2001 년 한국정보처리학회 춘계 학술발표논문집 제 8 권 제 1 호

패킷 스케쥴링 지원을 위한 Windows 기반의 라우터 설계 및 구현

유환석, 김기일, 김상하
충남대학교 컴퓨터과학과
e-mail : shkim@cclab.cnu.ac.kr

Design and Implementation of Router Based on Windows for Supporting Packet Scheduling

Hwan-Souk Yoo , Ki-Il Kim, Sang-Ha Kim
Dept. of Computer Science, Chung-Nam National University

요 약

최근 윈도우에서는 다양한 종류의 인터넷 소프트웨어가 개발되고 있다. 윈도우를 지원하기 위해 다양한 네트웍 API가 제공되고 있다. 인터넷 소프트웨어 중 보안, 인터넷 공유, 네트워크 교육용 소프트웨어 등은 기본적으로 라우팅에 관련된 기능을 포함하고 있어야만 한다. 그러나 일반적으로 사용자 계층에서 제공하고 있는 Winsock, NPP 등의 네트웍 API는 데이터 링크 계층에 직접적인 접근이 불가능하여 라우팅 기능 구현에 적당하지 않다. 따라서, 본 논문에서는 NDIS library를 이용하여 일반 개발자가 손쉽게 데이터 링크 계층의 프레임에 접근할 수 있는 Packet driver API를 설계하고 이를 기반으로 패킷 스케쥴링을 지원 할 수 있는 Windows 기반의 라우터를 설계 및 구현하였다.

1. 서론

인터넷이 일반화되면서 사용자들은 인터넷에 접근하기 쉬운 환경과 더욱 다양한 인터넷 소프트웨어를 요구하고 있다. 최근에는 Windows 환경의 사용자는 네트웍 보안 기능과 인터넷 회선 공유 기능등 라우팅 기능이 지원되는 소프트웨어를 사용하고 있으며 사용자의 요구사항이 더욱 다양해지고 있다. 사용자는 인터넷에 접근하기 위한 환경으로 Windows를 더 선호하고 있으며 개발자 역시 사용이 편리하고 다양한 개발 도구가 지원되는 Windows를 인터넷 소프트웨어 개발 환경으로 더 선호하고 있다.

Windows기반의 네트웍 API로 가장 많이 사용되는 것이 Winsock API(Windows Sockets)이다. Winsock은 사용이 쉽고 다양한 기능을 제공하지만 데이터 링크 계층 접근에 제한이 있다. 이 밖에 다른 API도 데이터 링크 접근에 불가능하거나 제한이 있다.

본 논문에서는 위의 제한을 극복하기 위해 NDIS (Network Driver Interface Specification) library를 이용하여 응용프로그램 레벨에서 데이터 링크 계층에 쉽게 접근할 수 있는 API를 설계 및 구현 하였다. 그리고 이를 기반으로 패킷 스케쥴링 기능이 가능한 라우터를 객체지향 개념을 기반으로 모델링,구현하였다.

이 논문의 다음과 같이 구성되어 있다. 2장에서는 Windows Network API의 구조 및 특징에 대해 살펴본다. 3장에서는 NDIS의 구조와 Windows Base Service를 이용한 Packet driver API에 대해서 알아본다. 4장에서는 스케쥴링 라우터 구현에 사용된 FIFO 큐와 CBQ에 대해 알아본다. 5장에서는 Packet driver를 이용한 패킷 스케쥴링 기능을 가진 라우터의 구현 모델에 대해 알아 본다. 5장에서는 결론과 향후 연구 방향에 대하여 살펴본다

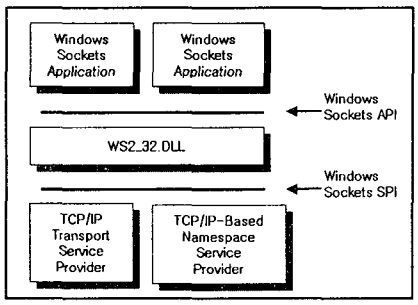
2. Windows 네트웍 API의 구조 및 특징

Windows는 인터넷 소프트웨어 개발을 위하여 몇 가지 API를 제공하고 있다. 이 장에서는 Windows 네트

윅 API에 대한 구조와 특징에 대해 알아보고 각 API 와 프로토콜 사이의 관계에 대하여 알아본다.

