이동 인터넷 서비스를 위한 IP-within-IP 프로토콜의 구현 및 성능 평가

정회원 정 진 우*, 천 정 훈*, 강 현 국*

Implementation and Analysis of IP-within-IP Protocol for Mobile Service.

Jin-Woo Jung*, Jung-Hoon Cheon*, Hyun-Kook Kahng* *Regualr members*

요 약

현재 다양한 형태의 망들이 연결되어 있는 인터넷과 이동 컴퓨팅 기술을 접목하여 새로운 통신 환경을 조성하려는 연구들이 일반화되어 가면서, 이동성에 의해 발생되는 문제들을 해결하기 위한 연구 또한 활발히 진행되고 있다. 기존의 인터넷이 사용자의 이동성을 지원하기 위해서는 네트워크 계층 및 전송 계층의 프로토콜 동작에 많은 변화가 요구된다. 이에 본 논문에서는 기존의 IP 경로설정 방법으로 전송될 수 없는 데이터그램을 특정 구간에서 유연하게 통과시킬 수 있는 터널링(Tunneling) 방법을 구현하여 이동 서비스에 적용할 수 있도록 하였다. IP 데이터그램 주소 변환 방법으로는 기존의 IP 네트워크와 호환성이 높은 IP-within-IP 방식을 사용하였으며, 그 기능을 Windows NT의 커널 내부에 프로토콜 형태로 구현하였다. 구현한 IP-within-IP 프로토콜은 커널 모드에서 동작하도록 구현함으로써 향상된 성능을 제공하며, 다이나믹 링크 라이브러리 형태로 사용자 인터페이스를 제공하므로 응용 프로그램 개발자들이 손쉽게 사용할 수 있도록 하였다. 텍스트 파일전송 프로그램을 사용하여 구현한 IP-within-IP 프로토콜의 성능을 검증하였다.

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 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. 서 론

최근 휴대용 컴퓨터가 널리 보급되고, 인터넷과 웹에 연결된 사용자의 수가 급증함에 따라 네트워크 하부구조의 변화가 요구되고 있다. 이러한 환경에서 이동 컴퓨터들이 네트워크의 접속점을 수시로 변경함에 따라, 사용자들은 연결의 끊김없이 통신하기 위한 통신 구조를 필요로 하게 되었다.

* 고려대학교 전자정보공학과
논문번호 : 98478-1103, 접수일자 : 1998년 11월 3일
※본 연구는 한국과학재단 목적기초연구(핵심연구과제 961-0905-0352) 지원으로 수행되었습니다.

하지만, 현재의 인터넷 프로토콜은 인터넷에 접속된 호스트의 접속점(attachment)은 고정되어 있고, 호스트의 IP 주소로 호스트가 접속되어 있는 망을 식별한다고 가정하고 있다. 즉, 호스트가 자신의 IP 주소 변경 없이 새로운 망으로 이동한다면, 호스트의 주소는 자신의 새로운 접속점을 나타내지 못하게 된다. 따라서, 이동 노드는 자신의 새로운 위치를 나타내는 IP 주소로 재설정 되어야 하지만, IP 주소를 변경 하게 되면 이미 설정된 트랜스포트 계층의 연결을 잃어버리게 되는 일이 발생한다. 그래서, IETF에서는 네트워크 계층에서 이동성을 지원하는 Mobile IP 프로토콜을 제안하였다. 위치정보 관리 측면에서 보면, 인터넷에서 호스트의 이동성을 지원하기 위한 Mobile IP 프로토콜은 홈 에이전트(home agent)와 외부 에이전트(foreign agent)로 구성된 두 계층 위치정보 서버 구조를 가지고 있으며, 호스트가 다른 기지국으로 이동할 때마다 위치정보를 등록하는 방식을 사용하고 있다.

본 논문에서는 기존의 IP 헤더에 동일한 구조를 가지는 IP 헤더를 추가하는 방법을 채택하여 IP-within-IP 프로토콜을 커널 레벨로 구현하였다. 이러한 기술은 기존의 소프트웨어와 하드웨어를 개선하지 않고 그대로 사용할 수 있고, 중간 라우터에 투명하다는 장점을 가지고 있다. 그리고, 텍스트 파일 전송 프로그램을 통해서 IP-within-IP 프로토콜이 성공적으로 IP 데이터그램을 캡슐화하여 목적지로 올바르게 전송하며, 터널링 기능을 정상적으로 수행하는 것을 확인 하였다.

본 논문의 구성은 2장에서는 터널링에 대한 전반적인 설명과 더불어 캡슐화와 캡슐화된 데이터의 이동노드로의 전달에 대해 서술하였고, 3장에서는 구현된 프로토콜의 구현구조와 송수신 처리과정을 다루었다. 4장에서는 구현 환경과 성능측정을 다루었고, 마지막으로 5장에서는 결론 및 향후방향에 대해서 나타내었다.

Ⅱ. 터널링(Tunneling)

2.1 캡슐화 방법을 사용한 터널링

호스트의 IP 주소는 네트워크를 지정하는 부분(네트워크 ID)과 호스트 컴퓨터를 지정하는 부분(호스트 ID)으로 나뉘어 표시된다(그림 1). 이러한 주소 형식을 갖는 시스템에서 단말로의 통신은 그 단말 주소의 네트워크 ID에 근거하여 라우팅 도메인의 라우터에게 전달된 후, 해당 라우팅 도메인의 라우터가 호스트 ID에 근거하여 호스트 컴퓨터에게 데이터그램을 전달함으로써 이루어 진다[16][17].

