하드웨어 기반 모바일 IPv6의 구현

준회원 김 혜 란*, 문 주 형**, 김 원 정***, 추 하 늘**, 지 서 영**, 종신회원 임 혜 숙**

Hardware-Based Mobile IPv6 Implementation

Hyeran Kim*, Ju Hyoung Mun**, Won jung Kim***, Ha Neul Chu**,
Suh Young Jhee** *Associate Members*, Hyesook Lim** *Lifelong Member*

요 약

모바일 IP는 휴대 단말이 새로운 링크로 이동 중에서도 통신을 지속할 수 있도록 하는 기능을 지원한다. 모바일 IPv6는 인터넷 프로토콜 버전 6에 기초하여 모빌리티 기능을 지원하는 프로토콜로서, 기존의 모바일 IPv4의 기능에 경로 최적화등의 새로운 기능이 추가되었다. 본 논문에서는 단말에 모빌리티를 제공하기 위한 기능들을 하드웨어로 처리하는 하드웨어 기반의 모바일 IPv6 구현에 관하여 다룬다. 하드웨어 기반의 모바일 IPv6는 기존의 소프트웨어에 기반한 방식에 비하여, 신속한 이동성을 지원하며, 이동 중에 발생할 수 있는 패킷손실을 최소화 하는 장점이 있다. 또한 CPU가 모빌리티 기능지원 부담에서 벗어나 응용프로그램의 수행에 전념할 수 있도록 함으로서, 전체적인 시스템 성능의 향상을 기대할 수 있다.

Key Words : Hardware implementation, Mobility, IPv6, Route optimization, binding

ABSTRACT

Mobile IP allows mobile end-systems to maintain on-going connections while moving to other links. Based on the Internet Protocol Version 6 (IPv6), mobile IPv6 provides a set of new mobility functions such as route optimization in addition to the functions in mobile IPv4. This paper describes the hardware-based mobile IPv6 implementation which provides all the mobility functionalities in hardware. The hardware-based mobile IPv6 provides faster mobility support than software-based implementation as well as it reduces the number of packet losses which can be caused during the movement. In end-systems equipped with hardware-based mobility support, the CPU can concentrate more on running application programs without wasting its effort for mobility support, and hence it is expected the overall performance improvement.

Ⅰ. 서 론

인터넷 사용자들은 언제 어디서나 인터넷에 접속하기를 바라고 있으며 휴대 전화나 휴대 단말기의 발전으로 그 수요 또한 크게 늘어나고 있다. 모바일 인터넷 프로토콜 (Mobile Internet Protocol, Mobile IP)은 이러한 휴대 노드들에게 고유의 IP 주소를 부여함과 동시에, 이동한 링크에서도 새로운 주소를 부여하여 이동 중에도 연결이 끊어지지 않고 통신을 지속할 수 있게 한다. 기존의 IPv4를 사용한 모바일 IP는 IPv4의 주소 공간 체계를 사용하기 때문에 주소 공간의 한계에 부딪히고 있고, 또한 삼각 라우팅 문제나 외부 에이전트(foreign agent)의 필요성에 따르는 단점이 있다. 모바일 IPv6는 IPv6의

※ 본 연구는 삼성전자 정보통신연구소의 지원으로 수행되었습니다.
* 삼성전자 정보통신총괄, ** 이화여자대학교 정보통신학과 SoC설계 연구실 (hlim@ewha.ac.kr), *** 넥스원퓨처
논문번호 : KICS2006-05-200, 접수일자 : 2006년 5월 8일, 최종논문접수일자 : 2006년 12월 21일

프로토콜을 기본으로 하고 있어, 이동한 링크에서 스스로 새로운 주소를 만들 수 있을 뿐만 아니라, 외부 에이전트를 필요로 하지 않는다. 따라서 모바일 기능을 지원하는 단말기는 링크의 지원에 관계없이 모바일 IPv6를 사용할 수 있게 된다. 또한 모바일 IPv6는 경로 최적화를 지원하여 경로에 따른 지연을 최소화 할 수 있다.

모바일 노드가 새로운 링크로 이동하였을 때, 신속히 이동을 감지하고, 새로운 바인딩을 성립시켜 사용자가 이동을 느끼지 않는 서비스를 지원하는 것은 앞으로의 모바일 IP 성패에 있어 매우 중요한 요인이라 할 것이다. 이는 이동을 감지하고, 새로운 바인딩을 성립시키기 위하여 교환, 처리되어야 하는 패킷들을 CPU의 간섭 없이 하드웨어가 만들어 내보내고 처리하도록 함으로서 가능한데, 이렇게 함으로서 링크의 전환에 보다 빨리 대응 할 수 있고 핸드오버로 인한 패킷손실을 최소화 시킬 수 있게 된다. 이 경우 CPU는 응용 프로그램에 더욱 전념 할 수 있어, 보다 나은 응용 프로그램 서비스를 가능하게 한다.

본 논문에서는 모바일 IPv6의 하드웨어 구현에 관하여 다룬다. 본 논문의 구성은 다음과 같다. 2장에서는 먼저 모바일 IPv6에 대해 살펴보고, 3장에서는 구현한 하드웨어의 기능을 자세히 설명한다. 4장에서는 여러가지 테스트 케이스를 사용하여 실험한 결과를 살펴보고, 5장에서 간단한 결론을 맺는다.

Ⅱ. Mobile IPv6

2.1 용어 정의

이동노드(mobile node, MN) 하나의 링크에서 다른 링크로 움직임이 가능한 노드를 말하는 것으로 홈주소를 이용해 다른 노드들과의 연결을 지속한다.

상대노드(correspondent node, CN) 이동노드와 통신을 하는 모든 노드를 말한다. 상대노드는 이동노드일 수도 있고 고정된 노드일 수도 있다.

홈주소(home address, HoA) 홈링크에서 이동노드에게 부여된 영구적인 주소로써 상대노드들은 홈주소를 이용해 이동노드와 통신 할 수 있다.

홈에이전트(home agent, HA) 홈링크에 있는 라우터로써 이동노드가 외부링크로 갔을 때 현재의 위치를 등록하는 라우터이다. 홈에이전트는 이 정보를 이용하여 이동노드로 들어오는 패킷을 현재 이동노드가 있는 위치로 전달한다.

외부링크(foreign link) 홈링크를 제외하고 이동노드가 머무는 모든 링크를 말한다.

획득주소(care-of-address, CoA) 획득주소는 이동노드가 외부링크로 갔을 때 그 곳에서 부여받는 일시적인 주소이다. 이동노드가 홈에이전트에게 현재 위치를 알릴 때 이 주소를 사용한다.

바인딩(binding) 바인딩은 유효시간 동안 이동노드의 홈주소와 획득주소를 연결하는 정보로써 상대노드는 이동노드의 위치가 바뀌어도 바인딩 되어진 홈주소를 이용해 계속적으로 통신을 유지한다. 이동노드가 새로운 획득주소를 얻거나 유효 시간이 지나도록 바인딩 정보를 유지하고 싶으면 바인딩을 업데이트 해야 한다.

바인딩캐시(binding cache, BC) 상대노드와 홈에이전트가 유지하는 테이블로서 받은 바인딩 정보를 저장한다. 바인딩 캐시에는 보낸 노드의 홈주소와 획득주소, 그리고 유효시간 등이 저장되어 있다. 이동노드에게 패킷을 보낼 때 홈주소를 이용해 이를 검색하여 획득주소를 얻게 되며, 그 주소를 패킷의 목적지 필드에 적어 이동노드의 현재 위치로 보낸다.

바인딩업데이트리스트(binding update list, BUL) 이동 노드가 유지하는 테이블로서 보낸 바인딩 정보를 저장한다. 바인딩 업데이트 리스트에는 바인딩을 보낸 노드의 주소와 이동노드의 획득주소, 유효시간 등이 저장되어 있고 유효시간이 다 되어가면 다시 업데이트 메시지를 보낼 수 있다. 이 리스트에 상대노드의 주소가 있을 경우 이동노드는 획득주소를 이용하여 홈에이전트를 거치지 않고 바로 상대노드에게 패킷을 보낼 수 있다.

