패킷 처리 프로그래밍을 위한 언어의 설계와 컴파일러의 구현

최예지, 고방원, 신경희, 유재우
숭실대학교 컴퓨터학부
e-mail:choiyeji@ssu.ac.kr

Design and Implementation of Packet Processing Programming Language and Compiler

YeJi Choi, BangWon Ko, KyoungHee Shin, ChaeWoo Yoo
Dept. of Computer Science & Engineering,Soongsil University

요 약

본 논문은 하드웨어에 독립적이고 패킷 처리 프로그래밍 개발의 효율성을 높이기 위하여 eFlowC 언어와 eFlowC 컴파일러를 제안한다. eFlowC 언어는 C 언어 기반의 고급언어이다. 그리고 기존의 C 문법에 패킷 처리 관련 기능을 위한 새로운 라이브러리를 추가하였다. eFlowC 컴파일러는 eFlowC 언어로 작성된 프로그램을 입력으로 받는다. 그리고 결과물로 가상 기계 목적 코드를 생성한다. 가상 기계 목적 코드는 언어 번역기의 입력 파일로 언어 번역기를 통하여 다양한 하드웨어 기계어로 번역이 가능하다.

1. 서론

네트워크의 사용량 증가로 의해 패킷의 분석과 재가공을 위한 방법이 요구되고 있다[1]. 패킷 프로세싱을 위한 언어는 대부분 저급언어로 구현되어 있다. 하지만 저급 언어는 개발 방법론을 적용하기 어려워 재사용성이 낮은 단점을 가진다. 이에 CloudShield 사에서는 네트워크 패킷을 위한 고급언어로 packetC 언어를 개발하였다. packetC는 저급언어로 설계된 다른 패킷 프로세싱 언어보다 프로그램의 개발이 용이하다. 하지만 CloudShield 사에서 제공하는 하드웨어 환경에서만 사용 가능하여 하드웨어에 종속적이라는 단점을 가진다. 그러므로 하드웨어에 독립적인 프로그래밍 언어와 개발 환경에 대한 연구가 필요하다. 이와 같은 연구로는 NetVM (NetWork Virtual Machine)이 있는데, NetVM은 패킷 프로토콜을 정의하는 XML 기반의 문서인 NetPDL과 프로그램의 흐름을 작성하는 고급언어인 NetPFL를 사용하여 프로그램의 요구사항을 만족시킨다. 이 방법은 패킷 데이터의 정의와 프로그램의 흐름을 분리하여 결합도를 낮추는 장점을 갖지만 개발자에게 새로운 언어에 대한 학습을 요구한다.

본 논문에서는 eFlowC 언어와 eFlowC 컴파일러를 개발하여 고성능 실시간 패킷 처리가 가능한 프로그래밍이 가능하도록 한다. eFlowC 언어는 C 기반의 고급언어로 설계하여 NetVM의 학습을 제공하지 않아도 된다. 그리고 eFlowC 컴파일러는 가상 기계 목적 코드를 생성하여 packetC의 단점인 하드웨어의 종속성을 없앤다. 또한 eFlowC 언어 및 eFlowC 컴파일러의 개발을 통하여 현재 패킷 처리 기술에 대한 높은 해외 의존도를 낮출 수 있고 더불어 해외에 제공되는 비용의 절감을 기대할 수 있다.

본 논문은 2장에서 관련 연구를 설명하고, 3장에서 eFlowC 언어에 대한 특징들을 설명한다. 그리고 4장에서 eFlowC 컴파일러 설계 및 구현 과정을 보이고 5장에서 연구 결과와 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 packetC

CloudShiled사의 packetC는 C언어의 문법을 기반으로 실시간 트래픽 제어를 위한 프로그래밍 언어이다[2]. 다양한 API를 통해 패킷 수정 및 생성, 병렬 처리와 같은 다양한 기능들을 제공한다.

