Journal of The Korea Society of Computer and Information
Vol. 19, No. 1, January 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.1.065

eFlowC : 네트워크 관리를 위한 패킷 처리 언어

고 방 원*, 유 재 우**

eFlowC: A Packet Processing Language for Network Management

Bang-Won Ko*, Jae-Woo Yoo**

요 약

본 연구는 패킷 처리를 위한 고급 프로그래밍언어 eFlow를 제안하고 이를 지원하는 개발환경을 구현한다. 또한 개발자들이 가장 익숙하고 배우기 쉬운 C 언어를 기반으로 하여 C 언어의 문법과 의미를 유지하면서, 패킷 처리에 불필요한 기능들을 제거하고 패킷 처리에 필요한 패킷 데이터, 데이터베이스 및 스트링 바이트 정보 검사, 이벤트 처리 등을 수행하기 위한 고급 프로그래밍 언어를 설계하고, 기존의 언어나 컴파일러 기술을 적용하면서 패킷 처리를 위해 필요한 언어의 기능과 컴파일 과정을 설명한다. X11 등과 같은 DPIC 장비에 활용하기 위해 이식성과 확장성을 고려한 가상 기계인 eFVM을 설계하고, 이를 위한 컴파일러와 시뮬레이터 및 디버거와 같은 개발 환경을 갖추어 실제 많이 사용되고 있는 다양한 응용 프로그램을 실험하여 제안한 언어의 효용성을 평가하고 있다. 패킷 처리를 위해 갖추어야 할 고급 언어의 기능과 형식 및 그 의미를 정의한 연구가 거의 없이 이루어진 실험에 의의가 있다.

▸ Keywords : 패킷 처리 언어, 컴파일러, DPIC, 가상 기계

Abstract

In this paper, we propose a high-level programming language for packet processing called eFlowC and it supporting programming development environment. Based on the C language which is already familiar and easy to use to program developers, eFlowC maintains the similar syntax and semantics of C. Some features that are unnecessary for the packet processing has been removed from C, eFlowC is highly focused on performing packet data, database, string byte information checking and event processing. Design high-level programming languages and apply an

•제1저자 : 고방원 •교신저자 : 유재우
•투고일 : 2013. 12. 12, 심사일 : 2014. 01. 05, 게재확정일 : 2014. 01. 11.
* 숭실대학교 컴퓨터학과(Ph.D course, Dept. of Computer Science, Soongsil University)
** 숭실대학교 컴퓨터학과(Professor, Dept. of Computer Science, Soongsil University)

existing language or compiler technology, language function and compilation process that is required for packet processing will be described. In order to use the DPIC device such as X11, we designed a virtual machine eFVM that takes into account the scalability and portability. We have evaluated the utility of the proposed language by experimenting a variety of real application programs with our programming environment such as compiler, simulator and debugger for eFVM. As there is little research that devoted to define the formats, meanings and functions of the packet processing language, this research is significant and expected to be a basis for the packet processing language.

▸ Keywords : Packet processing language, Compiler, DPIC, Virtual machine

I. 서 론

최근 네트워크를 이용하는 호스트의 숫자가 더욱 늘어나고, 여기에 스마트폰이나 노트북 등의 휴대용 기기들의 이용자들이 추가되면서 네트워크를 통과하는 데이터의 양이 크게 증가하였고, 전송되는 데이터의 종류도 멀티미디어 데이터를 포함하여 매우 다양해졌다. 개인 간의 통신도 전자 메일이나 채팅이 일상화되었고 일반 전화도 인터넷을 경유하는 디지털 통신 형태로 전환되면서 사람의 생활 패턴, 회사의 업무, 각종 연구 등에 있어 네트워킹이 필수가 되어있는 상태이다. 이를 위해 네트워크 상태의 정확한 관리와 모니터링은 필수적이며 고장이나 악의적인 이용에 대한 대응은 즉각적으로 이루어져야 하는데, 네트워크의 이용 목적과 구성 상태가 복잡해짐에 따라 네트워크를 모니터링하고 관리하는 것은 인력 집약적이며 많은 시간을 소모하는 업무가 되었다. 이와 같은 환경에서 네트워크를 효과적으로 관리하고 제어하기 위해서는, 바이러스 탐지나 법적으로 허용된 패킷 감시를 위한 DPIC (Deep Packet Inspection Control) 응용 프로그램이 요구되고 가능한 한 고속의 패킷 프로세싱을 수행할 수 있어야 할 것이다.

DPIC 응용 프로그램은 그림 1 과 같이 패킷의 전체 계층에 대한 검사를 수행할 수 있기 때문에 패킷의 데이터(Payload)까지도 확인할 수 있다[19][21]. 따라서 NIDS Evasion, DDoS 공격 방어, 음란 및 유해물 차단, P2P 사용 제어를 포함하여 QoS 관리, 정밀 과금, VoIP SIP 보호 등 네트워크를 모니터링하고 관리하는데 반드시 필요한 기술로서, 지금도 대용량 트래픽의 실시간 처리에도 적용되고 있으며 그 적용 범위는 점점 확대될 것으로 생각된다[21].

이러한 요구에도 불구하고 국내의 DPIC 응용 프로그래밍 방법과 그 개발 환경은 매우 열악한데, 대부분 저급 언어인 RAVE를 이용한 환경[3]과 어셈블리 언어 기반 X11[18] 등의 장비를 사용하고 있는 실정이다. 따라서 저급 언어 환경에서 응용 프로그램을 개발하므로 개발 기간과 생산성 및 유지보수에 많은 어려움이 있는 실정이며, 다른 플랫폼이나 기기로의 전환이 어렵고 비용도 많이 든다는 문제를 갖고 있다.

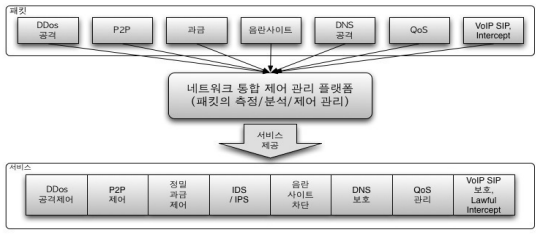

그림 1. 네트워크 통합관리 플랫폼
Fig. 1. Network Intergrated Management Platform

네트워크 통합 관리를 위한 DPIC와 같은 응용 개발에 있어서 기가급의 성능으로 진화하는 네트워크의 속도를 따라가기 위해서 많은 연구가 진행되고 있는데 이러한 움직임이 기존의 네트워크 프로세서를 이용한 DPI의 구현이 있고[20], FPGA(Field-programmable gate array)나 ASIC(application specific integrated circuit)을 이용하여 DPI를 구현하는 기술이 있다[5].

고급 프로그래밍 언어와 개발 환경이 필요한데 이는 세계

적으로도 아직 초기단계이고 연구 사례가 아주 드물게 나타나고 있어서 본 연구에 큰 의의가 있다고 하겠다.

