하드웨어 추상화 계층에 기반한 네트워크 스위치 소프트웨어의 설계 및 구현

김지현O 김준우 강경태 이원석 신현식
서울대학교 컴퓨터시스템연구실
{jhkimO, kesker, nicola, alnova2}@cslab.snu.ac.kr, shinhs@snu.ac.kr

Design and Implementation of Network Switching Software based on Hardware Abstraction Layer

Jihyun KimO Kyungtae Kang Junu Kim Wonseok Lee Heonsik Shin
Computer Systems Laboratory, Seoul National University

요 약

내장형 시스템의 일종인 네트워크 스위치는 소프트웨어의 하드웨어 의존성 때문에 그 개발에 어려움이 있다. 첫째, 하드웨어와 소프트웨어의 개발이 순차적으로 밖에 이루어지지 못하므로 개발 시간이 현저히 지연되며, 둘째, 하드웨어에 따라 소프트웨어가 이식되어야 하므로 개발 노력이 낭비된다. 특히 네트워크 스위치의 소프트웨어는 하드웨어에 의존적일 뿐 아니라, 소프트웨어 모듈 간에도 의존적인 요소가 존재하므로 개별적으로 개발된 소프트웨어들의 통합에 어려움이 있다. 본 논문에서는 네트워크 스위치 개발 시 앞서 언급한 내장형 시스템 개발 문제점을 해결하는 동시에, 소프트웨어 간의 의존성 역시 해결할 수 있는 소프트웨어 구조로써 가상의 스위치 계층을 설계하고 구현하였다. 또한 사례연구로써 OSI 2계층에서 동작하는 리눅스 기반의 스위치를 위한 각종 프로토콜을 본 논문에서 제안하는 가상의 스위치 계층을 기반으로 하여 개발하였으며, 개발 경험을 통하여 가상의 스위치 계층이 하드웨어와 소프트웨어 개발을 독립적으로 수행할 수 있도록 함으로써 스위치 개발 시간을 단축시키며, 또한 소프트웨어 통합 시 그 복잡도를 낮추고 소프트웨어의 신뢰성을 높이는 것을 검증하였다.

1. 서 론

최근 네트워크 기술의 발전에 힘입어 고속의 데이터 스위칭이 가능한 네트워크 스위치 및 라우터가 속속히 출현하고 있다. 이러한 네트워크 스위치 하드웨어의 발전에 비하여 스위치에 필요한 소프트웨어 프로토콜은 그 발전의 속도가 느리며 기존 시스템과의 호환성 문제로 프로토콜 자체의 혁신적이거나 잦은 변화는 힘들다.

그러나 프로토콜 메커니즘 자체에는 변화가 없음에도 불구하고, 네트워크 스위치 소프트웨어는 하드웨어 의존적인 성향 때문에 새로 개발되는 하드웨어에 따라 각 소프트웨어를 수정해 주어야 하는 불편함이 따른다.

또한 소프트웨어의 하드웨어 의존적인 성향 때문에 새로운 네트워크 스위치를 개발할 때 하드웨어와 소프트웨어를 병렬로 개발하는 데에 어려움이 있으므로 그 개발 시간이 현저히 지연되고 하드웨어와의 통합 과정에서 플랫폼에 맞도록 소프트웨어를 이식하는 작업으로 그 개발 효율 또한 떨어진다. [1]

이러한 상황을 고려하여 네트워크 스위치를 위한 소프트웨어를 효율적으로 개발하기 위해 요구되는 두 가지 사항을 인지하였다. 첫째, 네트워크 소프트웨어는 하드웨어와 독립적으로 개발될 수 있어야 한다. 하드웨어와 독립적으로 소프트웨어를 개발할 수 있다면, 하드웨어와 소프트웨어는 병렬적으로 개발될 수 있으므로 개발 기간을 단축시키며, 소프트웨어 디버깅할 때에 하드웨어의 결함에 대해 자유롭다. 둘째, 네트워크 소프트웨어는 재사용 가능하도록 구현되어야 한다. 앞서 언급한 것처럼 네트워크 소프트웨어는 하드웨어 의존적인 부분 때문에 그 알고리즘에 변화가 없음에도 불구하고 재사용을 위해서는 하드웨어 의존적인 부분에 대해 이식의 과정이 필요 했다. 그러나 하드웨어에 의존적인 부분을 해결하여 완전하게 재사용 가능하도록 개발한다면 개발자의 노력을 현저히 절약하게 한다.