2.2 NetVM(Network Virtual Machine)

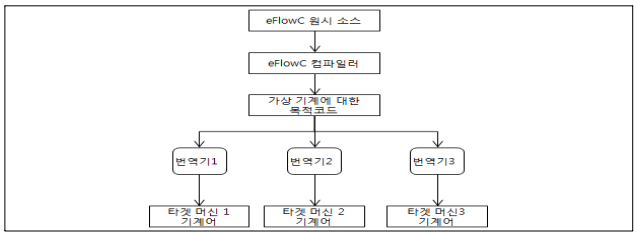
NetVM은 효율적인 패킷 프로세싱을 목적으로 하는 가상 기계이다. NetVM은 고급 프로그래밍 언어인 NetPFL로 개발된 네트워크 처리 프로그램과 프로토콜 데이터베이스의 역할을 하는 NetPDL을 사용하여 컴파일 한다. 그 결과로 NetVM 바이트코드인 NetIL을 생성한다. NetIL은 하드웨어 아키텍쳐로 네이티브 코드 안에서 네트워크 프로세서나 다른 하드웨어 시스템으로 번역되어 전달된다[3].

3. eFlowC 언어

3.1 개요

eFlowC 언어는 C 언어의 문법을 기반으로 하여 패킷 처리를 위한 목적으로 개발된 언어로 C 언어를 사용해 본 개발자가 쉽게 접근하여 프로그래밍을 할 수 있도록 하였

다. 다른 패킷 처리 언어는 타겟 머신에서 동작할 수 있는 목적 코드를 직접 생성하여 하드웨어에 종속적이다. 그러나 eFlowC 언어는 기존 언어들과 다르게 타겟 머신 기계어를 바로 생성하지 않고, 가상 기계어 목적 코드로 생성하여 타겟 머신의 종류에 상관없이 독립적으로 사용이 가능하다.



(그림 1) eFlowC의 시스템 구조도

다음 그림은 eFlowC 언어로 작성된 예제 프로그램이다.

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

int srcIP
int counter;

main() {
     srcIP = ipv4.sourceAddress;
     if (srcIP == 10.0.0.10) {
          counter++;
     }
     printi(counter);
}
end module
```

(그림 2) eFlowC 예제

헤더 파일로 packet.h 이 있는데 그 내부에는 시스템 상수, 패킷 라이브러리와 시스템 자료형인 PACKET, PIB에 대한 데이터 및 구조체가 정의 되어 있다.

3.2 eFlow 언어의 특징

(1) 시스템 자료형

PACKET과 PIB는 패킷을 처리하기 위하여 새롭게 추가된 자료형이다. PACKET은 패킷의 대한 타입이며, PIB는 패킷 정보 블록에 대한 구조체 정의이다. PACKET은 바이트의 배열로 표현되고 패킷을 받아 바이트 배열처럼 다룰 수 있다. 응용 어플리케이션에서 PACKET은 pkt 변수를 통하여 참조된다. PIB는 packet information block의 약자로 패킷 안에 포함되어 있는 다양한 네트워크 레이어와 그 오프셋에 대한 정보를 담고 있다. 응용어플리케이션에서 PIB는 pib 변수를 통하여 참조된다.

