Design and Implementation of Packet Driver for Accessing Data Link Layer in Windows 2000

Hwan-Souk Yoo, Sang-Ha Kim*
*Dept. of Computer Science, ChungNam University

Abstract

The current practice for designing and implementing network software is to use the pre-defined Winsock API, which provides the necessary functions for communication between applications and the network. However, this API does not provide the necessary functions for accessing the network protocol layer. Therefore, we have developed a new packet driver API that allows applications to access the network protocol layer directly.

1. Introduction

The current practice for designing and implementing network software is to use the pre-defined Winsock API, which provides the necessary functions for communication between applications and the network. However, this API does not provide the necessary functions for accessing the network protocol layer. Therefore, we have developed a new packet driver API that allows applications to access the network protocol layer directly.

2. Windows 2000 Network API

Windows 2000 provides a new network interface called Network Interface Card (NIC). This interface is used to communicate with other computers on the network. The NIC provides a set of functions that allow applications to access the network protocol layer directly. The Windows 2000 Network API provides a set of functions that allow applications to access the NIC.

3. Conclusion

In conclusion, we have developed a new packet driver API that allows applications to access the network protocol layer directly. This API provides a set of functions that allow applications to access the network protocol layer in a more efficient and flexible manner.

References


1117
2.1 Winsock (Windows Sockets)

Windows 계열 대부분의 인터넷 소프트웨어가 사용하고 있는 API 이다. Winsock 은 프로토콜이 아니라 전송 프로토콜 계층에 접근하기 위한 인터페이스를 제공하며 DLL 형식으로 Windows 시스템에 존재하고 있는 WS2_32.dll 파일이 이에 해당한다.

그림 1. Winsock의 구조 [1]

그림 1에서의 WS2_32.dll은 상위 계층으로는 API를 통하여 사용자 프로그램에 인터페이스를, 하위 계층으로는 SAPI(Service Provider Interface)를 통해 전송 프로토콜 계층에 인터페이스를 제공한다.[1]

Winsock은 전송 프로토콜 계층과 인터페이스를 이루고 있으므로 기본적으로 그에 해당하는 TCP나UDP에만 접근할 수 있다. 이를 확장하기 위해서는 방법으로는 소켓(socket)의 프로토콜 종류인 SOCK_RAW 하여 사용하는 방법이 있다. 이 방법은 ICMP의 접근에 적합하여 TCP, UDP, IGMP 접근을 위해 사용할 수 있다.

Winsock의 확장이 더 필요한 경우는 setsockopt() 함수와 IP_HDRINCL 옵션을 사용하여 IP 헤더(Header) 부분까지 확장이 가능하다. 이 방법은 소켓에게 보내지는 데이터의 일부분에 IP 헤더가 포함된 것을 알리는다고 전송 프로토콜 층에서는 IP 헤더를 만들지 않는 대신 사용자가 작성한 IP 헤더를 사용하게 한다. 그러므로 사용자가 IP 패킷 전체를 만들고 전송할 수 있게 된 것이다.[4]

하지만 Winsock에서는 전송에 대한 확장만이 허용할 뿐 수신에 대한 확장은 지원하지 않으므로 확장성을 위해서는 Network Monitor SDK에 NPP를 사용한다.[1]

2.2 NPP (Network Packet Providers)

MS(Microsoft)사에서는 시스템 관리에 위하여 SMS(Systems Management Server)를 개발하였다. 이 중에서 네트워크의 문제점 발견과 정보 수집을 위한 네트워크 모니터링 시스템에서는 데이터 밸크 계층의 프레임에 대한 수집이 필요하게 되었다. 이를 지원하기 위해 개발된 것이 NPP(Network Packet Providers)이었다.[1]

그림 2는 NPP의 구조를 나타내고 있다. NPP는 커널 계층의 네트워크 모니터링 시스템 드라이버(Nm nt.sys)로부터 얻은 네트워크 정보를 Network Monitor UI, Monitor Applications, NPP Applications 등에게 COM Interface를 통하여 전달한다. NPP는 프레임 수집 기능을 수행하는 부분과 트래픽에 대한 통계 정보를 관리하는 부분으로 구성되어 있다.

IDelayDC Interface는 트래픽에 대한 정보를 과열체다가 저장하여 결과를 알리며, IRTC Interface는 실시간 트래픽에 대한 정보를 수집하여 고정된 크기의 버퍼에 저장한다. IESP Interface는 수집된 프레임에 대한 자세한 통계정보를 파일로 제공해준다. IStats Interface는 각종 Filter 수준의 통계 정보를 제공해준다. (예, 수집된 프레임의 수와 크기, Multicasting을 통해 전송 받은 프레임의 수와 크기) [1]

하지만 NPP는 Monitoring을 위해 개발되었기 때문에 정보수집에 관련된 오퍼레이션을 수행할 수 있고, 데이터 맵크 레벨의 전송 기능은 지원하지 않는다.

