# 액티브 네트워크를 위한 프로그래밍 모델 설계*

김동영[0], 이영석, 손선경, 나중찬
한국전자통신연구원 정보보호기술연구본부 네트워크보안연구부
(kdy63281[0], yslee, sgsohn, njc)@etri.re.kr

# Programming Model Design for Active Network

Dong-Young Kim[0], Young-Seok Lee, Seon-Gyoung Sohn, Joong-Chan Na
Network Security Department, Information Security Technology Division, ETRI

**요 약**

액티브 네트워크 상의 각각의 노드는 패킷을 통해 전달된 프로그램을 수행시키는 수행환경을 가진다. 본 논문은 액티브 네트워크 노드에서 수행되는 프로그램을 작성하기 위한 자바기반의 프로그래밍 모델을 제안한다. 또한 이동코드의 적재 및 실행과정을 제어할 스크립트 언어를 설계하고, 제안된 모델을 적용할 수 있는 자바기반 수행환경의 구조를 기술한다.

## 1. 서론

기존의 네트워크에서 네트워크 노드의 역할이 패킷을 목적지까지 오류 없이 전달되도록 하는 것인데 비하여, 액티브 네트워크에서는 각각의 노드가 전달되는 패킷에 대하여 계산을 수행할 수 있고, 사용자가 노드에서 수행될 프로그램을 만들 수 있다. 각각의 노드에서 프로그램이 수행되기 위해서는 각 노드에는 프로그램을 수행시켜주는 수행환경이 존재해야 한다.

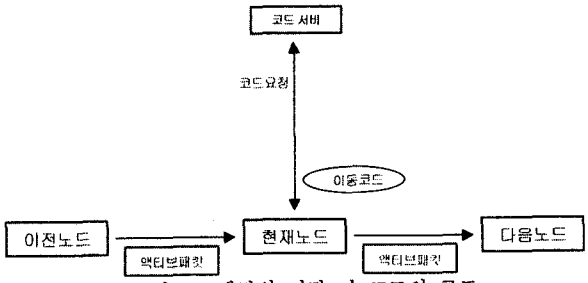
액티브 네트워크의 이동코드는 이종의 다양한 노드들에서 실행되며, 신뢰할 수 있고 안전성이 뛰어나야 한다. 이동코드를 기술하는 프로그래밍 언어는 이식성, 안전성, 보안성, 효율성과 같은 특징을 필요로 한다.

자바(JAVA) 언어는 객체지향 언어로서 이동성 및 보안성에 중점을 두어 설계되었으며, 자바로 제작한 애플릿은 이동코드의 한 예가 될 수 있다. 자바는 이동코드 프로그래밍 언어로서 적합한 몇 가지 특징을 가지고 있다. 언어적으로 포인터연산의 제거, 제한된 타입 캐스트 메모리관리 등의 특징으로 안전성이 뛰어나다. 시큐리티 관리자에 의한 자원접근의 제어와 자바코드에 대한 정적, 실행시의 에러체크 기능을 통한 보안 및 안전성, 네트워크를 통한 클래스 적재 등의 네트워크적 측면의 특징을 갖는다.

본 논문은 자바언어를 액티브 네트워크의 이동코드 기술언어로 사용할 경우, 이를 위한 프로그래밍 모델과 수행환경에 대한 설계를 기술한다.

## 2. 이동코드의 실행 모델

설계된 시스템은 네트워크를 통하여 동적으로 코드를 이동시키고 수행할 수 있는 환경을 제공한다. 이때 이동된 코드는 자바 언어로 프로그래밍 된 코드이어야 한다. 노드로 적재되어 실행되는 자바 이동코드 들은 코드서버에 의해 통합관리 되어지며, 액티브 패킷 내에는 코드서버로부터 이동코드를 적재하기위한 스크립트 언어와 수행환경과 이동코드에 의해 사용되는 데이터를 담고있다. 액티브 패킷이 노드에 도착하면 노드의 수행환경은 패킷 내에 포함된 스크립트를 해석하여 필요한 코드를 코드서버로부터 적재하여 실행시킨다. 액티브 패킷의 전달 및 수행과정은 <그림1> 과 같다.