```
typedef struct s_node {
                NODE_NAME name;
                int line;
                char *fn;
                struct s_type  *type;
                struct s_node *llink
                struct s_node *clink;
                struct s_node *rlink } A_NODE;
```

(그림 3) packet.h 파일에 선언된 PIB

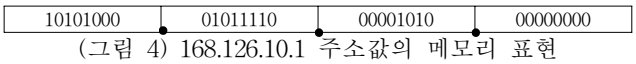
(2) 자료형과 상수

eFlowC 언어는 기본적으로 byte, int, long, short 형을 지원한다. eFlowC 언어에서는 C 언어에서 지원하고 있는 포인터 형과 불린 형, 부동소수점 형을 지원하지 않는다. 포인터 형의 경우에는 실행 시간 시에 충돌을 피하기 위해서[2] 주소 연산과 함께 eFlowC 언어에서 제거하였다. 불린 형은 true와 false에 대한 값이 packet.h 파일 안에 정의되어 있으므로 eFlowC 언어에서는 지원하지 않으며 부동소수점 형 또한 불필요하여 지원하지 않는다.

eFlowC 언어에는 데이터베이스는 확장된 자료형으로 다음 패킷이 처리되는 동안 데이터를 찾고자 할 때 사용한다. 데이터베이스는 배열로 선언되고 데이터 요소를 테이블 형태로 나타내며 필요한 데이터를 찾을 수 있게 해준다. 아래는 데이터베이스 선언의 문법 정의이다.

database *tyName identifier* [*constant_expression*]
identifier_init_clause_opt;

eFlowC 언어에서 네트워크 상수를 새롭게 추가하였다. 네트워크 상수는 32bit의 크기를 갖으며 정수형 데이터 타입이다.



(그림 4) 168.126.10.1 주소값의 메모리 표현

(3) 예약어

eFlowC 언어에서는 기존의 C 언어 예약어 목록에 buffer, catch, database, exit, module, offset, redefine, throw, try 라는 예약어를 추가하였다.

<표 1>은 새롭게 추가된 예약어에 대한 간단한 설명과 예제이다.

<표 1> eFlowC 언어에 추가된 예약어

예약어	설 명	예 제
module	매크로와 헤더파일 후에 선언	module(PACKET pkt)
buffer	변수를 버퍼 메모리에 저장	buffer byte bff[10000];

try/catch/ throw	에러 처리를 위해 사용	try{
		int a;
		throw(CC);
		}catch(CC) { ; }

database	데이터베이스 자료형에서 사용	database aa a[10];

exit	조건식에서 종료를 표시	if(a==b){
		b = 1;
		exit; }

end	프로그램의 마지막에 module 예약어와 함께 사용	end module

offset	구조체 필드의 시작 주소	struct test{
		int a; } t;
		int c = offset(t.a);

redefine	PIB 자료형의 논리적인 주소를 사용	struct Ipv4Struct{ byte Length; byte tos; short totalLength; ... int destAddress; } ipv4 redefine pib.l3Offset;

(4) 패킷 라이브러리

eFlowC 언어에서는 패킷 관련 라이브러리를 선언하였다. 패킷 관련 라이브러리는 db_delete, db_insert, db_match, str_find, pkt_insert, pkt_delete, pkt_replicate, pkt_requeue, str_find, printi 이다. <표 2>는 라이브러리에 대한 설명이다.

<표 2> 패킷 라이브러리

구 분	라이브러리	설명	에러
데이터베이스	db_insert(arg1,arg2)	데이터베이스에 데이터를 삽입	DB_INSERT_ERR
	db_delete(arg1,arg2)	데이터베이스에서 데이터를 삭제	DB_delete_ERR

패킷	pkt_insert(arg1,arg2,arg3,arg4)	패킷에 지정한 크기만큼의 데이터를 삽입	PKT_INSERT_ERR
	pkt_delete(arg1,arg2,arg3)	패킷에 지정한 크기만큼의 데이터를 삭제	PKT_DELETE_ERR
	pkt_replicate(arg1)	패킷의 사본을 생성	PKT_REPLICATE_ERR
	pkt_requeue()	패킷의 처리를 다음으로 연기하여 처리	PKT_REQUEUED_ERR
바이트 배열	str_find(arg1,arg2)	배열에서 특정 문자열을 검색	STR_FIND_ERR
디버깅	printi()	디버깅 결과 출력	-

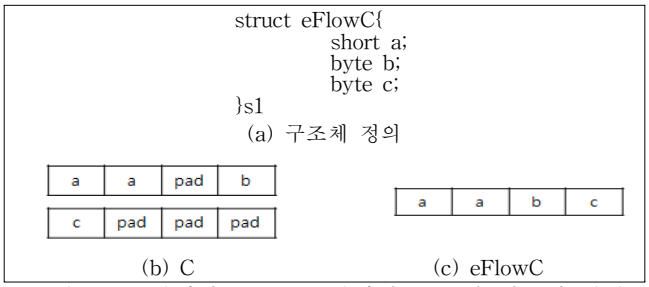
(5) try-catch-trhow문

eFlowC 언어에서는 에러 핸들링을 위하여 try-catch-throw문을 사용한다. try-catch-throw 문은 사용자가 직접 에러를 정의하여 핸들링 할 수도 있지만 printi를 제외한 나머지 패킷 라이브러리에 대한 에러를 핸들링 할 때 사용한다([표 2] 참조). eFlowC 언어는 에러 핸들링을 통하여 프로그램의 안정성을 높이고, 에러가 없는 코드를 작성할 수 있게 한다. 다음은 try-catch-throw 문에 대한 문법 정의이다.

