컴네 ARP 뿌시기

made by **mando**

▼ **Table of contents**

- 5조 레포지토리
- 이론 공부
  - ARP Protocol이란?
  - ARP 패킷 구조
    - Ethernet Frame
    - ARP Frame
  - ARP 데이터 flow
  - ARP cache table이란?
- 프로토콜 구조
- 각 프로토콜 역할
- ARP 흐름도 (순서도 넣기) 작성
- 실습 시나리오(유즈 케이스 다이어그램) 작성
  - 실습 시나리오 시퀀스 다이어그램 (보강 필요)
- 코드 구현 설명
  - [App Layer]
  - [IP Layer]
  - [ARP Layer]
  - [Ethernet Layer]
  - [NI Layer]

5조 레포지토리

https://github.com/troymerai/2023CN_ARPWithMFC

이론 공부

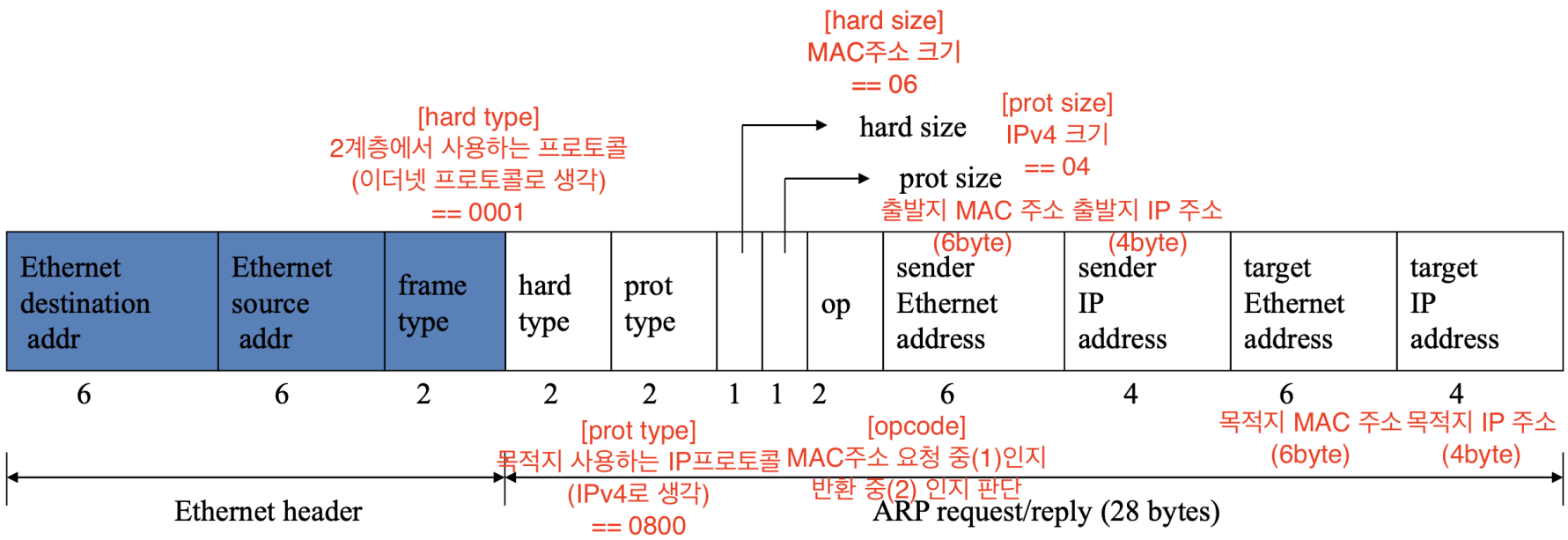
ARP Protocol이란?

ARP 프로토콜은 같은 네트워크 대역에서 통신을 하기 위해 필요한 MAC 주소를 IP 주소를 이용해서 알아오는 프로토콜이다.

같은 네트워크 대역에서 통신을 한다고 하더라도 데이터를 보내기 위해서는 7계층부터 캡슐화를 통해 데이터를 보내기 때문에 IP주소와 MAC 주소가 모두 필요하다. 이때, IP 주소는 알고 MAC 주소는 모르더라도 ARP를 통해 통신이 가능하다.

(평상 시에는 컴퓨터가 자동으로 ARP를 해주기 때문에 IP주소만 알아도 통신할 수 있다.)