2.1 Winsock (Windows Sockets)

Windows계열의 대표적인 API로 인터넷 소프트웨어 개발에 사용되고 있다. Winsock은 프로토콜을 포함하지 않으며, 단지 전송 프로토콜 계층에 접근하기 위한 인터페이스로 사용된다.



[그림 1] Winsock 구조

상위 계층으로는 API를 통하여 사용자 프로그램에 인터페이스를 제공하는 동시에, 하위 계층으로는 SPI(Service Provider Interface)를 통하여 전송 프로토콜 계층에 인터페이스를 제공한다.

Winsock은 전송 계층과 인터페이스를 이루고 있으므로 기본적으로 전송 계층까지 접근할 수 있다. 이를 확장하기 위한 방법으로 소켓(socket)의 프로토콜 종류를 SOCK_RAW로 선택하는 방법이 있다. 이 방법은 ICMP와 IGMP 접근을 위해 사용할 수 있다.

Winsock의 확장이 더 필요한 경우는 setsockopt 함수와 *IP_HDRINCL* 옵션을 사용한다. 이 방법은 소켓에게 전송되는 데이터의 앞부분에 IP헤더가 이미 포함된 것을 알려주고, 네트웍 계층에서는 사용자가 작성한 IP헤더를 사용하여 IP 패킷을 만들게 된다. 하지만 Winsock에서는 전송에 대한 확장만 가능하고 수신에 대해서는 불가능하다.

2.2 기타 네트웍 API

Winsock을 제외한 API로는 NPP(Network Packet Providers), IP Helper(Internet Protocol Helper), DNS API(Domain Name System API), DHCP API(Dynamic Host Configuration Protocol API), MADCAP(Multicast Address Dynamic Client Allocation Protocol) 등이 있다.

각 API별 지원 가능 계층과 프로토콜을 보면 **[표 1]** 과 같다. Windows에서 사용할 수 있는 API는 위의 요구 사항을 부분적으로만 만족하며 가장 중요한 기능인 데이터 링크 계층에서 사용자의 직접 전송을 지원하지 않는다.

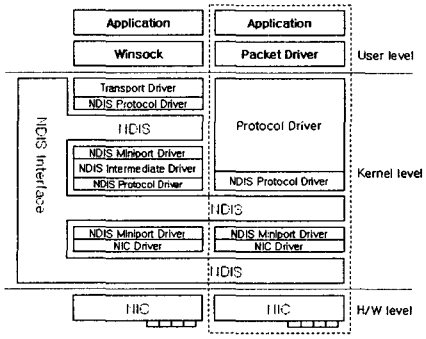
API	프로토콜	비고
Winsock (RAW Mode)	TCP,UDP Multicasting Broadcasting (ICMP) (IGMP)	Asynchronous & Overlapped 모드 지원
NPP	데이터링크 계층	Promiscuous Read
IP Helper	ARP	RARP, Proxy ARP, Gratuitous ARP 불가능

[표 1] API 와 프로토콜과의 관계

데이터 링크 계층 접근은 라우터의 포워딩(forwarding)과 같은 기능을 구현하기 위해서는 반드시 필요하다. 이 문제를 해결하기 위하여 NDIS를 이용한 Packet driver를 사용한다.