본 논문에서는 이러한 두 가지 요구사항을 충족시키는 네트워크 스위치를 위한 소프트웨어 구조인 가상의 스위치 계층을 설계하고 구현하였다.

2. 관련연구

2.1 네트워크 스위치의 구조

네트워크 스위치는 일반적으로 그림1과 같은 구조를 갖는다.

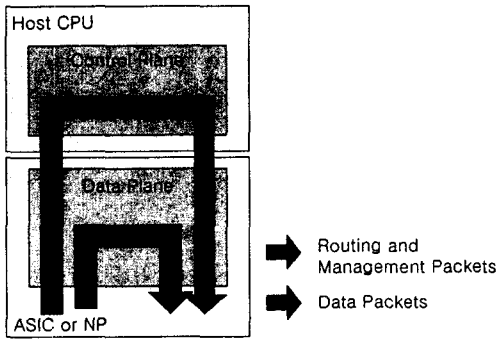

그림 1. 일반적인 스위치의 구조

그림1의 Control Plane은 범용 CPU 상에 구현되며, 제어 패킷, 라우팅 테이블 업데이트, 포워딩 테이블 업데이트, Interface 관리 등을 처리하는 네트워크 프로토콜이 실행된다. 그림1의 Data Plane은 NP(Network Processor, 네트워크 처리기)기반으로 구현되며, 일반 데이터 패킷을 스위칭한다. 이렇게 Control Plane과 Data Plane을 분리하는 구조는 기존 소프트웨어 기반 스위치의 성능 문제를 데이터 패킷의 스위칭을하드웨어로 처리함으로써 극복할 수 있도록 한다. 또 하나의 장점은 데이터 스위칭을 담당하는 하드웨어와 프로토콜 구현을 분리시킴으로써 각각이 서로의 결함에 대해 독립적으로 구현되므로 테스트 및 디버깅의 복잡도가 낮아진다.[2]

2.2 Control Plane 구현의 어려움

2.2.1 하드웨어 의존성

OSI 2계층에서 동작하는 프로토콜들은 Data Plane의 포워딩 동작에 영향을 주도록 각 물리적 link를 관리하므로, 각 link의 특성-속도, MTU (Maximum Transfer Unit), Link Type (Ethernet or ATM…)-을 기반으로 동작한다. 또한 스위치의 Data plane의 설계-포트의 수, Aggregator의 수 등-에 따라서도 Control Plane의 구현이 달라질 수 밖에 없으며, Data Plane의 하드웨어 구성요소와 그 디바이스 드라이버에 의존적으로 구현되어야 한다.

그러므로 Control Plane의 구현 시에 어떠한 data plane 구현에도 손쉽게 이식될 수 있도록 구현하는 것이 control plane구현의 중요한 고려사항이다.

2.2.2 소프트웨어 통합의 어려움

앞 절에서 언급한 것과 같이 OSI 2계층에서 동작하는 프로토콜은 주로 물리적 link를 논리적으로 관리하는 동작을 하며, Data Plane의 정보를 요구하기도 한다. 이 때 공통의 Data Plane 정보를 여러 프로토콜에서 사용하게 되므로, 각 프로토콜의 동작이 다른 프로토콜의 동작에 영향을 미치게 된다. 그러므로 각 소프트웨어의 개발 시에는 각 프로토콜의 올바른 동작을 보장하는 것은 물론, 다른 프로토콜의 동작에 결함을 일으키지 않도록 하여야 한다.