2.2 모바일 IPv6

모바일 IPv6는 노드가 인터넷 상에서 이동하는 동안에도 연결을 유지한 채로 통신을 지속 할 수 있도록 하기 위하여 만들어졌다. 응용계층 프로그램은 변하지 않는 주소인 홈주소만을 보기 때문에 노드의 이동과 상관없이 상대노드와의 통신을 지속할 수 있다.

이동노드가 홈 링크에 있을 때는 그림 1의 (a)와 같이 일반적인 IPv6 노드로써 동작하며, 패킷은 홈주소를 사용한 기존의 IP 라우팅 방법을 통해 전달된다. 모바일 IPv6는 이동노드가 외부링크로 이동했을 때 적용된다. 이동노드가 그림 1의 (b)와 같이 외부링크로 이동하면 먼저 외부링크의 라우터에게 프리픽스를 받아 획득주소를 얻는다. 이동노드는 획

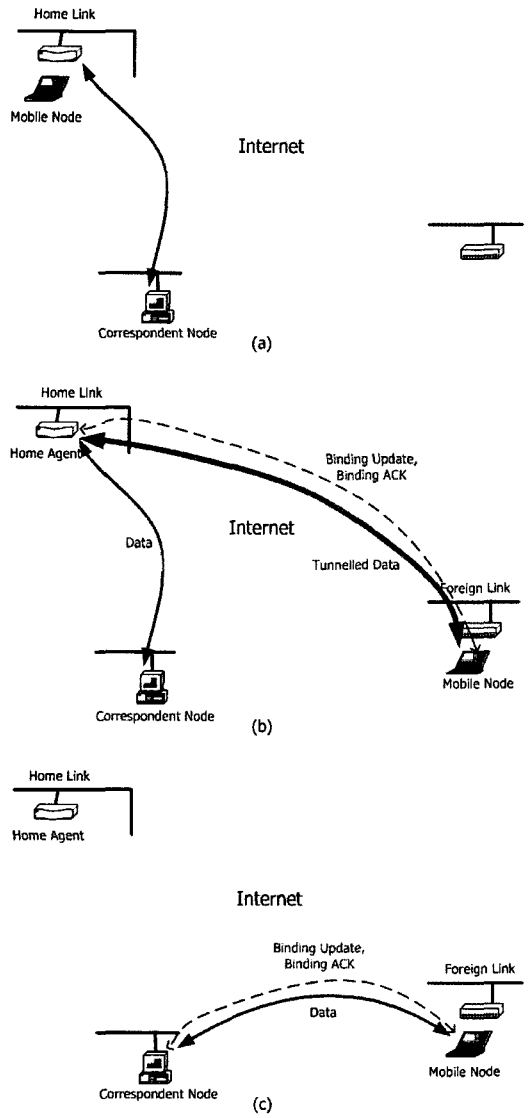


그림 1. 이동 노드의 동작

득주소 정보를 홈에이전트에게 알리기 위해 바인딩 업데이트 메시지를 보내게 된다. 홈에이전트는 바인딩업데이트 메시지를 받으면 그에 해당하는 바인딩 응답 메시지를 보내어 이동노드에게 바인딩이 성립하였음을 알린다. 이렇게 바인딩이 성립되고 나면 홈에이전트는 상대노드가 이동노드의 홈주소로 보내는 패킷을 대신 받아 이동노드의 획득주소를 사용해서 이동노드의 현재 위치로 보내준다. 또한 이동노드가 상대노드에게 패킷을 보낼 때에도 만들어진 패킷에 IP 헤더를 하나 덧씌워서 일단 홈에이전트에게로 보내면 홈에이전트는 이 패킷에 덧씌워진 IP 헤더를 벗겨서 상대노드에게 전달한다. 이렇게 함으

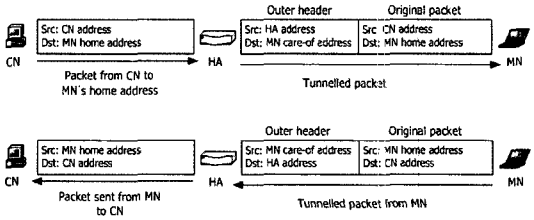


그림 2. 터널링 패킷

로써 이동노드는 상대노드와 통신을 지속 할 수 있다. 모바일 IPv6에서는 이러한 통신 방법외에 경로 최적화를 지원하여 상대노드와 직접 통신을 하는데, 그림 1의 (c)와 같다. 이동노드가 터널링을 통해 상대노드로부터 패킷을 받게 되면 이동노드는 상대노드의 주소를 목적지로 하는 바인딩업데이트 메시지를 보낸다. 상대노드가 이 메시지를 받으면 바인딩 응답 메시지를 보내어 바인딩이 성립하였음을 알린다. 이렇게 상대노드와 바인딩이 성립되고 나면 홈에이전트를 거치지 않고 직접 통신을 할 수 있게 된다.

그림 2에 그림 1(b)에서 설명한 홈에이전트를 통한 터널링 시의 패킷의 모양을 나타내었다. 그림에서 보여주는 바와 같이 홈에이전트는 이동노드와 상대노드의 패킷을 중계하여 이동노드의 현재 위치와 상관없이 상대노드가 같은 모양의 패킷을 받을 수 있도록 한다.

모바일 IPv6에 가장 중요한 요소로 바인딩을 들 수 있다. 바인딩은 자신의 현재 위치에 대한 정보를 홈에이전트 또는 상대노드에게 알리는 것으로 바인딩을 통하여 노드의 응용프로그램 계층에서는 여전히 홈주소를 사용할 수 있게 된다. 이러한 바인딩메시지는 모빌리티 헤더에 위치하게 되며 바인딩메시지로는 바인딩 정보를 보내는 바인딩업데이트 메시지와 그에 대한 응답으로 오는 바인딩응답 메시지가 있다.

획득주소를 이용하여 패킷을 보내는 경우 획득주소 외에 이동노드의 홈주소에 대한 정보를 나타내야 한다. 모바일 IPv6에서는 홈주소에 대한 정보를 나타내기 위하여 IPv6 확장 헤더 중 *목적지 옵션 헤더*와 *타입 2 라우팅 헤더*를 사용하였다. *목적지 옵션 헤더* 내의 홈주소 옵션은 이동노드가 패킷을 보낼 때 자신의 홈 주소에 대한 정보를 적어 보내는 데 사용된다. *목적지 옵션 헤더*는 최종 목적지에 도착했을 때 이 옵션을 사용하라는 것으로서 이 헤더를 받은 노드는 IP 헤더에 있는 목적지주소와 목

적지 옵션에 있는 주소를 서로 바꿈으로서, 이동노드의 홈 주소를 상위 계층에게 전달한다. 라우팅 헤더는 패킷의 경로를 지정하는 확장 헤더인데, *타입 2 라우팅 헤더*는 상대노드가 이동노드에게 패킷을 보낼 때 이동노드의 홈주소에 대한 정보를 알려준다. 이동노드의 홈주소를 라우팅 헤더 엔트리에 적음으로써 이동노드가 이 패킷을 받았을 때 라우팅 헤더의 주소와 IP 헤더의 목적지 주소를 서로 바꾸어서 상위 계층에게 전달하게 한다.

이러한 확장 헤더를 사용하여 홈 주소를 주고 받음으로써 상위 계층은 패킷의 실제 목적지와 출발지의 주소인 획득주소 대신 원래 주소인 홈주소를 전달받게 된다.

이동노드가 외부링크로의 이동을 감지하거나 그 링크에서 획득 주소를 받을 때는 *네이버 디스커버리* (neighbor discovery) 프로토콜을 이용하게 된다. 이 프로토콜은 또한 패킷을 보낼 때 첫번째 홉의 링크계층 주소를 알아내기 위해서도 사용된다. 이동노드는 현재 라우터의 알림(advertisement) 메시지가 들리지 않을 경우 자신이 다른 링크로 이동한 것을 의심하게 되며 그것을 확인하기 위해 찾기 (solicitation) 메시지를 보낸다. 이동노드가 새로운 링크로 이동하였을 경우 새로운 획득주소를 얻기 위해 이동한 링크의 프리픽스를 얻어 그것을 이용하여 획득 주소를 만들어서 사용할 수 있는데 이를 *상태 없는 자동설정*(stateless auto-configuration)이라 한다.