```
try compound_statement catch_list
catch( identifier ) compound_statement
catch( … ) compound_statement;
```

(6) 구조체의 메모리

eFlowC 언어와 C 언어는 구조체의 메모리 할당에서 차이를 보인다.



(그림 5) C 언어와 eFlowC 언어의 구조체 메모리 할당

위 그림에서 (a)는 구조체로 작성된 간단한 예제이고, (b)와 (c)는 s1에 대한 C 언어와 eFlowC 언어의 메모리 할당 형태이다. (b)의 경우 CPU 연산 속도를 위해 데이터를 4바이트 단위로 할당하는 메모리 구조로 인하여 바이트 패딩(byte padding)이 존재하지만[4] 그와 달리 (c)에서는 바이트 패딩이 존재하지 않는다. 바이트 패딩은 메모리의 낭비가 생기고 데이터의 오버헤드를 발생시키는 문제점이 있는데 eFlowC 언어에서는 바이트 패딩을 없앰으로써 그러한 문제점을 해결하였다.

(7) 배열

eFlowC 언어에서는 배열을 1차원 배열 또는 2차원 배열으로만 선언이 가능하며 C언어와는 다르게 배열의 크기를 지정하지 않고 배열 선언이 가능하다. 그리고 eFlowC 언어에서는 배열을 다룰 때 배열 슬라이스를 지원한다. 배열 슬라이스는 포인터 없이 바이트 배열을 참조하여 직접 접근할 수 있도록 해준다. 배열 슬라이스는 'a[start : end];'와 같은 형태이며 시작 오프셋과 지정 오프셋은 콜론 ':' 으로 구분한다. 사용 시 주의할 점으로 시작 오프셋(start)의 크기는 지정 오프셋(end)보다 작아야 한다.

4. eFlowC 컴파일러의 설계와 구현

4.1 구성

eFlowC 컴파일러는 eFlowC 원시 프로그램을 입력 파일로 받아 가상 기계에 대한 목적코드를 생성한다. 아래 (그림 5)는 본 컴파일러의 구조이다.

(그림 6)eFlowC 컴파일러의 구성

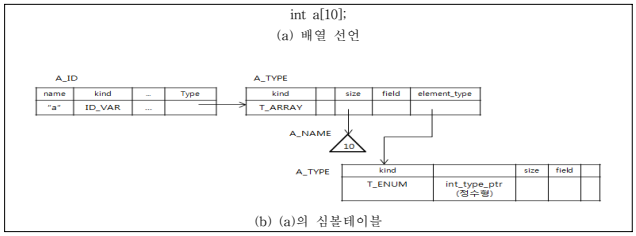
eFlowC 컴파일러는 (그림 6)과 같이 5단계를 거쳐 목적코드인 가상 기계어를 생성하게 된다. 본 컴파일러는 전반부(Front-end)와 후반부(Back-end)로 구성되는 2-pass compiler 방식을 사용하여 구현하고 있다[5]. 전반부에는 어휘 분석, 구문 분석, 의미 분석이 수행되며 후반부에선 코드를 생성하고 최적화 단계를 거쳐 가상 기계어 코드를 생성하게 된다. eFlowC 언어는 다중 스택 메모리 구조를 제공하는데, 사용되는 스택의 종류는 실시간으로 받은 패킷을 저장하는 스택과 일반 데이터를 저장하는 스택 그리고 메모리를 공유하기 위한 스택으로 나뉜다.

4.2 구현

eFlowC 문법은 스택 메모리 구조로 상향식 파싱이 가능하도록 정의되어 있다. 본 컴파일러의 구문 분석에서는 상향식 파싱의 기술 중에서도 LR 파싱을 사용하여 구현하였다. 왼쪽에 있는 터미널 노드에서 부터 의미 분석과 문법을 확인하며 루트 노드 쪽으로 트리를 구성한다. 상향식 파싱을 하기 위하여 문법은 BNF 표기법과 부문 지향적 정의 방법을 사용하여 기술하였다[5].

(1) 전반부