ARP 패킷 구조



Ethernet Frame

[Ethernet destination address Field]

Broadcast 주소

[Ethernet source address Field]

출발지의 MAC 주소

[frame type Field]

ARP 요청과 반환 시, 이 값은 0x0806으로 고정

ARP Frame

[hard type Field]

2계층에서 사용하는 프로토콜

이더넷 프로토콜만 있는 건 아니지만 이더넷 프로토콜이라고 생각하면 됨

0001 또는 1로 표기(이더넷 프로토콜일 경우)

[prot type Field]

목적지에서 사용하는 IP 프로토콜 (IPv4로 생각)

0800으로 표기

[hard size Field]

MAC 주소 크기

06 또는 6으로 표기

[prot size Field]

IPv4 크기

04 또는 4로 표기

[op(opertion) Field]

MAC 주소 요청 중 == ARP request(1로 표기)

MAC 주소 반환 중 == ARP reply(2로 표기)

IP 주소 요청 중 == RARP request(3으로 표기)

IP 주소 반환 중 == RARP reply(4로 표기)

[sender Ethernet address Field]

출발지 MAC 주소 (6byte)

[sender IP address Field]

출발지 IP 주소 (4btye)

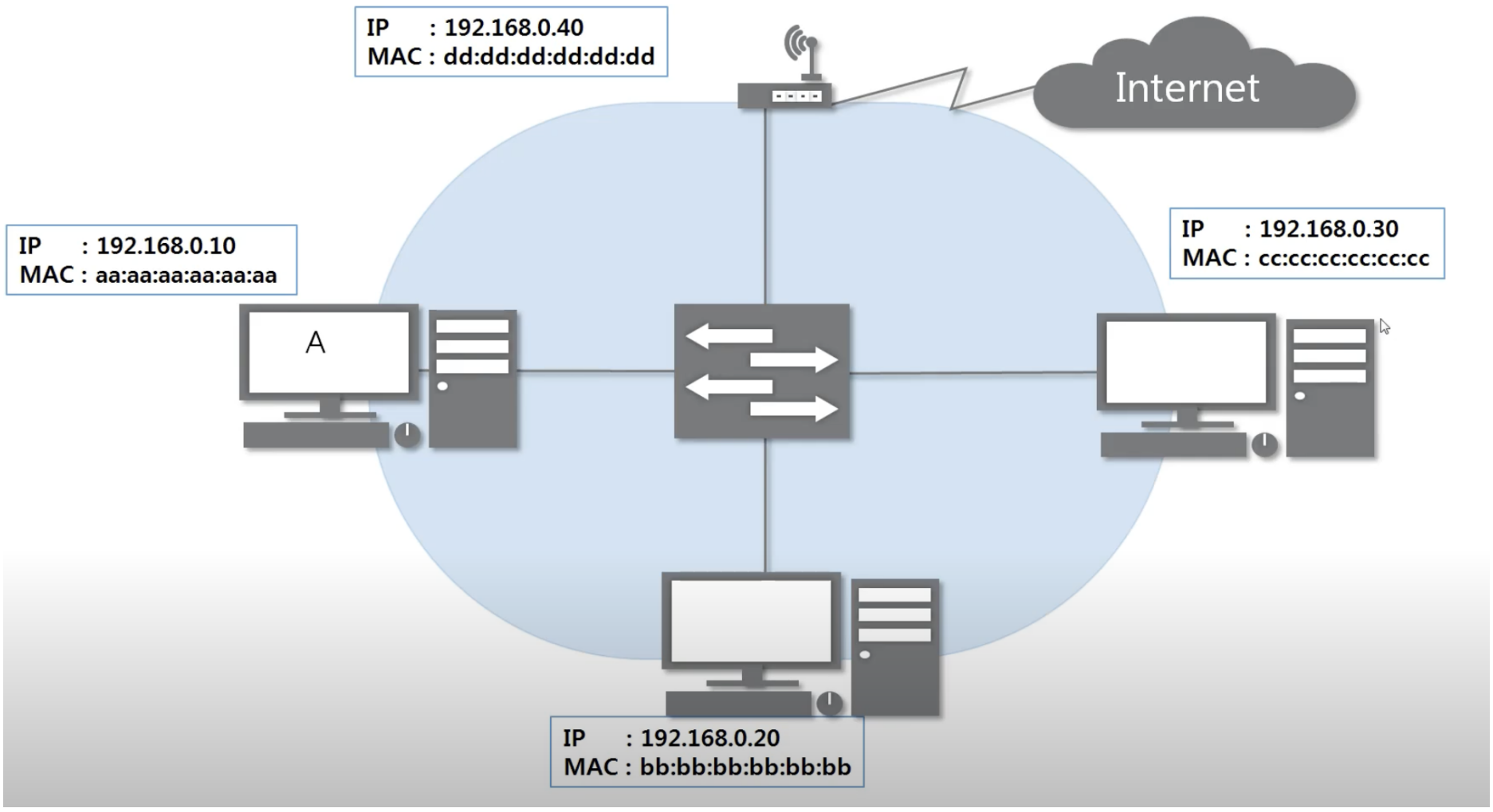
[target Ethernet address Field]

목적지 MAC 주소 (6byte)

[target IP address Field]

목적지 IP 주소 (4byte)

ARP 데이터 flow



같은 LAN 대역(같은 네트워크 대역)에 있는 컴퓨터 A가 컴퓨터 C와 통신하고싶지만 IP주소만 알고있는 상황이라고 가정

1. 컴퓨터 A가 ARP요청 프로토콜을 작성

ARP 요청 패킷에 이더넷 헤더를 붙여서 작성한다.