2.3 Neighbor Discovery

*네이버 디스커버리*는 IPv4가 수행하던 주소 해결 (address resolution, ARP) 기능에 더하여 네이버의 접근가능성(reachability)을 감지하고, IP 주소의 자동설정기능과 주소충돌 감지기능 까지를 수행한다. 하나의 단말기가 같은 링크내의 다른 단말기와 통신을 하고자 하는 경우에는 라우터의 도움없이 데이터링크 계층에서의 통신이 가능하므로, 상대노드의 데이터 링크 계층 주소인 MAC 주소를 알아야 하는데, 네이버 디스커버리의 주소해결기능이 그것이다. 외부링크에 있는 상대노드와 통신을 하고자 할 때에는 통신하고자 하는 패킷을 먼저 디폴트라우터에게 보내야 하며, 이경우 디폴트라우터는 패킷의 첫번째 홉이 되는데, 이때에는 디폴트라우터의 MAC 주소를 알아야 한다.

모든 *네이버 디스커버리 메시지*는 ICMPv6를 사용하여 보내진다. *네이버 디스커버리*는 ICMPv6 메시지 *type 133*으로부터 *type 137*까지의 *라우터찾기* (router solicitation), *라우터알림* (router advertisement), *네이버찾기* (neighbor solicitation), *네이버알림* (neighbor advertisement), *방향전환메시지* (redirect message)등의 5종류의 메시지를 사용한다.

먼저 *라우터찾기* (router solicitation) 메시지는 디폴트라우터를 검색하는데 사용되는데, 링크상의 모든 라우터에게 보내져야 하므로, *all-routers multicast* 주소를 사용하여 보내진다.

라우터알림 메시지는 *라우터찾기* 메시지에 대한 응답으로 보내지거나, *라우터찾기* 메시지가 없었더라도 주기적으로 보내지는 메시지이다. *라우터찾기* 메시지에 대한 응답으로 보내지는 경우에는 unicast 주소를 사용하여 *라우터찾기* 메시지를 보낸 노드에게만 보내지며, 주기적으로 보내지는 메시지인 경우에는 *all-nodes multicast* 주소를 사용하여 모든 노드에게 보내진다. *라우터알림* 메시지를 통하여 노드들은 디폴트라우터의 주소와 링크계층의 주소뿐만 아니라, 링크의 프리픽스와 링크의 최대전달단위 (maximum transfer unit, MTU)등의 정보를 얻게 된다. 이러한 정보들은 ICMPv6의 옵션헤더를 통하여 전달된다.

네이버찾기 메시지는 주소해결, 네이버 도달가능성감지 (neighbor unreachability detection, NUD), 주소충돌감지 (duplicated address detection, DAD) 등을 위하여 사용된다. *네이버찾기* 메시지의 *target address* 필드에는 찾기를 원하는 이웃하는 노드의 IP 주소를 쓴다. 이 필드는 *네이버찾기* 메시지의 목적지주소에 쓰여지는 주소값과 다를 수 있는데 그 이유는 *네이버찾기* 메시지의 목적지 주소에는 *solicited node multicast* 주소를 사용하기 때문이다. 어드레스충돌감지 기능을 위해서는 노드는 *unspecified* 주소를 *네이버찾기* 메시지의 근원지 주소 필드에 쓰고, 자신의 잠정 주소를 사용하여 계산된 *solicited node multicast* 주소를 목적지 필드에 쓰며, 자신의 잠정 주소를 *target address* 필드에 쓰게 된다. 만약 링크상의 다른 노드가 이 잠정 주소를 쓰고 있다면, *네이버응답* 메시지를 보내게 될 것이고, 이로서 주소가 충돌됨을 감지할 수 있다.

네이버알림 메시지가 *네이버찾기* 메시지에 대한 응답으로 보내지는 경우에는 *target* 주소 필드에 *네이버찾기* 메시지의 근원지 주소를 복사하여 사용한다. *네이버알림* 메시지가 *네이버찾기* 메시지 없이 보내지는 경우에는 이 메시지에 포함되는 링크계층 주소에 해당하는 IP 주소를 *target* 주소 필드에 적는다.

Ⅲ. 모바일 IPv6 하드웨어 구조

본 논문에서 구현한 모바일 IPv6를 위한 하드웨어 구조는 그림 3과 같다. 그림의 왼쪽이 외부로부터 패킷을 받는 Receive(Rx) 부분이고, 오른쪽이 CPU로부터 데이터를 받아 외부로 내 보내는 Transmit(Tx) 부분이다. Rx 부분은 PacketParser(PP)로부터 패킷을 받아 모바일 IPv6 프로세싱을 수행한다. Tx 부분은 CPU로부터 받은 응용 프로그램 데이터에 대하여 모바일 IPv6 프로세싱을 수행한 후, 조립하여 패킷으로 만들어 링크로 내보낸다.

각 블록은 다음과 같은 일을 한다. PacketParser (PP)는 외부 링크에서 오는 패킷을 받아 헤더를 분리하여 각 블록으로 보내주고, 데이터를 Rx_RAM에 저장하는 역할을 한다. Rx_Checksum(RxCS)은 입력 패킷의 체크섬(checksum)을 사용하여 입력된 패킷의 에러 여부를 확인하는 블록으로서, PP로부터 입력 패킷의 데이터에 대한 체크섬을 받고, Rx_moblie(RxMP) 블록으로부터 슈도 헤더(pseudo header)를 받아, 입력 패킷의 체크섬과 계산된 체크섬이 맞는지 확인하는 역할을 한다. NodeState(NS) 블록은 현재 노드의 상태를 파악하는 블록으로서, 노드가 홈 링크(home link)에 있는지, 외부 링크(foreign link)에 있는지를 파악한다. 또한 Neighbor-Discovery(ND) 블럭으로부터 이동 정보를 받아 새로운 링크로 움직인 상태인지를 파악하여 다른 모바일 블록으로 전달하는 역할을 한다.

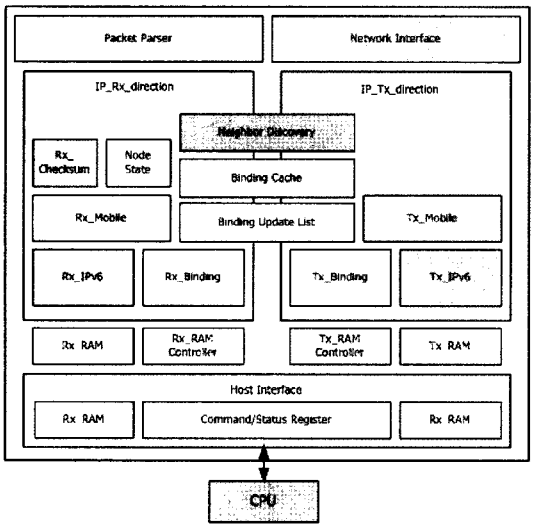


그림 3. Top level 블록도

Rx_Mobile 블록은 PP로부터 모바일 IPv6 헤더를 받아 처리하는 역할을 한다. Rx_Binding은 바인딩(binding) 정보를 받아 Binding_Cache(BC) 블록과 Binding_Update_List(BUL) 블록에 전달한다. BC에는 노드가 받은 바인딩 정보가 저장되어 있고, BUL에는 보낸 바인딩 정보가 저장된다.

Tx_Binding은 바인딩 메시지를 만드는 프로세싱을 수행하고, Tx_Mobile 블록은 모바일 IPv6 헤더를 만드는 역할을 수행한다. Network_Interface(NI)는 각 블록에서 만들어진 헤더와 CPU로부터의 응용 프로그램 데이터를 조합하여 하나의 패킷을 만들어 내보내는 역할을 한다.