3. NDIS와 Base Service를 이용한 Packet Driver

NDIS는 Windows 커널 내에 프로토콜 스택을 구성하기 위한 명세서이다. [그림 2]는 NDIS를 사용하여 프로토콜 스택을 구성한 것이다. 각 NDIS Driver는 DriverEntry함수에서 다른 Driver와 인터페이스로 사용할 엔트리 포인트를 제공하고 있다.



[그림 2] NDIS 를 이용한 프로토콜

NDIS는 다음과 같은 기능을 제공한다.

- NDIS Miniport Driver

하드웨어와 상위 계층간의 통신을 담당한다. NIC을 통한 데이터 전송,수신 기능이 있으며, 상위 계층의 드라이버를 위한 인터페이스 가지고 있다. 대부분 NIC 제조 업체에서 제공한다.

- NDIS Intermediate Driver

새로운 계층을 추가하여 새로운 프로토콜 추가, 패킷 필터링, 다른 종류의 네트워크 미디어간의 패킷 교환 등의 기능을 수행할 수 있다.

- NDIS Protocol Driver

상위 계층과 하위 계층의 Driver사이에서 인터페이스로 사용된다. [그림 2]는 응용 계층에서 데이터 링크 계층 접근을 위한 구조를 나타내고 있다. 사용자가 정의한 Protocol Driver가 NDIS Protocol Driver Interface를 사용하여 NIC의 Miniport Driver에 접근할 수 있게 되고 그에 따라 데이터 링크 계층에 사용자가 직접 접근할 수 있게 된다.

Packet Driver 구현에 중요한 NDIS Protocol Driver에 대하여 자세히 살펴본다.

3.1 NDIS Protocol Driver 등록

Windows 시스템이 부팅과정 중 NT Service가 시작되어 해당 Driver를 Loading 과정 중, DriverEntry 함수가 호출되어 사용자 정의 Protocol Driver의 등록이 시작된다. NDIS_PROTOCOL_CHARACTERISTICS 구조체와 NdisRegisterProtocol 함수를 사용하여 Protocol Driver의 이름, 버전을 등록하고 각 핸들에 대한 함수를 지정하게 된다. 이는 하위 계층의 수행결과를 상위 계층으로 전달하기 위한 방법으로 하위 계층의 수행결과에 해당하는 핸들이 가르치고 있는 함수를 호출함으로써 이루어진다. 특별히 핸들의 이름에 xxxCompleteHandler인 핸들은 비동기 모드 수행을 위한 함수로 비동기 모드를 사용하려면 반드시 구현하여야 한다.

```
SendCompleteHandler   =   PacketSendComplete
ReceivePacketHandler  =   PacketReceivePacket
```

[예제 1] NDIS Protocol Driver 의 Entry 등록

[예제 1]과 같이 설정되어 있다면 Protocol Driver가 패킷을 보내기위해 NdisSendPackets 함수를 호출하고 정상적으로 수행되면 운용체제 시스템에 의해 Protocol Driver의 PacketSendComplete 함수가 수행 되어 실행된 결과를 알 수 있다.

3.2 Windows Base Service를 이용한 Packet Driver

응용 계층에서는 Kernel의 사용자 정의 Protocol Driver에 접근하기 위해 Base Service를 사용한다. 이는 유닉스(UNIX)의 시스템 콜(System call)에 해당한다.

UNIX 계열의 OS는 모든 장치를 파일로 취급하며 시스템 콜을 사용하여 이를 관리한다. Windows도 이와 유사한 방법을 사용한다. 사용자 정의 프로토콜에 등록 과정에 정의된 핸들함수와 Base Service 함수는 일대일 대응관계 가지며 있으며, 응용 계층에서 Base Service를 이용하여 커널 계층의 사용자 정의 프로토콜 드라이버에 접근 및 관리가 가능하다.

Windows에 사용 할 수 있는 모든 통신 장치에 SymbolicLink 이름이 할당되어 있다. Packet Driver는 Windows 시스템에 등록되어 있는 NIC(Network Interface Card)의 SymbolicLink 이름을 레지스트리(Registry)에서 찾고 이를 이용하여 응용 계층에서 사용자 정의 프로토콜에 정의 되어 있는 기능을 통해 NIC을 제어 할 수 있다.

