능동 응용을 고려한 능동 노드 구조

안상현* 김경춘*0 손선경** 손승원**
* 서울시립대학교 컴퓨터·통계학과
** 한국전자통신연구원 정보보호기술연구본부
ahn@venus.uos.ac.kr, netiv@orgio.net0, {sgsohn, swsohn}@etri.re.kr

Active Node Architecture considering Active Applications

Sanghyun Ahn* Kyeongchoon Kim*0 Seon-Gyoung Sohn** Sung-Won Sohn**
* Dept. of Computer Science & Statistics, University of Seoul
** Information Security Technology Division, ETRI

요 약

능동 네트워크 기술은 차세대 프로토콜 개발 방식으로 등장한 기술로서, 현재 여러 연구소와 대학에서 능동 노드의 구조와 능동 패킷의 형식을 제안하고 있다. 그렇지만, 현재까지 제안된 능동 노드의 구조와 능동 패킷의 형식을 살펴보면, 능동 패킷의 특성을 고려하지 않고 있다. 최근 인터넷은 사용자들의 요구사항을 보장하기 위해 패킷의 특성을 구분하고, QoS를 보장하기 위해 많은 연구가 진행되고 있으며, 능동 네트워크 영역에서도 이러한 기술이 적용되어야 한다. 본 논문에서는 이를 위해 능동 패킷의 특성에 따라 패킷의 형태를 네가지로 구분하고, 라우팅 방법을 세가지 형태로 구분하여 반영시킨 능동노드의 구조와 능동 패킷의 형식을 제안한다.

1. 서론

능동 네트워크 기술은 차세대 프로토콜 개발 방식으로 등장한 기술로서, 현재 여러 연구소와 대학에서 능동노드의 구조[5,6,7,12,13,15,16,17]와 능동 패킷의 형식[2,3,4,8,9,10,11,15]을 제안하고 있다. 우리는 현재까지 제안된 여러 능동노드의 구조와 패킷형식에 대해 살펴보았으며 몇가지 고려되고 있지 못하는 문제들을 발견했다. 능동 패킷의 트래픽 특성과 능동 네트워크 상에서의 라우팅에 관한 것이다. 초기 인터넷의 가장 큰 특징중의 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 트래픽의 특성에 따라 구분하지 않고 단일한 특성을 가지는 패킷으로 처리하고 있다. 본 논문에서는 능동 패킷의 특성에 따라 패킷의 형태를 네가지로 구분하고, 라우팅 방법을 세가지 형태로 구분하여 반영시킨 능동 노드의 구조와 능동 패킷의 형식을 제안한다.

본 논문의 2장에서는 제안된 능동노드의 전체적인 구조를 기술하고, 3장에서는 트래픽 특성과 라우팅을 고려한 능동 패킷의 형식과 라우팅 방법, 마지막으로 4장에서는 결론과 향후 연구 방향에 대해서 기술한다.

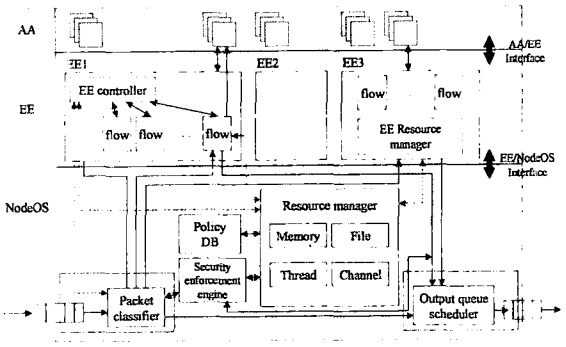
2. 제안된 능동노드 구조

[그림 1]은 제안된 능동노드의 구조를 보여준다.

능동노드의 구조는 크게 노드운영체제(Node Operating System), 실행환경(Execution Environment), 능동응용(Active Application)으로 구분된다.

노드운영체제는 DARPA 능동 네트워크 노드운영체제 인터페이스[6]를 따른다.