eFlowC 컴파일러는 구분 분석과 동시에 구문 트리를 생성하고, 자료형이나 함수 또는 변수들의 이름을 분석한다. 구문 분석기에선 프로그램에서 사용하는 각종 자료형이나 함수, 변수들에 대한 모든 정보를 저장하고 관리하기 위하여 심볼 테이블을 사용한다. 심볼 테이블은 type.h 파일에 자료형에 대한 정보를 담고 있는 구조체와 사용되는 모든 식별자들의 정보를 담고 있는 구조체 그리고 노드에 대한 정보를 담고 있는 구조체 세 개의 구조체로 정의되어 있다. 심볼 테이블에는 자료형과 함수, 변수의 이름과 에러 처리 시 필요한 라인 번호, 각각의 자식 노드들의 오프셋과 타입의 크기 등에 대한 다양한 정보를 담고 있으며 메모리 할당 및 주소의 정보를 계산할 수 있다.



(그림 7) 심볼 테이블 구조 그림

(그림 7)에서 (a)는 간단한 배열 선언이고, (b)는 (a)에 대한 심볼 테이블을 그림으로 나타낸 것이다. 구문 분석기에서는 축약된 형태의 추상 구문 트리를 생성하는데 트리의 각 노드들은 세 개의 터미널 노드를 가지고 서로 연결된다. 프로그램의 구조와 명령문은 구문 트리로 변환된다. 구문 트리는 MODULE을 루트로 하여 원시 프로그램과 동일한 의미 정보를 보관하게 되며 C 언어 계열의 다른 언어와 새로운 언어에서도 적용될 수 있다. 의미 분석기는 구문 분석기를 통하여 생성된 구문 트리를 분석하여 프로그램에서 자료형이나 이름, 메모리 주소 등이 알맞게 사용되었는지 검사 한다.

(2) 후반부

본 컴파일러에서는 모듈화를 통하여 코드 생성기를 구현한다. 가상 기계 목적 코드를 생성하기 전에 코드의 최적화 과정이 필요한데 코드의 최적화는 생성한 코드에 대해서 불필요하게 반복되는 부분 등을 없애 프로그램의 실행 효율이 증진될 수 있도록 해준다.

4.3 컴파일 결과

eFlowC 컴파일러는 직접 타겟 머신 기계어를 생성하지 않고, 가상 기계 목적 코드를 만든다. 가상 기계 목적 코드는 스택 기반의 메모리 구조를 가지고 있으며 메모리는 상수를 저장하는 데이터 메모리와 지역 변수 및 전역 변수를 위한 로컬 메모리, 연산을 위한 스택 메모리로 구성되어 있다. 가상 목적 코드는 eFlowC 언어의 문법 구조와 구문 트리를 바탕으로 생성된다.

아래의 표는 가상 기계어를 생성할 때 사용되는 명령어의 종류와 그에 해당하는 대표적인 명령어에 대한 설명이다.

<표 3> 가상 기계어 명령어 분류

분 류	설 명	명령어	설명
초기화	실행 전 초기화	INT	지정한 크기를 메모리에 할당
데이터 이동	메모리 내에서 데이터를 이동	STOB	BYTE 크기만큼 메모리를 할당하여 저장
흐름 제어	흐름을 제어	JPC	지정한 레이블로 이동
패턴 매칭	메모리의 값을 비교	EQL	같은 레벨에 있는 값이 같은지 검사
수식 계산	간단한 표현식을 계산	ADD	두 개의 값을 더함

eFlowC 컴파일러에서는 eFlowC 언어의 문법 구조와 의미에 맞게 가상 기계 언어의 명령어를 선택하여 적절하게 대입시켜 목적 코드를 생성한다.

(그림 8)은 실제 구현되어 출력된 컴파일의 결과에 대한 예제로 입력파일로 (a)를 받아 컴파일하여 그에 해당하는 목적코드인 (b)가 생성된다.

