능동 응용의 특성을 고려한 능동 노드 구조
(Active Node Architecture considering the Characteristics of Active Applications)

안 상 현 † 김 경 춘 †† 손 선 경 ††† 손 승 원 †††
(Sanghyun Ahn) (Kyeongchoon Kim) (Seon-Gyoung Sohn) (Sung-Won Sohn)

요 약 능동 네트워크는 기존 프로토콜 개발 방식이 가지는 새로운 프로토콜의 수용 능력에 있어서의 한계와, 장시간의 표준화 작업 등의 문제를 해결하기 위해 등장한 새로운 네트워킹 방식이다. 지금까지 많은 연구소와 대학에서 능동 노드의 구조와 능동 패킷의 형식을 제안해 왔으나, 현재까지 제안된 능동 네트워크에서 고려되지 못한 문제들이 여전히 남아 있다. 그중의 하나가 서로 다른 능동 응용들의 요구사항에 따라 특성화되는 능동 패킷과 라우팅 방법에 대한 것이다. 따라서 본 논문에서는 다양한 능동 응용들의 요구사항에 따른 능동 패킷의 기능을 파악해서 이를 반영한 능동 패킷의 종류를 네 가지로 구분하고, 각 특성에 맞는 라우팅 방법을 제안한다. 또한 이들 사항을 반영한 능동 노드 구조도 제시하였다.

키워드 : 능동 네트워크, 능동 노드, 능동 응용, 능동 패킷, 능동 라우팅

Abstract The active network is a new networking approach to allow new protocols to be developed easily by solving the limitation of the existing protocol development procedure and the long protocol standardization process. So far many research institutes and universities have proposed active node architectures and active packet formats, but still there are some problems needed to be solved. One of them is the active packet format and the routing approach which consider the requirements of various active applications. Therefore, in this paper, we figure out the requirements of various active applications and design four types of active packet formats that reflect these requirements and propose routing schemes appropriate for these characteristics. Also we propose an active node architecture which reflect these requirements.

Key words : Active Network, Active Node, Active Application, Active Packet, Active Routing

1. 서 론

최근 인터넷에서는 웹의 등장과 함께 빠른 속도로 트래픽이 증가하고 있으며 멀티미디어 서비스와 같은 초기 네트워크 개발 당시에는 고려되지 않았던 다양한 서비스들이 요구되고 있다. 이를 위해 개발자들은 새로운 프로토콜을 개발하는데 많은 노력을 기울이고 있지만 다양하고 급격하게 변화하는 사용자의 요구사항을 만족시키지 못하고 있다. 오늘날의 네트워크 하부구조가 본질적으로 정적이므로, 새로운 기술 및 서비스가 인터넷에 적용되기 위해서는 기존 프로토콜과의 호환성을 가지고 개발되어야 하며 장시간의 표준화 작업을 거쳐야 한다. 이러한 기존 네트워크 프로토콜 개발 방식이 가지는 문제점을 해결하고자 제시된 기술이 능동 네트워크(active network)이다.

기존 네트워크에서 스위치나 라우터는 IP 패킷을 목적지 주소와 경로 배정 테이블에 따라 적절한 인터페이스로 포워딩(store-and-forward)하는 단순한 패킷 전달 기능을 수행한다. 반면, 능동 네트워크에서는 새로운 프로토콜이 각 능동 노드에서 수행될 프로그램 코드로 작성되며, 이러한 능동 코드들은 능동 패킷내에 데이타와 함께 포함되어 전송된다. 능동 패킷을 수신한 노드는 패킷내에 포함된 코드를 해석하고 실행한 후 그 결과를 다른 노드로 전송한다(store-compute-forward). 각 노

† 종신회원 : 서울시립대학교 컴퓨터통계학과 교수
(corresponding author임)
ahn@venus.uos.ac.kr
†† 학생회원 : 서울시립대학교 컴퓨터통계학과
netiv@venus.uos.ac.kr
††† 비 회 원 : 한국전자통신연구원 네트워크보안연구부 연구원
sgsohn@etri.re.kr
swsohn@etri.re.kr
논문접수 : 2002년 3월 5일
심사완료 : 2002년 9월 27일

드에서 수행될 프로그램 코드를 작성하는 방식으로 새로운 프로토콜을 개발할 수 있기 때문에, 기존 프로토콜과의 호환성을 반드시 유지할 필요가 없으며 좀 더 빠르게 프로토콜을 개발할 수 있고, 또한 라우터의 추가적인 역할을 필요로 하는 프로토콜을 개발할 수 있는 장점이 있다(능동 응용은 프로그램으로 작성된 프로토콜들이다).

능동 네트워크의 구성요소는 다음과 같다.

능동 노드는 능동 네트워크를 구성하는 가장 기본적인 구성요소로서 능동 패킷을 분류하고 패킷내에 포함된 코드를 해석하고 실행한 후 그 결과를 다른 능동 노드로 전달하는 기능을 수행한다. 능동 노드는 크게 노드운영체제, 실행환경, 능동 응용으로 구성된다. 노드운영체제는 다른 능동 노드로부터 능동 패킷을 받아서 이를 적절한 실행환경으로 분배하고, 실행환경에서 해당 능동 패킷을 처리하기 위해 필요한 노드 자원을 관리(할당/제어/반환)하며, 실행환경으로부터의 결과를 다른 노드로 전달하는 기능을 수행한다. 실행환경은 일종의 가상 기계(virtual machine)로 노드운영체제로부터 전달받은 능동 패킷내에 포함된 코드를 해석하고 실행한 후 그 결과를 노드 자신의 능동 응용이나 노드운영체제로 전달하는 작업을 수행한다. 이때 수행하고자 하는 실행코드는 반드시 노드에 적재되어 있어야 하며, 만약 필요한 실행코드를 노드 자신의 능동 응용으로부터 얻을 수 없으면 다른 노드로부터 전달받아야 한다. 이때 필요한 기술이 코드 요청 기술이다. 현재까지 제시되고 있는 방법으로는 이전 능동 노드로부터 전달받는 방법과 코드 서버로부터 전달받는 방법이 있다.

능동 패킷은 사용자 데이타와 함께 각 능동 노드에서 수행될 프로그램 코드를 포함하며, 일반 IP 패킷과 구별된다. 기존 일반 라우터에서는 능동 패킷을 처리할 필요가 없으며, 능동 노드에서만 능동 패킷을 적절한 실행환경으로 분류하고 해당 실행환경에서 패킷에 포함된 코드를 해석하고 실행한다. 능동 패킷은 크게 두 가지 형식이 결합된 형태를 가진다. 첫째는 해당 패킷이 능동 패킷이고 패킷내에 포함된 코드를 수행할 실행환경을 명시한 형식[1, 2, 3, 4]이며, 둘째는 각 실행환경에 따른 패킷 형식이다[5, 6, 7, 8].