해외의 기술로서는 가장 대표적으로, DPIC 응용 프로그램 개발을 위한 고급 언어로 CloudShield사에서 개발한 PacketC 언어가 최근 발표되었다[4]. PacketC 언어는 자료구조나 영령어가 C 언어와 유사한 고급 언어로서 많은 개발자에게 익숙한 Ecplise 기반의 IDE를 제공하고 있지만 관련회사 개발환경에 제한되어 있고, 실행 환경이 특정 하드웨어에 종속적이라는 단점을 갖고 있어서, 국내에 도입하더라도 다양한 환경과 기기에 바로 적용하는 것이 어렵고 추가적으로 PacketC를 사용하기 위해서는 높은 비용을 지불해야 한다는 점 때문에 핵심 기술이 특정 개발사에 의존되어 DPIC 기술 확보에 어려움이 있을 것으로 생각된다.

본 논문에서는 이와 같은 필요성에 부합하여 네트워크 통합 관리 플랫폼 상의 DPIC 응용 프로그램 개발을 효율적으로 수행할 수 있는 고급 언어인 eFlowC를 개발하여 제시한다. 제시하고자 하는 C언어의 틀을 그대로 유지하면서 패킷 처리에 필요한 여러 가지 명령과 데이터 타입을 지원하도록 만든 패킷 처리에 적합한 언어이다. 여기에 개발자가 더욱 편리한 개발이 가능하도록 시뮬레이터와 디버거를 포함하여 프로그램 개발 생산성을 높일 수 있도록 구현하였다. 국내의 통합 관리 플랫폼 환경에서 사용하기 적합한 DPIC 응용 프로그램 개발용 고급 언어인 eFlowC와 그 컴파일러를 개발함으로서 프로그램 개발 효율과 재사용성을 높일 수 있고 보다 편리한 사용자 친화적인 개발 환경을 제시하여 국내의 네트워크 모니터링과 관리에 기여할 수 있을 것으로 생각된다.

본 연구에서 2장은 DPIC의 역할 및 용도, 고급 언어에서의 기능을 검토하고, 3장에서는 제안한 eFlowC 언어의 기능과 특징 그리고 패킷 처리 상황에 대해 기술한다. 4장에서는 eFlowC 프로그래밍을 지원하기 위한 개발환경을 기술하고, 5장에서는 eFlowC 언어에 대한 실험과 그 결과를 보이고, 6장에서는 결론에서는 본 논문에서 제시한 eFlowC 언어와 그 컴파일러 그리고 eFVM에 대해 요약한다.

II. DPIC

DPIC 기술을 이해하기 위해서 우선은 패킷의 구조에 대해서 개괄적인 정의가 필요하다. 패킷은 IP 네트워크상에서 전송되는 가장 작은 단위를 의미하는 것으로 헤더와 페이로드(Payload: Data)로 크게 구분할 수 있다. 헤더에는 패킷 전송을 위한 주요 정보가 들어가는데 크게 IP 헤더와 전송계층 헤더의 두 가지로 구분할 있다. IP 헤더가 가장 바깥쪽에 있는데 IP 주소를 기반으로 목적지를 찾거나 수신하며, 전송계층의 정보를 가지는 헤더는 일반적으로 TCP 프로토콜을 사용하는 TCP 헤더이다. 페이로드는 실제 데이터를 담고 있는 부분인데 패킷의 콘텐츠나 내용에 해당하는 부분이다[19].

DPI 는 패킷 검사 중 최고 수준이며, 패킷의 심층부라 할 수 있는 애플리케이션 계층의 데이터까지를 검사할 수 있기 때문에 데이터의 콘텐츠를 중심으로 패킷을 검사하는 것도 가능하다. 어떤 응용 프로그램이 어떤 패킷을 생성하는지는 누적한 데이터에 기반하여 구분할 수 있고, 초당 수십만의 패킷을 실시간으로 처리할 수 있기 때문에 대규모 네트워크에서 패킷 검사에 적합한 방식이다. 거기에 트래픽 패턴에 의한 예측과 검사를 실행할 수 있어서 암호화된 패킷이라도 대응이 가능한 검사 방식이다[18].

BEREC(Body of European Regulators for Electronic Communication)에서는 트래픽 관리를 네트워크의 위치와 계층이 어디냐에 따라 세 가지로 구분하는데[19], 그림 2 와 같이 분류를 하여, 트래픽 관리 타입 1은 네트워크의 위치와는 상관없이 네트워크 계층에서 이뤄지는 트래픽 관리로, 타입 2와 3은 애플리케이션 계층에서 이뤄지는 트래픽 관리로, 타입 2는 네트워크 end point에서 타입 3은 네트워크 내부 노드에서 실행된다. 이런 타입들 중 DPI 기술에 가장 근접한 것은 타입 3으로 패킷의 가장 심층 부분을 검사하고 트래픽 필터링이나 트래픽 변경과 같은 기술을 포함한다. 따라서 미리 지정된 트래픽 관리정책에 따라 개별 IP 패킷이 분류, 전달, 지연 및 폐기된다.

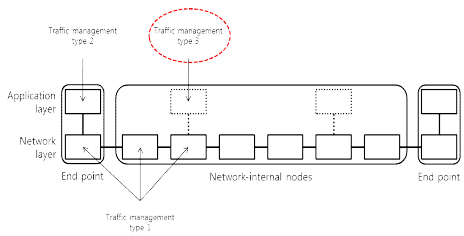

그림 2. 트래픽 관리의 분류 및 DPI 기술의 적용 부분
Fig. 2. Classification of Traffic Management and Application of DPI

DIP 기술은 앞으로 더욱 많이 사용될 것이고 점차 적용 범위가 확대될 것으로 예상이 된다. 어떤 경우이든 DPI 나 DPIC 응용 프로그램은 실시간으로 인터넷 트래픽을 분석할 수 있어야하고 다양한 기능들을 하나의 장비에 구현할 수 있어야 한다. 실시간 인터넷 트래픽 분석을 위해서는 네트워크 장비의 발전이 우선되어야 하지만 다양한 응용 프로그램으로

부터 생산되는 여러 가지 트래픽의 종류에 적합한 DPIC 응용 프로그램의 개발도 시급하다. 특히 하나의 장비에 다양한 기능들이 포함되려면 특정 장비에 적합하도록 개발한 DPIC 응용 프로그램을 한 번 개발하고 이후에 또 다른 네트워크 장비로 쉽게 이식시킬 수도 있어야 한다.

III. eFlowC 언어의 설계

1. 언어의 특징

eFlowC 는 네트워크 장비에 프로그램을 할 때 타켓 머신에 대한 직접적인 저급 언어 프로그래밍이 아니라면, 보편적으로 하드웨어 제어를 위한 프로그램 개발에 개발자들이 가장 많이 사용하여 익숙한 C 언어의 문법을 기반으로, 그 기능과 틀을 그대로 유지하고 있으며 패킷처리를 위한 기능을 추가하여 DPIC 응용 개발에 적합하도록 설계되었다. eFlowC 언어는 실시간으로 들어오는 패킷의 IP 헤더와 페이로드를 분류하고 분석해 처리하는 프로그래밍을 가능하게 하고 타겟 머신에 대한 프로그래밍 뿐만 아니라 다른 플랫폼을 사용하는 머신으로의 이식이 쉬운 장점을 가지고 있다.