<그림 1> 패킷의 전달 과 코드의 로드

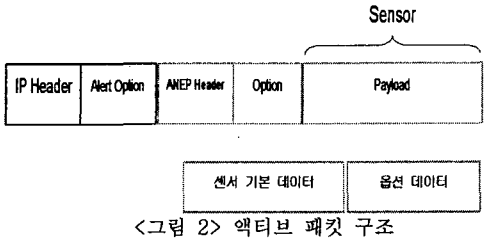
또한 수행환경에서 실행될 코드가 액티브 패킷에 의해 동작하지 않고 수행환경이 실행을 시작할 때 적재되어 실행될 수 있다.

스크립트언어는 이동코드의 적재 및 실행을 제어하기 위한 것으로 수행환경에 의해 해석되어 실행된다.

이동코드와 데이터는 <그림 2>와 같은 액티브 패킷을 사용하여 전달한다.

IP Header, Alert Option, ANEP(Active Network Encapsulation Protocol) Header 부분은 NodeOS에서 처리되어지고, Payload 부분만 수행환경으로 전달된다. 여기서 Payload 정보부분을 센서라고 한다.

* 본 논문에서 제안된 연구는 정보통신부연구과제로 지원되었다.




<그림 2> 액티브 패킷 구조

센서는 다시 센서 기본 데이터와 옵션 데이터 영역으로 나누어진다. 센서 기본 데이터 영역은 센서 아이디, 이전 센서 아이디, 센서타입, 생성일시, 소스주소, 목적지 주소, 자바 이동코드를 적재하기 위한 스크립트 언어를 포함한다. 옵션 데이터 영역은 센서의 종류에 따라 다른 정보가 들어간다. 센서가 다시 다른 노드로 전송될 때에는 NodeOS에서 IP Header와 ANEP Header를 생성하여 전송한다. 수행환경에서는 센서 안의 데이터들이 자바 클래스의 객체로 만들어져 이동코드에 의해 사용되어진다.

### 3. 이동코드의 프로그래밍 모델

본 시스템에서 사용하는 이동코드는 자바언어를 사용하여 작성할 수 있다. 그러나 특정한 모델에 따라 프로그램을 작성해야만 수행환경에서 적재하여 실행이 가능하게 된다. 수행환경에서는 이동코드에서 사용할 수 있는 API 들을 제공한다. 이 API 들은 액티브 노드에서 제공하는 기능들을 클래스의 형태로 제공한다. 제공되는 클래스들 중 중요한 클래스는 아래와 같이 Mcode(모든 이동코드에 대한 최상위 클래스), Sensor (액티브 패킷에 의해 전달되는 데이터를 추상화), NodeManager(액티브 노드에 대한 정보를 제공)이다.

- 이동코드(Mcode)
  모든 이동코드가 계승해야 하는 추상 클래스이다. 이것은 자바 애플릿을 만들기 위해서는 Applet 클래스를 계승해야 하는 것과 같다. <표 1>과 같이 Mcode는 이동코드를 위한 기본적인 기능, 메소드, 보안모델을 가진다. 이동코드는 McodeMain 메소드를 재정의하여 구현하여야 하며, 코드가 시스템으로 적재되면 McodeMain 메소드가 새로운 스레드로서 동작하게 된다. 수행환경에서 새로운 액티브 패킷을 받게 되면 패킷 내의 정보에 따라 이동코드를 수신하여 로드하고, 적재된 이동코드에 데이터를 전달하고 실행시킨다.

| 이름 | 기능 |
|---|---|
| m_rcvSensor | 수신된 액티브 패킷 데이터 |
| McodeMain() | 이동코드가 실행될 때 이동코드의 메인 스레드로 호출되는 메소드 |

<표 1> Mcode 클래스 어트리뷰트와 메소드

- 액티브 패킷(Sensor)
  Sensor 클래스는 액티브 패킷을 추상화 한 것으로 액티브 패킷 안에 있는 데이터를 나타내는 클래스이다. 수행환경은 패킷 내의 데이터를 Sensor 클래스의 객체로 만들어 이동코드로 전달한다. <표 2>와 같이 Sensor 클래스에서는 패킷의 데이터를 읽고, 변경하고, 다시 패킷의 형태로 다른 노드로 전송하는 메소드들을 지원한다.