현재까지 능동 네트워크 분야에서는 능동 노드 구조[1, 2, 3, 4, 5, 6, 7, 8]와 능동 패킷의 형식에 관한 연구[8, 9, 10, 11, 12, 13, 14, 15]가 중점적으로 연구되고 있다. 우리는 현재까지 제안된 여러 능동 노드의 구조와 패킷형식, 코드 요청 방식에 대해 살펴보았으며 몇가지 고려되고 있지 못하는 문제들을 발견했다. 이들 문제는 능동 패킷의 트래픽 특성과 능동 네트워크 상에서의 라우팅에 관한 것이다. 현재 인터넷의 가장 큰 특징 중의 하나가 best-effort 전달방식이다. 모든 IP 패킷은 단일한 특성을 가지는 독립된 IP 패킷들로 고려되며, 연구 또한 서로 독립된 IP 패킷을 네트워크의 상태에 따라 어떻게 효과적으로 라우팅할 것인가에 맞추어져 왔다. 그렇지만 최근 인터넷 사용자 증가와 응용의 다양성에 따라, 인터넷의 연구분야는 응용에 따른 패킷의 특성을 구별하고 요구되는 성능을 보장해주기 위한 방법을 모색하는데 초점이 맞춰지고 있다. QoS 보장과 고속의 트래픽 처리에 관한 연구분야들이 그것이다. 이러한 인터넷의 발전과정과 비슷하게 현재까지의 능동 네트워크 연구 분야에서도 아직까지 능동 패킷을 패킷의 특성에 따라 구분하지 않고 모두 단일한 특성을 가지는 패킷으로 처리하고 있다. 능동 패킷은 능동 응용의 요구사항에 따라 특성을 구분할 수 있으며, 본 논문에서는 능동 응용의 특성에 따라 크게 네 종류로 능동 패킷을 구분하고 이를 우리가 제안하는 능동 노드의 구조와 능동 패킷의 구조, 능동 패킷의 라우팅 방법 등에 반영시켰다.

본 논문의 구성은 다음과 같다. 2절에서는 제안된 능동 노드의 전체적인 구조를 기술하고, 3절에서는 트래픽 특성과 라우팅을 고려한 능동 패킷의 형식과 능동 노드에서의 처리과정, 코드 요청 방법 등 주요 기능을 기술하며, 마지막으로 4절에서는 결론 및 향후 연구 방향에 대해서 기술한다.

2. 제안된 능동 노드 구조

능동 노드의 구조는 크게 노드운영체제(Node Operating System), 실행환경(Execution Environment), 능동 응용(Active Application)으로 구분된다.

능동 응용은 능동 패킷을 처리하기 위한 실행코드이다. 사용되는 횟수가 작거나 실행코드의 크기가 작은 경우는 능동 패킷의 데이타에 포함되어 전달되고, 자주 사용되거나 실행코드의 크기가 큰 경우는 능동코드를 전달하는 패킷을 통해 능동 노드에 미리 적재되는 특성을 가진다. 예를 들어, 실행코드를 위해 필요한 라이브러리의 경우 미리 적재되어야 한다.

실행환경은 능동 패킷을 받고 실행코드를 통해 패킷을 처리한 후 결과를 다른 응용이나 노드운영체제로 전달하는 기능을 수행한다. 실행환경은 일종의 가상머신(virtual machine)과 같다.

노드운영체제는 다른 능동 노드로부터의 능동 패킷을 받아서 이를 적절한 실행환경으로 분배하고, 실행환경에서 해당 능동 패킷을 처리하기 위해 필요한 노드 자원을 관리(할당/제어/반환)하며, 실행환경으로부터의 결과

를 다른 노드로 전달하는 기능을 수행한다

그림 1은 제안된 능동 노드의 구조를 보여준다.

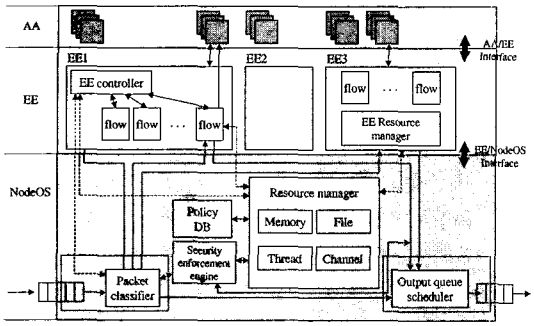


그림 1 능동 노드 구조

2.1 노드운영체제

노드운영체제는 기본적으로 DARPA 능동 네트워크 노드운영체제 인터페이스[10]를 따른다. [10]에서의 노드운영체제는 크게 패킷 분류기(packet classifier), 자원 관리기(resource manager), 출력 큐 관리기(output queue scheduler)로 구성된다.

2.1.1 패킷 분류기

패킷 분류기는 다른 노드로부터의 패킷을 분류하고 패킷이 처리될 적절한 개체로 전달한다. 패킷은 실행코드의 실행을 수반하는 능동 패킷이거나 일반적인 패킷 전달 기능만 필요로 하는 IP 패킷일 수 있다. 따라서 우선 능동 패킷인지 IP 패킷인지를 구별하고, 만약 일반 IP 패킷인 경우는 코드 실행을 필요로 하지 않기 때문에 곧바로 출력 큐(output queue)로 전달한다. 능동 패킷인 경우는 분류기내의 분류 키(demux key) 테이블에 등록된 분류 규칙과 비교해서 만약 일치된 플로우가 존재하면 기설정된 해당 플로우로 전달하고, 그렇지 않은 경우는 패킷에 명시된 실행환경으로 전달한다. 분류 키 테이블의 항목을 등록하거나 삭제하는 작업은 실행환경과의 상호동작을 통해 이루어지며, 자세한 패킷 분류과정과 패킷의 처리 방법은 다음절(3.4절 참조)에서 기술한다.

2.1.2 자원 관리기

자원 관리기는 실행환경에서 패킷을 처리하기 위해 필요한 자원을 관리(할당/제어/반환)한다. 실행환경으로부터의 논리적인 자원 요청에 대해 물리적인 자원을 효율적이고 공정하게 관리하기 위해, DARPA 노드운영체제 인터페이스[10]에서는 크게 다섯 가지 추상화를 정의하고 있다.

노드의 물리적인 자원은 크게 "메모리(memory)", "쓰레드(thread)", "채널(channel)", "파일(file)"로 구분한다.