실행환경은 자원을 관리하는 방법에 따라 다중 플로우 실행환경(multiple flow EE)과 단일 플로우 실행환경(single flow EE)의 두가지 형태로 존재한다. 다중 플로우 실행환경은 실행환경에 속한 플로우들의 자원을 직접 관리하지 않고, 단지 플로우를 생성하고 생성된 플로우가 자원을 할당받도록 노드운영체제에게 요청해 주는 역할을 수행하며, 실행환경 제어기(EE controller)가 이 작업을 수행한다. 실행환경 제어기는 또한 플로우 생성시 이후의 해당 플로우로의 능동 패킷이 전달되도록 하기 위해 패킷 분류기(packet classifier)에게 분류 키(demux key) 테이블에 해당 플로우에 대한 분류 규칙을 등록하게 한다. 그리고 플로우 종료시에는 패킷 분류기에게 해당 분류 키 정보를 삭제하도록 요청한다. 실행환경 제어기는 이외에도 능동 패킷과 코드를 해석하고, 필요한 능동 코드를 적재하는 기능을 수행한다. 단일 플로우 실행환경은 노드운영체제로부터 필요한 자원을 할당받고, 실행환경 자신이 패킷 플로우를 생성하고 직접 플로우들의 자원을 관리한다. 이를 수



[그림 1] 능동노드 구조

행하는 개체가 실행환경 자원 관리자(EE resource manager)로서 기본적으로 실행환경 제어기와 같은 작업을 수행하지만, 직접적으로 좀더 정교한 플로우 관리(fine grained flow management)를 수행하는 차이점이 있다. 실행환경 자원 관리자는 실행환경내에서의 운영체제라고 할 수 있다.

실행환경을 두가지 형태로 구분하고, 좀더 복잡한 실행환경 자원 관리자를 사용하는 것은 다음과 같은 장점이 있기 때문이다. 첫째, 확장성을 가진다. 다중 플로우 실행환경에서는 플로우의 수가 증가함에 따라 노드운영체제가 수행해야 할 작업들(플로우 구별, 스케쥴링, 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 스케쥴링 기법을 사용할 수 있다.

3. 능동 패킷의 형식과 라우팅 방법

능동응용은 패킷들간의 연관성에 따라 다음과 같이 분류할 수 있다. 능동응용은 능동 패킷이 서로 독립적으로 처리되는 응용과 같은 특성을 가지며 능동 패킷들간의 상호운영성을 필요로 하는 응용으로 구분할 수 있다. 능동응용은 또한 능동코드를 수행하는 노드의 위치에 따라 구분할 수 있다. 능동 패킷의 처리가 임의의 능동노드에서 수행될 수 있는 능동응용과 지정된 능동노드에서만 처리될 필요가 있는 능동응용이 있다. 이에 따라 능동 패킷의 라우팅 방법이 달라지게 된다.

3.1 패킷 플로우

능동 패킷은 패킷의 특성과 처리과정에 따라 다음 네가지 패킷 플로우로 구분된다.

· Non A flow

Non A flow 패킷은 같은 능동응용에 의해 생성되는 능동 패킷들들이 서로 독립적으로 전송 및 처리되는 패킷이다. 따라서 능동노드에서는 non A flow를 위한 자원을 위해 플로우를 기설정하거나 패킷 분류 키 테이블에 분류 항목을 유지할 필요가 없다. 패킷 분류기는 non A flow를 단지 실행환경으로 전달해주면 된다. 실행환경은 패킷을 받고 패킷에 포함된 능동 코드를 수행한 후 그 결과를 노드 자신의 다른 능동응용이나 다른 노드로 전달함으로써 패킷 처리과정을 마친다. 같은 능동 응용에 속한 non A flow에 속한 패킷들은 서로 독립적으로 패킷이 처리되며 서로 다른 경로를 통해 전달될 수 있기 때문에, 코드 요청 주소가 같더라도 각 패킷마다 코드 요청 주소가 명시된다.