| 이름 | 기능 |
|---|---|
| Sensor() | 센서 생성 |
| send() | 센서 전송 |
| sendtoASM() | 센서의 내용을 ASMS로 전달 |
| getSensorInfo() | 센서 기본 데이터 읽기 |
| setSensorInfo() | 센서 기본 데이터 설정 |
| getOptionInfo() | 센서 옵션 데이터 읽기 |
| setOptionInfo() | 센서 옵션 데이터 설정 |

<표 2> Sensor 클래스 어트리뷰트 및 메소드

- 노드(NodeManager)
  노드 자체를 추상화해주는 클래스로, <표 3>과 같이 노드에 대한 정보를 얻고 설정할 수 있도록 해준다.

| 이름 | 기능 |
|---|---|
| getMyAddress() | 현재 노드의 주소를 반환 |
| getASMSAddress() | ASMS 노드의 주소를 반환 |
| getCodeServerAddress() | 코드서버의 주소를 반환 |
| sameNetwork() | 현재 노드가 특정 주소와 동일한 도메인에 속하는지 검사 |

<표 3> NodeManager 클래스의 메소드

다음의 프로그램은 기술한 프로그래밍 모델을 사용하여 인터넷 보안분야에 적용한 예이다. 네트웍의 사이버 공격에 대한 역 추적을 수행하는 예제이다.

```
import eSMART.asi.*;
import eSMART.ee.*;
import eSMART.common.*;

public class TracingSensor extends Mcode
{
  TraceInfo tin;
  SensorInfo sin;

  public void McodeMain(String[] args)
  {
    sin = m_rcvSensor.getSensorInfo();
    tin = (TraceInfo) m_rcvSensor.getOptionInfo();

    if (NodeManager.getMyAddress().equals(
          NodeManager.getAsmsAddress()))
    {
      m_rcvSensor.sendtoASM();
    }
    else // 현재 노드가 secure node인 경우
    {
      if (NodeManager.sameNetwork(tin.getDaddr()))
      {
        Asi.sessionBlock(tin.getSaddr().getHostAddress(),
                         tin.getDaddr().getHostAddress(),
                         tin.getDport(), tin.getProtocol());
      }
      else if (NodeManager.sameNetwork(tin.getSaddr()))
      {
        Asi.sessionBlock(tin.getSaddr().getHostAddress(),
```

```
                    tin.getDaddr().getHostAddress(),
                    tin.getDport(), tin.getProtocol());
        sin.setDIA(NodeManager.getAsmsAddress());
      }
      m_rcvSensor.send();
    }
  }
}
```

### 4. 스크립트 언어의 설계

수행환경 제어 스크립트 언어는 자바 이동코드의 적재 및 실행을 위한 정보를 전달한다. 다음 리스트는 수행환경에서 실행되는 자바 이동코드 및 고정형 코드를 적재하고 실행시키기 위한 스크립트 언어 문법의 EBNF 포맷이다.

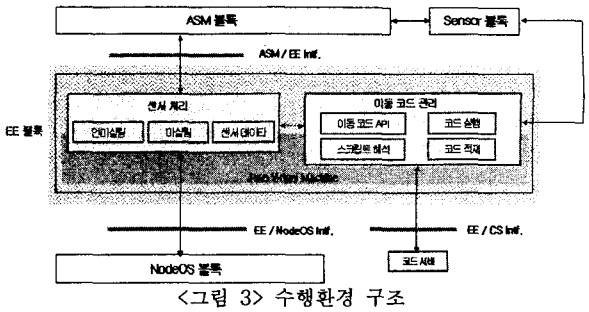
```
<ESL_program> ::= <run_stmt> *
<run_stmt> ::= 'run' <integer_num> <arguments>
                    <install_stmt> <location_stmt>
<arguments> ::= '(' <argument_list> ')'
<argument_list> ::= <string> ( ',' <string> ) *
                  | empty
<install_stmt> ::= 'after' 'install' <version_stmt>
                | empty
<version_stmt> ::= 'version' <version_string>
                  | empty
<version_string> ::= <string>
<location_stmt> ::= 'loc' <location_string>
                  | empty
<location_string> ::= <string>
<integer_num>  ::=  ['1'-'9'] ['0'-'9']*
<string> ::= '"'
      (   (~['"','₩','₩n','₩r'])
        | ('₩'
            ( ['n','t','b','r','f','₩','''','"']
            | ['0'-'7'] ( ['0'-'7'] )?
            | ['0'-'3'] ['0'-'7'] ['0'-'7']
            )
          )
      )*
      '"'
```