- 메모리 : 실행환경에서 실행코드를 적재하고, 입·출력 채널에 패킷을 저장하며, soft-state 데이타를 저장하기 위해 필요한 메모리를 제공한다.
- 채널 : 각 플로우에 대한 논리적인 채널을 제공한다. 능동 패킷을 처리하기 위해 설정된 플로우는 각각 하나의 입력채널(input channel)과 출력채널(output channel)을 가지며, 일반 IP 패킷은 컷쓰로우채널(cut-through channel)을 통해 처리된다.
- 쓰레드 : 코드의 실행과 채널상의 패킷 처리 등을 위해 필요한 CPU 자원을 제공한다.
- 파일 : 한 플로우에 대한 지속적인 데이타나 플로우들 간의 공유 데이타를 위한 저장 장치를 제공한다.

같은 특성을 가지는 하나의 패킷 플로우를 처리하기 위해 필요한 모든 자원은 "플로우(flow)"로 추상화한다. 하나의 패킷 플로우가 노드운영체제에 들어왔을 때 필요한 채널, 메모리, 쓰레드, 파일은 하나의 플로우로 구별되며, 노드운영체제는 각 플로우별로 자원을 할당/제어/반환한다.

2.1.3 출력 큐 처리기

출력 큐 처리기는 패킷 분류기나 플로우의 출력채널로부터 돌어온 출력 패킷을 다른 노드로 전달한다. 이때 패킷 분류기로부터 받은 일반 IP 패킷들과 각 플로우의 출력채널로부터의 패킷에 대한 네트워크 대역폭을 공정하게 분배해야 한다.

이외에도 자원 관리기에서 플로우에 대한 자원을 할당할 때 적용할 정책을 갖고 있는 정책 데이타베이스(policy database)와 패킷 분류기와 출력 큐 처리기에서 필요한 능동 패킷의 인증, 암호화, 권한부여 등과 같은 능동 패킷의 보안과 노드 자신의 보안을 보장하는 보안 강제 엔진(security enforcement engine)이 있다.

2.2 실행환경

실행환경은 노드운영체제로부터 능동 패킷을 받고, 적절한 실행코드를 수행한 후 결과를 노드 자신의 응용이나 노드운영체제로 전달하는 작업을 수행한다. 수행하고자 하는 실행코드는 능동 응용으로부터 가져올 수 있지만, 만약 필요한 실행코드가 노드 자신에게 없으면 다른 노드로부터 가져 와야 한다. 코드 요청 방식은 다음 절(3.5절 참조)에서 살펴보도록 하겠다.

코드를 수행하기 위해서는 노드운영체제로부터 필요한 자원을 할당받아야 한다. 입력 채널을 위한 메모리와 채널에서의 패킷 처리를 위한 쓰레드, 코드를 적재하기 위한 메모리, 코드를 실행하기 위한 CPU 시간 등이 필

요하다.

자원을 관리하는 방법에 따라 실행환경은 크게 두 가지 형태로 존재한다.

· 다중 플로우 실행환경(multiple flow execution environment)

다중 플로우 실행환경은 실행환경에 속한 플로우들의 자원을 직접 관리하지 않고, 단지 플로우를 생성하고 생성된 플로우가 자원을 할당받도록 노드운영체제에게 요청할 뿐이다. 이를 수행하는 개체가 실행환경 제어기(EE controller)이다.

· 단일 플로우 실행환경(single flow execution environment)

단일 플로우 실행환경은 노드운영체제로부터 필요한 자원을 할당받고, 실행환경 내에서 패킷 플로우에 대한 구분과 자원 관리를 수행한다. 이를 수행하는 개체가 실행환경 자원 관리자(EE resource manager)이다.

2.2.1 실행환경 제어기

실행환경 제어기는 능동 패킷이 수행되기 위해 필요한 기능들을 수행한다.

· 능동 패킷/코드 해석(active packet/code interpretation)

노드운영체제로부터의 능동 패킷은 헤더의 "Type" 필드(3.3절 참조)를 통해 패킷 플로우의 종류가 구별되며, 이에 따라 패킷의 처리 과정이 달라진다. 새로운 플로우를 만들거나, 패킷에 포함된 능동 코드를 해석하고 필요한 코드를 메모리에 적재하고 해당 코드를 수행하거나, 실행환경 자신에 대한 관리 기능을 수행한다.

· 능동 코드 적재(active code loading)

패킷에 포함된 능동 코드를 수행하기 위해 필요한 코드가 노드에 없는 경우, 다른 노드로부터 능동 코드를 전달받아야 한다. 자세한 내용은 다음절에서 다루기로 하겠다.

· 플로우 관리(coarse-grained flow management)

새로운 플로우를 생성하는 경우, 해당 플로우에 대한 분류 키를 분류 키 테이블에 등록함으로써 이후 해당 플로우로 패킷이 전달될 수 있게 하고, 노드운영체제에게 해당 플로우에 대한 자원 할당을 요청한다. 즉, 실행환경 자신에게 할당된 자원중의 일부를 새로운 플로우에게 할당하지 않고, 노드운영체제가 직접 관리(할당/제어/반환)한다. 플로우가 종료되는 경우, 해당 플로우에 대한 분류 키를 분류 키 테이블에서 삭제해 종료된 플로우로 패킷이 전달되지 못하게 한다.

2.2.2 실행환경 자원 관리자

실행환경 자원 관리자는 위 실행환경 제어기와 마찬가지로 능동 패킷이 수행되기 위해 필요한 기능들을 수행한다. 따라서 기본적으로 실행환경 제어기에서와 같은 기능들을 수행하지만, 플로우의 자원을 직접 관리하는 좀더 정교한 플로우 관리(fine-grained flow management)를 제공하는 차이점이 있다. 즉, 새로운 플로우를 생성할 때, 자신이 노드운영체제로부터 할당받은 자원중의 일부를 해당 플로우에게 할당/제어/반환한다. 다시 말해 실행환경 자원 관리자는 실행환경 내에서의 운영체제라고 할 수 있다.

실행환경 자원 관리자는 실행환경을 좀더 복잡하게 만들지만 다음과 같은 장점을 가지고 있다. 첫째, 확장성을 가진다. 다중 플로우 실행환경에서는 플로우의 수가 증가함에 따라 노드운영체제가 수행해야 할 작업들(플로우 구별, 스케줄링, 감시)의 처리량이 많아지지만, 실행환경 자원 관리자를 가지는 단일 플로우 실행환경에서의 노드운영체제는 플로우의 수에 영향받지 않는다. 둘째, 노드운영체제의 안전성이 보장된다. 잘못된 능동 코드 수행시 이는 실행환경에서 처리되며 노드운영체제는 영향받지 않는다. 셋째, 실행환경은 자치성(autonomy)을 보장받는다. 실행환경은 플로우를 처리하기 위해 필요한 작업을 노드운영체제와 독립적으로 수행할 수 있다. 예를 들어, 노드운영체제와 서로 다른 스케줄링 기법을 사용할 수 있다.

