CLICK Modular Router를 위한 자기 재설정 기법*

류진형°, 김혜진, 이재국, 김형식
충남대학교 대학원 컴퓨터공학과
jinsto@cnu.ac.kr, angella@csalab.cnu.ac.kr, {empire, hskim}@cs.cnu.ac.kr

A Self-reconfiguration Scheme for CLICK Modular Router

Jin-Hyong Ryu°, Hye-Jin Kim, Jae-Kook Lee, Hyong-Shik Kim
Dept. of Computer Engineering, Chungnam National University

요 약

라우터 옵션중의 일부는 특정 환경에서만 필요한 기능들을 구현한다. 이러한 옵션들은 일반적인 환경에서 라우터의 패킷처리 효율을 저하시킨다는 점에서 미리 정의된 조건에 의해 라우터가 옵션 등을 포함한 설정을 동적으로 변경함으로써 라우터의 처리성능을 향상시킬 수 있다. 본 논문에서는 CLICK Modular Router 위에서 동적으로 설정을 재구성하는 기법을 제안하고 설계, 구현한다.

1. 서 론

기존의 라우터 옵션 중에는 특정 환경에서만 필요한 옵션들이 존재한다. 예를 들면 필터링을 처리하는 모듈은 평소에는 유용하지만 라우터에 많은 부하가 걸린다면 패킷처리 효율을 저하시키게 된다. 또, 시스코 라우터의 WRED(Weighted Random Early Detection)[1]옵션은 평소에는 필요하지 않지만 라우터에 일정량 이상의 부하가 발생하면 유용한 옵션이다. 하지만 특정 조건을 벗어나는 경우에는 필요하지 않거나 라우터에 부하를 가중시킨다. 따라서 라우터가 조건을 판단하여 동적으로 설정을 변경할 수 있는 기법이 필요하다. 기존의 하드웨어 라우터의 경우엔 관리자가 직접 시스템의 변경을 위해 필요한 옵션을 on/off 해야 한다. 원할 경우 라우터의 관리자가 직접 필요한 옵션을 변경해야 한다. 본 논문에서는 소프트웨어 라우터인 CLICK Modular Router[2]를 이용하여 관리자의 개입 없이 동작 중인 라우터가 동적으로 설정을 변경하기 위한 기법을 제안하고 설계 및 구현, 시험한다.

본 논문의 구성은 다음과 같다. 2장은 라우터가 처리중인 패킷의 손실 없이 동적으로 설정을 변경하기 위한 기법을 소개하고 3장은 자기 재설정을 위한 라우터 시스템을 설계한다. 4장에서는 자기 재설정을 위한 라우터를 실제 구현하고 5장에서는 구현된 시스템의 동작성을 시험한다. 마지막으로 6장에서는 결론을 맺는다.

* 본 연구는 "대학 IT 연구센터 육성지원사업"의 지원을 받아 수행한 연구 결과임.

2. 라우터의 동적 재설정 기법

2.1. CLICK Modular Router

CLICK Modular Router(이하 CLICK)는 MIT LCS's Parallel and Distributed Operating System group과 Mazu Networks, ICSI Center, UCLA가 공동으로 개발한 모듈화 된 소프트웨어 라우터로, 유연하고 구성이 용이하며 확장성이 뛰어나다[3].

CLICK은 엘리먼트(element)라고 불리는 간단한 패킷처리 모듈들의 조합으로 구성되는데 각각의 엘리먼트는 패킷분류나 큐잉, 스케줄링과 같은 단순한 기능을 한다. 엘리먼트 사이의 연결은 'push connection' 또는 'pull connection'을 통하여 이루어지며, 이를 따라 패킷이 흐르게 된다. 사용자는 목적에 부합하는 엘리먼트들을 선택하고 조합함으로써 다양한 종류의 라우터를 쉽게 구성할 수 있다. 또한, CLICK에서는 사용자가 직접 새로운 엘리먼트를 생성하여 추가할 수 있으며, 좀 더 구체적인 기능을 할 수 있도록 둘 이상의 엘리먼트를 조합하여 하나의 복합 엘리먼트(compound element)를 만들 수 있다. CLICK은 이와 같은 패킷처리를 위한 엘리먼트와 함께 라우터에 필요한 여러 가지 정보를 설정하는 엘리먼트를 제공한다. 사용자는 이 엘리먼트를 사용하여 네트워크 정보나 스케줄 정보 등을 직접 지정한다.