패킷 처리 특성상 불필요한 실수형 데이터 타입이나 포인터 타입을 지원하지 않으며, 대신 패킷처리에 필요한 데이터 타입 으로 데이터베이스 타입, 스트링 검사를 위한 부분스트링 배열 타입, 패킷 과 PIB (Packet Information Block) 데이터 타입, IP 주소 데이터 등을 지원하도록 하였다. C 언어의 대부분의 명령문을 그대로 채택하였으며, 패킷의 삽입과 제거 및 변경을 위한 명령, 테이터 베이스의 변경 및 검색, 패킷에서의 스트링 검사, 패킷 처리 과정에서의 에러 처리나 이벤트 처리 등을 지원하기 위한 명령이나 라이브러리 함수 등을 지원하도록 설계되었다.

C 언어가 가지는 블록구조의 일반적 특징을 그대로 유지하도록 하여 변수나 명칭들의 스코프 규칙, 컴파일단위와 프로그램실행시의 메모리 구조 등을 같은 방법으로 생각할 수 있도록 하였다. 패킷처리를 목적으로 설계된 언어로서 일반적인 메모리는 광범위한 메모리 접근을 허용하지만 안전하지 않는 네트워크 도메인에서 신뢰할 수 있는 응용 프로그램이나 보안 시스템에서 메모리 접근을 제한하도록 하였다.

2. eFlowC 언어의 구성

2.1 예약어

eFlowC 언어의 예약어는 C 언어와 대부분 일치하며, 패킷 처리 등을 하기 위한 기능을 설계하면서 새로운 기능에 대한 데이터 타입과 명령을 위한 키워드는 기존의 키워드나 부호를 많이 사용하도록 하였으며, 추가로 byte, module, buffer, try-catch, throw, database, exit, offset, redefine 등을 정의하였다.

2.2 시스템 테이터 타입

그림 3 과 같이 네트워크를 통해 전송되는 패킷을 처리하기 위한 목적으로 PACKET과 PIB를 정의한다. 패킷은 크게 두 부분으로 나뉘는데, 앞부분은 TCP/IP 프로토콜에 대한 정보를 가지고 있고, 뒷부분은 실제 전송되는 데이터를 가지는 페이로드로 구성되어 있다. 패킷을 처리하기 위해서는 실시간으로 전송되는 패킷을 저장하는 공간이 필요로 하고, 또한 TCP/IP 프로토콜의 정보를 저장하는 공간이 필요하다.

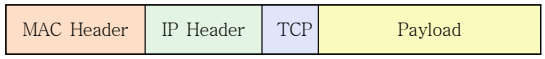

그림 3. 패킷 구조
Fig. 3. Packet Structure

네트워크를 통해 실시간으로 전송되는 패킷을 저장하고 패킷의 정보를 저장하여 효율석인 분석과 처리를 위해 시스템 데이터타입을 다음과 같이 정의하였다.

```
typedef byte PACKET[9 * 1024 - 1];
typedef struct {
        enum PacketAction action;
        int             logAccelTarget;
        int             length;
        int             flags;
        enum L2Type     l2Type;
        enum L3Type     l3Type;
        enum L4Type     l4Type;
        int             l2Offset;
        int             mplsOffset;
        int             l3Offset;
        int             l4Offset;
        int             payloadOffset;
} PIB;
```

실시간으로 전송되는 패킷을 받아 바이트 배열처럼 처리가 가능하고 정의된 바이트 배열의 크기는 패킷의 TCP/IP 프로

토콜의 정보과 데이터를 가지고 있는 페이로드를 담기 위하여 최대 크기인 9K 바이트로 정의한다. eFlowC 언어에서 다룰 수 있는 패킷의 프로토콜은 네트워크 OSI 계층의 2 레이어부터 7 레이어까지의 정보를 저장하며 PACKET 데이터 타입은 eFlowC 응용 어플리케이션에서 PACKET 변수를 통하여 참조한다. PIB 데이터는 네트워크 OSI 계층의 2 레이어부터 4 레이어까지의 프로토콜을 정보를 표현하며, 현재 패킷의 프로토콜의 종류, 헤더, 주소에 대한 결과를 저장하며 eFlowC 응용 어플리케이션에서 PIB 변수를 통하여 참조한다.

2.3 기본적 데이터 타입과 상수

eFlowC 언어는 기본적 데이터 타입으로 int, long, short, byte 타입 등을 갖는다. byte 타입은 C 언어에서 지원하지 않지만 char 과 유사하다. C 언어에서와 같은 10진, 8진, 2진, 16진 상수와 문자 상수 및 스트링상수 외에 IP주소를 표현하기 위한 네트워크 상수를 지원한다. eFlowC 언어에서는 네트워크 IP와 패킷 데이터를 다루고 있어서 IP 주소를 처리하기 위하여 네트워크 상수를 새롭게 정의하여 지원한다. 범용적인 목적으로 사용되는 C 언어와는 달리 패킷을 처리하기 위한 목적으로 설계되어 실수형 데이터 타입은 필요하지 않다. 포인터 타입은 실행 시간 중에 발생하는 충돌을 방지하기 위해 주소연산과 함께 지원하지 않는다.

2.4 수식과 연산자

수식은 가장 간편한 명령으로서 이에 사용 되는 연산자는 일반 C 언어의 연산자와 대부분 일치하며 그 우선순위와 결합 순위도 같도록 하였다. 우선순위에 따라 정리하면 다음과 같다.

```
++, --
!, ~, sizeof, offset, 타입변환
/, *, %
+, -
<<, >>
<, >, <=, >=
==, !=
&
^
|
&&
||
=, +=, -=, /=, *=, %=, <<=, >>=, &=, ^=, |=
```

이러한 연산자는 변수 등의 초기화에도 사용되며 그 방법도 다음의 예와 같이 C 언어와 동일하도록 하였다.

```
module (PACKET pkt)
int srcIP;
int count=0;
main()
  srcIP=ipv4.sourceAddress;
  if (srcIP==10.10.2.105) {
    counter++;
  }
  printi(counter);
}
end module
```

2.5 함수의 정의와 호출

함수는 현대 프로그래밍언어와 방법에서 중요한 역할을 하고 있으므로 C 언어와 같은 형태를 취하도록 하였으며 함수의 선언과 호출방법도 동일하게 하였다. 재귀적 호출이나 가변적인 매개변수의 타입과 개수도 허용하며 prototype을 선언할 수 있도록 하였다. 매개변수의 전달은 call—by-value 방식만을 지원한다.