3. 주요 기능 설명

본 절에서는 능동 응용의 요구 사항에 따라 능동 패킷의 특성을 구분하고 이를 능동 노드 구조, 능동 패킷 구조, 능동 패킷 라우팅 방식 등에 반영시키는 것에 대해서 다룬다.

우선 능동 응용은 패킷들간의 연관성에 따라 다음과 같이 분류할 수 있다.

능동 응용은 능동 패킷이 서로 독립적으로 처리되는 응용과, 같은 특성을 가지면서 능동 패킷들간의 상호운영성을 필요로 하는 응용으로 구분할 수 있다. 예를 들어, 전자의 경우에 해당하는 응용으로는 ping이 있다. Ping 응용에서 같은 목적지로 여러 개의 패킷을 보내는 경우라도, 각 패킷들은 서로 아무런 관계를 가지지 않기 때문에 패킷을 처리한 결과를 저장할 필요가 없다. 후자의 경우에 해당하는 응용으로는 멀티캐스트 프로토콜이 있을 수 있다. 멀티캐스트 프로토콜들은 하나의 멀티캐스트 세션을 설정하기 위해 우선 멀티캐스트 트리를 구성한다. 멀티캐스트 트리는 라우터에서 해당 그룹에 대해 멀티캐스트 패킷을 전달하기 위해 필요한 상위링크(uplink)와 하위링크(downlink)를 설정하는 것으로 구성

된다. 이 트리 정보는 세션이 설정되어 있는 동안 연속적인 이후의 멀티캐스트 패킷을 처리하기 위해 노드에 저장되어야 한다. 즉, 같은 세션에 속하는 멀티캐스트 패킷들은 같은 특성을 가지고 있으며 이후의 패킷에 영향을 준다.

능동 응용은 또한 능동코드를 수행하는 노드의 위치에 따라 구분할 수 있다.

능동 패킷의 처리가 임의의 능동 노드에서 수행될 수 있는 능동 응용과 지정된 능동 노드에서만 처리될 필요가 있는 능동 응용이 있다.

본 논문에서는 위에서 기술한 두 가지 범주에 따른 패킷의 특성을 반영하기 위해, 우선 전자를 위해 능동 패킷을 네 가지 패킷 플로우로 구분해서 정의하고, 후자를 위해 세 가지 라우팅 방법을 제안한다.

3.1 패킷 플로우

능동 패킷은 패킷의 특성 및 처리과정에 따라 다음 네 가지 패킷 플로우로 구분된다.

· Non A-flow

Non A-flow 패킷은 같은 능동 응용에 의해 생성되는 다른 능동 패킷들과는 무관하게 독자적으로 전송 및 처리되는 패킷이다. 따라서 능동 노드에서는 같은 특성을 가지는 non A-flow에 속한 패킷들도 각각 독립적으로 처리하기 때문에 non A-flow를 위한 자원을 기설정할 필요가 없다. 패킷 분류기는 non A-flow에 속한 패킷을 기설정된 플로우로 전달할 필요가 없으며, 단지 패킷을 실행환경으로 전달하면 된다. 이는 패킷 분류기에서 패킷 분류를 위해 참조하는 분류 키 테이블에 해당 non A-flow에 대한 항목을 유지하지 않음으로써 가능하다. 실행환경은 패킷을 받고 패킷에 포함된 능동 코드를 수행한 후 결과를 노드 자신의 다른 능동 응용이나 다른 노드로 전달함으로써 패킷 처리과정을 마친다.

실행환경에서 능동 패킷의 능동코드를 수행하기 위해서는 우선 능동코드가 노드에 적재되어 있어야 하며, 만약 노드에 적재되어 있지 않다면 해당 코드를 다른 노드에게 요청해야 한다. 같은 non A-flow에 속한 패킷들은 패킷이 서로 독립적으로 처리되며 패킷의 전달 경로가 일정하지 않기 때문에 코드 요청 주소가 같더라도 각 패킷마다 코드 요청 주소가 명시되어야 한다.

· A-flow

A-flow 패킷은 같은 능동 응용에 의해 생성되는 다른 패킷들과 상호 연관성(시간, 자원 할당, 데이타 공유)을 갖는 패킷이다. 상호 연관성을 유지하기 위해서는 능동 응용의 소스에서 목적지까지의 중간 능동 노드에서 이러한 패킷의 처리와 실행결과에 대한 상태 정보를 유지해야 한다. 즉, 중간 능동 노드에서 해당 A-flow에 대한 플로우를 생성하고 A-flow가 종료될 때까지 이를 지속적으로 유지해야 하며, 패킷 분류기가 같은 A-flow에 속한 패킷들을 같은 플로우에 전달시켜 주기 위해 분류 키 테이블에 해당 A-flow에 대한 항목을 유지해야 한다. 노드에서 A-flow에 대한 플로우를 기설정하기 위해서는 A-flow를 고유하게 구별할 수 있는 정보와 코드 요청 주소 등에 대한 정보를 필요로 한다. Non A-flow 패킷은 각 패킷이 독립적으로 처리되기 때문에 코드 요청 주소와 같은 정보를 패킷마다 명시해야 하지만, A-flow 패킷의 경우는 패킷들이 같은 특성을 가지며 같은 경로로 전달되기 때문에 이러한 정보는 송신지에서 목적지까지의 중간 노드에서 플로우가 기설정될 때만 필요하다. 따라서 하나의 A-flow에 속한 각 패킷에 중복된 정보를 포함시키는 것을 피하면서 플로우의 생성과 종료 등에 관련된 작업을 수행하도록 하기 위해 신호 능동 플로우(signaling flow)를 정의하였다. 신호 능동 패킷을 통해 소스에서 목적지까지의 중간 노드들에서 플로우가 생성되고 이후 A-flow에 속한 패킷들이 전달되기 때문에, A-flow에 속한 패킷을 받은 노드는 패킷 분류기를 통해 빠르게 패킷을 분류하고, 기생성된 플로우를 사용해 처리하기 때문에 패킷을 고속으로 처리할 수 있다. 더욱이 신호 능동 패킷에서 패킷의 분류키와 코드 적재에 필요한 정보를 제공하기 때문에, A-flow 패킷의 헤더는 단지 기본적으로 필요한 Version과 Type 부필드(3.3절 참조)만 존재하는 간단한 형태의 패킷이 된다.