그림 1. IP 주소 형식
Fig. 1 IP Address Format

그러나, 이동 노드가 다른 네트워크로 이동을 할 경우, 위와 같은 IP 주소 체계와 경로설정 방법으로는 이동한 노드에게 데이터 전송이 이루어질 수 없다. 이러한 문제점을 해결하고자 여러가지 캡슐화 방법이 제안되고 있다. 캡슐화 방법은 캡슐화 구조에 따라서 IP-within-IP 캡슐화와 Generic Routing Encapsulation (GRE), 그리고 Minimal Encapsulation의 세 가지 형태가 존재한다. 하지만, Generic Routing Encapsulation 방법과 Minimal Encapsulation 방법은 추가되는 헤더의 길이가 작은 반면(8 또는 12바이트)에 기존의 IP 헤더를 변화 시키거나, IP 패킷이 분리되는 단점이 있어서 본 논문에서는 IPv4를 사용하면서 중간 라우터들에게 투명한 IP-within-IP 캡슐화 방법을 사용하여 구현하였다.

2.1.1 IP-within-IP 캡슐화

IP-within-IP 캡슐화는 그림2에서 보듯이 데이터그램의 기존 IP 헤더 앞에 Outer IP 헤더를 추가하는 방법이다[2].

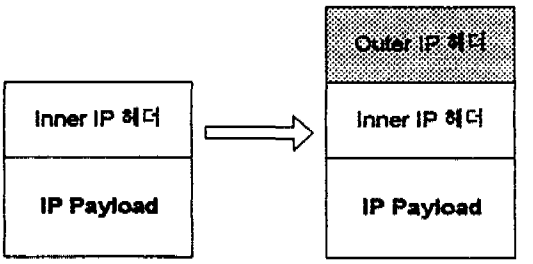


그림 2. IP within IP 패킷 포맷
Fig. 2 IP within IP Packet Format

Outer IP 헤더의 근원지 주소(source address)와 목적지 주소(destination address)는 터널의 종단점을 나타내고, Inner IP 헤더의 근원지 주소와 목적지 주소는 데이터그램의 원래 송신자와 수신자를 나타낸다. Outer IP 헤더는 Inner IP 헤더와 동일한 헤더 구조(그림 3)를 가지지만 각각의 필드에 Inner IP 헤더와는 다른 설정 값을 가지게 된다.

Outer IP헤더에서 변경되어 유지되는 몇 가지 필드를 살펴보면, 전체 길이를 나타내는 ip_len은 Outer IP 헤더와 Inner IP 헤더, 그리고 데이터를 포함하는 캡슐화 된 IP 데이터그램의 전체 길이를 나타낸다. 터널의 종점에 도달한 데이터그램이 캡슐화 된 것인지를 판단하는 기준으로 사용되는 ip_p는 4의 값을 가지게 된다[4].

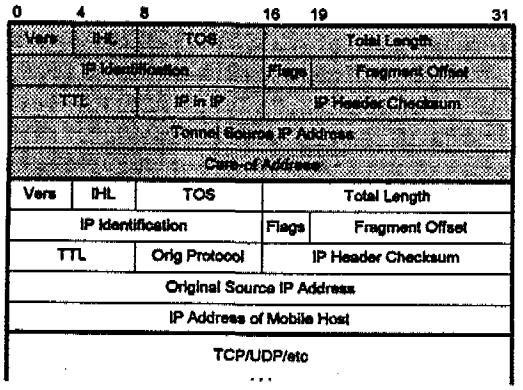


그림 3. Outer IP 헤더 구조
Fig. 3 Structure of Outer IP Header

위와 같은 캡슐화 기법은 다양한 목적(eg. 멀티캐스팅, 다중 프로토콜 운용)으로 사용될 수 있으나, 본 논문에서는 이동 서비스를 지원하기 위한 관점에서 프로토콜 드라이버 형태로 구현하였다[1].

2.2 Mobile IP에서의 동작

그림 4에서 HA는 홈 에이전트이고, FA는 외부 에이전트이다. 만약 이동 노드(Mobile Node, MN)가 홈 네트워크를 벗어난 경우에는 현재 위치한 네트워크에서 할당 받은 주소를 홈 에이전트에게 등록하게 된다. 등록은 홈 에이전트에 직접 등록하거나, 현재 위치한 네트워크의 외부 에이전트를 통해 등록하게 된다. 홈 에이전트는 다른 노드(Correspondent Node, CN)가 이동 노드에게 전송하는 모든 데이터그램을 수신하여, 이동 노드가 자신의 네트워크에 위치하지 않을 경우에는 이동 노드의 등록 정보를 살펴본 후 현재 이동 노드가 위치한 네트워크로 데이터그램을 터널링(tunneling)하게 된다. 위에서 설명한 것과 같이 홈 에이전트는 원래 홈 네트워크에 위치해 있던 노드가 다른 네트워크로 이동했을 때, 그러한 이동 노드에게 패킷을 터널링 해 주기 위해서 이동 노드에 대한 상태 정보를 항상 유지하고 있어야 한다. 이렇듯 홈 에이전트가 유지하는 이동 노드의 상태 정보를 홈 리스트(home list)라고 한다. 외부 에이전트도 홈 에이전트처럼 상태정보를 테이블 형태로 유지하게 된다. 외부 에이전트가 유지하게 되는 테이블은 자신의 네트워크 주소를 갖지 않는 노드들이 자신의 네트워크로 이동을 해 오면, 등록 절차를 통해 이동 노드들의 홈 주소와 의탁주소(care of address)를 알게 된다. 이러한 이동 노드들의 상태 정보를 유지하는 테이블을 방문자 리스트(visitor list)라고 한다. 이동 노드의 상태정보 리스트(Home List, Visitor List)는 이동 호스트의 이전 위치에 도달한 패킷을 새로운 위치로 전달하기 위한 전달점(forwarding pointer)을 제공한다.