2.6 복합 데이터 타입

열거형 데이터 타입, 구조체(struct), 공용체(union), 및 배열을 지원하도록 하였다. 응용프로그램의 보안성과 신뢰성을 위해 배열의 경우 동적 할당을 허락하지 않으며, 패킷 처리를 위해 부분 배열을 접근하기 위해 아래의 예와 같이 슬라이스 (slice)를 지원한다. 슬라이스의 크기는 콜론(:)을 이용하여 그 범위를 표현하도록 하였다. 배열 슬라이스를 통해 배열에 저장된 패킷에 접근하여 배열 메모리의 지정한 범위에서 원하는 데이터를 검색하거나 삭제하고 처리할 수 있도록 하였다. 또한 C 언어와는 다르게 배열 전체를 다른 배열로 편리하게 치환할 수 있도록 하고 있다.

```
byte a[10], b[10];
byte start=0, end=4;
a[start:end]="korea";
b=a;
```

구조체와 공용체도 모두 C 언어와 같은 문법과 의미로 사용된다. 구조체 안에서의 bit field 들 허용하며, byte 나 bit

field 타입의 구조체는 다른 데이터 타입과 달리 패킷처리의 특성상 메모리 할당을 연속해서 하는 밀착형태 (packed)의 주소체계를 가지도록 하였다.

구조체의 특정 위치에 또 다른 구조 정보를 담기위해 구조체의 아래의 예와 같이 redefine 할당자를 제공한다.

```
struct Ipv4struct {
  byte versionAndLength;
  byte tos;
  short totalLength;
  short identification;
  short fragment;
  byte ttl;
  byte protocol;
  short checksum;
  int sourceAddress;
  int destinationAddress;
} ipv4 redefine pib.l3Offset;
```

ipv4는 구조체 변수로 새롭게 메모리에 할당되지 않고 패킷의 tcp 정보, 즉 PIB 가 알려주는 pib.i30Offset에 위치한 패킷의 구조정보를 정의할 수 있도록 하였다.

2.7 테이터 타입 연산자와 검사

C 언어와 같이 새로운 데이터 타입을 정의할 수 있도록 (typedef) 하였으며 컴파일러는 static type 검사를 한다. C 언어와는 달리 패킷처리의 특성상 프로그램의 안정성을 위하여 비교적 강한 (strong) 타입 검사를 하도록 하였다. 타입 변환 연산 (type cast) 이 명시적으로 가능하며 정수와 byte 사이에서만 타입이 다른 경우 묵시적으로 타입 변환이 이루어지도록 하였다.

2.8 일반 명령과 제어문

자주 사용되는 수식문 이외에 C 언어와 같이 선택문 (if-else문, switch-case문), 반복문(while문, do-while문, for문), 분기문 (goto문, break문, continue문, return문) 등의 문법과 그 사용방법을 같도록 하였다.

2.9 패킷 처리 명령

실시간으로 네트워크를 통해 전송되는 패킷을 처리하기 패킷을 효율적으로 분석하기 위한 명령으로서 수식이나 연산자 형태를 취하지 않고 다음과 같이 사전 정의된 라이브러리를 호출하는 형식으로 설계되었다.

표 1. 패킷 처리 라이브러리
Table 1. Packet Processing Library

라이브러리 명칭	설명	에러 이벤트
pkt_insert (arg1, arg2, arg3, arg4)	arg1 패킷의 arg2 위치에 arg3 크기의 btye를 arg4 byte 배열로부터 삽입	PKT_INSERT_ERR
pkt_delete (arg1, arg2, arg3)	arg1 패킷의 arg2 위치로부터 arg3 크기의 데이터 byte를 삭제	PKT_DELETE_ERR
pkt_replicate (arg1)	패킷의 사본 생성	PKT_REPLICATE_ERR
pkt_requeue()	패킷의 처리를 다음으로 연기	PKT_REQUEUE_ERR

다음은 패킷 라이브러리의 간단한 사용 예이다

```
byte data[10]={1,2,3,4,5,6,7,8,9,10};
pkt_delete(pkt,34,sizeof(TcpProtocol));
pkt_insert(pkt,16,2,data);
```

2.10 예외처리

패킷, 데이터베이스 및 byte배열이나 스트링 검색과 연산 과정에서 일어나는 에러 혹은 예외처리를 위해, 아래의 예와 같이 C++ 나 Java 와 같은 형태의 try-catch문과 이벤트 발생을 위한 throw문을 지원한다. try-catch 문은에러를 다루기가 쉽고 프로그램의 구조를 비교적 명확하게 한다. try블록 안에서는 반드시 에러나 예외처리를 위한 관련 이벤트를 발생하도록 의미적으로 제한하고 있다. 패킷의 연산, 데이터베이스 연산이나 스트링 검색 등에는 묵시적으로 에러 이벤트를 발생할 수 있도록 사전에 정의된 에러의 종류를 제공한다.

throw 명령을 통하여 명시적으로 사용자가 정의하는 이벤트를 발생시킬 수 있으며, 이런 명시적 이벤트도 try-catch 문을 통하여 처리할 수 있도록 하였다. 다음은 묵시적으로 발생한 에러 이벤트를 처리하는 예이다.

```
byte string[2][10] = {"korea","seoul"};
main (PACKET pkt, PIB pib)
{
   int result;
   try {
result=str_find(pkt[pib.payloadOffset:end],string[1])
      pib.action=FORWARD_PACKET;}
   catch (STR_FIND_ERR) {
      pib.action=DROP_PACKET;}
}
```

2.11 데이터베이스

데이터베이스는 데이터의 집합으로 패킷 데이터와 정보를 다루기 위하여 새롭게 정의된 복합 구조체이다. 데이터베이스는 패킷에 포함된 IP 소스, 목적 주소, 프로토콜과 같은 데이터와 다음 패킷이 처리되는 동안의 데이터 결과를 저장한다. 데이터베이스는 보통 1차원 배열로 정의되고 데이터 요소를 테이블 형태로 나타내어 필요한 데이터를 검색하여 찾는다. 데이터베이스는 일치하는 IP source 주소를 찾고, 데이터를 검색하기 위해서 레코드를 사용한다.

eFlowC 언어에서 데이터 베이스는 메모리의 특수 영역에 할당하므로 항상 전역변수로 정의하도록 제한하고 있다. 데이터베이스에 데이터 레코드와 함께 마스크 (mask)를 저장한다. 마스크는 데이터 레코드와 함께 사용되어 관심 있는 데이터의 집합을 찾는데 용이하다. 마스크는 구조체 필드에 관점에서 정의되어 정수 타입으로 정의하도록 한다. 마스크는 그 특성상 데이터베이스 또는 레코드 부분에 명시적으로 사용하거나 정의할 수 없다. 그러나 사용자 정의 타입으로 데이터베이스를 선언했을 시에는 직접 마스크 필드에 액세스가 가능하다. 마스크는 모든 마스크 비트는 1의 값을 기본 값으로 가지거나 처음 초기 값으로 사용한다. 데이터베이스 요소의 데이터 부분과 마스크 부분은 일치되는 값을 가져야하는데, 데이터 필드에는 IP 주소가 저장되고 마스크 필드에는 네트워크 환경에 맞는 IP 주소를 연산이 가능하도록 하는 마스크 값이 포함된다. 마스킹은 보통 자동적으로 생성되도록 하고 있으며 각 레코드에서 정의할 수도 있다. 테이터베이스를 접근하는 과정에서 오류가 일어나면 DB_READ_ERR 이벤트를 발생한다.