A-flow는 고속의 패킷 처리와, QoS 보장을 요구하는 실시간 응용에 적합하다.

· 신호 능동 플로우(signaling flow)

신호 능동 패킷은 A-flow에서 필요한 플로우 생성 및 종료시 필요한 정보와 코드 적재시 필요한 정보를 교환하기 위해 사용되는 패킷이다.

따라서 신호 능동 패킷은 해당 A-flow에 대한 플로우의 생성/종료 요청을 명시하고, A-flow를 구별하기 위해 필요한 Demux_Key 정보와 해당 A-flow의 코드가 위치한 주소 정보를 포함한다. 신호 능동 패킷은 선택적으로 코드 적재 과정의 효율성을 위해 패킷의 데이타 부분에 다른 노드에서 필요할 만한 일반적인 능동 코드를 포함시킴으로써 코드 요청 횟수와 비용을 줄일 수 있다.

· 관리 플로우(management flow)

관리 패킷은 서로 다른 노드에 위치한 같은 실행환경들간의 관리 정보 교환을 위해 사용되는 패킷으로, 본

논문에서는 아직 자세한 패킷의 형식과 처리 방식을 정의하지 않은 상태이다.

3.2 라우팅

능동 패킷의 라우팅은 크게 세 가지 방식으로 구분된다. 다음 세 가지 라우팅 방식은 인터넷에서 사용되는 IP 패킷의 라우팅 방법과 같은 개념을 가지지만, 라우팅에 관여하는 라우터가 해당 실행환경을 적재하고 있는 능동 노드이며 라우팅 테이블이 실행환경 특정(EE-specific) 라우팅 테이블이라는 차이점이 있다.

· hop-by-hop 라우팅

모든 능동 패킷은 기본적으로 hop-by-hop 라우팅으로 동작한다. 일반 IP 패킷의 라우팅 방식과 같으며, 능동 패킷이 전달될 다음 능동 노드는 실행환경이 자신의 실행환경 특정 라우팅 테이블을 참조하거나 능동 패킷의 처리결과를 통해서 결정된다. 따라서 송신지에서 목적지까지의 능동 노드가 임의의 노드가 될 수 있으며, 중간 노드들 전체에서 능동 패킷의 처리를 필요로 하는 응용에 적합하다. 예를 들어, 원격지에 있는 공격자를 찾는 능동 응용의 경우 송신지부터 공격자가 있는 목적지 노드까지의 모든 중간 능동 노드에서 능동 패킷이 처리되어야 하며, 각 중간 능동 노드에서의 처리결과는 목적지 노드로의 다음 능동 노드가 된다.

· direct 라우팅

이 라우팅 방식은 송신지에서 목적지까지의 중간 능동 노드에서의 패킷처리를 필요로 하지 않는 경우에 적합하다. 예를 들어, 위에서 공격자가 있는 목적지를 찾았을 때 송신자는 공격자가 있는 네트워크에 있는 능동 노드에게 공격자로부터의 패킷을 외부로 전달하지 않도록 필터링을 요청할 수 있다. 이러한 경우 송신지에서 목적지(공격자 네트워크의 라우터)까지의 중간 능동 노드는 이 능동 패킷을 처리할 필요가 없으며, 능동 패킷은 직접 목적지로 전달되어야 한다.

· source 라우팅

이 라우팅 방식은 IP 패킷의 source 라우팅과 같은 방식으로 송신지와 목적지간의 중간 능동 노드들 가운데 몇몇 선택된 능동 노드만 해당 능동 패킷을 처리하도록 요청하는 응용에 적합하다.

3.3 패킷 구조

능동 패킷은 위에서 제시한 패킷 플로우와 라우팅 방식을 표현해야 하며, 본 논문에서 제시하는 능동 패킷의 형식은 그림 2에 나타나 있다.

능동 패킷의 형식은 패킷 플로우의 종류에 따라 달라진다. 또한 능동 응용이 어느 계층에서 구현되는가에 따라 IP 헤더나 UDP 헤더와 같은 하위 계층 프로토콜들의 헤더가 오며, 그 다음에 실행환경을 구분하기 위해 필요한 ANEP[2] 헤더가 온다. ANEP 헤더 뒤에는 본 논문에서 제시하는 능동 패킷 특정 헤더가 온다.

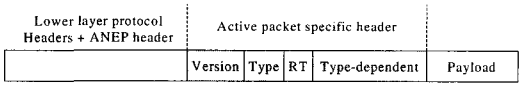


그림 2 능동 패킷 형식

능동 패킷 특정 헤더는 Version, Type, Routing Type, Type-dependent 필드들로 구성된다.

· Version 필드 (3비트) : 실행환경의 버전
· Type 필드 (2비트) : 패킷 플로우에 따른 능동 패킷의 종류. 신호(signaling) 능동 패킷 (00), 관리(management) 능동 패킷 (01), non a-flow 능동 패킷 (10), a-flow 능동 패킷 (11)
· Routing Type 필드 (2비트) : 라우팅 방법. hop-by-hop 라우팅 (00), direct 라우팅 (01), source 라우팅 (10)
· Type-dependent 필드 (가변길이) : 능동 패킷의 종류와 라우팅 방법에 따른 서로 다른 형식의 부필드(subfield)들을 가짐

좀 더 빠른 패킷 처리를 위해서 각 필드들은 바이트 단위의 경계로 구분되며, Version, Type, Routing Type과 Type-dependent 헤더의 첫 번째 비트는 한 바이트를 이룬다. Routing Type과 type-dependent는 능동 패킷의 형식에 따라서 사용되며, 사용되지 않을 경우는 예약(reserved; R) 필드로 남겨둔다.

신호 능동 패킷의 형식은 그림 3에 나타나 있다.

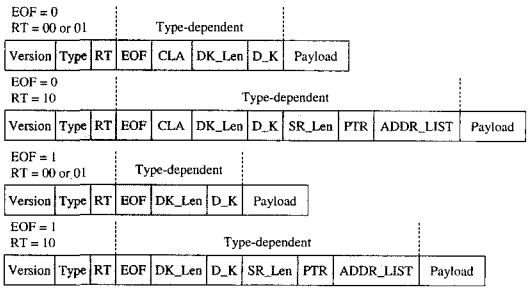


그림 3 신호 능동 패킷 형식

신호 능동 패킷은 End of A-Flow(EOF: 1비트), Code Location Address(CLA: 4바이트), Demux_Key length(DK_Len: 1바이트), Demux_Key(D_K: 가변) 부

필드들을 가진다. 그리고 라우팅 방법에 따라 source route length(SR_Len: 1바이트), current pointer(PTR: 1바이트), address list(ADDR_LIST: 가변) 부필드를 가진다. EOF 부필드는 A-flow의 시작(EOF: 0)과 종료(EOF: 1)를 나타낸다.