그 밖에 Rx_IPv6와 Tx_IPv6 블록은 각 프로토콜에 해당하는 헤더 프로세싱을 담당하며, Rx_RAM에는 받은 패킷의 응용 프로그램 데이터가 저장된다. 저장된 데이터는 Host Interface(HI)를 통하여 CPU 에 전달된다. CPU로부터 외부로 나가야 하는 응용 프로그램 데이터는 HI 블록을 통하여 Tx_RAM에 저장되고, Tx_IPv6블록으로부터 IP 헤더가 만들어 진후, NI를 통하여 외부로 전달된다.

3.1 Data path / Control path

3.1.1 데이터 패킷 플로우

데이터 패킷에는 TCP, UDP 패킷 등이 해당되는데, 데이터 패킷 플로우를 그림 4에 보였다. 데이터 패킷을 받은 경우 PP는 IP 헤더를 Rx_Mobile 블록에 전해주고, Rx_Mobile 블록은 모바일 관련 기능을 제거한 40 바이트의 IP 헤더를 Rx_IPv6 에 전달한다. 그리고 Rx_IPv6는 IP 헤더 프로세싱을 수행하고, 그 결과를 CPU에게 주게 된다. 패킷의 TCP 혹은 UDP 데이터 부분은 PP가 직접 Rx_RAM에 저장하여 HI를 통하여 CPU가 읽어가도록 한다.

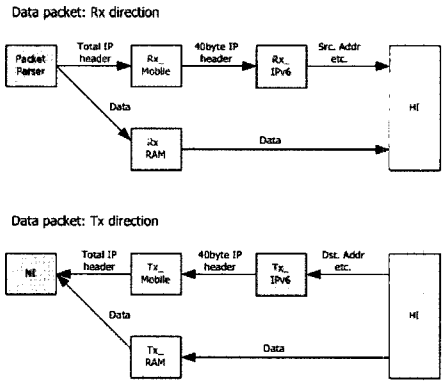


그림 4. 데이터 패킷 플로우

보내고자 하는 패킷이 있을 경우, CPU가 주는 명령에 따라 Tx_IPv6에서 IP헤더가 만들어 지는데, Tx_IPv6는 IP 헤더를 만들어 Tx_Mobile에게 전달하며 Tx_Mobile 에서 모바일 기능과 관련된 헤더를 만들어서 NI에게 전달한다. 보내고자 하는 응용 프로그램 데이터는 HI를 통하여 Tx_RAM에 쓰여지게 되고, NI는 Tx_RAM으로부터 데이터를 읽어, 헤더와 데이터를 순서에 맞게 조립하여 내 보내게 된다.

3.1.2 바인딩 관련 패킷 플로우

바인딩 관련 패킷은 CPU의 간섭 없이 모바일 관련 블록들에 의해 프로세싱되고, 또한 만들어져서 내보내어 지는데, 그림 5에 바인딩 관련 패킷의 플로우를 보였다. 바인딩 관련 패킷에는 바인딩업데이트(BU), 바인딩응답(BA) 패킷이 있고, 이러한 패킷은 IPv6의 모빌리티 헤더(mobility header)에 실려서 전송된다. 이러한 패킷을 받았을 경우 PP는 모빌리티 헤더를 제외한 IP 헤더는 Rx_Mobile 블록에 전달하고 모빌리티 헤더와 바인딩 관련 정보는 Rx_Binding 블록에 전달한다. Rx_Mobile 또한 IP 헤더에서 나온 바인딩 관련 정보를 Rx_Binding에게 전달한다. Rx_Binding 블록은 들어온 바인딩 패킷이 BU인 경우 BC에, BA인 경우 BUL에 전달한다. BU를 받았을 때 그에 따르는 BA를 보내거나 새로운 BU를 보내고자 할 때에는 Tx_Binding에 바인딩 패킷을 만들어 보내라는 요청을 하게 되고 Tx_Binding은 모빌리티 헤더와 바인딩 정보를 만든 후, Tx_Mobile에 IP 헤더를 만들라는 요청을 한다. Tx_Binding와 Tx_Mobile에서 만들어진 헤더 및 바인딩 정보는 NI로 전달 되고 NI는 패킷을 조립하여 내보내게 된다.

Binding packet: Rx_Tx direction

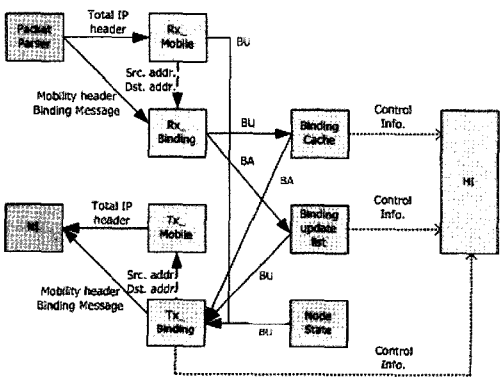


그림 5. 바인딩 관련 패킷 플로우

3.1.3 ICMP 관련 패킷 플로우

ICMP 관련 패킷 또한 CPU의 간섭 없이 ND 블록에 의해 프로세싱되고, 또한 만들어져서 내보내어 지는데, 그림 6에 ICMP 관련 패킷의 플로우를 보였다. ICMP 관련 패킷에는 *네이버 디스커버리* (neighbor discovery) 관련 패킷과 ICMP 에러 메시지가 있다. 이러한 패킷을 받은 경우 PP는 패킷을 ND로 전달한다. ND에서는 받은 패킷 정보를 사용하여, 자신이 갖고 있는 리스트를 업데이트 시키거나 패킷에 대한 처리를 하고 CPU에게 필요한 정보를 전달한다. *네이버 디스커버리*를 하고자 할 때나, 받은 패킷에 문제가 있을 경우 ICMP 패킷을 만들어 내보내게 되는데 이러한 경우에도 ND 블록에서 직접 패킷을 만들어 NI로 내보내고, 필요한 정보만 CPU에게 전달한다.

ICMP packet: Rx_Tx direction

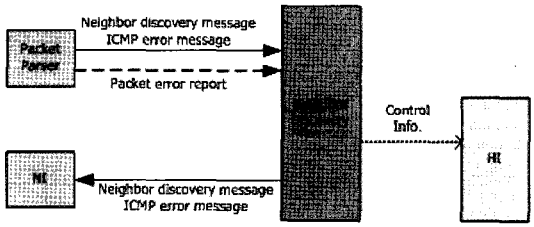


그림 6. ICMP 관련 패킷 플로우

3.2 각 블록의 구조

3.2.1 Host Interface

HI는 그림 3에서 보이는 바와 같이 CPU와 모바일 IPv6 하드웨어와의 통신을 담당하는 블록으로, CPU의 슬레이브 모드로 동작한다. HI의 컨트롤 정보는 CPU로부터 받는 버스 폭에 대한 정보와 웨이트(wait)에 관련된 정보, 하드웨어 블록들에 내려지는 명령, 보내는 패킷을 위한 헤더 정보, 하드웨어 블록들로부터 CPU로 보내지는 상태 정보, 받은 패킷의 헤더 정보 등 패킷의 송수신에 필요한 정보들이 포함된다. 컨트롤 정보는 HI 내부의 Command/Status Register(CSR)을 통해 전달된다.

HI의 데이터 플로우는 CPU와 모바일 IPv6 하드웨어 사이에서 데이터를 전달하는 역할을 하는데, 데이터를 수신하는 Rx 플로우와 데이터를 패킷으로 만들어 내보내는 Tx 플로우로 구분된다. Rx 플로우의 경우에는 Rx_RAM로부터 가져온 데이터를 내부의 RxFIFO를 통해 CPU가 읽어가도록 하고, Tx 플로우의 경우에는 CPU로부터 전송할 데이터를 읽

어와 내부의 TxFIFO를 통해 Tx_RAM에 저장하여 NI가 읽어가 패킷을 만들어 보낼 수 있도록 한다.