위의 문법에 의해 생성되는 코드의 예는 다음과 같다. 코드의 의미는 코드서버에서 코드타입 200에 해당하는 코드의 Version 2.1 코드를 코드서버(129.254.242.44)로 부터 적재하여 실행시키는 것이다. 이때 실행되는 이동코드의 아큐먼트로 "10" 이 전달된다.

```
run  200  ("10")
      after install  version  "2.1"
      loc  "129.254.242.44";
```

### 5. 자바기반의 수행환경 구조

수행환경은 <그림 3>과 같이 센서 처리(Sensor Processor) 부분과 이동 코드 관리(Mobile Code Manager) 부분으로 구성되어 있다. 수행환경 내의 두 구성요소는 자바가상기계(JVM: Java Virtual Machine) 상에서 수행된다.



<그림 3> 수행환경 구조

수행환경내의 센서 처리 부분은 액티브 패킷의 내용을 읽는 언마샬링, 전달 가능한 패킷의 형태로 만드는 마샬링, 센서데이터 관리기능을 가지며, ASM 및 NodeOS과 외부 인터페이스를 가진다. 이동 코드 관리기는 코드 서버와 외부 인터페이스를 가지고 스크립트 언어를 해석, 이동코드를 로드하는 코드적재, 적재된 이동코드를 실행하는 코드실행, 이동코드에 의해 호출되는 API를 지원하는 기능을 가진다.

수행환경은 NodeOS, ASM(Active Security Manager) 및 코드서버와 인터페이스를 가진다. NodeOS는 액티브 패킷의 수신 및 전송의 역할을 담당한다. 그리고 ASM은 액티브 네트워크를 보안분야에 사용하기 위한 보안 관리기로 네트워크 보안을 담당한다. 코드서버는 수행환경에서 수행될 자바 이동코드를 저장관리 한다. 코드서버는 LDAP 디렉토리 시스템을 사용하여 구성되어 있다.

### 6. 결론

액티브네트워크를 위한 자바기반 프로그래밍 모델을 설계하고 이동코드의 실행과정, 이동코드 적재를 위한 스크립트언어를 기술하였다. 설계된 모델은 자바를 기반으로 하였으며 자바언어가 가지는 장점들을 공유한다. 본 모델의 유용성을 확인하기 위해서는 다양한 분야의 응용을 통한 검증작업이 요구된다.

### 참고문헌

[1] 나중찬, 손승원, 박치항, "액티브 네트워크 프로그래밍 언어 동향", ETRI 주간기술동향, 2001

[2] David Wcthcrall, "Developing Network Protocols with the ANTS Toolkit", Aug, 1997.

[3] David J. Wetherall, John V. Guttag and David L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", IEEE OPENARCH'98, Apr,.1998.

[4] S.Merugu, S.Bhattacharjee, Y.Chae, M.Sanders, K.Calvert and E.Zegura, "Bowman and CANEs : Implementation of an Active Network", *37th Annual Allerton Conference,* Monticello, IL, Sep, 1999

[5] Stephen F. Bush and Amit B. Kulkarni, "Active Networks and Active Network Management", p.9-26, 2001

[6] Tim Owen, "Programmer's guide to the SafetyNet language", Dec,1999

[7] Pankaj Kakkar with Jonathan Moore and Mike Hicks, "The PLAN Tutorial", Nov, 24, 1997