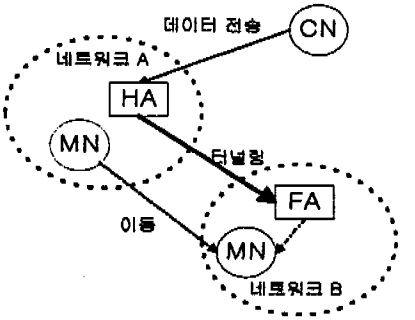


그림 4. 이동 노드로의 데이터 전달
Fig. 4. Deliver data to mobile node

Ⅲ. IP-within-IP 프로토콜의 구현

본 연구에서는 터널링 구현에서 변화를 요구하지 않는 트랜스포트 계층(eg, TCP, UDP, etc)은 고려하지 않았으며, 인터넷상에서 데이터그램 전달을 위한 IPv4와 인캡슐레이션 기능을 Windows NT 커널 내부에 프로토콜로 구현하였다[13][14]. IP-within-IP 프로토콜을 커널 모드로 작성하였는데, 이를 통하여 다음과 같은 장점을 가지게 된다.

- 커널 모드의 IP-within-IP 프로토콜은 사용자 모드의 프로그램보다 자원 할당면에서 높은 우선 순위를 가지게 된다.
- 메모리에 대한 접근을 가상 주소로 하지 않고 실제 주소로 함으로써 메모리에 대한 접근 시간을 줄일 수 있다.
- 터널링은 IPv4 의 동작을 제어할 수 있어야 하는데, 그와 같은 기능은 커널 모드에서만 구현될 수 있다.

본 논문에서 구현한 IP-within-IP 프로토콜은 이

동노드가 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 5)의 회색 부분과 같다.

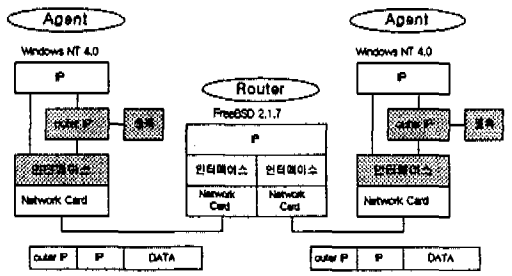


그림 5. IP-within-IP 의 구현 범위
Fig. 5 Scope of Implementation

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 드라이버 개발 시 NT시스템에서 요구하는 루틴(WIN32 API, Dispatches Routine, etc)들을 작성하였다.

3.1 IP-within-IP 프로토콜의 구현 구조

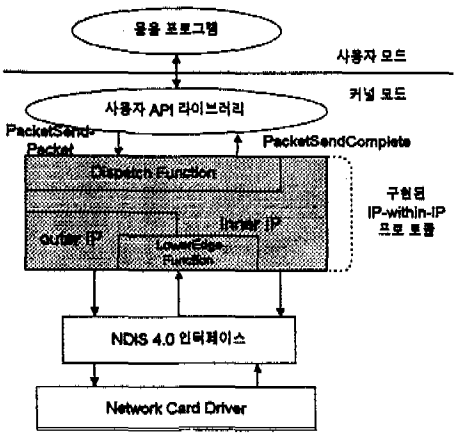


그림 6. IP-within-IP 프로토콜의 구현구조
Fig. 6 Structure of IP-within-IP Protocol

IP-within-IP 프로토콜의 구현 구조는 (그림 6)과 같으며, 각 모듈의 기능은 다음과 같다.

- 사용자 API 라이브러리 : 상위 사용자의 명령(Command)에 의해서 호출된 내부 함수를 수행하여 사용자에게 서비스를 제공한다[6][10].
- 디스패치 함수 : 사용자의 호출이 있을 때 미리 지정된 동작을 수행하는 함수이다. 구현된 IP-within-IP 프로토콜을 사용자 계층에서 사용할 수 있도록 라이브러리 형태로 구현 하였다.
- Inner IP : 기존의 IP헤더를 처리하는 기능을 수행한다. IP 헤더 처리 기능은 FreeBSD 2.1.7의 IP 처리 모듈 소스 코드를 사용하여 구현하였으며, 구현 범위에서 패킷 분할과 재조합 처리는 제외하였다.
- Outer IP : 인캡슐레이션된 IP 헤더를 처리하는 기능을 수행한다. Inner IP 헤더 처리 모듈과 유사한 형태로 구현하였다.
- Lower-Edge 함수 : 하부 네트카드의 서비스가 필요하다면 NDIS를 통해서 네트워크카드 드라이버를 구동시킨다[6][10]. 이벤트 관리 큐와 출력 큐를 생성하여 네트워크 카드의 동작을 제어 하도록 구현하였다.

3.2 IP-within-IP 프로토콜의 등록 과정

3.2.1 디스패취 함수

Windows NT에서는 일관된 형태의 I/O를 수행하기 위하여 모든 I/O 요구들을 IRP(I/O Request Packet)형태로 처리한다. 본 구현에서도 IP-within-IP 프로토콜을 커널에 위치 시키기 위해, IRP를 사용하였다. 즉 사용자가 Windows NT 시스템 서비스를 호출하면 커널 내부의 I/O 매니저가 사용자의 요청을 IRP 형태로 구성하여 IP-within-IP 프로토콜에게 전달한다[10]. I/O 매니저가 사용자의 요구에 따라 I/O 동작을 수행하도록 하기 위해서 프로토콜 초기화 단계에서 미리 I/O 매니저에 의하여 호출될 디스패취 함수들을 등록하여야 한다. 구현된 디스패치 함수의 종류 및 기능은 표 1과 같다.

