리 모듈이 수행되는 과정을 보면 그림 4와 같다.

(1) 가장 먼저 수신 이벤트 처리 모듈에서 데이터그램의 수신을 프로토콜에게 알려준다.

(2) 프로토콜에서는 수신된 데이터그램이 최종목적지에 도달하였는지를 검사하고, 만약 최종 목적지에 도달한 것이라면, 데이터그램이 캡슐화 된 것인지를 검사한다.

(3) 수신된 데이터가 캡슐화 된 것이라면, 이동 노드에 대한 상태정보 리스트(Visitor List)를 검색하여 목적지 노드가 네트워크 안에 존재하는지를 검색하게 된다.

(4) 목적지 노드가 방문자 리스트(Visitor List)에 존재하지 않는다면, 다시 홈 에이전트로 캡슐화하여 전송하기 위해서 Outer IP 헤더 처리 모듈을 호출하게 된다.

(5) 목적지 노드가 방문자 리스트에 존재한다면, Outer IP 헤더를 제거한다.

(6) Outer IP 헤더가 제거된 패킷은 Inner IP 헤더 처리를 위해 Inner IP 헤더 처리 모듈에게 넘겨지고, 데이터그램 조합 처리와 헤더 처리 작업이 수행된다.

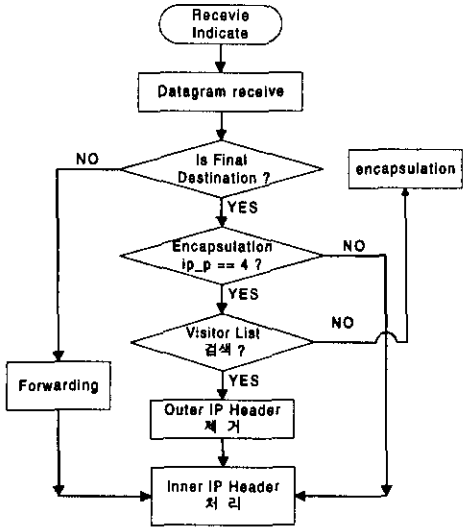

그림 4. 수신 데이타그램 처리과정

2. IP-within-IP 프로토콜의 성능분석

2.1 IP-within-IP 프로토콜의 테스트 환경

본 논문에서는 Windows NT 4.0의 운영 체제를 갖는 펜티엄급 PC 2대와 FreeBSD 2.1.7의 운영 체제를 갖는 펜티엄급 PC 1대를 이더넷으로 연결하여 테스트하였다. Windows NT 4.0의 PC 2대에 기본적인 IP 기능과 IP 캡슐화 기능을 프로토콜 형태로 구현하였으며, 중간에 라우터 기능을 수행할 수 있는 FreeBSD 2.1.7의 PC 1대를 사용하였다. 디버깅 프로그램으로는 Microsoft 사에서 제공하는 SDK (Software Developm-ent Kit)의 WinDbg를 사용하였고, 디버깅 함수는 DDK(Device Driver Kit)에서 제공되는 함수를 사용하였다.

그림 5와 같이 호스트 A는 홈 에이전트로 동작하며, 호스트 B는 외부 에이전트로써 동작하도록 구성하였다. Windows NT 커널에 구현된 IP-within-IP 프로토콜의 동작을 확인하기 위해 DLL 형태로 구현된 사용자 API 함수를 이용하여 텍스트 파일전송 프로그램을 구현하여 기능 시험을 수행하였다.

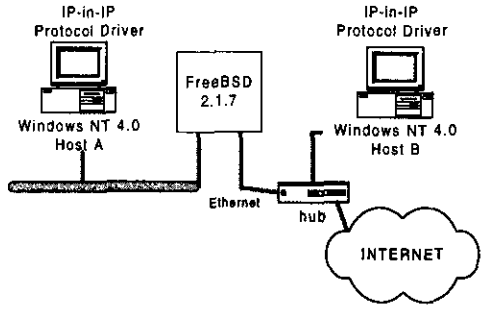

그림 5. 테스트 환경

호스트 A에서 전송한 40개의 패킷이 올바르게 터널링 되었는지를 알아보기 위해서 FreeBSD가 설치된 컴퓨터를 모니터링 하였다. 그 결과는 <표 1>과 같은데, 라우터가 40개의 패킷을 수신하여 기존의 IP 패킷과 동일한 방식으로 포워딩(forwarding) 시킨다는 것을 확인할 수 있다.

<표 1> 모니터링 결과

```
ip:
        2185 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 with bad options
        0 with incorrect version number
        0 fragments received
        0 fragments dropped (drop or out of space)
        0 fragments dropped after timeout
        0 packets reassembled ok
        2117 packets for this host
        0 packets for unknown/unsupported protocol
        [redacted]
        0 packets sent with fabricated ip header
        0 output packets dropped due to no bufs, etc.
        0 output packet discarded due to no route
        0 output datagrams fragmented
        0 fragments created
        0 datagrams that can't be fragmented
```

2.2 성능 분석

IP-within-IP 프로토콜은 이벤트 구동 방식으로 구

현되었으므로 디버깅이나 성능 또한 이벤트 중심으로 분석하였다. 사용자 계층에서의 처리율은 파일 전송 프로그램으로 다양한 크기의 파일을 전송한 후 수신이 완료된 시간을 측정하여 처리 속도의 평균값을 계산하여 산출하였다. 즉, 파일 전송 프로그램이 PacketSend Packet를 호출한 이후부터 PacketSendComplete 이벤트가 수신된 시점사이의 차이이다. 인캡슐레이션한 패킷으로 인해 발생한 오버헤드를 알아보기 위해서, 데이터 송·수신에 소요된 시간과 계산된 데이터 처리율을 터널링 되지 않았을 때와 비교하여 나타내면 <표 2>와 같다.