2.3 IP Helper (Internet Protocol Helper)

IP Helper는 Internet Protocol을 사용하는 컴퓨터의 관리에 중점을 두는 API라고 볼 수 있다.

IP Helper는 다음과 같은 기능을 가지고 있다.
- 네트워크 설정에 관련된 정보 관리

그림 2. NPP의 구조 [1]

- 네트워크 인터페이스에 관련된 정보 관리
- 로컬 컴퓨터의 IP 계층 설정 관리
- ARP 기능 수행 및 ARP Table 관리
- 프로토콜의 통계정보
- Routing Table 관리
- 네트워크 설정의 변경에 대한 알림 기능

IP Helper에서는 사용자 계층에서 ARP를 직접 수행할 수 있으며, ARP Table의 인립, 추출, 삭제 기능하. 하지만 단순 ARP만 지원하므로 RARP, Proxy ARP, Gratuitous ARP는 구현 불가능하다.

IP Helper 기능은 NDIS Library 를 사용하여 모두 구현 가능하지만 사용자 계층에서는 커널의 NDIS 에 직접 접근할 수 없으므로 IP Helper를 사용하여 개발자에게 보다 편리한 개발 환경을 제공해준다고 있다.

2.4 API와 프로토콜의 관계

표 1은 지금까지 알아본 API와 프로토콜과 대하여 API별 지원 가능한 프로토콜을 요약한 것이다.
<table>
<thead>
<tr>
<th>Winsock (RAW Mode)</th>
<th>TCP</th>
<th>UDP (IP)</th>
<th>IP Multicasting</th>
<th>IP Broadcasting (ICMP) (IGMP)</th>
<th>Asynchronous Read/Write 기능</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFP</td>
<td>DataLink Layer</td>
<td>Promiscuous</td>
<td>Read</td>
<td>Promiscuous Read</td>
<td>RARP, Proxy ARP, Gratuitous ARP 동작 가능</td>
</tr>
<tr>
<td>IP Helper</td>
<td>ARP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

표 1. API와 프로토콜들과의 관계

일반적으로 네트워크 프로그래밍을 위한 API는 다음과 같은 요구 사항이 있다.
- 가능한 낮은 데이터 링크 계층까지 접근 가능.
- Promiscuous 모드로 동작 가능.
- 사용자 정의 프로토콜을 사용할 수 있어야 함.
- 사용자 계층에서 쉽게 접근 가능.
- Asynchronous 모드로 동작 가능.
- 네트워크 인터페이스의 정보 열람 가능.
- 로컬 컴퓨터의 네트워크 설정 정보를 열람 가능.
- 장치 드라이버 계층의 정보를 획득/설정 가능.

Windows 2000에서 사용할 수 있는 API는 위의 요구사항을 보다 잘 만족하며 가장 중요한 기능인 데이터 링크 계층에서 사용자의 직접 접근을 지원하지 않는다. 이 기능은 라우터의 포워딩(forwarding)과 같은 기능을 구현하기 위해서는 반드시 필요하다.

이런 문제를 해결하기 위하여 NDIS Library를 이용한 packet driver를 사용한다.

3. Packet Driver의 설계 및 구현


3.1 NDIS (Network Driver Interface Specification)

그림 3은 NDIS를 사용하여 프로토콜 스택을 구성한 것이다. NDIS는 다음과 같은 기능을 제공한다.

- NDIS Miniport Driver
  하드웨어와 상위 계층간의 통신을 담당한다. NIC을 통한 데이터 전송 및 수신 기능이 있으며, 상위 계층의 트래픽을 위한 인터페이스 가지고 있다. 대부분 NIC 제조업체에서 제공한다.

- NDIS Intermediate Driver
  새로운 계층을 추가하여 새로운 프로토콜 추가, 패킷 필터링, 다른 종류의 네트워크 미디어간의 패킷 교환 등의 기능을 수행할 수 있다. 상위 계층 인터페이스로 NDIS Miniport Driver를 사용하여 하위 계층 인터페이스로 NDIS Protocol Driver를 사용한다.

- NDIS Protocol Driver

그림 4는 사용자 계층에서 데이터 링크 계층 접근을 위한 Packet Driver의 구조를 나타내고 있다. 사용자의 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 Protocol Driver에 대하여 자세히 알아본다.

3.1.1 NDIS Protocol Driver 등록


예) SendCompleteHandler = PacketSendComplete

다음과 같이 정의되어 있다면 Protocol Driver가 패킷을 보내기 위해서 마다 NdisSendPackets() 함수를 호출하고 정상적으로 수행된 경우 데이터 테이블에 의해 Protocol Driver의 PacketSendComplete() 함수가 수행되어 그 결과를 알 수 있다.