3.2.2 Rx_Mobile

모바일 IPv6의 기능을 수행하는 블록이다. PP에서 모빌리티 헤더를 제외한 IP 헤더를 받아 데이터를 처리하여 다른 블록으로 전달해준다. PP에서는 데이터를 바이트 단위로 전해주며 Rx_Mobile 블록으로 첫 번째 바이트를 줄 때 *start* 신호를 같이 띄워주고 마지막 바이트와 *end* 신호를 같이 띄워준다. 데이터를 받으면 헤더의 길이와 *next header* 필드를 보고 헤더의 유형을 결정한다. 들어온 패킷의 마지막 *next header* 필드가 모빌리티 헤더, 즉 135 이면 바인딩 패킷으로 간주하고 그렇지 않으면 데이터 패킷으로 간주한다.

들어온 패킷이 바인딩 패킷인 경우, 이동노드의 홈주소(MNHoA) 정보와, 상대노드의 홈주소(CNCoA)를 Rx_Binding 블록으로 전달한다. 이는 Rx_Binding 블록으로 들어오는 바인딩 메시지에는 홈 주소를 포함하고 있지 않으므로 Rx_Mobile 에서 그 정보를 전달해주어야 하기 때문이다. 들어온 패킷이 데이터 패킷이면 홈주소를 사용한 40byte의 헤더를 만들어서 Rx_IPv6 블록으로 전달한다. 원칙적으로 상위 계층에서는 바인딩이나 획득주소에 대해 알고 있지 않으므로 모빌리티 관련 헤더는 Rx_IPv6 블록으로 전달되지 않는다.

들어오는 모든 패킷은 체크섬이 맞는지 확인 되어야 한다. Rx_Mobile에서는 체크섬 계산을 위해 슈도헤더 (pseudo header)를 만들어서 RxCS 블록에 전달한다.

Rx_Mobile의 또 다른 기능은 터널링 된 패킷을 받았을 경우, 경로 최적화를 위하여, Tx_Binding에게 BU을 보내라고 요청하는 것이다. 상대노드가 이동노드의 획득주소를 모르는 경우 상대노드가 보내는 패킷은 홈 에이전트를 통하여 터널링 된 후 이동노드에게 전달되는데, Rx_Mobile에 터널링 된 패킷이 들어오면 Rx_Mobile 블록은 경로 최적화를 하려 하고 그것을 위해 패킷을 보낸 상대노드에게 획득 주소를 알려주기 위해 바인딩 업데이트 메시지를 보낸다.

3.3.3 Rx_Binding

바인딩 메시지를 받아 처리하는 블록이다. 바인딩 메시지에는 BU와 BA가 있다. PP로부터 모빌리티 헤더와 바인딩메시지를 받는데, 모빌리티 헤더와 바인딩메시지에는 홈주소에 대한 정보가 포함되어 있지 않으므로 Rx_Mobile로부터 이동노드의 홈주소와 상대노드의 홈주소를 받는다. 모빌리티 헤더의 MH type 필드를 보고 BU 패킷인 경우에는 MNHoA, CNHoA, CNCoA, *sequence number*, *lifetime*들을 BC (binding cache) 블록에 전달한다. BA 패킷인 경우에는 MNHoA, MNCoA, CNHoA, *sequence number*, *lifetime*, *status*들을 BUL(binding update list)에게 전달한다.

3.3.4 Binding Cache

받은 바인딩 업데이트를 저장하는 곳으로 상대노드의 획득주소에 대한 정보를 테이블의 형태로 유지하고 있다. BC의 엔트리에는 MNHoA, CNHoA, CNCoA, *sequence number*, *lifetime*을 저장하고 있고, 또한 각 엔트리의 유지를 위하여 *valid bit*, *lifetime counter*, *hit bit*를 갖고 있다.

Rx_Binding으로부터 바인딩 데이터를 받으면 MNHoA, CNHoA를 사용하여 일치하는 엔트리가 있는지 순차적으로 검색한다. 일치하는 엔트리가 있으면 *sequence number*를 비교하여 들어온 숫자가 저장되어 있는 것 보다 큰지 확인한 후, 해당 엔트리의 정보를 업데이트 한다. 들어온 *sequence number*가 엔트리에 저장되어 있는 것보다 크지 않으면 에러로 간주하고, BA 패킷의 *status* 필드에 *sequence number* 에러 라고 표시하여 내 보내게 된다. 일치하는 엔트리가 없으면 새로운 엔트리에 저장한다. 받은 BU 메시지에 대한 BA를 보내기 위해 Tx_Binding에 BA 패킷을 만들어 내 보내라고 요청한다.

Tx_Binding 블록은 패킷을 보낼 때 목적지 노드의 CoA를 알고 있는지 확인하기 위해 BC 검색을 의뢰한다. BC 블록은 검색을 통해 일치하는 엔트리가 있으면 *ack*, *match* 신호와 함께 상대노드의 CoA를 Tx_Binding에 전달하고, 일치하는 엔트리가 없으면 *ack* 신호만 보내준다.

BC 블록은 적당한 수의 엔트리를 유지하고 사용하지 않는 엔트리를 삭제하기 위하여 다음과 같은 과정을 백 그라운드 프로세싱으로 수행한다. 일단 테이블에 데이터가 저장되면 *valid bit*과 *hit bit*을 1로 set 하고 들어온 *lifetime*을 *lifetime counter*에 저장한다. 그리고 valid 한 엔트리에 대해 *lifetime counter*를 하나씩 감소시키고 일정시간마다 *hit bit*을 0으로 set한다. *Lifetime counter*가 임계 값 보다 작아지고 *hit bit*이 set 되어 있으면 해당 엔트리를

계속 사용 중인 것으로 간주하여 Tx_Binding에 바인딩 리퀘스트(BR)을 요청하여 binding을 유지할 수 있다. *Lifetime counter*가 임계 값 보다 작으며 *hit bit*이 0인 경우에 그 엔트리는 사용되지 않는 것으로 간주하여 *Lifetime counter* 가 0이 되면 엔트리를 삭제한다.

3.3.5 Binding Update List (BUL)

보내는 바인딩 업데이트 메시지를 저장하는 곳으로 이동노드의 CoA와 그것을 아는 상대노드에 대한 정보를 테이블의 형태로 가지고 있다. BUL 블록의 테이블 엔트리에는 CNHoA, MNCOA, MNHoA, *sequence number*, *lifetime*, *H bit*를 저장하고 있고 또한 각 엔트리의 유지를 위해서 *valid bit*, *lifetime counter*, *hit bit*를 갖고 있다. 그리고 응답패킷을 기다리는 과정을 위해 각 엔트리 마다 *Retrans_times*, *wait ack counter*, *state*를 저장하고 있다.

Tx_Binding 블록은 바인딩 업데이트를 보내고 난 후, 그 정보를 BUL에 전달한다. BUL 블록은 받은 정보가 BUL 블록에서 요청한 것이면 해당 엔트리를 업데이트하고, 다른 블록에서 요청한 바인딩은 새로운 엔트리에 저장한다. 또한 보낸 바인딩업데이트 메시지에 대한 바인딩응답 패킷을 기다린다. *State*를 *wait ACK* 상태로 하고 *wait ACK counter*를 하나씩 증가시킨다. *Wait ACK counter*가 정해놓은 시간보다 넘게 되면 다시 업데이트 메시지를 보내도록 Tx_Binding 블록에 요청한다. 이러한 재전송은 정해놓은 횟수까지 반복하고 그 때까지 바인딩응답 패킷이 오지 않으면 그 엔트리는 무효한 것으로 간주한다. Rx_Binding 블록으로부터 바인딩응답 패킷을 받으면 바인딩응답 패킷에 해당하는 엔트리를 찾기 위해 MNHoA, CNHoA을 사용하여 일치하는 엔트리가 있는지 순차적으로 검색한다. 일치하는 엔트리가 있으면 *sequence number*를 비교하여 들어온 숫자가 저장되어 있는 것과 일치하는지 확인한 후, 일치하는 경우 해당 엔트리의 정보를 업데이트한다.

Tx_Binding 블록은 패킷을 보낼 때 상대노드가 이동노드의 획득주소를 알고 있는지를 확인하기 위해 BUL 검색을 의뢰한다. BUL 블록은 검색을 통해 일치하는 엔트리가 있으면 *ack*, *match* 신호와 함께 MNCoA를 Tx_Binding 블록으로 전달하고 일치하는 엔트리가 없으면 ack 신호만 보내준다.