데이터베이스 연산을 위해서는 일반적인 수식에서 데이터베이스의 각 레코드를 가지고 치환이나 비교하는 연산이외에 레코드의 삭제, 삽입을 위해 다음과 같은 라이브러리 함수를 제공한다.

표 2. 데이터베이스 라이브러리
Table 2. Database Library

라이브러리 명칭	설명	에러 이벤트
db_insert(arg1, arg2)	데이터베이스에 레코드 삽입	DB_INSERT_ERR
db_delete(arg1, arg2)	데이터베이스에서 레코드 삭제	DB_DELETE_ERR

eFlowC 언어의 문법은 많은 부분이 C 언어[13]와 같으며 새롭게 추가되거나 변경된 부분은 부록에 yacc 언어 형식으로 수록하였다.

IV. 개발환경

1. eFVM 가상 기계와 시뮬레이터

본 연구에서 eFlowC 언어를 컴파일하여 저급의 언어로 변역하기 위한 목적 기계로 그림 4 와 같은 가상기계 eFVM을 설계하였다. 패킷 프로세싱은 복합 단계 파이프라인과 프로세서 배열이 가장 적합하다. 실제적으로 프로세싱은 몇몇 보통의 속도 프로세싱 유닛으로 나뉘게 된다. 결과적으로 eFVM 은 패킷 프로세싱 엘리먼트 (Packet Processing Element) 로 만들어진 모듈식식 구조를 가지게 된다. 이를 ePPE 라고 하며 ePPE 는 실제로 네트워크 프로세서의 마이크로 엔진을 가상화한 것이다.

파이프라인 방법은 다수의 프로세스가 다른 작업을 연속적으로 실행하는 방법으로 패킷이 하나의 프로세스에서 실행된 이후 다른 프로세스로 이동하면서 순차적으로 패킷이 처리되는 방법이고 병렬로 실행되는 방법은 동일한 작업을 하는 프로세스가 동시에 여러 패킷을 처리하는 방법이다. 그러므로 eFVM은 이와 같은 대표적인 두 방법을 모두 지원해야 하며, 그러기 위해서 가상의 프로세서인 ePPE를 사용한다. ePPE는 프로세서가 실행하기 위해 요구되는 메모리 관리와 스케쥴링, 메모리 접근을 제공하고 있으며 실제 실행에 있어서는 지금 실제 사용되고 있는 프로세서의 명령어를 생성하여 프로세스를 실행한다.

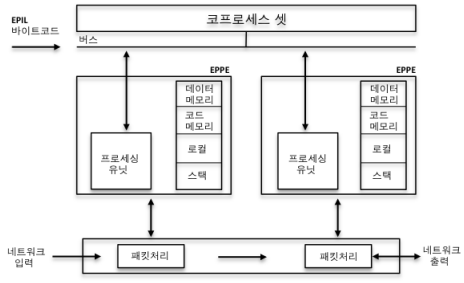

그림 4. eFVM의 구조
Fig. 4. Structure of eFVM

코프로세서(coprocessor) 는 구체적인 운영방법을 제공하는 블랙박스와 같은 응용 프로그램으로 보여지는 반면 코프로세서의 일괄적인 인터페이스는 서로 다른 플랫폼으로의 이식성을 보장한다. 그리고 그 구현은 플랫폼에 따라 다양하다. 어떤 아키텍쳐는 하드웨어 가속화, 코프로세서를 제공하지 않

기 때문에 그 기능은 소프트웨어에 의해 구현되며 특수 목적의 기능을 제공하는 아키텍쳐 프로세서는 직접적으로 eFVM과 일대일로 맵핑된다.

eFlowC 컴파일러는 다양한 하드웨어 플랫폼에서 패킷 처리 기능의 구현을 간단하게 하고 효율적인 실행을 하기 위한 가상 기계인 eFVM에서 사용하기 위한 기계어 코드를 생성한다. 가상 기계어는 스택 기반의 메모리 구조를 가지고, 메모리는 상수를 저장하는 데이터 메모리, 지역 변수와 전역 변수를 위한 로컬 메모리, 연산을 위한 스택 메모리로 구성된다. 가상 기계어는 eFlowC 언어의 문법과 구문 트리를 기반으로 생성된다. eFVM의 가상 기계의 명령은 스택 관리 (INIT, POP, POPB, PUSH, OFFSET, CNV, MIN, STO, STOB, STX, STXB, LIT, LOD, LDI, LDIB, LDX, LDA, TSTAMP_S, TSTAMP_US), Bit 조작 (COMPL, AND, OR, NOT, BAND, BOR, BXOR, SHL, SHR), 산술 연산 (INC, DEC, ADD, ODD, SUB, MUL, MOD, DIV), 흐름 제어 (JMPEQ, JMPNEQ, JMPLE, JMPGE, JMPGTR, JMPLEQ, RET, CALL, JUMP, JUMPC, JUMPT, JUMPF, CMP, CALLLIB, JEQ, JNEQ), 데이터 전송 (DBYTECPY, DWORDCPY), 시스템 명령 (OUT, HALT, BRKPOINT), 코프로세서 (COPINIT, COPIN, COPOUT, COPRUN)를 지원하며, 프로그램의 실행을 위해 시뮬레이터를 제작하여 설치하였으며 디버거 같은 개발환경과 연동하여 실행할 수 있도록 구현하였다.

2. 컴파일러 구조

eFlowC 컴파일러는 본 논문에서 제안하는 패킷 처리용 위한 고급 언어인 eFlowC 로 작성한 프로그램을 입력으로 하여 가상 기계어 목적 코드를 생성한다. eFlowC 컴파일러의 개괄적 구조는 그림 5 와 같이 구성되었다. 컴파일러는 전단부(front-end)와 후단부(back-end)로 나뉘어 지는데, 전단부에서는 어휘 분석과 구문 분석 및의미 분석과 같은 일반적인 컴파일 과정을 수행하며, 후단부에서는 분석한 정보를 바탕으로 기본적인 최적화 등의 과정을 거쳐 최종 결과물인 가상 기계의 목적 코드를 생성한다.

Preprocessing을 위해 기존의 cpp를 그대로 사용하고 있으며, 어휘 분석을 위하여 lex를 사용하였다. 구문분석을 위하여 부록을 포함하는 eFlowC 언어의 문법과 이를 신택스 트리로 변환하기 위한 정의를 설계하고, yacc을 이용하여 파싱과 변수의 스코프 등 일차적 의미 분석을 병행하여 신택스 트리로 변환하는 과정을 수행한다.