```
#include "packet.h"
module(PACKET pkt)
main {
 int a,b,c
 a=1;
 b=2;
 c = a+b
}
end module
```

```
INT    0, 9264
       CAL    0, main
       HALT   0, 0
.global 0
.global 9216
       .WORD   0
       .WORD   0
       .WORD   0
       .WORD   0
       .WORD   0
       .WORD   0
       .WORD   0
       .WORD   0
       .WORD   0
       .WORD   14

       .WORD   33
```

(a)

```
main:  .WORD   0

       INT    0, 24
       LDA    1, 12
       LIT    0, 1
       STXB   0, 4
       POPB   0, 4
       LDA    1, 16
       LIT    0, 2
       STXB   0, 4
       POPB   0, 4
       LDA    1, 20
       LDA    1, 12
       LDIB   0, 4
       LDA    1, 16
       LDIB   0, 4
       ADD    0, 0
       STXB   0, 4
       POPB   0, 4
       RET    0, 0
```

(b)

(그림 8) 실제 구현된 컴파일 결과 예

5. 결 론 및 향후 연구 방향

본 논문에서는 실시간 패킷 처리를 위한 eFlowC 언어와 eFlowC 컴파일러를 설계하였다. eFlowC는 C 언어의 구조와 문법을 기반으로 설계된 언어이다. 그리고 패킷을 처리하기 위하여 새로운 상수와 라이브러리 자료형을 생성하였다. 또한 C에서 사용되는 포인터 형과 부동 소수점형 등과 같은 불필요한 구조는 eFlowC에선 제공하지 않는다. eFlowC 컴파일러는 실시간으로 패킷을 메모리에 저장하여 패킷 처리를 가능하게 하였다. eFlowC 컴파일러의 결과물로는 가상 기계어가 생성된다. 가상 기계어는 언어 번역기를 통하여 어느 하드웨어에도 번역될 수 있다. 또한 가상 기계어 목적 코드를 생성함으로써 eFlowC 언어는 하드웨어로부터 독립적으로 사용될 수 있다.

향후 연구로는 생성된 가상 기계어를 입력으로 받아 특정 타겟 머신 기계어로 번역할 수 있도록 해주는 언어번역기에 대한 활발한 연구가 필요하다. 또한 아직 구현되어 있지 않은 보안과 병렬처리, 패킷의 생성과 같은 추가적인 기능을 제공해 줄 수 있는 라이브러리에 대한 연구가 필요하다.

6. 참고 문헌

[1] 이현신, 김명섭 "실시간 트래픽 분석을 위한 운영체제 판별 방법에 관한 연구", 한국통신학회 논문지, Vol. 36, No. 5, pp. 443-450, 2011.05.

[2] Ralphduncan, Peder Jungck, Ralph "PacketC Language for High Performance Packet Processing", Proceedings of the IEEE International Conference on High Performance Couputing and Communications, pp. 450-457, 2009.

[3] M. Baldi and F. Risso. "A framework for rapid development and portable execution of packet-handling applications" In ISSPIT 2005: Proceedings of the 5th IEEE International Symposium on Signal Processing and Information Technology, Athens, Greece, December 2005. 2005.

[4] Brian W. Kernighan, Dennis M.Ritchie, "*The C Programming Languages*",Prentice-Hall, 1988.

[5] Alfred V.Aho, Monica S.Lam, Ravi Sethi, Jeffrey D.ullman, "*Compilers Principles, Techniques, & Tools*", Addison-Wesley, 2007.