4. 패킷 스케쥴링 메커니즘

이번 장에서는 본 논문에서 구현하고자 하는 FIFO Queuing 과 CBQ 스케쥴에 대하여 간단히 살펴본다.

스케쥴링의 기본 기능은 링크상의 전송을 기다리는 패킷들 사이의 전송 순서를 조정하는 것이다.

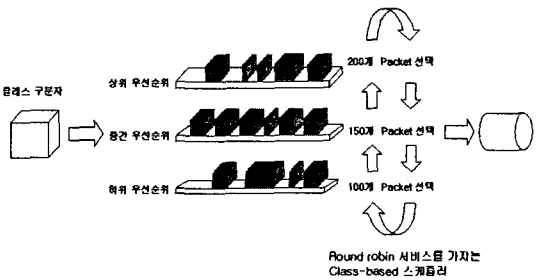
- FIFO Queuing

패킷은 도착하는 순서와 동일한 순서로 서비스를 받게 된다. 이 메커니즘은 차별화 서비스를 제공하기는 어렵지만 구현이 용이하다. 특히 서비스를 기다리는 큐에 삽입, 삭제의 시간이 상수에 비례하고 각 플로우별 상태를 유지할 필요가 없다.

FIFO 정책은 지연과 비율을 보장하는 방법을 제시하고 있지 않다. 지연에 대한 제한을 구현하는 하나의 방법은 서비스를 기다리는 큐의 크기를 제한하는 것이다.

- Class-based Queuing

우선 순위 큐잉 방식의 단점인 우선권이 있는 클래스를 제외한 타 클래스 트래픽의 서비스가 완전히 거부하는 경우를 방지하기 위해서 최근에 제안된 방식이다. CBQ는 우선 순위 큐잉 방식의 변형으로써 하나의 출력 큐 대신에 여러 개의 출력 큐를 두어서 우선순위를 정하고 각 큐별로 서비스 되는 트래픽 양을 조절할 수 있는 방식이다. CBQ는 각 클래스별로 정해진 양의 대역폭을 보장 할 수 있는 방식으로 알려져 있지만, 실제로 완전 보장이기 보다는 우선 순위 큐잉 방식보다는 약간 더 완화된, 클래스별로 자원이 완전 고갈되는 것을 막으면서도 각 클래스에 적절한



[그림 3] CBQ 의 구조

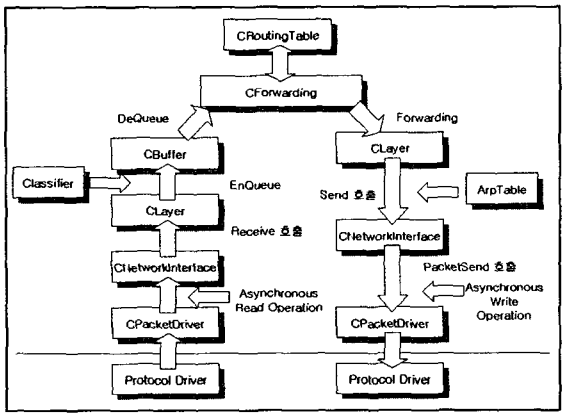
서비스를 제공할 수 있는 방법으로 인식되어야 한다.

5. 라우터 구현

이번 장에서는 Packet Driver와 사용자 정의 프로토콜 드라이버를 이용하여 Router 구현에 대해 살펴 보겠다. 개발에 사용된 툴을 살펴보면 Windows 2000 DDK(Driver Development Kit), Microsoft® Platform SDK(Software Development Kit), Microsoft Visual C++을 사용하였다.