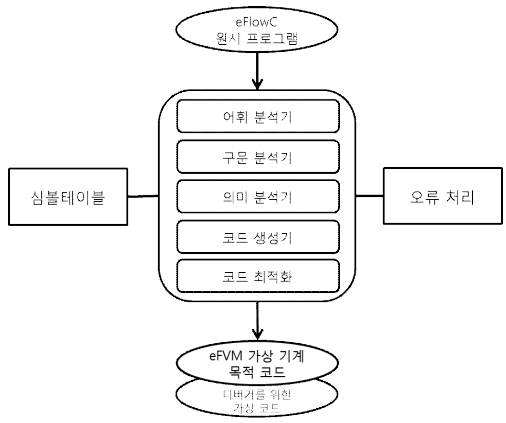

그림 5. eFlowC 컴파일러 구성
Fig. 5. Constitution of eFlowC Compiler

의미 분석 과정에서는 데이터 타입 검사, 변수의 메모리 할당 주소계산, 초기화 수식 검사, 수식의 정당성 과 타입 변환 검사, 레이블 사용의 정당성 검사, try-catch 문의 정당성 검사, 함수 호출과 매개변수 검사, 패킷이나 데이터베이스 구조와 사용의 적합성 검사 등을 수행한다.

코드 생성 과정을 위해서 신택스 트리로부터 4-tuple의 중간 코드로 변환하고 다시 가상기계어로 변환하기 위한 패턴을 정의하고 이를 C 언어로 구현하였다. 지역적인 최적화 기능등 기본적인 최적화를 수행하도록 하였으며 추후의 보다 고급의 최적화 기술을 구현할 수 있도록 모듈화 하였다.

컴파일러에서 디버깅 옵션이 설정되어있는 경우 가상 기계 목적 코드 외에도 디버거를 위하여 가상 코드를 추가로 생성하는데 원시 프로그램 소스 및 디버거의 각 명령문에서 사용가능한 모든 명칭에 대한 정보를 수록하도록 하였다.

3. 디버거

eFlowC 프로그래밍을 지원하기위해 edb 디버거를 구현하였는데, 많이 사용하는 gdb의 일반적 기능과 일치하도록 설계하였으며, 일반 프로그램과 다른 패킷과 pib 구조내용 및 데이터베이스 등 패킷처리에서 나타나는 정보를 추적할 수 있도록 하였다. 구현한 edb 는 컴파일러 실행 시에 옵션으로 주어진 경우에 생성되는 가상 기계어 프로그램과 디버거 수행을 위한 정보 (즉, 원시프로그램 구조, 원시프로그램에서 사용하는 모든 명칭과 메모리 할당 정보 등)을 입력으로 하여 가상 기계 시뮬레이터와 같이 실행되도록 하였다. 가상 기계어 프로그램을 위한 디버거 구현 기술이 잘 알려져 있지 않음으로, 디버거를 위한 추가적인의사코드 설계하여 이를 운용하고 디

버거의 각 명령을 처리하기 위한 알고리즘을 개발하여 구현하였다.

디버거의 명령은, 원시프로그램출력(list), 오류의 역추적(backtrace), 중단점 제어(break), 중단점 삭제(delete), 명칭정보 출력(whatis), 변수나 패킷 내용 출력(print), 중단점이나 변수와 함수의 정보 출력(info), 중단점활성화(enable, disable), 실행제어(run,step,next,continue,return), 감시점 설정(watch), 함수호출(call) 사작과 종료(edb,quit) 명령문을 지원하고 있다.

V. 실험

eFlowC 언어와 컴파일러는 DPIC 응용 프로그램을 만들기 위해서 제안되었으며 일반적인 C 언어의 기능도 거의 유사하게 가지고 있다. 그러나 DPIC 응용 프로그램은 패킷 처리를 해야 하는데, 개발자들이 이런 응용 프로그램을 효율적으로 개발할 수 있는 기능을 제안한 eFlowC 언어가 가질 수 있도록 구현하였다. 실험은 리눅스 페도라를 사용하는 시스템으로 인텔(Intel) 제논 CPU, 2GB 메모리를 사용하는 플랫폼이다. 본 연구에서 구현한 eFlowC 언어의 프로그램이 적합하게 사용되고 활용될 수 있는가를 테스트하기 위하여 패킷 처리 기능을 중심으로, 세 가지로 나누어 실험을 수행하였다.

첫째는, 관련분야에서 많이 사용될 수 있는 대표적 프로그램 72개를 선택하여 실험을 수행하였다. eFlowC 언어의 거의 모든 데이터 타입과 명령을 수록하고 있으며 특히 패킷, 데이터베이스 타입의 자료 및 예외처리를 하는데 중점을 두었다. 저급언어서 다루는 모든 프로그래밍 기술이 eFlowC 언어로 표현되고 사용에 편리한가를 실험하였으며, 저급언어에 비한 프로그램 개발의 생산성이나 신뢰성을 측량하지는 않았다.

둘째는, 컴파일러의 기능을 실험하였다. 기능의 정확성을 증명하기 위한 좋은 기술이 없으므로 앞서 언급한 대표 프로그램의 번역된 목적 코드를 일일이 검토하였으며, 시뮬레이터를 통하여 올바로 실행되는가를 검사하였다.

셋째는, 개발환경의 기능을 실험하였다. 어셈블러나 디버거 등의 개발환경이 새로운 기술이나 알고리즘을 가지고 개발한 것은 아니나, 패킷처리에 필요한 특별한 데이터들과 명령을 다루고 있어서 이들이 정확하게 기능하는지를 검토하였다.

다음의 실험예는 패킷의 삽입과 삭제를 수행하는 원시 코드를 테스트 한 것이다. 변수의 초기화, 기본적인 try-catch 문, 패킷 insert 및 delete를 테스트 할 수 있다. 그림 6 은 소스 코드이며, 그림 7 은 실행 후의 결과를 보여준다. 소스 코드에서 pkt_insert()와 pkt_delete()를 사용한 결과가 정확하게 도출되는 것을 확인할 수 있는데, 이런 라이브러리를 이용해서 패킷 처리 응용 프로그램을 쉽게 작성할 수 있다.

```
#include "packet.h"
module (PACKET pkt)
#include "packet1.h"

byte barr[4] = {6,7,8,9};
int main() {
        int    i = 10;
        try {
                pkt_insert(pkt,16,3,barr);
                pkt_delete(pkt,34,i);
                printi(i);
        }
        catch(PKT_INSERT_ERR){ printi(96); }
        catch(PKT_DELETE_ERR){ printi(97); }
}
end module
```

그림 6. 소스 코드-1
Fig. 6. Source Code-1

```
result: 10

정상 패킷

00deadbeef0000debb00000100084500
01ee0000000400160600a0a02690a0a
023300006c570003002a000102030405

삽입, 삭제 진행한 패킷

00deadbeef0000debb00000100084500
060708 01ee0000000400160600a0a02
690a2a0001020304050607080 90a0b0c
삽입된 패킷 : 060708
삭제된 패킷 : 023300006c570003002a
```