CLICK의 동작은 정의된 CLICK 언어로 사용자가 작성한 설정파일을 라우터에 설치함으로써 가능하고 이러한

언어는 쉽고 간단하게 정의되어 있다. 또, CLICK은 처리 중인 패킷 정보를 잃지 않으면서 설치된 설정파일을 관리자가 동적으로 변경할 수 있는 기능을 제공한다. 다음은 이러한 기법 두 가지를 설명한 것이다.

2.2. 핫 스와핑(Hot swapping)

CLICK에서 동적으로 새로운 설정파일을 재설정하면 기존에 저장된 패킷 정보를 잃어버리고, 설정파일에 에러가 존재한다면 기존에 동작 중인 라우터는 동작하지 않는다. 그러나 핫 스와핑은 설정파일을 변경하면서 에러가 발생할 경우 기존에 라우터 설정에 영향을 미치지 않으며, 그렇지 않은 경우 처리중인 패킷을 잃어버리지 않고 새로운 설정파일을 라우터에 적용시킬 수 있는 기능을 제공한다.

2.3. 핸들러(Handlers)

각각의 엘리먼트에 미리 정의된 핸들러가 리눅스의 /proc파일 시스템아래 CLICK 디렉토리에 존재한다. 이러한 핸들러는 각각 읽기, 쓰기, 읽기/쓰기 속성을 가지며 사용자가 CLICK의 상태를 확인하거나 설정을 변경하는 매개가 된다. 핸들러는 엘리먼트 내에 존재하는 엘리먼트 핸들러와 엘리먼트 외부에 통합적으로 존재하는 전역 핸들러로 나눌 수 있다. 엘리먼트는 기본적으로 7개의 핸들러를 가지고 있으며 표 1은 이중 5개의 중요 기본 엘리먼트 핸들러에 대한 종류와 설명을 나타낸 것이다.

표 1 기본 엘리먼트 핸들러 종류

이 름	설 명	모드
class	엘리먼트가 속해 있는 클래스	읽기
name	엘리먼트 이름	읽기
config	엘리먼트의 설정 정보	읽기 또는 읽기/쓰기
ports	엘리먼트의 입력과 출력 포트의 개수와 연결종류	읽기
handlers	해당 엘리먼트 가지는 엘리먼트의 종류	읽기

표 1의 핸들러들을 제외한 다른 핸들러들은 엘리먼트의 특성에 따라 다양하다. 예를 들면 Queue 엘리먼트의 'length' 핸들러는 사용자가 Queue에 저장된 패킷의 수를 확인하는 기능을 가진 읽기 핸들러이다. 라우팅 테이블 엘리먼트의 'add_route','del_route' 핸들러는 사용자가 라우팅 테이블 정보를 삽입하거나 삭제할 수 있는 기능을 가진 쓰기 핸들러이다.

전역 핸들러의 예를 들면 'config' 핸들러는 현재 라우터의 설정을 사용자에게 제공하거나 사용자에게 설정파일을 받아서 재설정할 수 있는 기능을 제공하는 읽기/쓰기 핸들러이다. 중요 전역 핸들러의 종류와 설명은 표 2와 같다.

표 2 중요 전역 핸들러

이 름	설 명	모드
config	현재 라우터 설정을 사용자에게 제공하고 사용자가 라우터 설정을 변경할 수 있다.	읽기/쓰기
hotconfig	핫 스와핑으로 현재 라우터 설정을 변경할 수 있다.	쓰기
error	CLICK이 에러를 출력한다.	읽기
message	CLICK이 메시지를 출력한다.	읽기
priority	CLICK프로세스의 우선순위 변경	쓰기
list	현재 라우터가 사용 중인 엘리먼트의 목록	읽기
stop	현재 라우터의 패킷 처리를 중지시킨다.	쓰기
version	CLICK의 버전을 출력한다.	읽기

'hotconfig'라는 전역 핸들러는 핫 스와핑을 할 수 있는 전역 핸들러로써 이를 이용하면 핸들러를 이용한 핫 스와핑이 가능하기 때문에 핸들러는 핫 스와핑을 대체할 수 있고, 본 논문에서의 핫 스와핑은 핸들러를 이용하는 방법으로 한정한다.

핸들러는 사용자뿐만 아니라 동작 중인 엘리먼트 간에 핸들러를 이용하여 정보를 주고받는 기능도 한다. 이러한 방법을 이용하면 엘리먼트 간에 서로 유기적인 동작이 가능하다.