5.1 모델링

라우터의 구성 요소를 객체지향 기반으로 모델링 하여 기존의 프로토콜 계층 수정과 새로운 계층 삽입이 용이하다.



[그림 4] 라우터의 클래스 구조

라우터는 다음과 같이 구성되어 있다.

- **CLayer 클래스**

라우터 구현에 필요한 프로토콜 계층 구현에 이용된는 가상 클래스(Virtual class)이다. 프로토콜 구현에 직접 사용되지 않지만 계층 성형에 필요한 가상 맴버 함수(virtual member function)을 정의하고 있다. 이는 공통된 인터페이스를 제공하고 각 계층의 독립성을 보장해 줄 수 있다.

CLayer부터 상속 받은 CEthernetLayer, CIpLayer, CArpLayer 클래스는 CLayer의 가상 맴버함수를 계층에 특성에 맞게 재정의하여 사용한다. 또한 프로토콜 스택(Protocol Stack)을 구성하기 위해 각 계층의 AddUpDownLayer 가상 멤버함수를 호출하여 프로토콜 스택을 구성한다.

- **CBuffer 클래스**

IP헤더의 서비스 종류(Type of Service Field)을 참조하여 패킷의 우선 순위를 구별하는 CBQ와 기존의 FIFO Queuing을 구였다. CBQ는 Classifier를 포함하며, CLayer부터 받은 IP 패킷을 서비스 종류에 따라 3가지의 우선 순위를 갖는 큐에 저장하고 Round Robin 서비스 방식에 따라 dequeue 된다. FIFO는 고정된 크기의 drop-tail 방식의 큐를 갖는다.

- **CForwarding 클래스**

CBuffer부터 받은 IP Packet을 CRoutingTable을 참조하여 forwarding에 관련된 정보를 얻는다. 이를 이용하여 다음 홉으로 포워딩 하기 위한 데이터 링크 계층의 패킷을 만들고 포워딩 될 네트웍 인터페이스를 결정한다.

- **CRoutingTable 클래스**

라우팅에 필요한 정보를 저장한다. 자료 저장을 위해 CList 템플릿(template)을 사용하였다.

6. 결론

본 논문에서는 Windows에서 위한 API의 문제점을 알아보고 이를 해결하기 위해 NDIS와 Window Base Service를 이용한 Packet driver API와 사용자 정의 프로토콜을 설계하고 이를 기반으로 패킷 스케쥴링을 지원 가능한 라우터를 구현하였다. 응용 계층은 Base Service를 이용하여 커널 계층의 사용자 정의 Protocol Driver에 접근 가능하며, 데이터 링크 계층의 접근 할 수 있다.

응용 계층과 커널 계층의 입출력에 있어서, 사용자 계층보다 커널 계층에게 더 많은 자원이 할당되므로 응용 계층에 위치한 Packet Driver와 버퍼의 성능이 커널에 구현되어 있는 것보다 상대적으로 낮게 나타난다. 이를 해결하기 위해 사용자 정의 Protocol driver에 기존의 버퍼를 확장하여 구성하는 방법과 응용 계층에서의 효율적인 비동기 모드 운영을 위한 임계 영역에 대한 연구가 필요하다.

참고 문헌

[1] MSDN, Microsoft® Platform Software Development Kit(April 2000) Documentation, Microsoft Corporation, 2000.
[2] MSDN, Windows 2000 Driver Development Kit (March 2000) Documentation, Microsoft Corporation, 2000.
[3] MSDN, MSDN Library Visual Studio 6.0, Microsoft Corporation, 1998.
[4] Anthony Jones and Jim Ohlund., *Network Programming for Microsoft Windows*, Microsoft®Press, 1999.
[5] Geoff Huston, *Internet Performance Survival Guide, QoS Strategies for Multiservice Networks*, John Wiley & Sons, Inc., 2000.
[6] Wright Stevens, *TCP/IP Illustrated, Volume II The Implementation*, Addison-Wesley, 1995.