그림 7. 실행 결과-1
Fig. 7. Execution Result-1

다음의 실험 예는 원시프로그램이 패킷을 처리하는 중에 데이터베이스 자료를 구성하는 것이다. 데이터베이스를 선언하고 레코드의 삽입, 삭제 및 업데이트 기능을 테스트 한다. 정확히는 데이터베이스 aclDb와 myrec를 선언하고 aclDb의 비어있는 공간을 찾아 myrec를 삽입, aclDb에 3번째 멤버를 삭제, 1번째 멤버를 readData 로 업데이트하는 프로그램이다. 그림 8 은 소스 코드이고, 그림 9 는 eFlowC가 실행된 후에 출력되는 실행 결과로서 본 연구에서 제안한 데이터베이스 데이터 타입과 그에 따른 라이브러리가 정확하게 동작되고 있음을 보여준다. 데이터베이스 타입과 라이브러리 또한 패킷 처리를 위한 응용 프로그램 개발에 편리하게 사용될 수 있다.

```
#include "packet.h"
module (PACKET pkt)
#include "packet1.h"

typedef database AclStruct {
        int srcIp; int destIp;
        short srcPort; short destPort;
        } DATABASE;
        DATABASE aclDb[6] = {
              {{10.10.20.80, 10.10.10.50, 25, 8}, {~0, ~0, ~0, ~0}},
              {{10.10.20.81, 10.10.10.51, 25, 8}, {~0, ~0, ~5, ~0}},
              {{10.10.20.82, 10.10.10.52, 25, 8}, {~0, ~0, ~5, ~0}},
              {{10.10.20.83, 10.10.10.53, 25, 8}, {~0, ~0, ~5, ~0}},
              {{10.10.20.84, 10.10.10.54, 25, 8}, {~0, ~0, ~0, ~0}},
              {{0.0.0.0, 0.0.0.0, 0, 0}, {0, 0, 0, 0}}
        };
database AclStruct myrec
              = {{10.10.30.100,10.20.30.50, 29, 10},{~0,~0,~0,~0}};
 int main() {
        DATABASE readData, readData2;
        readData2 = aclDb[4];
        aclDb[1].srcIp = ipv4.sourceAddress;
        aclDb[1].destIp = ipv4.destinationAddress;
        readData = aclDb[2];
        try {
                db_insert(aclDb,myrec); // insert
                db_delete(aclDb,3); // delete
                db_update(aclDb,1,readData); //update
                db_match(aclDb, readData2); //match
        }
         catch(DB_INSERT_ERR){ printi(97); }
        catch(DB_DELETE_ERR){ printi(98); }
        catch(DB_UPDATE_ERR){ printi(99); }
        catch(DB_MATCH_ERR){ printi(100); }
}
end module
```

그림 8. 소스 코드-2
Fig. 8. Source Code-2

```
{(data=(srcIp=10.10.20.80,destIp=10.10.10.50,srcPort=0.0.0.25,destPort=0.0.0.8),
mask=(srcIp=255.255.255.255,destIp=255.255.255.255,srcPort=0.0.0.255,destPort=0.
0.0.255)),(data=(srcIp=10.10.20.82,destIp=10.10.10.52,srcPort=0.0.0.25,destPort=
0.0.0.8),mask=(srcIp=255.255.255.255,destIp=255.255.255.255,srcPort=0.0.0.250,de
stPort=0.0.0.255)),(data=(srcIp=10.10.20.82,destIp=10.10.10.52,srcPort=0.0.0.25,
destPort=0.0.0.8),mask=(srcIp=255.255.255.255,destIp=255.255.255.255,srcPort=0.0
.0.250,destPort=0.0.0.255)),(data=(srcIp=0.0.0.0,destIp=0.0.0.0,srcPort=0.0.0.0,
destPort=0.0.0.0),mask=(srcIp=0.0.0.0,destIp=0.0.0.0,srcPort=0.0.0.0,destPort=0.
0.0.0)),(data=(srcIp=10.10.20.84,destIp=10.10.10.54,srcPort=0.0.0.25,destPort=0.
0.0.8),mask=(srcIp=255.255.255.255,destIp=255.255.255.255,srcPort=0.0.0.255,dest
Port=0.0.0.255)),(data=(srcIp=10.10.30.100,destIp=10.20.30.50,srcPort=0.0.0.29,d
estPort=0.0.0.10),mask=(srcIp=255.255.255.255,destIp=255.255.255.255,srcPort=0.0
.0.255,destPort=0.0.0.255))}
```

그림 9. 실행 결과-2
Fig. 9. Execution Result-2

대표 프로그램의 실험을 통하여 제안하는 네트워크 통합 관리용 DPIC 응용 프로그램 개발을 위한 고급 언어는 패킷 처리를 위한 특수한 데이터 타입과 라이브러리가 정확하게 동작하도록 구현되어 있으며, 패킷 데이터, 네트워크 상수, 데이터베이스 등은 패킷을 처리하고 다룰 때 개발자 입장에서 매우 편리하도록 되어 있으며 부분 배열 회수나 그 밖의 패킷, 데이터베이스 라이브러리들도 패킷 처리 응용 프로그램을 작성할 때 편리함이나 활용도를 높여줄 수 있다. 저급 언어로 프로그래밍 하던 것을 고급 언어 환경으로 변화시켜 개발자들에게는 편리함을, 소프트웨어 생산 업체에게는 생산성을 높여 줄 수 있을 것으로 기대한다.

VI. 결론

본 논문에서 제안한 eFlowC 언어는 기존의 C 언어를 기반으로 하되 패킷 처리를 위한 네트워크 상수 테이터, 패킷 과 패킷 정보 블록 테이터, 데이터베이스 데이터 등을 지원하게 설계하였고, 에러 이벤트 등의 예외처리 기능을 포함하여 이들 데이터들을 효율적으로 처리할 수 있는 라이브러리를 다양하게 지원한다. 현재 패킷 처리 분야에서 개발하고 사용하고 있는 저급언어 프로그램을 분석하여 72개의 다양한 응용 프로그램을 실험하여 정확하게 실행되고 있음을 실험하여, 본 논문에서 제안하는 eFlowC 언어의 장점에 해당되는 패킷 처리 관련 기능이 제대로 동작하는지 실험을 통해 기능 검증을 수행하였고, 정확하게 동작한 실행 결과를 확인할 수 있었다.

본 연구의 eFlowC 언어와 컴파일러의 구현을 통해 다음과 같은 장점을 얻을 수 있다. 첫째, 저급 언어를 대체하여 프로그램의 생산성을 높일 수 있을 것으로 기대하고, 둘째, 국내 핵심 기술을 확보함으로써 해외에 지급되는 사용료를 줄일 수 있어 국내의 트래픽 분석에 대한 연구가 활발하게 진행될 수 있고, 셋째, 고급 프로그래밍 언어를 사용하여 코드의 재사용성이 증가하여 응용 프로그램의 개발에 사용되는 비용을 절감할 수 있고, 넷째, C 언어와 유사한 문법적 구조를 가짐으로 개발자들의 새로운 언어를 익히는데 드는 시간과 비용을 절감할 수 있을 것이다. 그리고 마지막으로 가상 기계 목적 코드의 출력을 통해 표준화된 하드웨어의 종속성을 낮출 수 있다.