Demux_Key에 포함되는 패킷 분류 정보는 가변이며, Demux_Key length 필드에 길이가 명시된다. Demux_Key에 포함되는 패킷 분류 정보는 IPv6[16]와 같이 특정 플로우의 구별을 위해 플로우에 속한 모든 패킷들에 부여되는 일정한 값(비트열)이 아닌 Bowman[15]에서와 같이 패킷을 구별하는 방법을 명시하는 정보이다. 예를 들어, 송신자의 IP 주소는 1.1.1.1, 수신자의 IP 주소는 2.2.2.2, UDP 포트 번호가 3333인 패킷들을 하나의 플로우로 정의하는 경우, Demux_Key의 값을 ((src_ip=1.1.1.1)&(dsr_ip=2.2.2.2)&(udp=3333))과 같이 나타낼 수 있다. Demux_Key에 포함되는 정보의 형식은 아직 정의하지 않았으며, 향후 패킷 분류기를 구현하는 과정에서 좀더 효율적인 표기 방식을 정의할 것이다. 위와 같이 Demux_Key에 플로우를 구별하는 방법을 명시함으로써 A-flow 패킷은 플로우의 생성과 종료시 플로우 구별을 하기 위해 필요한 Demux_Key 필드를 포함하지 않아도 된다.

EOF가 0일 때에는, EOF 비트 다음에 CLA, Demux_Key 부필드들이 따라온다. CLA 부필드는 해당 A-flow 능동 패킷들을 처리하기 위해 필요한 코드가 존재하는 노드의 주소에 대한 정보를 나타낸다. CLA 부필드는 이전 능동 노드의 주소나 코드 서버의 주소, 또는 0값을 가질 수 있다. 0값은 Payload 부분에 코드가 포함되어 있음을 의미하며 신호 능동 패킷을 처리할 때 Payload에 포함된 능동코드가 적재된다. CLA가 0이 아닌 값을 갖고 있을 경우에는 능동 패킷의 Payload 부분은 존재하지 않으며, 이후의 A-flow 패킷을 처리할 때 필요한 능동코드를 요청할 주소로서 사용된다. Demux_Key 부필드는 해당 A-flow 능동 패킷들을 구분하기 위한 정보를 나타내며, 실행환경은 이 정보를 패킷 분류자에게 전달함으로써 분류 키 테이블에 해당 항목이 생성되고, 이후 A-flow 능동 패킷이 기설정된 해당 플로우로 전달되도록 한다.

EOF가 1일 때에는, EOF 비트 다음에 Demux_Key 부필드가 따라온다. 실행환경은 해당 Demux_Key 부필드에 명시된 분류 키를 가진 플로우를 종료시키고, 패킷 분류자에게 해당 플로우에 대한 항목을 분류 키 테이블에서 삭제하도록 요청한다.

RT가 10(source 라우팅)일 때에는 SR_Len, PTR, ADDR_LIST 부필드가 따라오며, 각 부필드들(SR_Len, PTR, ADDR_LIST)의 의미와 사용방법은 IP 패킷의 source route option과 같다.

Non A-flow 패킷의 형식은 그림 4에 나타나 있다.

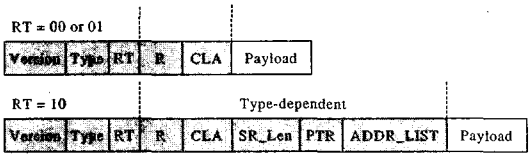


그림 4 Non A-flow 능동 패킷 형식

Non A-flow 능동 패킷은 CLA 부필드를 가지며, RT가 10인 경우에는 신호능동 패킷의 경우와 같이 SR_Len, PTR, ADDR_LIST 부필드를 가진다. 그리고 각 부필드는 신호 능동 패킷에서와 같은 의미를 지니고 처리된다.

A-flow 능동 패킷의 형식은 그림 5에 나타나 있다.

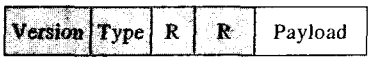


그림 5 A-flow 능동 패킷 형식

A-flow 능동 패킷은 Type-dependent 필드가 존재하지 않는다. 앞서 설명한 것과 같이 A-flow 패킷은 신호 능동 패킷을 통해 플로우가 생성되고 라우팅 경로가 설정된 후에 전송되기 때문에 추가적인 정보가 필요하지 않으며, 따라서 type-dependent 필드가 필요하지 않다. 이와 같이 A-flow 능동 패킷 헤더 구조를 간단히 함으로써 A-flow 능동 패킷을 고속으로 처리할 수 있는 장점이 있다.

3.4 능동 패킷 처리 과정

플로우의 종류와 그에 따른 노드의 각 구성요소들에서의 패킷 처리 과정을 좀 더 자세히 살펴보면 다음과 같다.

패킷을 처리하기 위해서는 우선 패킷의 종류를 구분하고 적절한 구성요소로 패킷을 전달하는 분류 과정이 필요하다.

3.4.1 패킷 분류 과정

패킷의 분류 과정은 크게 2단계로 구분된다.

그림 6은 패킷의 분류 과정을 나타낸다.

첫 번째 분류 과정은 패킷 분류기에서 수행되며, 패킷 내의 ANEP[2] 헤더의 유무에 따라 일반 IP 패킷인지 능동 패킷인지를 결정한다. ANEP 헤더가 존재하지 않

으면 일반 IP 패킷이며, 패킷을 곧바로 출력 큐로 전달한다. ANEP 헤더가 존재하면 해당 패킷은 능동 패킷이며, ANEP 헤더의 "Type ID" 필드와 능동 패킷 특정 헤더의 "Type" 필드와 분류 키 테이블을 통해 해당 능동 패킷을 전달할 실행환경이나 플로우를 선택한다. 우선 능동 패킷을 분류 키 테이블의 항목과 일치하는지 검사해서, 만약 일치하는 항목이 존재하면 이는 A-flow 능동 패킷이며 패킷을 해당 플로우의 입력 채널로 전달하며, 일치하는 항목이 존재하지 않으면 ANEP 헤더에 명시된 실행환경으로 패킷을 전달한다.