· A flow

A flow 패킷은 같은 능동응용에 의해 생성되는 다른 패킷들과 상호 연관성(시간, 자원 할당, 데이터 공유)을 갖는 패킷이다. 상호 연관성을 유지하기 위해서는 능동응용의 소스에서 목적지까지의 중간 능동노드에서 이러한 패킷의 처리와 실행결과에 대한 상태 정보를 유지해야 한다. 즉, 중간 능동노드에서 해당 A flow에 대한 플로우를 생성하고 A flow가 종료될 때까지 이를 지속적으로 유지해야 하며, 패킷 분류기가 같은 A flow에 속한 패킷들을 같은 플로우에 전달시켜주기 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. 우리는 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 설정된 이후에 A flow에 속한 패킷들이 전달되기 때문에, 해당 A flow에 속한 패킷을 받은 노드는 따로게 패킷을 분류하고, 기생성된 플로우를 사용하여 고속으로 패킷을 처리할 수 있다. 더욱이 신호 능동 패킷에서 패킷의 분류 키와 코드 적재에 필요한 정보를 제공하기 때문에, A flow 패킷의 헤더는 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 flow에 대한 플로우 생성 및 종료시 필요한 정보와 코드 적재시 필요한 정보를 교환하기 위한 패킷이다. 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. 코드 적재 과정의 효율성을 위해 선택적으로 신호 능동 패킷 데이터 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 세가지 방식으로 구분된다. 다음 세가지 라우팅 방식은 인터넷에서 사용되는 IP 패킷의 라우팅 방법과 같은 개념을 가지고 있지만, 라우팅에 관여하는 라우터가 해당 실행환경을 적재하고 있는 능동노드이며 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, 능동패킷이 전달된 다음 능동노드는 실행환경이 자신의 실행환경 특정 라우팅 테이블을 참조하거나 능동 패킷의 처리결과를 통해서 결정된다. 따라서 송신지에서 목적지까지의 능동노드가 임의의 노드가 될 수 있고, 중간 노드들 전체에서 능동패킷의 처리를 필요로 하는 응용에 적합하다

· direct 라우팅

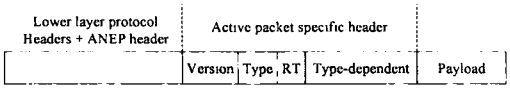
이 라우팅 방식은 송신지에서 목적지까지의 중간 능동노드에서의 패킷처리를 필요로 하지 않는 경우에 적합하다.

· source 라우팅

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

3.3 패킷 구조

[그림 2]는 능동 패킷의 일반적인 형식을 보여준다.



[그림 2] 능동 패킷 형식

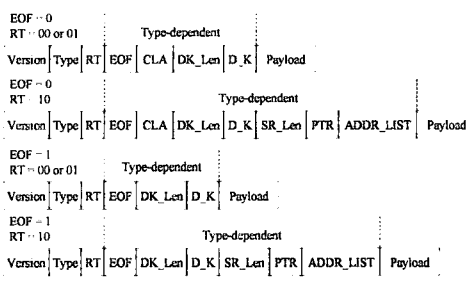
능동 패킷의 형식은 패킷 플로우의 종류에 따라 달라진다. 또한 능동응용이 어느 계층에서 구현되는가에 따라 IP 헤더나 UDP 헤더와 같은 하위 계층 프로토콜들의 헤더가 오며, 그 다음에 실행환경을 구분하기 위해 필요한 ANEP[3] 헤더가 온다. ANEP 헤더 뒤에는 본 논문에서 제시하는 능동 패킷 특정 헤더가 온다.

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.

좀더 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 3]에 나타나 있다.



[그림 3] 신호 능동 패킷 형식

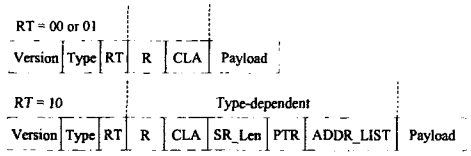
신호 능동 패킷은 End of A-Flow(EOF: 1비트), Code Location Address(CLA: 4바이트), Demux_Key length(DK_Len: 1바이트), Demux_Key(D_K: 가변) 부필드들을 가진다. 그리고 라우팅 방법에 따라 source route length(SR_Len: 1바이트), current pointer(PTR: 1바이트), address list(ADDR_LIST: 가변) 부필드를 가진다. EOF 부필드는 A flow의 시작(EOF: 0)과 종료(EOF: 1)를 나타낸다.

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 능동노드의 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 4]에 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 5]에 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.

3.4.1 패킷 분류 과정

패킷의 분류 과정은 크게 2단계로 구분된다.

첫 번째 분류 과정은 패킷 분류기에서 수행되며, 패킷내의 ANEP[3] 헤더의 유무에 따라 일반 IP 패킷인지 능동 패킷인지를 결정한다. ANEP 헤더가 존재하지 않으면 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.

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.