BUL 블록은 BUL 테이블을 유지하기 위해서 유효한 엔트리에 대해 *lifetime counter*를 하나씩 감소시키고 일정시간마다 *hit bit*을 0으로 set한다. *Lifetime counter* 가 임계 값 보다 작아지고 *hit bit*이 set 되어 있으면 해당 엔트리를 계속 사용 중인 것으로 간주하여 Tx_Binding에 바인딩 업데이트 메시지를 요청하여 바인딩을 유지할 수 있게 한다. 임계 값 보다 작거나 *hit bit*이 0인 경우에 그 엔트리는 사용되지 않는 것으로 생각하고 *lifetime counter* 가 0이 되면 엔트리를 삭제한다. 또한 NS로부터 다른 링크로 이동 했다는 신호를 받으면 유효한 엔트리의 *state*를 *move* 상태로 하여 순차적으로 BU를 보내 리스트를 업데이트 하고 홈 링크로 돌아온 경우 *state*를 *move* 상태로 하되 보내는 BU의 *lifetime*을 0으로 하고 MNCoA를 MNHoA와 같게 하여 바인딩을 취소시킨다.

3.3.6 Tx_Binding

바인딩 메시지를 만드는 블록이다. 바인딩메시지 요청은 주변 4개의 블록에서 올 수 있다. 링크를 이동한 경우 NodeState (NS) 블록으로부터 바인딩 업데이트 메시지를 만들라는 요청이 올 수 있고, 터널링 패킷을 받은 경우 Rx_Mobile 블록으로부터 경로 최적화를 위한 바인딩업데이트 메시지를 만들라는 요청이 올 수 있다. 또한 BUL의 임의 엔트리의 *lifetime*이 임계값에 도달했을 경우 BUL 블록으로부터 바인딩업데이트 메시지에 대한 요청이 올 수 있으며, BU 메시지를 받았을 경우 BC 블록으로부터 바인딩업데이트 요청이 올 수 있다. 이렇게 각 블록에서 온 바인딩 요청은 순차적으로 처리된다. 바인딩메시지에 대한 요청이 오면 해당하는 메시지를 각 헤더 필드에 저장하고 체크섬을 계산한다. 또한 메시지의 IP 헤더를 만들기 위해 Tx_Mobile에 헤더를 만들라고 요청한다. 체크섬 계산이 끝나면 데이터를 조립하여 NI에게 전달하여야 한다. NI에게 데이터를 내보내겠다는 요청을 하여, NI로부터 응답을 받으면, 한 클럭에 2 바이트씩을 전달하며, 마지막 2 바이트와 함께 *end* 신호를 띄워 데이터가 끝임을 알린다.

3.3.7 Tx_Mobile

모바일 IP 헤더를 만드는 블록이다. Rx_IPv6 블록으로부터 IPv6 헤더를 받거나 Tx_Binding 블록으로부터 MNHoA, CNHoA, 길이 정보 등을 받아 모바일 IPv6 헤더를 만들게 된다. 각 블록에서 오는 데이터의 진원지, 목적지 주소는 홈주소이므로 BC, BUL 검색을 통해 CoA를 얻고 헤더의 형태를 결정

하여야 한다. IP 블록에서 온 데이터의 경우에는 BUL, BC를 모두 검색하여 CoA에 대한 정보를 얻고, 바인딩 메시지 중 BU 이면 이미 이동노드의 CoA는 결정된 상태이므로 BC만을 검색한다. 그렇지 않고 BA이면 상대노드의 CoA에 대한 바인딩응답 패킷이므로 BUL만 검색하여 이동노드의 CoA를 사용할지를 결정한다. 그리고 현재 이동노드가 홈링크에 있으면 BC만 검색하고 BUL은 검색하지 않는다. 이렇게 들어온 데이터와 검색 결과에 따라 헤더의 형태를 결정하고 모바일 헤더를 만들어서 NI에게 전달한다. 또한 Tx_Mobile은 MAC 헤더를 만들기 위하여 ND 블록에게 가장 외곽에 있는 목적지 주소를 전달한다.

3.3.8 NodeState

NodeState (NS) 블록은 ND로부터 외부링크로 이동했는지, 홈링크에 있는지에 대한 정보를 받아서 상태를 유지하며 그 상태를 다른 모바일 관련 블록에 전해주는 역할을 한다. 홈링크에 있다가 ND로부터 외부링크로 이동했다는 신호와 함께 획득주소를 전달 받으면 NS는 Tx_Binding 블록에 요청하여 홈에이전트에게 바인딩을 보내게 한다. 또한 새로운 링크로 다시 이동하게 되는 경우에 BUL 블록에 신호를 주어서 현재 BUL 리스트를 업데이트 하라는 신호를 전달한다. 이동노드가 다시 홈링크로 돌아오는 경우에도 BUL 블록에게 모든 바인딩을 없애라는 신호를 전달한다.

3.3.9 Neighbor Discovery (ND)

ND block은 목적지 IP 주소로 패킷을 보내야 할 때 들르는 첫번째 홉의 MAC 주소를 찾아내는 기능을 수행한다. 또한 디폴트 라우터 선택, 네이버 도달가능성감지(neighbor unreachability detection), 그리고 이동 감지 (movement detection)를 수행한다. ND 블럭은 RxND, NDP, 그리고 TxND로 이루어져 있다. RxND는 PP로부터 라우터 알림(router advertisement), 라우터 찾기(neighbor solicitation), 그리고 네이버 알림(neighbor advertisement) 패킷을 받아들여 패킷에 에러가 없는 지를 확인한 후 들어온 메시지가 에러가 없는 경우에만 NDP에게 그 정보를 전달한다. TxND는 NDP의 명령에 따라 ND 메시지를 만들어 NI가 준비될 때까지 패킷을 저장해 놓았다가 NI에게 보내주는 역할을 한다. NDP는 주소 변환에 필요한 네이버캐쉬 (neighbor cache)와 디폴트 라우터 리스트와 프리픽스 리스트를 관리한다. 그리고 ND 메시지를 보내야 할 경우에, TxND 블럭에게 ND 메시지를 만들라는 신호를 보내준다.

ND 블록은 네이버캐쉬의 정보를 관리하고 IP 주소의 변환을 수행한다. NI 블럭은 프리픽스 리스트의 프리픽스들과 목적지 주소를 비교하여, 목적지 주소가 같은 링크에 있는 경우 (on-link)에 ND에게 알고자 하는 IP 주소와 함께 요청 신호를 보낸다. 이 요청을 받은 ND 블록은 네이버캐쉬에서 일치하는 정보를 검색하고 해당 정보가 캐쉬에 존재하는 경우 MAC 주소에 대한 정보를 NI 블럭으로 전달한다. 캐쉬에 찾는 정보가 없는 경우 NS 패킷을 만들어 같은 링크상의 다른 모든 호스트들에게 전달한다. 보낸 NS에 대한 응답으로 요구된 네이버 알림 (solicited neighbor advertisement) 메시지가 오면 이 패킷으로부터 원하는 주소 정보를 얻게 된다. 이 때 해당하는 네이버캐쉬의 엔트리는 *REACHALBE* 상태가 된다. 만약 일반적인 네이버 알림(unsolicited neighbor advertisement)이나 라우터 알림(router advertisement) 메시지를 받은 경우에는 네이버캐쉬의 해당 엔트리는 *STALE* 상태가 된다. 또한 reachable time 동안 요청을 받지 않은 네이버캐쉬 엔트리는 reachable time 이 지나면 *REACHABLE*상태에서 *STALE*상태로 바뀌게 된다. *STALE*상태에서 요청을 받으면 *DELAY*상태가 되며, 타이머를 *DELAY FRIST PROBE TIME* (5sec)으로 조정한다. *DELAY FIRST PROBE TIME*이 지나도록 해당 네이버로부터 패킷을 받지 못 하면, *DELAY*상태에서 *PROBE* 상태로 이동하게 된다. *PROBE*상태에서는 *Retrans Timer*에 정해진 시간에 한번씩 NS 패킷을 내어 보낸다. *MAX_UNICAST_SOLICIT*보낸 후 Reachable Time이 지나도록 해당 노드로부터 패킷을 받지 못하면, 그 네이버는 도달 가능하지 않은 것으로 판단한다. *DELAY*, 혹은 *PROBE*상태에서 해당 노드로부터 패킷을 받는 다면, *REACHABLE*상태로 옮겨가게 된다. 쓰지 않는 엔트리를 지우는 에이징 (aging) 기능을 제공하는데, *STALE*상태에서 10초 동안 사용되지 않으면, 그 네이버와는 통신하지 않는 것으로 판단하여 해당 엔트리를 지우게 된다.