두번째 분류 과정은 실행환경에서 수행되며, 패킷의 "Type" 필드의 값에 따라 각각 신호 능동 패킷(값: 00), 관리 패킷(값: 01), non A-flow 패킷(값: 10)으로 분류한다.

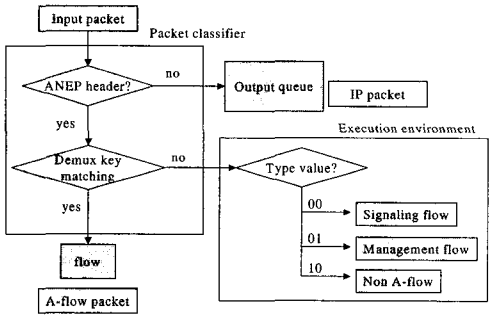


그림 6 패킷 분류 과정

3.4.2 패킷 처리 과정

플로우의 종류에 따른 능동 패킷의 처리과정은 다음과 같다.

· 신호 능동 플로우

신호 능동 패킷은 패킷 분류과정을 거쳐 실행환경으로 전달된다. 실행환경은 신호 플로우의 EOF 필드의 값에 따라 다른 과정을 수행한다. EOF 필드가 0인 경우(플로우 생성 요청)는 해당 A-flow를 위한 하나의 플로우를 생성하고 필요한 자원을 노드운영체제에게 요청한다.

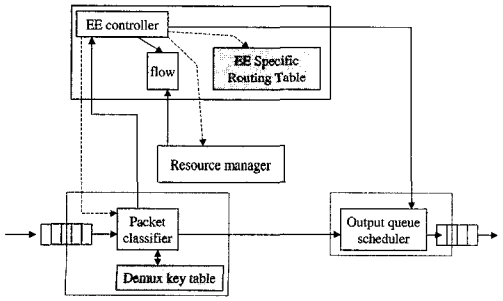


그림 7 신호 능동 패킷 처리 과정

Demux_Key 필드에 명시된 정보를 통해 패킷 분류기에게 해당 A-flow에 대한 분류 규칙을 분류 키 테이블에 등록하도록 요청한다. 다음으로 CLA의 값이 0이 아닌 경우는 CLA에 명시된 능동 노드의 주소를 생성된 플로우에게 할당함으로써 해당 플로우가 CLA 필드에 명시된 노드에게 코드 적재를 요청하게 한다. CLA의 값이 0인 경우는 패킷의 Payload 부분에 포함된 능동 코드를 해석하고 코드 캐시에 적재한다.

신호 능동 패킷은 A-flow에 대한 플로우 생성/종료와 경로 설정을 위해 사용되는 플로우이며 A-flow의 패킷에는 이에 대한 정보가 없기 때문에, A-flow에 속한 패킷이 일정한 경로로 전달되기 위해서는 능동 노드에 패킷이 전달될 다음 능동 노드의 주소를 유지해야 한다. 이 정보는 플로우 생성시 실행환경 특정(EE specific) 라우팅 테이블에 저장되며, 플로우 종료시 삭제된다. 실행환경 특정 라우팅 테이블은 [flow id, 다음 능동 노드 주소] 형식의 항목을 가진다. 다음 능동 노드의 주소는 패킷의 "Routing Type" 필드와 부필드(source 라우팅의 경우), 실행환경이 가지고 있는 능동 노드로 구성된 네트워크의 토폴로지 정보를 통해 얻을 수 있다. 먼저 라우팅 방식이 hop-by-hop(값: 00)인 경우는 실행환경에서 유지하고 있는 토폴로지 정보를 바탕으로 다음 능동 노드를 찾을 수 있으며, source 라우팅인 경우(값: 10)는 헤더에 명시된 경로 정보를 통해 다음 능동 노드를 찾을 수 있다. Direct 라우팅의 경우는 패킷의 송신지에서 목적지로 곧바로 전달되기 때문에 다음 능동 노드의 주소는 목적지 주소가 된다.

마지막으로 플로우의 생성이 끝나고 능동 노드 자신이 목적지가 아니면 위의 과정을 통해 얻은 다음 능동 노드로 신호 능동 패킷을 전달해서 목적지까지의 중간 능동 노드에서 플로우의 생성이 이루어지도록 한다.

EOF 값이 1인 경우(플로우 종료 요청)는 Demux_Key에 명시된 플로우를 종료시키고, 분류 키 테이블과 실행환경 특정 라우팅 테이블에서 해당 항목을 삭제하고, 자신이 목적지가 아니면 다음 능동 노드로 패킷을 전달해서 목적지까지의 중간 능동 노드에서 플로우를 종료시키도록 한다.

· A-flow

A-flow는 패킷 분류 과정을 거쳐 노드에 기설정된 플로우의 입력 채널로 전달된다. 패킷은 기설정된 플로

우의 자원을 이용해 처리되며, 실행 결과를 노드 자신의 능동 응용이나 실행환경 특정 라우팅 테이블에 명시되어 있는 다음 능동 노드로 전달한다.

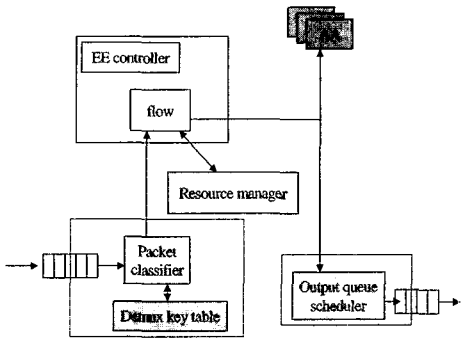


그림 8 A-flow 패킷 처리 과정

· Non A-flow

Non A-flow는 패킷 분류과정을 거쳐 실행환경으로 전달된다. Non A-flow 패킷 또한 A-flow 패킷과 같이 능동 코드 수행을 필요로 하지만, 기설정된 자원(즉, 플로우)이 없기 때문에 실행환경은 일시적인 플로우를 생성하고 해당 non A-flow 패킷을 처리한 후 플로우를 곧바로 종료시킨다. 능동 패킷을 실행하기 위해 필요한 능동 코드가 없는 경우는 CLA에 명시된 능동 노드에게 코드를 요청하며, 실행 결과는 노드 자신의 능동 응용이나 RT에 명시된 라우팅 방법에 따라 다음 능동 노드의 주소로 전달한다.