표 1. IP-within-IP 프로토콜의 디스패취 함수
Table 1. Dispatch Functions of IP-within-IP Protocol

Dispatch함수	기 능
IPIPClose	주소 객체의 등록 해제.
IPIPCreate	주소 객체 등록.
IPIPIoControl	요청에 대한 생성 및 제거.
IPIPWrite	데이터 전송 처리.
IPIPRead	데이터 수신 처리.
IPIPUnload	드라이버의 등록 해제.

사용자는 CreateFile 함수를 호출하고, CreateFile

함수는 자신의 주소 객체를 등록하기 위해서 내부에 정의된 오픈 함수(IPIPCreate 함수)를 호출한다. 호출된 오픈 함수(IPIPCreate 함수)는 사용자 주소 객체를 지정하고 필요한 초기화 동작을 수행한다. 사용자가 수행한 CretateFile이 성공적으로 완료되었으면 그 후 주소 객체 등록과 해제를 제외한 모든 요청들은 IOCTL 메시지로 IPIPIoControl 함수를 호출함으로써 수행된다.

사용자가 IP-within-IP 프로토콜의 연결을 해제하고자 경우에는 CloseFile 함수를 호출하여 자신의 주소 객체를 해제 시킨다. 해제 동작은 I/O 매니저에 의해 프로토콜의 해당 함수가 호출되고 이 함수는 다시 내부에 정의된 함수(IPIPClose 함수)를 호출한다.

3.2.2 Lower-Edge 함수

Lower-Edge 함수는 NDIS를 통해 네트워크 접속 카드로부터 패킷을 수신하였을 경우나, NDIS에 요청한 송신의 비 동기적 완료를 통보해 주는 경우와 같이 NDIS를 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. 구현한 Lower-Edge 함수의 종류 및 기능은 표 2와 같다. 드라이버가 초기화 될 때 Lower-Edge 함수들은 NDIS 인터페이스에 등록되어 커널 모드로 동작할 수 있게 된다.

표 2. Lower-Edge 함수
Table 2. Lower-Edge Functions

Lower-Edge 함수	기 능
IPIPAdapter Complete	하부 망 접속 카드가 준비되었음을 비 동기적으로 I/O 매니저에게 알림.
IPIPCloseAdapter Complete	하부 망 접속 카드에 대한 사용자의 서비스 종료 요청이 완료되었음을 비 동기적으로 입출력 관리자에게 알림.
IPIPSend Complete	데이터 송신이 종료되었음을 비 동기적으로 입출력 서비스에 알림.
IPIPTransferData Complete	데이터 수신이 종료되었음을 비 동기적으로 입출력 서비스에 알림.
IPIPReceive Indicate	데이터가 망 접속 인터페이스를 통해 도착했음을 비 동기적으로 드라이버에게 통고 함.
IPIPRequest Complete0	사용자의 요청(주소 지정 및 질의, 패킷필터링)이 종료되었음을 동기적, 또는 비 동기적으로 입출력 서비스에 알림.

3.3 IP-within-IP프로토콜의 동작

IP-within-IP 프로토콜은 하부 네트카드 드라이버로부터 PDU(Protocol Data Unit)를 수신하는 동작과 PDU를 송신하는 동작, 그리고 상위 사용자로부터 SDU(Service Data Unit)를 수신하는 동작과 사용자에게 SDU를 전달하는 동작, 그리고 수신된 PDU를 처리하는 동작으로 구성된다. 각각의 수행 과정을 다음에 나타내었다.

3.3.1 SDU 송수신 처리

구현된 Win32 API는 사용자 모드로부터 데이터를 커널 모드로 복사하기 위한 모듈과 수신된 데이터를 커널 모드로부터 사용자 모드로 복사하기 위한 모듈로 구성된다. 상위 사용자로부터 데이터 전송 명령이 발생하면, Win32 API는 무엇을 사용자 버퍼로부터 프로토콜 로 일정한 크기(최대 14000바이트)이하의 메시지 사이즈로 분할하여 전달한다. 이런 제약은 구현된 IP-within-IP 프로토콜에는 트랜스포트 계층의 프로토콜 동작이 존재하지 않기 때문에 네트워크 계층에서 분할할 수 있는 데이터크기를 하고 있기 때문에 발생한다.

그림 7은 사용자가 송신할 데이터를 IP-within-IP로 전달하는 동작을 보여주는 흐름도이다.

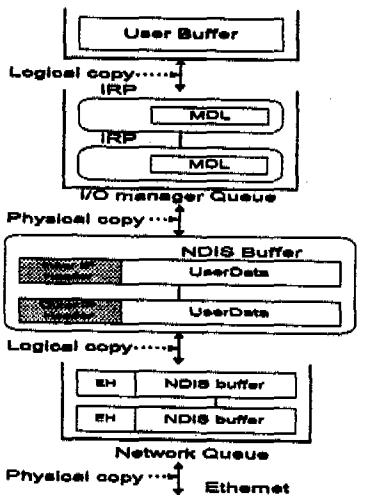


그림 7. 데이터의 메모리 복사
Fig. 7 Memory Copy of Data

상위 사용자로부터 데이터 수신 요청이 왔을 경우에는 전달(Delivery) 이벤트가 생성된다. 데이터 수신 요청에는 사용자 버퍼를 가리키는 포인터가 포함되어 있어 IP-within-IP드라이버는 이 영역으로

데이터를 전달한다.