3. 자기 재설정을 위한 시스템 설계

CLICK의 경우에 동적으로 여러 설정파일을 유지할 수 없기 때문에 핫 스와핑과 핸들러의 방법으로 라우터를 재설정을 하기 위해서는 외부 프로그램의 도움을 받아야 한다. 외부 프로그램은 핸들러를 통해서 주기적으로 특정 조건을 확인하고 조건이 만족되면 핸들러를 활용하여 라우터를 재설정한다. 본 논문에서는 데몬 프로세스가

이러한 외부 프로그램의 역할을 맡는다. 그림 1은 라우터의 자기 재설정을 위한 시스템 구조를 간단히 도식화한 것이다.

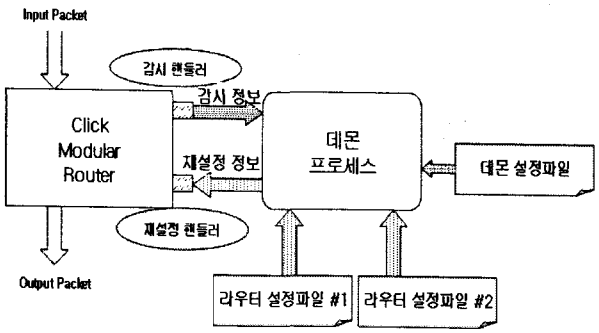

그림 1 자기 재설정을 위한 라우터 시스템 구조

그림 1에서 데몬 프로세스는 핸들러를 통해 CLICK과 통신하는 역할을 하며 CLICK 프로세스와 동시에 시작된다. 데몬 프로세스는 최초에 데몬 설정파일을 읽어 자신을 설정한다. 이 설정파일은 다음과 같은 설정정보로 구성되어 있다.

< 실행주기, 감시 핸들러 경로, 비교연산자, 목표값, 재설정 핸들러 경로, 설정파일 경로 >

위와 같은 정보는 하나의 조건이 하나의 튜플(Tuple)로 구성된다. 조건은 여러 개가 존재할 수 있으며 동시에 만족하는 조건이 둘 이상일 경우 설정파일에 먼저 존재하는 조건이 우선한다. 감시 핸들러 경로는 데몬 프로세스가 설정된 실행 주기로 감시해야 하는 핸들러의 경로를 의미하고 비교연산자는 감시 핸들러를 읽는 값과 비교수치를 비교하기 위한 연산자이다. 만약 이 비교연산의 결과가 참이 되면 재설정 핸들러에 설정파일을 씀으로써 라우터를 재설정한다. 이렇게 재설정 핸들러를 조건마다 다르게 설정할 수 있게 한 이유는 상황에 맞게 전역 핸들러나 엘리먼트 핸들러를 선택적으로 사용할 수 있게 하기 위함이다.

4. 자기 재설정을 위한 데몬 프로세스 구현

자기 재설정을 위한 데몬 프로세스를 구현하는 환경은 표 3과 같다.

표 3 구현 환경

CPU	펜티엄4 1.4GHz
Memory	256Mbyte
OS	Linux(Ubuntu)
Programming Language	C 언어
CLICK Version	Version 1.5

우선 설정파일을 읽기 위한 자료구조를 그림 2와 같이 정의한다.

```
typedef struct _condition_t
{
  int          nInterval; // 실행주기
  char         pReadHandler[MAX_NAME]; //감시 핸들러
  int          nConditioner; //비교연산자
  long         lCondition; //목표값
  char         pWriteHandler[MAX_NAME]; //재설정 핸들러
  char         pConfigureFile[MAX_NAME]; //설정파일
} condition_t;
```

그림 2 설정파일을 위한 자료구조

그림 2의 자료구조는 앞에서 제시한 설정정보와 같은 순서로 배열되어 있다. 비교연산자는 숫자로 변환되어 저장되며 0부터 차례로 <, <=, >, >= 순서가 된다.

또한, 다음과 같이 구현에 필요한 함수를 정의한다.

```
int readConfigure( condition_t* conf_t );
int compare( condition_t conf );
int reConfigure( condition_t conf );
```

첫번째 함수는 데몬 설정파일을 읽어와 그림 2의 자료구조에 할당하는 함수이고 두번째 함수는 하나의 조건을 입력받아 조건이 만족하면 참을 반환하는 비교함수이다. 세번째 함수는 만족하는 조건을 입력받아 실제로 재설정을 하는 함수이다. 위에서 표현하지 않은 main함수에서는 반복적으로 현재의 라우터 설정 조건을 제외한 조건들을 비교하고 만족하면 라우터를 재설정한다. 이를 소스코드로 나타내면 그림 3과 같다.

```
condition_t condition[CONF_MAX];//설정을 위한 자료구조
while(1)
{
  int nResult = 0; //함수의 반환값을 저장하기 위한 변수
  int nIndex; //인덱스 변수
  for( nIndex = 0; nIndex < nConfigCount ; nIndex++ )
  {
    // 현재의 설정은 비교하지 않는다.
    if( gCurrentConfigure == nIndex )
        continue;
    // 조건을 비교한다.
    if( compare( contidion[nIndex] ) )
    {
        //조건이 같다면 설정 변경
        nResult = reConfigure( condition[nIndex] );
        if( nResult < 0 )
           '에러를 출력한다.'
        else if( nResult == 0 )
        {
            // 현재의 설정 인덱스를 갱신한다.
            gCurrentConfigure = nIndex;
            성공을 출력한다.
        }
    }
  }
  sleep(n); //설정된 주기 동안 기다린다.
}
```