실험을 위해 설계한 eFVM은 가상 기계로서 디버거와 시뮬레이터와 함께 개발자가 한 번의 응용 프로램 개발로 다양한 프로세서에서 동작할 수 있도록 이식이 가능하기 때문에 네트워크 통합 관리용 DPIC 응용 개발에 다양성과 생산성을 높여줄 수 있도록 하였다. 특히 기능 검증을 통하여 보인 결과들이 패킷 처리를 편리하게 만드는 기능들이므로 많은 개발자들이 사용할 수 있고 eFlowC 언어의 활용도를 높일 수 있을 것이다. 따라서 eFlowC 언어를 이용하면 기존의 ASIC와 같이 하드웨어로 로직을 구성하거나 기계어 수준의 저급 언어로 개발하는 방법보다 적은 비용, 고효율로 네트워크 처리 응용 프로그램을 개발할 수 있다.

참고문헌

〔1〕 A.V.Aho, M.S.Lam, R.Sethi, J.D.Ullman, *"Compilers Principles, Techniques and Tools"*, Addison-Wesley, 2007.

〔2〕 M.Baldi, F.Risso, "A Framework for Rapid Development and Portable Execution of Packet -Handling Applications", Signal Processing and Information Technology, pp.233-238, 2005.

〔3〕 CloudShield Technologies Inc., *"PacketWorksTM Integrated Development Environment"*, CloudShield, 2007.

〔4〕 CloudShield Technologies Inc., *"PacketC Unleashed"*, CloudShield, 2008.

〔5〕 S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. W. Lockwood, "Deep Packet Inspection Using Parallel Bloom Filters", Hot Interconnects 11(HotI), pp.44-51, Stanford, CA, USA, Aug. 2003.

〔6〕 L.Degioanni, M.Baldi, D.Buffa, F.Risso, F.Stirano, G.Varenni, "Network Virtual Machine(NetVM): A New Architecture for Efficient and Portable Packet Processing Applications", Proceedings of the 8th International Conference on Telecommunications, pp.163-168, 2005.

〔7〕 R.Duncan and P.Jungck, "PacketC Language for High Performance Packet Processing", Proceedings of the 11'th IEEE International Conference on High Performance Computing and Communications, Seoul, Korea, pp. 450-457, June 2009.

〔8〕 R.Duncan, P.Jungck and K.Ross, "PacketC Language and Parallel Processing of Masked Databases," *Proceedings of the 2010 International Conference on Parallel Processing,* pp. 472-481, 2010.

〔9〕 R.Duncan, P.Jungck, K.Ross and S.Tillman, "Packet Content Matching with packetC Searchsets", Proceedings of the International Conference on Parallel and Distributed Systems, 2010.

〔10〕 B. Gatliff, *"Embedding with GNU : GNU debugger"*, Embedded Systems Programming, 1999

〔11〕 B.W. Kernighan, D. M.Ritchie, *"The C Programming Languages"*, Prentice-Hall, 1988.

〔12〕 Kevin R. Fall, W. Richard Stevens, *"TCP/IP illustrated volume 1"*, Addison-Wesley, 2011.

〔13〕 S.P, Harbisonn G.L, Steele Jr., *"C: A REFERENCE MANUAL"*, Tartan Inc., 1995.

〔14〕 B. L, Kurtz, *"Formal Syntax and Semantices of Programming Language"*, Louisiana Tech University, 1995.

〔15〕 O.Morandi, F.Risso, Pierluigi, Rolando, S.Valenti, P.Veglia, "Creating Portable and Efficient Packet Processing Applications", Creating Portable and Efficient Packet Processing Applications, pp.51-85, Mar. 2011.

〔16〕 H. Park ,D. Xu, J.Park, J.H.Ji, G.Woo "Design of On -Chip Debug System for embedded processor," *Procedeeings of 2008 SoC Design Conference,* Dept. of Electron. Eng., Pusan Nat. Univ. 2008.

〔17〕 R.Stallman, R.Pesch, S.Shebs, *"GNU User Manual: Debugging With GDB (TheGNU Source-level Debugger)"*, GNU Presss, 2002

〔18〕 Xelerated, *"Xelerator X11 Network Processor User's Guide Part1-Programmable Pipeline"*, Xelerated Inc., 2010.

〔19〕 Y.R. Kang, "Internet Traffic Control and DPI", KISDI, Vol. 25(8), pp.23-48, May. 2013.

〔20〕 D.Y. Kim, H.Y. Cho, "Technology of Security Engine Based on Hardware", *ICAT 2003*, Apr. 2003.

〔21〕 S.Y. Shin, D.H. Kang, K.Y. Kim, J.S. Jang, "Analysis of DPI Technology", *Electronics and Telecommunications Trends,* Vol. 19(3), pp.117~124, 2004. 06

〔22〕 C.W. Yoo, BangWon Ko, *"A Study on the Application Development Environment and Simulator Architecture of High Speed In-line Network Computing"*, Final Report, ETRI, 2012

부록

```
%start program
%%
program
        : declaration_list  MODULE ( parameter_list )
          translation_unit END MODULE
〈중략〉
storage_class_specifier
        : BUFFER
        | STATIC
        | TYPEDEF
type_specifier
        : struct_or_union_specifier
        | enum_specifier
        | typedef_name
        | bit_specifier
bit_specifier
        : BIT ( constant_expression )
〈중략〉
struct_or_union
        : STRUCT
        | UNION
        | DATABASE
init_declarator
        : declarator
        | declarator = initializer
        | declarator REDEFINE unary_expression
〈중략〉
statement
        : labeled_statement
        | compound_statement
        | expression_statement
        | selection_statement
        | iteration_statement
        | jump_statement
        | throw_statment
        | try_statement
try_statement
        : TRY compound_statement  catch_list
catch_list
        : CATCH ( IDENTIFIER ) compound_statement
        | catch_list CATCH ( IDENTIFIER )
          compound_statement
throw_statment
        : THROW      IDENTIFIER ;
〈중략〉
primary_expression
        : IDENTIFIER
        | INTEGER_LITERAL
        | NETWORK_LITERAL
        | CHARACTER_LITERAL
        | STRING_LITERAL
        | ENUM_LITERAL
        | ( expression )
        ;
〈중략〉
```

저 자 소 개

고 방 원

2005 : 동아대학교 컴퓨터학부 공학사.
현 재: 숭실대학교 컴퓨터학부 석박사통합과정.
관심분야: 프로그래밍 언어, 컴파일러, HCI, XML
Email : bangwon.ko@gmail.com

유 재 우

1976: 숭실대학교 전자계산학과 공학사.
1978: 한국과학기술원 전산학과 공학석사.
1985: 한국과학기술원 전산학과 공학박사
현 재: 숭실대학교 정보과학대학 컴퓨터학부 교수
관심분야: 프로그래밍 언어, 컴파일러, 인간과 컴퓨터 상호 작용
Email : cwyoo@ssu.ac.kr