3.3.2 PDU 송수신 처리

구현된 프로토콜은 크게 세가지 처리 모듈을 가진다. 하나는 Inner IP 헤더를 처리하는 모듈, 또 다른 하나는 Outer IP 헤더를 처리하는 모듈, 그리고 마지막으로 하부 통신망 접속 카드의 이벤트를 처리하는 모듈이다. 그림 8은 하부 네트워크 카드 드라이버로부터 PDU를 수신하는 동작을 세가지 모듈과 각각에서 데이터가 처리되는 과정으로 나타내고 있다.

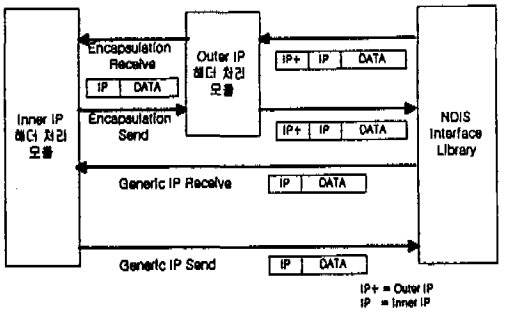


그림 8. IP-within-IP 처리 모듈
Fig. 8 IP-within-IP process module

각 모듈의 기능을 살펴보면, Inner IP 헤더 처리 모듈은 기존의 IP 헤더를 처리하는 모듈이다. 따라서, 패킷이 올바르게 전달된 것인지를 확인하고, 만약 오류가 존재한다면 수신 패킷을 제거한다. 또한 패킷에 캡슐화가 필요한지 검사하기 위해서 이동노드의 상태정보를 유지하는 리스트(Home list)를 검색하게 된다. 두 번째로 Outer IP 헤더 처리 모듈은 이동노드 상태정보 리스트(Home list)에 등록되어 있는 외부 에이전트를 목적지로 하는 캡슐화된 패킷을 생성하게 된다. 패킷 수신시에는 패킷을 역캡슐화 시키는 기능을 수행한다. 세 번째로 하부 통신망 이벤트 처리 모듈은 패킷의 전송 및 수신 인터럽트의 처리를 담당하는 모듈이다.

윈도우의 한 객체가 데이터를 전송하고자 할 때, 각각의 모듈에서 처리되는 과정은 다음과 같다(그림 9).

(1) 기존의 IP 헤더를 처리하는 부분은 송신할 데이터에 Inner IP 헤더를 추가하고 필요한 IP 헤더 필드를 처리한다. 만약 송신할 메시지가 네트워크 카드의 MTU(최대 전송 단위, e.g. 1500)보다 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 IP헤더 처리 모듈은 캡슐화 방법으로 IP-within-IP 인캡슐레이션을 사용한다. 따라서 Inner IP 데이터처리 모듈에서 넘겨진 패킷에 홈 리스트의 외부 에이전트 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.

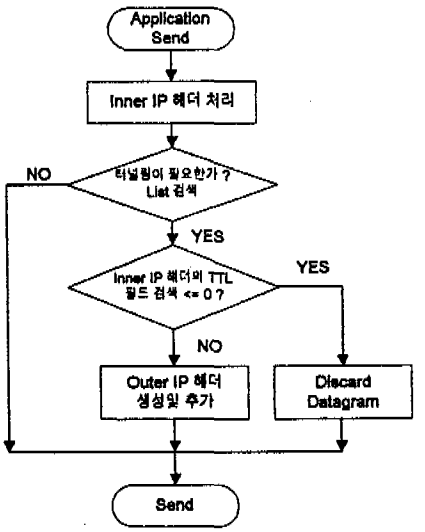


그림 9. IP-within-IP 데이터그램 전송과정
Fig. 9 Transmission process of IP-within-IP

하위의 네트워크 카드가 데이터그램을 수신했을 때 각각의 처리 모듈이 수행되는 과정은 다음과 같다(그림 10).

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

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

(3) 수신된 데이터가 캡슐화 된 것이라면, 이동 노드의 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 IP헤더가 제거된 패킷은 Inner IP 헤더 처리를 위해 Inner IP 헤더 처리 모듈에게 넘겨지고, 데이터그램 조합 처리와 헤더 처리 작업이 수행된다.

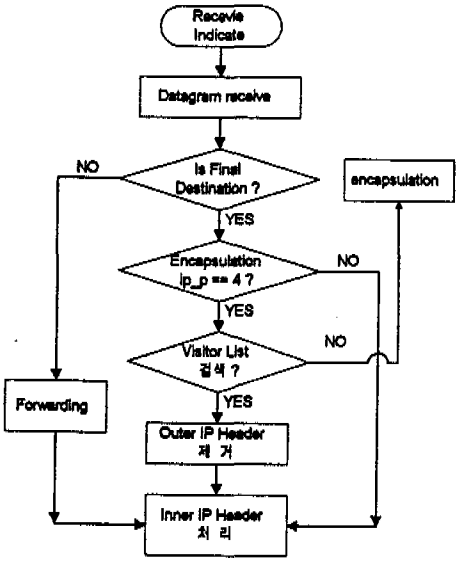


그림 10. 수신 데이터그램 처리과정
Fig. 10 Process of receive datagram

3.4 구현된 IP-within-IP 프로토콜의 동작

3절에서 설명한 함수들의 기능을 송·수신 동작으로 나누어 나타내면 다음과 같다.