근데, ARP 패킷에도 목적지 MAC 주소를 써야하고 이더넷 헤더에도 목적지 MAC 주소를 써야하는데 모르지 않나?

먼저 ARP Frame을 살펴보자

[request - ARP Frame]

00 01		**08 00**
06	04	00 01
aa aa aa aa		
aa aa		c0 a8
00 0a		00 00
00 00 00 00		
c0 a8 00 1e		

출발지 IP 주소는 16진수로 변환한 것이다.

목적지 MAC 주소는 모르니까 00 00 00 00 00 00 으로 비워둔다.

목적지 IP 주소도 16진수로 변환하여 적는다.

그렇다면 이더넷 헤더는 어떨까?

[request - Ethernet Frame]

FF FF		**FF FF**
FF FF		aa aa
aa aa aa aa		
08 06		

목적지의 MAC 주소를 모르기때문에 FF FF FF FF FF FF로 작성한다. 이진수로 생각하면 1로 꽉채운 것이다.

== broadcast라는 뜻

이렇게 이더넷 헤더를 broadcast로 지정하면 스위치로 연결된(== 같은 네트워크 대역) 모든 장비에게 프레임 전송

2. 컴퓨터 A가 스위치(L2 장비)한테 프레임 전송
3. 스위치는 L2장비니까 이더넷 헤더만 까서 확인
4. broadcast니까 스위치에 연결된 모든 장비에 프레임 재전송
5. 프레임을 받은 장비들(컴퓨터 C 포함)은 이더넷 헤더를 까고 이더넷 헤더에 적힌 MAC주소를 확인, broadcast임을 확인하고 ARP 프레임을 까봄(L3 장비니까)
6. 각 컴퓨터는 목적지 IP주소를 확인하고 자신의 IP 주소와 다르면 discard, 같으면 출발지 MAC 주소에 자신의 MAC 주소(컴퓨터 C의 MAC 주소)를 넣어서 ARP reply 패킷의 ARP Frame 작성

00 01		**08 00**
06	04	00 02
cc cc cc cc		
cc cc		c0 a8
00 1e		aa aa
aa aa aa aa		
c0 a8 00 0a		

7. ARP reply 패킷의 Ethernet Frame에서도 어디서 온 프레임인지 알기 때문에 도착지 MAC 주소(컴퓨터 A의 MAC 주소)를 잘 쓸 수 있음

aa	aa	aa	aa
aa	aa	cc	cc
cc	cc	cc	cc
08	06		

8. ARP - reply 패킷을 받은 컴퓨터 A는 다 까서 컴퓨터 C의 MAC 주소를 확인하고 자신의 ARP cache table을 갱신

ARP cache table이란?

나(내 컴퓨터)와 통신했던 컴퓨터들의 MAC 주소와 IP 주소들을 하나씩 맵핑 시켜놓은 것

하지만, cache table이기에 일정 시간이 지나면 사라진다. 사라지고 나서 다시 그 컴퓨터와 통신하려면 ARP를 통해 cache table을 채워넣고 통신한다. (수동으로 등록해서 영구적으로 사용하는 방법도 있음)