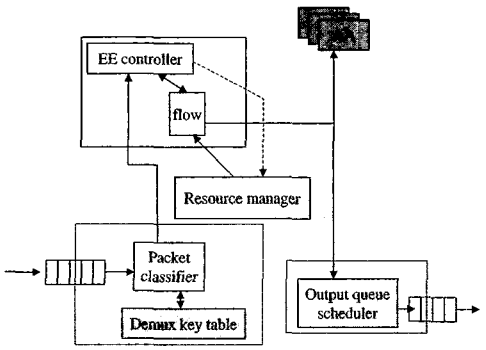


그림 9 Non A-flow 패킷 처리 과정

3.5 능동 코드 적재 과정

능동 코드 적재 과정은 능동 코드가 있는 노드의 위치에 따라 구분된다. 이전 노드로부터 능동 코드를 가져오는 방법과 기설정된 코드 서버로부터 능동 코드를 받아오는 방법이 있다. 본 논문에서는 이 두 가지를 모두 지원할 수 있는 구조를 사용하도록 제안한다. 이는 서로 다른 능동 코드 요청 상황과 배포 방법을 만족시킬 수 있기 때문이다. 예를 들어, 코드를 가지고 있지 않은 송신지의 경우 필요한 코드는 코드 서버로부터 전달받아야 한다. 그리고 다음 능동 노드는 이전 노드로부터 필요한 코드를 전달받을 수 있다. 또한 능동 코드가 송신지, 중간 능동 노드, 목적지 노드에서 필요한 코드가 분리되어 있는 경우 중간 능동 노드는 송신지로부터 필요한 코드를 받을 수 없고 코드 서버에게 코드를 요청해야 한다.

능동 코드가 위치한 노드(코드 요청 주소)를 선택하기 위해서는 우선 능동 패킷(신호 능동 패킷, non A-flow 패킷)의 CLA에 명시된 노드를 선택하고, 만약 CLA에 대한 정보를 패킷으로부터 전달받지 못한 경우는 기설정된 코드 서버를 선택한다.

그림 10은 코드 적재 과정의 예를 보여준다.

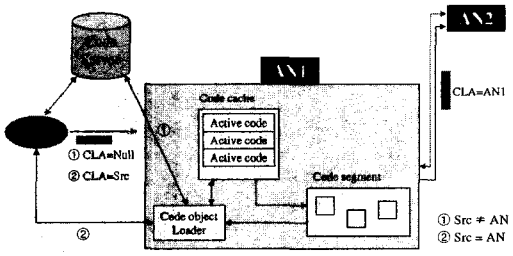


그림 10 코드 적재 과정

①의 경우는 송신지에서 CLA의 값을 설정하지 않고 능동 패킷을 보내고, AN1에 필요한 코드가 없는 경우에 해당한다. 이 경우 AN1의 코드 객체 적재기(code object loader)는 필요한 코드를 코드 서버로부터 할당받아야 한다. 이 경우에는 CLA의 값에 코드 서버의 주소를 지정할 수도 있다.

②의 경우는 송신지에서 CLA에 자신의 주소를 설정함으로써 AN1이 이전 노드로부터 코드를 요청받을 수 있도록 한 경우이다.

4. 결론 및 향후 연구 방향

다양한 능동 응용의 요구사항을 위해 능동 패킷을 A-flow, non A-flow, 시그널링 플로우, 관리 플로우로 구분하고, 다양한 라우팅 방법 (hop-by-hop, direct, source)을 지원할 수 있는 능동 노드의 구조를 제안하

였다. 제안된 능동 노드에서는 능동 응용들을 좀더 빠르고 효율적으로 개발할 수 있다. 그러나 아직까지 관리 플로우에 대해서는 정확한 동작 과정이나 패킷 형식에 대해 정의하지 못했으며, 실행환경들간에 필요한 동작들을 연구함으로써 관리 플로우에 대한 정의를 향후 내려야 할 것이다. 또한 본 논문에서 제시한 능동 노드의 구조를 구현하고 성능을 평가하는 과정이 필요하며, 이 과정에서는 성능, 안전성 및 보안, 프로그래밍 언어 등에 관한 추가적인 연구를 수행할 예정이다.

참 고 문 헌

[1] D. Wetherall, et. al., "The Active IP Option," 7th ACM SIGOPS European Workshop, 1996.

[2] D. Alexander, et. al., "Active Network Encapsulation Protocol (ANEP)," http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt, 1997.

[3] D. Decasper, et. al., "Simple Active Packet Format (SAPF)," 1998.

[4] T. Wolf, et. al., "Tags for High Performance Active Networks," IEEE OPENARCH, 2000.

[5] B. Schwartz, et. al, "Smart packets for Active Networks," BBN Technology, Jan. 1998.

[6] D. Alexander, et. al., "The SwitchWare Active Network Architecture," IEEE Network, May/June 1998.

[7] M. Hicks, et.al, "PLANet: An Active Internetwork," IEEE INFOCOM, 1999.

[8] D. Wetherall, et. al., "ANTS: a toolkit for building and dynamically deploying network protocols," Open Architectures and Network Programming 1998 IEEE, 1998.

[9] G. Alex, et. al., "A Flexible IP Active Networks Architecture," IWAN 2000 Conference, 2000.

[10] L. Peterson (Editor), "NodeOS Interface Specification," DARPA AN NodeOS Working Group, 1999.

[11] K. Calvert, "Architectural Framework for Active network," Active Network Working Group, 1999.

[12] Sumi shoi, et. al, "Design of a Flexible Open Platform for High-Performance Active Networks," 1999.

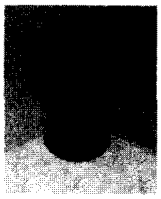
[13] T. Wolf, et. al, "A Scalable High-Performance Active Network Node," IEEE Network, 1999.

[14] P. Tullmann, et. al., "A Java-oriented OS for Active Network Nodes," IEEE Journal on Selected Areas of Communication. Volume 19, Number 3, March 2001.

[15] S. Merugu, et. al., "Bowman: A Node OS for Active Networks," Proceedings of IEEE Infocom 2000, March 2000.

[16] S. Deering, et. al., "Internet Protocol, Version 6 (IPv6) Specification," Requrest For Comment 1883, December 1995.

안 상 현
정보과학회논문 : 정보통신
제 29 권 제 1 호 참조



김 경 춘
2001년 2월 서울시립대학교 전산통계학과 졸업. 2001년 3월 ~ 현재 서울시립대학교 컴퓨터통계학과 석사 과정. 관심분야는 네트워크 보안, 능동 네트워크, 멀티캐스트

손 선 경
1999년 2월 전남대학교 전산학과 졸업. 2001년 2월 전남대학교 전산통계학과 석사. 2001년 2월 ~ 현재 한국전자통신연구원 연구원. 관심분야는 능동 보안, 네트워크 보안, Active Network

손 승 원
정보과학회논문 : 정보통신
제 29 권 제 2 호 참조