가) 데이터 송신 동작 : 상위 사용자의 데이터 송신 요청 시(PacketSendPacket), IP-within-IP 프로토콜은 전달 받은 데이터 메시지를 프로토콜 동작에 의해서 IP-within-IP PDU 형태로 재구성(IPIPWrite)하게 되며, 이들 재구성된 PDU들은 Lower-Edge 함수(IPIPSend)를 이용하여 네트워크 카드 드라이버에게 전달된다. 하위 네트워크 카드 드라이버 역시 자신의 내부 함수를 이용하여 송신 동작의 상태나 수행 결과(IPIPSendComplete)를 IP-within-IP 프로토콜에게 통보하게 된다.

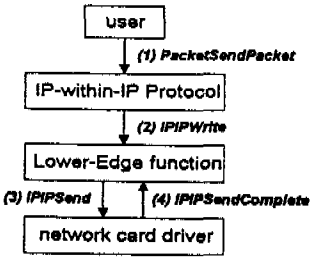


그림 11. 데이터 송신 동작
Fig. 11 Data Sending Operation

나) 데이터 수신 동작 : 네트워크 카드 드라이버가 상대 호스트로부터 패킷들을 수신하였을 경우 네트워크 카드 드라이버들은 패킷 수신을 NDIS 인터페이스 라이브러리를 이용하여 IP-within-IP 프로토콜에게 통보(IPIPReceiveIndicate)하고, IP-within-IP 드라이버 역시 NDIS 인터페이스를 사용하여 수신된 데이터들을 전달받게 된다(IPIPTransferDataComplete).

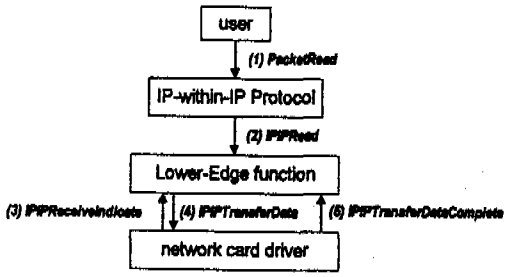


그림 12. 데이터 수신 동작
Fig. 12 Data Receiving Operation

Ⅳ. IP-within-IP 프로토콜의 테스트 환경 및 성능 평가

4.1 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 있

는 FreeBSD 2.1.7의 PC 1대를 사용하였다. 디버깅 프로그램으로는 Microsoft사에서 제공하는SDK (Software Development Kit)의 WinDbg를 사용하였고, 디버깅 함수는 DDK(Device Driver Kit)에서 제공되는 함수를 사용하였다.

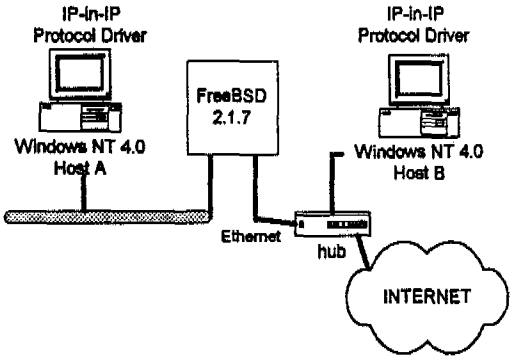


그림 13. 테스트 환경
Fig. 13 Test Environment

표 3 .모니터링 결과
Table 3. Monitoring Result

Ip :
1 : 전체 2185개의 패킷을 수신함
2 : bad header checksums은 0개 발생함
3 : 최소 크기보다 작은 사이즈를 갖는 것은 0개 존재함
4 : (데이터 사이즈 < 데이터 길이)인 것은 0개 존재함
5 : (헤더 길이 < 데이터 사이즈) 인 것은 0개 존재함
6 : (데이터 길이 < 헤더 길이) 인 것은 0개 존재함
7 : bad 옵션을 갖는 것은 0개 존재함
8 : 잘못된 버전 번호를 갖는 것은 0개 존재함
9 : 수신된 fragments는 0개 존재함
10 : 버려진 fragments는 0개 존재함(drop or out of space)
11 : 타임아웃이후에 버려진fragments는 0개 존재함
12 : reassembled ok인 패킷은 0개 존재함
13 : 이 호스트에 대한 패킷은 2117개 존재함
14 : 알 수 없고/지원하지 않는 프로토콜을 위한 패킷은 0개 존재함
15 : 40 개의 패킷이 포워드됨
16 : forwardable하지 않는 패킷은 0개 존재함
17 : 40개의 패킷이 redirect되어 송신됨
18 : 이 호스트로부터 40 개의 패킷이 송신됨
19 : 합성된 ip 헤더를 갖는 패킷은 0개 송신됨
20 : 버퍼가 없어서 버려진 output 패킷은 0개 존재함
21 : 경로설정이 되지 않아서 버려진 패킷은 0개 존재함
22 : fragmented된 output 데이터그램은 0개 존재함
23 : 생성된 fragments는 0개 존재함
24 : fragmented될 수 없는 데이터그램은 0개 존재함

테스트 환경은 그림 13과 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.

호스트 A에서 전송한 40개의 패킷이 올바르게 터널링 되었는지를 알아보기 위해서 Free BSD가 설치된 컴퓨터를 모니터링한 결과를 표 3에 나타내었다.

1번째 라인을 통해 라우터 컴퓨터에 전체 2185개의 패킷이 도착했음을 알 수 있다. 그리고, 2~12 번째 라인을 통해서 라우터에 수신된 패킷이 기존의 패킷처럼 인식되어 처리됨을 볼 수 있다. 15번째 라인과 17~18 번째 라인을 통해서 40개의 패킷이 아무 문제없이 포워드 되는 것을 알 수가 있었다.