DRIVER_OBJECT 구조체는 사용자가 커널 계층으로 접근하기 위해 WIN32 API 사용할 때 WIN32 API I/O 함수와 Protocol Driver의 정의되어 있는 함수와의 대응 관계를 나타낸다. 사용자가 CreateFile()함수를 사용하려면 하위 계층에서 IRP_MJ_CREATE에 해당하는 함수가 반드시 존재해야 한다.

Ex) MajorFunction[IRP_MJ_CREATE] = PacketOpen

3.1.2 I/O Control 및 Request/Query

사용자 계층에서 I/O Control을 위하여 WIN32 API 중 DeviceIoControl() 함수를 사용하면 커널 계층에서는 Protocol Driver의 정의에 의해 PacketIoControl() 함수가 실행된 다. PacketIoControl은 두가지의 방법으로 Request를 처리한다. Request 가 NDIS Driver에 관련된 것이면 PACKET_OID_DATA 구조체에 의해 NDIS 드라이버에 정보를 요청한다. 하지만 일반적인 I/O Control이면 커널의 상
위 드라이버가 하위 드라이버에게 정보를 요청하거나 절의
하는데 가장 많이 쓰이는 IRP Function 을 사용하여 처리한
d.[2]

<table>
<thead>
<tr>
<th>Packet Driver</th>
<th>WIN32 API</th>
<th>Protocol Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>PacketOpenAdapter</td>
<td>CreateFile</td>
<td>PacketOpen</td>
</tr>
<tr>
<td>PacketSendPacket</td>
<td>WriteFile</td>
<td>PacketWrite</td>
</tr>
<tr>
<td>PacketReceivePacket</td>
<td>ReadFile</td>
<td>PacketRead</td>
</tr>
<tr>
<td>PacketRequest</td>
<td>DeviceIoControl</td>
<td>PacketIoControl</td>
</tr>
<tr>
<td>PacketWaitPacket</td>
<td>OverlappedResult</td>
<td>CompleteFunction</td>
</tr>
</tbody>
</table>

표 2 계층별 함수의 대응관계

3.2 WIN32 API 를 이용한 WIN32 Packet Driver

사용자 계층의 Packet Driver 는 WIN32 API 를 사용하여
구성되어 있고, 사용자의 편의를 위하여 클래스로 구성을
있다. 기존의 C 스타일로 쓰이던 Packet Driver 를 Visual
C++에서 사용할 수 있게 클래스화하고 이를 상속 받아
ChetworkInterface 클래스로 확장
하였다. ChetworkInterface 클래스에는 Asynchronous 수행을 위한
정적 크기의 tail-drop 버퍼를 가지고 있으며, 스레드 생성
을 위한 부분과 여러 쓰레드에서 버퍼에 동시에 접근을 방지하는 Critical Section을 설정하였다.

Chost 클래스에서는 시스템에 설치되어 있는 NIC 을 발견
하고 사용자에게 알려준다. 사용자는 NIC의 이름을 이용하
며 ChetworkInterface 클래스를 초기화 동적 관리를 위하여
Chetwork 클래스에 추가한다. Chetwork 클래스에는 모든
NIC에 대한 포인터를 가지고 있기 때문에 Chetwork 클래스
버퍼로 연결하여 Repeater 를 구현하였다.

그림 6은 리피터를 Class Diagram으로 나타낸 것이다.

그림 5 Packet Driver를 사용한リ피터의 구조

그림 4. Packet Driver의 구조

만 가지면 모든 NIC은 저장할 수 있다.

표 2는 사용자 계층에서 Protocol Driver까지의 계층적 구조를 나타낸 것이다.

3.3 리피터(Repeater) 구현

그림 5는 리피터를 Packet Driver API를 사용하여 모델링
한 것이다. 리피터의 원리는 NIC#1으로 들어온 데이터 링크
계층의 프레임을 변환하여 NIC#2로 보내어 내보내는 것이다. 반대로도 동작한다. NIC#1의 입력버퍼를
NIC#2의 출력버퍼로, NIC#2의 입력버퍼는 NIC#1의 출력

4. 결론

본 논문에서는 Windows 2000에서 인터넷 소프트웨어 개
발을 위한 API의 문제점을 알아보고 이를 해결하기 위해
NDIS 를 이용한 packet driver API를 설계하고 구현하였다.
Packet driver API는 사용자 레벨에서 WIN32 API를 사용하여
커널의 Protocol Driver를 접근하여 데이터 링크 레벨의 프레
임을 조작할 수 있다.

사용자 계층과 커널 계층의 입출력에 있어서, 사용자 계
층보다 우선 순위가 높은 커널 계층에 더 많은 자원이 할
당되므로 packet driver 성능이 커널의 Driver보다 상대적으
로 낮게 감소한다. 이를 해결하기 위해 Protocol driver에 사용
자 버퍼를 구성하는 방법과 사용자 계층에서의 비동기화 버
퍼에 대하여 연구가 필요하다.

4. 참고 문헌