그러나 각 프로토콜의 개발자는 다른 모든 프로토콜의 동작을 완벽히 이해하고 상호관계를 고려하여 구현한다는 것이 쉬운 일은 아니며, 이 때 각 프로토콜의 구현의 복잡도는 증가하고 소프트웨어의 신뢰도는 떨어진다.

그러므로 각 프로토콜이 독립적으로 구현되며 각 소프트웨어의 올바른 동작을 보장하는 동시에, 독립적으로 개발된 프로토콜들의 통합을 용이하고 안전하게 할 수 있는 방법론이 필요하다.[3]

3. 가상의 네트워크 스위치

앞서 살펴본 Control Plane 구현의 어려움을 극복하고자 본 논문에서는 Control Plane 소프트웨어 구조로써 가상의 스위치 계층을 설계하였다. 가상의 스위치 계층은 Control Plane에 구현되는 프로토콜들을 Data Plane의 구현에 완전하게 독립적이 되도록 분리하는 동시에, Control Plane에 구현되는 각 프로토콜의 통합을 쉽고 신뢰성 있게 할 수 있는 가상의 Data Plane 환경을 제공한다.

3.1 가상의 네트워크 스위치 개요

가상의 네트워크 스위치 계층은 그림2의 Data Plane에 해당하는 스위치 하드웨어를 추상화하고, Control Plane에 의해 유지되는 논리적인 스위치의 상태를 통합하여 관리한다. 가상의 스위치 계층이 삽입된 스위치의 구조는 그림2와 같다.

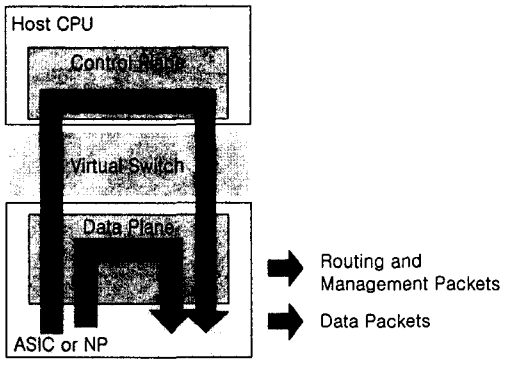

그림 2. 가상의 스위치 계층이 삽입된 스위치의 구조

3.2 가상의 네트워크 스위치의 설계

가상의 스위치 계층은 Data Plane을 추상화하여 가상의 Data Plane 역할을 하는 동시에, Control Plane에 의해서 제어되는 Data Plane의 논리적인 구성을 Control Plane의 모든 소프트웨어에 공통으로 제공하여야 한다. 본 논문에서 설계한 가상의 스위치가 제공하는 가상의 스위치는 다음 그림3과 같다.

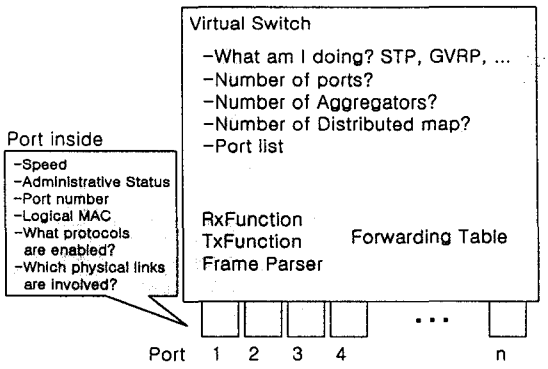

그림 3. 가상의 스위치 계층

가상의 스위치 구성을 세부적으로 설명하면 다음과 같다.

우선, 스위치 일반 정보는 그림3의 상단에서 보는 바와 같이 스위치의 Control Plane에서 동작하고 있는 프로토콜, 포트의 수, Aggregator의 수, Distributed map의 수,포트 정보의 연결 리스트 등을 유지한다.