표 3에서 알 수 있듯이 라우터가 40개의 패킷을 수신하여, 기존의 IP 패킷과 동일한 방식으로 포워딩(forwarding) 시킨다는 것을 볼 수 있다. 즉, 다양한 텍스트 파일 전송을 통해 IP-within-IP 프로토콜이 성공적으로 IP 데이터그램을 캡슐화하여 목적지로 올바르게 전달하는 것을 확인할 수 있었다.

4.2 성능 분석

IP-within-IP 프로토콜은 이벤트 구동 방식으로 구현되었으므로 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 PacketSend를 호출한 이후부터 PacketSend Complete 이벤트가 수신된 시점 사이의 차이이다(그림 5).

수치적 분석을 위해 사용된 기호들은 다음과 같다.

T_{op} - 기존 IP의 전체 패킷 전송 처리시간 (sec)

T_{ip} - 터널링된 IP 전체 패킷 전송 처리시간 (sec)

M_t - 전체 전송 패킷의 크기 (bit)

I_p - Innner IP 헤더 처리시간 (예, 분할 및 재조합 처리 시간, sec)

O_p - Outer IP 헤더 처리시간 (예, 검사합, 상태정보 리스트 검색 처리 시간, 헤더 생성 처

리 시간)

C_m - 네트워크 드라이버의 실제 전송 시간 (프로토콜로부터 네트워크 드라이버의 메모리 복사)

F_p - 데이터그램을 응용계층으로부터 프로토콜로 복사하는데 소요된 시간(sec)

Λ_{op} - 기존의 IP프로토콜에 대한 패킷 처리율 (bps)

Λ_{tp} - 터널링된 IP 프로토콜에 대한 패킷 처리율

기존 IPv4의 송신 패킷 처리율과 전체 패킷처리 시간은 식(1)과 식(2)와 같다.

$$\Lambda_{op} = M_r / T_{op} \tag{1}$$

$$T_{op} = I_p + F_p + C_m \tag{2}$$

IP-within-IP 프로토콜의 송신 패킷 처리율(식 (3))과 전체 패킷 처리 시간(식 (4))은 다음과 같다.

$$\Lambda_{tp} = M_r / T_{tp} \tag{3}$$

$$T_{tp} = O_p + I_p + F_p + C_m \tag{4}$$

수치학적 해석을 위해서 사용된 실측값은 C_m, T_{op}, T_{tp}로써, C_m은 데이터 전송을 네트워크 카드에 요청하는 NDIS 인터페이스 함수 즉, NdisSend를 수행하는데 소요되는 시간을 평균한 결과이다(표 5).

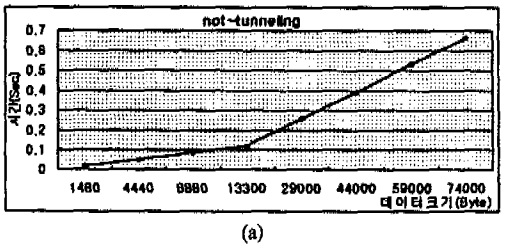
표 4. 네트워크 카드 드라이버의 송신 시간
Table 4. Sending time of Network Card Driver

Data Size	Time(sec) C_m	Throughput (Mbps),
13300 byte	0.0181	5.606
29000 byte	0.0391	5.659
44000 byte	0.0593	5.661
59000 byte	0.0798	5.641
74000 byte	0.0998	5.657

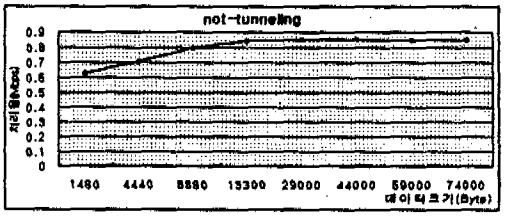
표 4를 사용하여 C_m의 평균 값을 계산한 결과 약 5.65Mbps를 얻을 수 있었다.

응용계층에서 터널링을 이용하여 데이터 송신 시간(T_{tp})를 측정하고 이에 따라 계산된 데이터 처리율 (Λ_{tp})을 터널링 되지 않았을 경우의 데이터 처리율 (Λ_{op})과 비교하여 나타내면 그림 14와 그림 15와 같다.

그림 15의 (b)에서 주목할 만한 사항은 일정한 크기 이하의 데이터를 전송할 경우에 발생하는 처리율 감소이다. 이것은 이더넷 MTU(Maximum Transmission Unit)가 1514바이트로 제한됨으로써 발생하는 문제이다. 1480바이트의 데이터를 보내고자 할 때, 기존의 IPv4에서는 Inner IP 헤더 20바이트를 추가하여 전송하면 된다. 그러나, IP-withinIP 프로토콜은 Inner IP 헤더 20바이트 외에 Outer IP 20바이트를 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(약14800 byte 이하), 하나의 패킷을 추가적으로 전송 하는 것은 큰 오버헤드로 작용하지만, 대량의 데이터 전송에서는 비교적 작은 오버헤드로 작용한다는 것을 알 수 있다.



(a)



(b)

그림 14. 터널링 되지 않는 패킷의 처리시간(a)과 처리율(b)
Fig. 14 Throughput time (a) and Throughput rate (b) of Non-Tunneling Packet

데이터 송·수신 이벤트 처리로 인한 프로토콜의 기능에 영향을 주는 요인, 즉 메모리 복사 처리율 (F_P)은 약 78.6Mbps를 나타내었다.