<표 2> 터널링된 패킷 처리시간과 처리율

Data Size (단위 : 바이트)	Not-Tunneling		Tunneling		Overhead
	Time T_{op}	Through-put (Mbps)	Time T_{tp}	Through-put (Mbps)	
13300	0.120	0.845	0.124	0.818	3.1
29000	0.260	0.851	0.262	0.844	0.82
44000	0.394	0.852	0.397	0.845	0.82
59000	0.532	0.846	0.537	0.838	0.91
74000	0.665	0.849	0.672	0.840	1.06

<표 2>에서 고려되는 사항은 일정한 크기 이하의 메시지를 전송할 경우에 발생하는 처리 감소율이다. 이것은 이더넷의 최대 전송 단위가 1540 바이트로 제한됨으로써 발생하는 문제이다. 1480 바이트의 데이터를 보내고자 할 때, 기존의 IPv4 에서는 Inner IP 헤더 20 바이트를 추가하여 전송하면 된다. 그러나, IP-within-IP 프로토콜은 Inner IP 헤더 20 바이트 외에 Outer IP 20 바이트를 추가하여 패킷을 생성하게 된다. 여기서 발생하는 문제점은 IP 패킷의 전체 길이(1480 +20 + 20)가 MTU를 초과하게 된다는 것이다. 이런 경우 특정한 크기의 패킷은 두번에 나뉘어 전송될 수 밖에 없으며, 따라서 터널링 되지 않은 패킷보다 하나의 패킷을 전송하는 시간이 더 소요되게 된다. 작은 양의 데이터를 전송할 경우(14800 byte 이하), 하나의 패킷을 추가적으로 전송 하는 것은 큰 오버헤드로 작용하지만, 대량의 데이터 전송에서는 비교적 작은 오버헤드로 작용한다는 것을 알 수 있다. 그러므로, Mobile IP 프로토콜을 기존의 IP 프로토콜에 적용시키게 되면 이동 노드에게 이동 서비스를 제공할 수 있게 된다.

III. 결론

본 연구에서는 Inner IP와 Outer IP 라는 2단계 주소체계를 갖는 패킷의 터널링을 통해 이동 서비스를 실현하도록 하였다. 이 방법을 사용하게 되면, 다른 캡슐화 방법보다 좀더 큰 오버 헤드를 갖고, 데이터그램을 디캡슐레이션 할 수 있는 터널의 종점이 없다면 사용될 수 없다는 문제점이 있다. 반면, 기존의 IP 헤더를 변화 없이 사용함으로써 비교적 처리가 간단하고, 기존의 IP 헤더와 호환성이 높으며, 중간 노드들은 기존의 IP 데이터그램과 같은 방식으로 처리함으로 투명성이 보장된다는 것이다. 또한, 본 연구에 의해 개발된 구현 결과는 Windows NT의 커널상에서 프로토콜 구현기술 확보와 기존 네트워크의 S/W와 H/W를 수정하지 않고도 이동 통신 서비스를 제공할 수 있다는 점에서 의미 있는 성과를 거두었다고 하겠다.

본 논문에서는 라우팅 메커니즘만을 고려하였고, ICMP 에시지를 사용한 agent discovery 와 등록 절차는 차후 연구 과제로 남아있다. 그리고, 라우터에서의 터널링 기법을 시험하고, 마지막으로 Mobile IP의 완전한 기능(agent discovery, 등록 등)을 구현하기 위한 연구가 요구된다. 또한 이번 구현 범위에서 향후 고려사항으로 남겨 둔 ICMP 처리 문제와 보안 문제, IP 프로토콜의 주소체계를 확장 시킨 경로 최적화의 연구가 필요할 것으로 사료된다.

참고 문헌

[1] Perkins, C., "IP Mobility Support", RFC 2002, October 1996.

[2] Perkins, C., "IP Encapsulation within IP", RFC 2003, October 1996.

[3] W. Simpson, "IP in IP Tunneling", RFC 1853, October 1995.

[4] Reynolds, J.Postel, "Assigned Numbers", STD2, RFC 1700, USC/Information Sciences Institute, October 1994.

[5] J.Postel, "Internet Protocol", STD 5, RFC 791, September 1981.

[6] Helen Custer, "Inside WINDOWS NT", Microsoft Press, 1993

[7] Microsoft WINDOWS NT Device Driver Kit, "Network Drivers", 1993

[8] Microsoft WINDOWS NT Device Driver Kit,"Kernel mode Driver Reference", 1993

[9] Douglas E. Comer & David L. Stevens, "Internetworking with TCP/IP Volume I", pp. 59-137, Prentice Hall, 1991.

[10] Art Baker, " The WINDOWS NT Device Driver Book", Prentice Hall, pp. 1-77, 1996

[11] Microsoft WINDOWS NT Device Driver Kit,"NDIS 3.0 Network Driver Design Guide", 1992