Control Plane의 프로토콜은 Data Plane에 물리적으로 존재하는 링크를 논리적으로 Enable/Disable하기도 하고 물리적으로 여러 개의 링크를 묶어서 논리적으로는 하나의 포트로 사용하기도 한다. 그러므로 Control Plane에서 사용하게 되는 링크는 물리적인 링크가 아니라 논리적으로 구성되어 있는 포트이며, 가상의 스위치는 이 포트에 대한 정보를 유지하여 Control Plane에 제공한다. 포트가 유지하는 정보는 그림3의 좌측 Port inside와 같다.

또한, Data Plane의 기본 동작인 Rx, Tx, Frame Parser (Classifier)에 대한 정의를 포함하며, Control Plane에 의해 제어되며 각 데이터의 egress 포트를 결정하여 주는 Forwarding Table을 유지한다.

4. 사례연구: OSI 2계층 네트워크 스위치의 개발

4.1 스위치의 하드웨어 구성: Data Plane

그림3은 전체 시스템 구성이며, 그림4는 그림3 하단에 보이는 Network board 부분을 상세히 한 것이다.

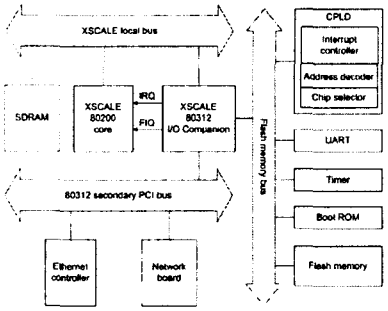

그림 4. 스위치 하드웨어의 구성

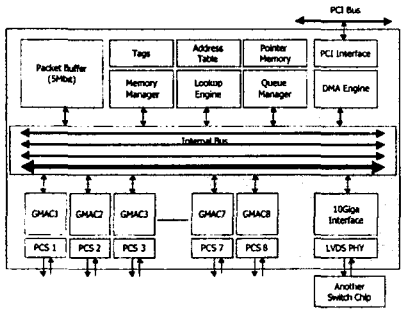

그림 5. 네트워크 보드 구성

4.2 스위치의 소프트웨어 구성: Control Plane

4.2.1 운영체제

Xscale CPU에서 동작하도록 이식된 리눅스 2.4.7-10 버전의 커널을 사용하며, 커널의 이미지를 생성하여 Flash memory에 복사하고 부팅한다. 파일시스템은 별도의 개발 호스트에서 NFS(Network File System)을 통하여 다운로드 받는다.

4.2.2 하드웨어 추상화 계층의 구현

하드웨어 추상화 계층을 정의하는 주요 데이터 형을 기술하면 다음과 같다.

```
#include <linux/skbuff.h>
struct VirtualSwitch{
        unsigned char cMac[ETH_ALEN];        // 스위치 MAC주소
        struct net_device *dev;              // 리눅스 net_device 구조체
        int iPortNum;                        // 포트 수
        int iAggNum;                         // Aggregator 수
        int I DistmapNum;                    // Distributed map 수
        struct PortState *portList;          // 포트정보 연결 리스트
        int iSTPEnabled;                     // STP의 구동 여부
        int iLACPEnabled;                    // LACP의 구동 여부
        int iGVRPEnabled;                    // GVRP의 구동여부
        int iSNMPEnabled;                    // SNMP의 구동여부
};
struct PortState{
        struct PortInfo   default;           // 포트의 디폴트 정보
        struct PortInfo   info;              // 포트의 현재 정보
        struct PortUp     up;                // 포트의 물리적/논리적 연결
        int iSTPEnabled;                     // STP구동 여부
        int iLACPEnabled;                    // LACP구동 여부
};
struct PortInfo{
        int iPortNumber;                     // 포트 번호
        unsigned char cMac[ETH_ALEN];        // 포트의 MAC 주소
};
struct PortUP{
        int iUp;                             // 링크의 물리적 연결 여부
        int iEnabled;                        // 포트의 논리적 연결 여부
};
```