네이버캐쉬는 20개의 엔트리를 가지며, 엔트리가 라우터인지 호스트인지를 나타내는 *IsRouter*와 *state*, IP 주소와 MAC 주소로 이루어져 있다.

디폴트 라우터 리스트는 라우터 알림(router advertisement)을 받을 때마다 업데이트 된다. 이동노드가 홈링크가 아닌 다른 링크에 가 있을 때 패킷을 보내게 되는 디폴트 라우터는 홈에이전트 역할을 해줄 수 있는 (*H bit*이 set) 라우터를 사용하게

된다. 디폴트 라우터의 변경은 디폴트 라우터를 통한 통신이 실패할 때 사용된다. 디폴트 라우터 리스트는 5개의 엔트리를 가지며, 각각의 엔트리는 홈에이전트, *Lifetime*, 라우터 IP 주소로 이루어져 있다.

프리픽스 리스트는 라우터 알림(router advertisement)의 옵션인 프리픽스 옵션을 통해 업데이트된다. 프리픽스 리스트는 5개의 엔트리를 가지며, 각 엔트리는 *prefix*, *prefix 길이*로 이루어져 있다.

Ⅳ. 실 험

모바일 IPv6 하드웨어는 베릴로그 (verilog) 언어를 사용하여 구현하였다. Verilog-XL 시뮬레이터를 사용하여 실험하였으며, 삼성전자에서 제공한 STD130 라이브러리 0.18 micron 기술을 사용하여 합성하였다. 상대노드, 홈에이전트, 외부링크의 라우터가 보내는 여러 종류의 패킷을 만들어 입력으로 사용하여, 설계한 하드웨어가 그에 해당하는 동작을 잘 수행하는지 살펴보았다. 표 1은 본 논문에서 설계한 모바일 IPv6 하드웨어의 테스트 케이스이다. 컨트롤 플로우와 3가지 데이터 패킷의 플로우에 따라 테스트 케이스를 분류하였다.

그림 7은 CPU의 요청으로 데이터 패킷을 보내는 경우 HI의 동작을 나타낸 것이다. CPU에서 XrDATA를 받아 그것을 TxFIFO(T0~T7)에 쓰고 그것을 TxRAM(B0~B7)에 읽어서 저장하는 과정이다. 보내고자 하는 데이터를 다 저장하면 그것을 NI에게 알리고 NI가 패킷의 헤더와 데이터를 조립하여 패킷을 내 보낸다.

그림 8은 데이터 패킷을 보내는 경우 IP헤더 처리에 대한 과정이다. HI가 TxIP 블록으로 헤더 정보와 요청(HI_Snd_TxUDP)을 보내면 IP헤더를 만들어서 TxMP블록에 보내고 그것을 받아 BC와 BUL을 검색하여 적절한 헤더의 형태를 결정한다. 그렇게 헤더

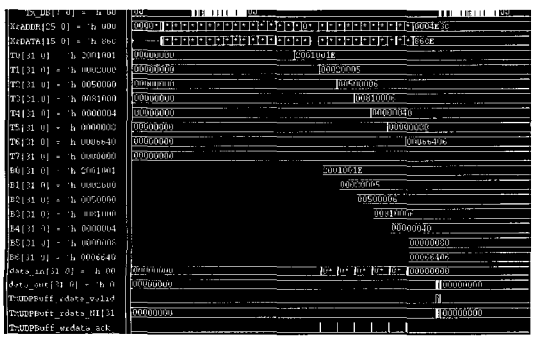


그림 7. CPU의 요청으로 데이터 패킷을 보내는 경우(1)

표 1.Test Cases

컨트롤 플로우
1. CPU가 CSR 레지스터에 값을 쓰고 그 CSR 레지스터에서 값을 읽어감
2. 받은 데이터를 RxRAM을 거쳐 HI의 RxFIFO를 통해 CPU가 읽어감
3. CPU로부터 전송할 데이터를 받아 내부의 TxFIFO를 통해 TxRAM에 저장하여 NI가 내보냄
4. 각 블록들의 상태 레지스터를 CPU가 읽어간 후, 이 레지스터는 clear 됨
데이터 패킷 플로우
1. 다양한 헤더의 형태를 처리하여 상위 계층으로 전달
2. HI가 받은 패킷의 헤더 정보를 CSR에 쓰고 CPU가 읽어감.
3. 보내고자 하는 데이터 패킷이 있으면 CPU가 헤더정보를 CSR에 쓰고 HI가 읽어 IP 블록에 전달
4. CPU로부터 IP 헤더를 받아 BUL/BC를 검색하여 적당한 헤더 형태를 만들어 NI로 내보냄
바인딩 패킷 플로우
1. 다른 링크로 이동시 홈 에이전트에게 바인딩 패킷을 보내고 응답을 받음
2. BUL의 요청으로 바인딩 패킷을 보내고 응답을 받음
3. 터널링 패킷을 받았을 경우 바인딩 패킷을 보내고 응답을 받음
4. 바인딩 패킷을 받았을 경우 응답을 보냄
5. 바인딩 패킷을 받았을 경우 BC에 저장
6. 바인딩 패킷을 보내는 경우 BUL에 저장
7. BC/ BUL 의 엔트리 관리
9. BU에 대한 응답이 오지 않으면 재전송
8. 에러 패킷의 처리
ICMP 패킷 플로우
1. RA의 옵션처리
2. 새로운 프리픽스를 받았을 때 이동 감지
3. NS를 받았을 때 응답으로 NA를 보냄
4. 처음 받는 RA인 경우에 RS를 보냄
5. 네이버캐쉬의 타이머와 상태를 관리
6. 네이버 접근 가능성 감지(neighbor unreachability detection)을 위한 NS를 보냄
7. 접근 가능하지 않은 네이버를 네이버캐쉬에서 지움
8. 패킷의 MAC 헤더를 만들기위해 네이버캐쉬에서 MAC 주소를 찾음
9. 외부링크에서 홈링크로 돌아왔을 때, 홈으로 귀환을 감지
10. 홈으로 귀환했을 때 홈에이전트의 MAC 주소를 찾음

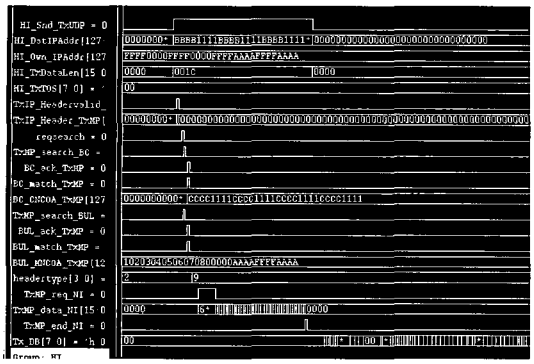


그림 8. CPU의 요청으로 데이터 패킷을 보내는 경우(2)

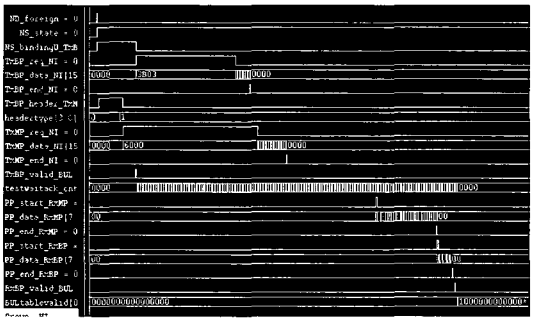


그림 9. 홈 링크에서 외부링크로 이동 했을 경우

를 만들고 NI에게 요청을 보내면 NI가 헤더를 받아 데이터와 조립하여 Tx_DB로 패킷을 내보낸다.