측정된 값들(C_m, T_{tp}, T_{op}, F_p) 과 식 (2)와 식 (3)을 이용하면, 기존 IP 헤더 처리시간()과 Outer IP() 헤더 처리시간을 계산할 수 있다.

$$I_p = T_{op} - (F_p + C_m) \tag{5}$$

$$O_p = T_{tp} - (I_p + F_p + C_m) \quad (6)$$

식 (5)와 식 (6)을 통해 얻어진 O_p를 사용하여 IP-within-IP 프로토콜의 오버헤드(O_p/T_{op})를 산출한 결과 약 0.01% 이하의 값을 얻을 수 있었다. 이러한 결과를 바탕으로 구현한 IP-within-IP 프로토콜은 기존의 IP 전송 방법과 비교할 때, 무시할 수 있을 정도의 작은 오버헤드를 갖게 되므로 Mobile IP 프로토콜에 적용하는데 문제가 없다는 결론에 도달하였다.

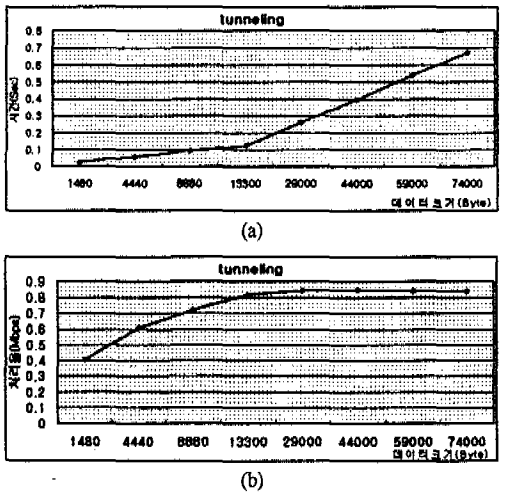


그림 15. 터널링된 패킷의 처리시간(a)과 처리율(b)
Fig. 15 Throughput time (a) and Throughput rate (b) of Tunneling Packet

V. 결론 및 향후 과제

본 연구에서는 Inner IP와 Outer IP 라는 2단계 주소 체계를 갖는 주소 변환 방법을 사용해서 이동 서비스를 실현하도록 하였다. 이 방법을 사용함으로써 발생하는 문제점은 다른 캡슐화 방법보다 좀더 큰 오버 헤드를 가진다는 점과 데이터그램을 디캡슐레이션 할 수 있는 터널의 종점이 없다면 사용될 수 없다는 것이다. 이와 비교할 때 장점은 기존의 IP 헤더를 변화 없이 사용함으로써 비교적 처리가 간단하고, 기존의 IP 헤더와 호환성이 높아 이동한 노드에 데이터 전달이 원활히 이루어 질 수가 있다는 것이다. 또한 중간 노드들은 기존의 IP 데이터그램과 같은 방식으로 처리함으로써 투명성이 보장된다는 것이다.

본 연구를 통해 개발된 구현 결과는 Windows NT 의 커널상에서 프로토콜 구현기술 확보 측면과 기존 네트워크의 S/W와 H/W를 수정하지 않고도 이동 통신 서비스를 제공할 수 있다는 점에서 의미있는 성과를 거두었다고 하겠다.

본 논문에서는 라우팅 메커니즘만을 고려하였으며, ICMP 메시지를 사용한 agent discovery와 등록 절차는 차후 연구 과제로 남아있다. 이번 연구에서는 터널링 기법을 통한 라우팅 메커니즘을 일반 호스트에서 다루었으나, 다음 단계에서는 라우터에서 이 기능을 시험하고, 마지막으로 Mobile IP의 완전한 기능(agent discovery, 등록)을 구현하기 위한 연구가 요구된다. 또한 이번 구현 범위에서 향후 고려사항으로 남겨 둔 ICMP 처리 문제와 보안 문제, 그리고 IP 프로토콜의 주소체계를 확장 시킨 경로 설정의 최적화에 관한 연구가 필요할 것으로 생각된다.

참 고 문 헌

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

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

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

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

[5] Postel, J., Editor, "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] AL STEVENS, "Memory-Resident Utilities, Screen I/O AND Programming Techniques", 세화, pp. 38-121, 1991.

정 진 우(Jin-Woo Jung) 정회원
1998년 2월 : 고려대학교 정보공학과 학사
1998년 3월~현재 : 고려대학교 전자정보공학과 석사과정
<관심분야> : 고속통신 프로토콜 설계 및 구현, 인터넷 트랜스포트 프로토콜, 인터넷 이동 통신 프로토콜, 멀티캐스트 프로토콜

천 정 훈(Jung-Hoon Cheon) 정회원
1996년 2월 : 고려대학교 정보공학과 학사
1998년 2월 : 고려대학교 정보공학과 석사
1998년 3월~현재 : 고려대학교 전자정보공학과 박사과정
<관심분야> : 고속통신 프로토콜 설계 및 구현, 인터넷 트랜스포트 프로토콜, 인터넷 이동 통신 프로토콜, 초고속 통신망

강 현 국(Hyun-Kook Kahng) 정회원
1982년 : 고려대학교 전자공학과 학사
1984년 12월 : 미국 미시간 대학교 컴퓨터공학 석사
1990년 6월 : 미국 조지아공과대학교 컴퓨터 통신공학 박사
1991년 7월~1994년 2월 : 한국전자통신원구원, 정보통신표준연구센타, 선임연구원
1993년 3월~현재 : ITU-T SG7 한국대표
1993년 3월~현재 : ISO/IEC SC6 한국대표
1994년 3월~현재 : 고려대학교 전자 및 정보공학부 부교수
1996년 5월~현재 : 개방형컴퓨터통신연구회 Internet-KIG TCP/IP WG 위원장
<관심분야> : 고속통신 프로토콜 설계 및 구현, 인터넷 이동 통신 프로토콜, 인터넷 스위칭 기술, 인터넷 트랜스포트 프로토콜 등