4.2.3 하드웨어 추상화 계층에 기반한 프로토콜의 구현

본 논문에서 제안하는 가상의 스위치 계층을 기반으로 하여 OSI 2계층에서 동작하는 프로토콜인 STP, LACP, GVRP를 구현하고 신뢰할만한 Test Suite를 통하여 프로토콜 동작의 무결성을 검증하였다.[4]

5. 결론 및 향후 연구

본 논문에서는 신뢰성 있는 네트워크 스위치 소프트웨어를 보다 효율적으로 개발할 수 있도록 하는 소프트웨어 구조로써 가상의 스위치 계층을 설계하고 구현하였다. 또한, 사례연구로써 OSI 2계층에서 동작하는 네트워크 스위치 소프트웨어를 디바이스 드라이버 추상화 계층을 기반으로 개발하였다.

가상의 스위치 계층은 하드웨어를 추상화하여 가상의 Data Plane을 Control Plane에 제공하며, 이를 통해서 우리가 얻을 수 있는 이득은 다음과 같이 정리할 수 있다.

첫째, 하드웨어와 소프트웨어를 독립적으로 개발할 수 있다. 이로써 소프트웨어의 개발이 하드웨어와 병렬적으로 이루어질 수 있으며 전체 개발 시간을 현저히 단축 시킨다. 그리고 개발된 소프트웨어의 디버깅 및 테스트 시에 하드웨어의 결함에 대해 자유로울 수 있으므로 개발의 복잡도가 낮아진다. 또한 하드웨어와 독립적으로 개발된 소프트웨어 모듈들은, 하드웨어에 맞게 이식된 가상의 스위치 계층을 기반으로, 완벽하게 재사용 가능하다.

둘째, 프로토콜 모듈간의 통합이 용이하다. 네트워크 스위치를 위한 프로토콜 소프트웨어는 스위치의 하드웨어 자원, 즉 Data Plane을 논리적으로 재구성하여 사용한다. 이때 각 프로토콜 모듈에서 유지하는 스위치의 논리적인 정보를 가상의 스위치 계층에서 통합하여 관리함으로써 각 프로토콜 모듈간의 충돌을 방지하며 각 소프트웨어 모듈의 통합을 용이하게 하고, 각 프로토콜 구현의 복잡도를 낮춘다. 또한 각 모듈에서는 디바이스 드라이버 추상화 계층에서 유지하는 공통된 정보를 접근하여 이용함으로써 보다 신뢰도 높은 소프트웨어 개발을 할 수 있다.

셋째, 가상의 스위치 계층을 기반으로 개발된 소프트웨어는 재사용 가능하다. 하드웨어 환경이 달라지면 이에 맞게 디바이스 드라이버 추상화 계층을 이식하여, 그 위에서 실행되는 모든 소프트웨어 모듈은 완벽히 재사용 가능하다.

본 논문이 기여하는 바는 네트워크 스위치 소프트웨어 개발 시 하드웨어와 독립적인 소프트웨어 개발의 방법론으로써 가상의 스위치 계층을 설계하였으며, Data Plane을 추상화하고 Control Plane의 소프트웨어 통합을 용이하게 하는 가상의 스위치 계층이 가져야 할 명세를 정의한 것이라 할 수 있다.

참고문헌

[1] Sungjoo Yoo, Introduction to Hardware Abstraction Layers for SoC, Proceedings of the DATE IEEE, 2003

[2] FutureSoft White Paper, Challenges in Building Network Processor Based Solutions, http://www.futsoft.com, 2003

[3] T.SRIDHAR, Control and Data Plane Issues in Communications Software, http://www.futsoft.com, 2003

[4] InterOperability Laboratory, http://www.iol.unh.edu/

[5] John Renwick, Interface Management API Implementation Agreement, Network Processing Forum, 2002