그림 3 재설정 기능에 대한 소스코드

5. 시험

자기 재설정 게이트웨이를 구현하기 위하여 간단한 프로토타입을 설계하였다. 먼저 시험을 위해 그림 4와 같이 두 개의 라우터 설정파일과 한 개의 데몬 설정파일을 만들었다.

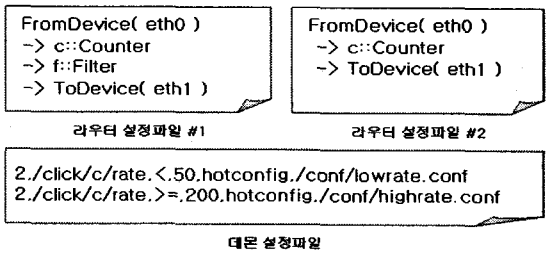

그림 4 시험을 위한 설정파일 내용

'라우터 설정파일 #1'은 Network1에서 패킷을 받아 통계를 내고 미리 정의된 규칙에 의해 패킷을 필터링 후 Network2에 전달한다. 그리고 '라우터 설정파일 #2'는 Network1에서 패킷을 받아 통계를 내고 필터링 없이 Network2에 패킷을 전달한다. '데몬 설정파일'의 내용은 2초를 주기로 패킷 처리율(packet/s)을 계산하여 50 미만이면 lowrate.conf 설정파일 내용으로 핫 스와핑 하고 200 이상이면 highrate.conf 설정파일 내용으로 핫 스와핑하는 것이다. 시험을 위해 그림 5와 같이 시스템을 구성하고, 라우터에 자기 재설정 게이트웨이 설정을 적재한 후 Network1에서 Network2로 트래픽을 발생시켜 테스트 한다.

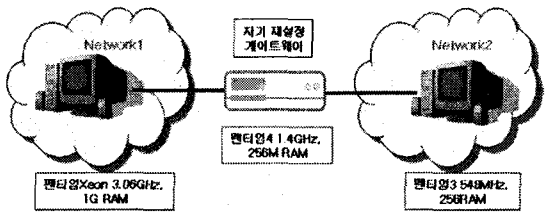

그림 5 시험환경

시스템을 동작시키고 핸들러를 통해 라우터 설정을 살펴보면 패킷 처리율은 약 45(packet/s)였다. 패킷 처리율은 50 미만이므로 라우터 설정파일1의 설정이 라우터에 적재됐다. 트래픽을 발생시켜 라우터에 부하를 주고 다시 앞의 실험을 반복한다. 이때의 패킷 처리율은 약 3927(packet/s)이므로 라우터 설정파일2의 설정이 라우터에 적재됐다. 실험결과는 패킷 처리율에 따른 라우터의 자기 재설정이 가능하다는 것을 보여준다. 비록 실험의 조건이 패킷 처리율이었지만 이러한 조건과 설정은 상황에 따라 많은 응용이 가능하다.

6. 결론

본 논문은 라우터가 처한 다양한 상황에 관리자가 즉각 대처하기는 불가능하다고 판단하여 미리 정의된 다양한 상황에서 처리중인 패킷의 손실 없이 라우터 스스로 설정을 변경하고 상황을 대처하는 시스템을 설계하고 구현, 시험하였다.

현재 시스템은 패킷 처리율에 한정하여 라우터의 설정을 변경하지만 앞으로 다양한 응용을 통해 시스템을 발전시킬 계획이다.

참고문헌

[1] *Cisco Corporation. Distributed WRED. Technical report.* http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/wred.htm, October 1999

[2] Eddie Kohler. "The Click modular router", MIT, November 2000.

[3] 김혜진, 이재국, 김형식 "보안 게이트웨이용 패킷처리 구조," 제32회 한국정보과학회 추계발표회 논문집, 32권, 2호, 2005년 11월.

[4] CLICK Modular Router (http://pdos.csail.mit.edu/CLICK/)