그림 9는 홈 링크에서 외부 링크로 이동 했을 때 바인딩 업데이트 패킷을 홈 에이전트에 보내고 응답을 받는 과정이다. ND로부터 이동 했다는 신호(ND_foreign)와 획득 주소를 받으면 NS는 TxBP블록에 홈에이전트에게 바인딩 업데이트 패킷을 보내라고 요청을 한다. TxBP는 바인딩 메시지를 만들고 TxMP에 IP 헤더를 만들어 달라고 요청을 한다. TxMP는 요청에 따라 IP 헤더를 만들어 NI로 내보내고 TxBP는 바인딩 업데이트 정보를 BUL에 전달한다. BUL은 그 정보를 저장하고 그에 해당하는 응답 패킷을 기다린다. PP로부터 바인딩 응답 패킷을 받으면 RxBP는 그것을 BUL에 전달하고 BUL은 증가 시키고 있던 ack_cnt를 중지하며 해당 엔트리의 valid bit을 1로 바꾸게 된다.

그림 10는 상대 노드로부터 바인딩 업데이트 패킷을 받았을 경우이다. RxBP는 업데이트 패킷을 받으면 그 정보를 BC에 전달한다. BC는 같은 상대 주소에 대한 엔트리가 있는지를 검색하고 엔트리가 없으므로 새로운 엔트리에 데이터를 저장하고 valid bit을 1로 만든다. 그리고 TxBP에게 받은 패킷에 대한 응답으로 바인딩 응답 패킷을 만들어 보내라

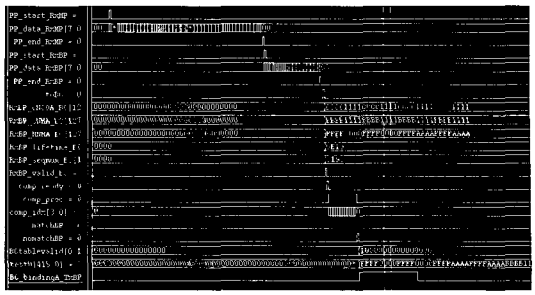


그림 10. BU를 받았을 경우

표 2. 합성결과

블록	Area
Packet Parser (PP)	6354
Rx_Checksum	3049
Rx_Mobile	22540
Rx_Binding	9068
Binding Cache	92644
Binding Update List	146542
Tx_Binding	26062
Tx_Mobile	33939
NodeState	3835
RxND	11308
TxND	67100
NDP	135636
ICMP	44878
NI	112433
Total	711388

고 요청한다.

설계한 하드웨어를 테스트해 본 결과 모바일 기능이 CPU의 간섭없이 제대로 동작하는 것을 확인하였다. 이동 노드가 외부링크로 움직인 경우 제안하는 하드웨어 구조는 10 클락 주기 시간 이내에 이동을 감지하여 바인딩 메시지를 만들어 낸다. 클락 주파수가 100MHz라고 가정할 때 이동감지까지 100nsec이하의 시간만이 소요되어 빠른 이동감지 성능을 보인다. 또한 바인딩과 네이버 찾기 기능들이 하드웨어로 빠르게 동작하여 모바일 기능의 전반적인 성능향상을 기대할 수 있다.

표 2은 설계한 하드웨어의 합성 결과이다. 세부 블록은 18개의 블록으로 나누어져 있으며 블록 별로 다음과 같은 area를 보였다. 전체 블록은 728351의 area 결과를 보였다.

V. 결 론

모바일 IPv6를 위한 하드웨어를 설계하고 구현하였다. 본 논문에서 구현된 하드웨어는 이동노드가

새로운 링크로 움직였을 때에 10 클락 구간 이내에 이동을 감지할 수 있으며, 바인딩에 필요한 패킷들 역시 빠른 시간 안에 만들어 내 보내는 것을 확인할 수 있었다. 바인딩 패킷들에 따르는 응답 패킷 또한 신속히 처리함으로 효율적인 이동성 지원이 가능하다. 이동성 지원을 위한 모든 기능들이 CPU의 간섭없이 하드웨어에 의하여 처리됨으로 이동지원 처리속도가 매우 빠를 뿐만 아니라, CPU는 응용프로그램에 보다 충실 할 수 있으므로, 단말기의 전체적인 성능향상이 기대된다.

참 고 문 헌

[1] H. Soliman, "Mobile IPv6: Mobility in a Wireless Internet," *Pearson Education*, Inc., 2004

[2] C. Perkins, "IP Encapsulation within IP," RFC2003, Oct. 1996, http://www.rfc-editor.org/rfc/rfc2003.txt

[3] C. Perkins, "Minimal Encapsulation within IP," RFC2004, Oct. 1996, http://www.rfc-editor.org/rfc/rfc2004.txt

[4] C. Perkins, "IP Mobility Support for IPv4," RFC3220, Jan. 2002, http://www.rfc-editor.org/rfc/rfc3220.txt

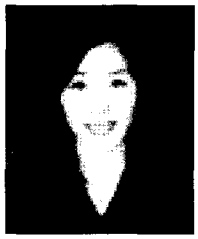
[5] D. Johnson and C. Perkins, "IP Mobility Support for IPv6," RFC3775, Jun. 2004, http://www.rfc-editor.org/rfc/rfc3775.txt

[6] S. Deering and R. Hinden, "Internet Protocol, Version 6, Specification," RFC2460, Dec. 1998, http://www.rfc-editor.org/rfc/rfc2460.txt

[7] T. Narten and W. Simpson, "Neighbor Discovery for IP Version 6," RFC2461, Dec. 1998, http://www.rfc-editor.org/rfc/rfc2461.txt

[8] A. Conta and S. Deering, "Generic Packet Tunneling in IPv6 Specification," RFC2473, Dec. 1998, http://www.rfc-editor.org/rfc/rfc2473.txt

김 혜 란 (Hyeran Kim) 준회원



2003년 2월 이화여자대학교 정보통신학과 학사

2006년 2월 이화여자대학교 정보통신학과 석사

2006년 2월~현재 삼성전자 정보통신총괄

<관심분야> Router나 switch 등의 network 관련 설계, TCP/IP 관련 설계

문 주 형 (Ju Hyoung Mun) 준회원

2005년 2월 이화여자대학교 정보통신학과, 학사

2005년 3월~현재 이화여자대학교 정보통신학과, 석사과정

<관심분야> Router나 switch 등의 Network 관련 SoC 설계

김 원 정 (Wonjung Kim) 준회원

2003년 2월 이화여자대학교 정보통신학과 학사

2006년 2월 이화여자대학교 정보통신학과 석사

2006년 3월~현재 넥스원퓨처 시스템연구소 연구원

<관심분야> Router나 switch 등의 Network 관련 SoC설계, 멀티미디어 네트워크

추 하 늘 (Ha Neul Chu) 준회원

2005년 8월 이화여자대학교 정보통신학과, 학사

2005년 9월~현재 이화여자대학교 정보통신학과, 석사과정

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

지 서 영 (Suh Young Jhee) 준회원

2005년 2월 이화여자대학교 물리학과, 학사

2005년 3월~현재 이화여자대학교 정보통신학과, 석사과정

<관심분야>Router나 switch 등의 Network 관련 SoC 설계, Multimedia Streaming

임 혜 숙 (Hyesook Lim) 종신회원

1986년 2월 서울대학교 제어계측공학과 학사

1991년 2월 서울 대학교 제어계측공학과 석사

1996년 12월 The University of Texas at Austin, Electrical and Computer Engineering, Ph.D.

1986년 8월~1989년 2월 삼성 휴렛 팩커드 연구원

1996년 11월~2000년 7월 Lucent Technologies, Murray Hill, NJ, Member of Technical Staff

2000년 7월~2002년 2월 Cisco Systems, San Jose, CA, Hardware Engineer

2002년 3월~2006년 2월 이화여자대학교 정보통신학과, 조교수

2006년 3월~현재 이화여자대학교 정보통신학과, 부교수

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, 통신 관련 SoC 설계