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 능동노드의 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 능동노드에 패킷이 전달될 다음 능동노드의 주소를 유지해야 한다. 이 정보는 플로우 생성시 실행환경 특정(EE specific) 라우팅 테이블에 저장되며, 플로우 종료시 삭제된다. 실행환경 특정 라우팅 테이블은 [flow id, 다음 능동노드 주소] 형식의 항목을 가진다. 다음 능동노드의 주소는 패킷의 "Routing Type" 필드와 부필드(source 라우팅의 경우), 실행환경이 가지고 있는 능동노드로 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 능동노드를 찾을 수 있으며, source 라우팅인 경우(값: 10)는 헤더에 명시된 경로 정보를 통해 다음 능동노드를 찾을 수 있다. Direct 라우팅의 경우는 패킷의 송신지에서 목적지로 곧바로 전달되기 때문에 다음 능동노드의 주소는 목적지 주소가 된다.

마지막으로 플로우의 생성이 끝나고 능동노드 자신이 목적지가 아니면 위의 과정을 통해 얻은 다음 능동노드로 신호 능동 패킷을 전달해서 목적지까지의 중간 능동노드에서 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 능동노드로 패킷을 전달해서 목적지까지의 중간 능동노드에서 플로우를 종료시키도록 한다.

· A-flow

A-flow는 패킷 분류 과정을 거쳐 노드에 기설정된 플로우의 입력 채널로 전달된다. 패킷은 기설정된 플로우의 자원을 이용해 처리되며, 실행 결과를 노드 자신의 능동응용이나 실행환경 특정 라우팅 테이블에 명시되어 있는 다음 능동노드로 전달한다.

· 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 능동노드에게 코드를 요청하며, 실행 결과는 노드 자신의 능동응용이나 RT에 명시된 라우팅 방법에 따라 다음 능동노드의 주소로 전달한다.

4. 결론 및 향후 연구 방향

본 논문에서는 다양한 능동응용의 요구사항을 위해 능동 패킷을 트래픽의 특성에 따라 a-flow, non a-flow, 신호 능동 플로우, 관리 플로우로 구분하고, 몇가지 라우팅 방법(hop-by-hop, direct, source)을 지원할 수 있는 능동노드 구조를 제안했다. 따라서 능동응용은 능동노드에서 제공하는 이러한 기능을 이용하여 좀더 빠르고 효율적으로 개발될 수 있다. 우리는 아직까지 관리 플로우에 대해서는 정확한 동작 과정이나 패킷 형식에 대해서는 정의하지 못하였으며, 실행환경들 사이에서 필요한 동작들을 연구함으로서 관리 플로우에 대한 정의를 내려야 할 것이다. 다음으로 본 논문에서 제시한 능동노드의 구조를 구현하고 성능을 평가하는 과정이 필요하다. 이 과정에서 성능, 안전성 및 보안, 프로그래밍 언어 등에 관한 추가적인 연구를 수행할 것이다.

감사의 글

본 연구는 2001년도 한국전자통신연구원 정보보호기술본부 위탁연구과제에 의한 것임.

참고문헌

[1] J. Postel, "Internet Protcocol," RFC 791, Sep. 1981.
[2] D. Wetherall, et. al., "The Active IP Option," 7th ACM SIGOPS European Workshop, 1996.
[3] D. Alexander, et. al., "Active Network Encapsulation Protocol (ANEP)," 1997.
[4] B. Schwartz, et. al, "Smart packets for Active Networks," BBN Technology, Jan. 1998.
[5] G. Alex, et. al., "A Flexible IP Active Networks Architecture," IWAN 2000 Conference, 2000.
[6] L. Peterson (Editor), "NodeOS Interface Specification," DARPA AN NodeOS Working Group, 1999.
[7] K. Calvert, "Architectural Framework for Active network," Active Network Working Group, 1999.
[8] D. Decasper, et. al., "Simple Active Packet Format (SAPF)," 1998.
[9] T. Wolf, et. al., "Tags for High Performance Active Networks," IEEE OPENARCH, 2000.
[10] D. Alexander, et. al., "The SwitchWare Active Network Architecture," IEEE Network, May/June 1998.
[11] M. Hicks, et.al, "PLANet: An Active Internetwork," IEEE INFOCOM, 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] D. Decasper, et. al, "DAN: Distributed Code Caching for Active Networks," IEEE INFOCOM, 1998
[15] D. Wetherall, et. al., "ANTS: a toolkit for building and dynamically deploying network protocols," Open Architectures and Network Programming 1998 IEEE, 1998.
[16] P. Tullmann, et. al., "A Java-oriented OS for Active Network Nodes," IEEE Journal on Selected Areas of Communication. Volume 19, Number 3, Mar. 2001.
[17] S. Merugu, et. al., "Bowman: A Node OS for Active Networks," Proceedings of IEEE Infocom 2000, Mar. 2000.