complete는 20분마다 incomplete는 3분마다 삭제

프로토콜 구조

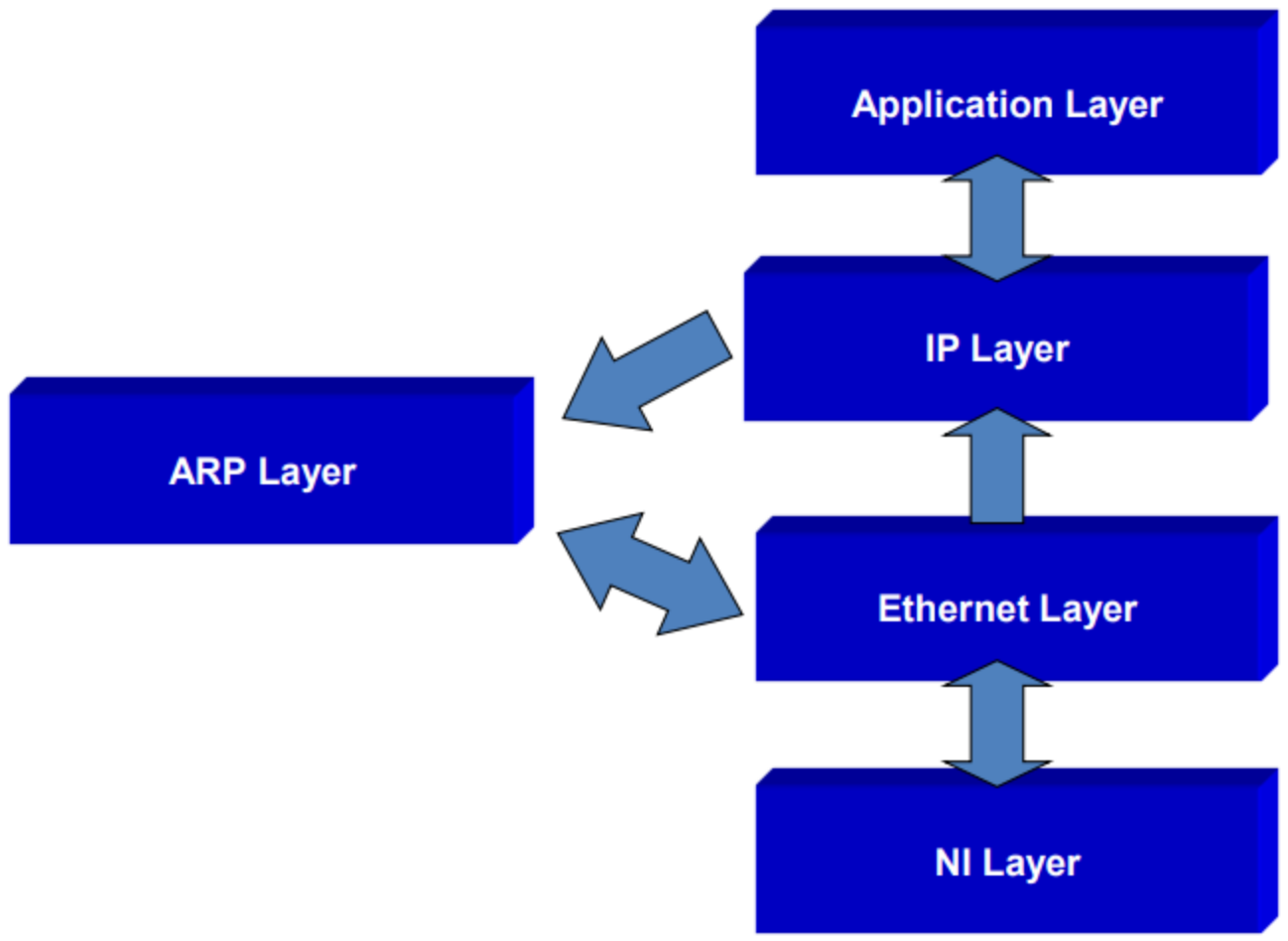


Figure 1. Layered Architecture

각 프로토콜 역할

[application layer]

GUI 부분을 담당한다.

사용자가 네트워크 어뎁터를 선택할 수 있는 UI를 제공하고 해당 어뎁터의 MAC 주소와 IP 주소를 보여준다.

목적지 호스트의 IP 주소를 입력하는 UI를 제공한다.

ARP cache table을 GUI로 구현하여 보여준다.

ARP 요청을 보낸다.

[IP layer]

IP와 관련된 로직이 정의된 곳이다.

IP 주소를 통해 호스트를 구분한다.

ARP Layer로 ARP request 패킷을 전송한다.

[ARP layer]

ARP 프로토콜의 로직이 정의된 곳이다.

ARP request, reply 패킷을 처리한다.

ARP cache table 값을 업데이트한다.

[ethernet layer]

Ethernet과 관련된 로직이 정의된 곳이다.

MAC 주소를 설정한다.

MAC 주소를 통해 호스트를 구분한다.

이더넷 헤더 타입 변수를 통해 레이어를 구분하여 전송한다.

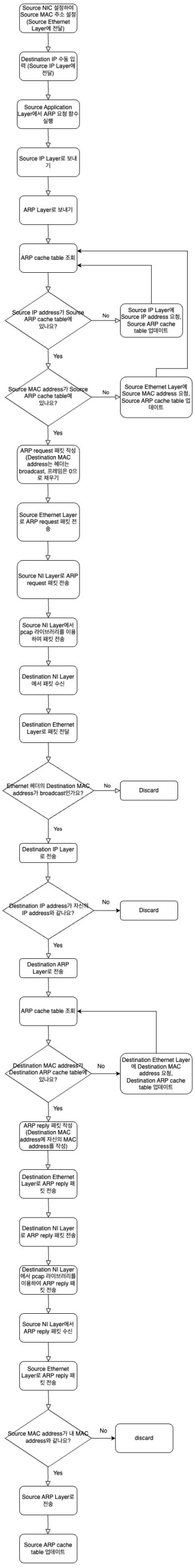
[NI layer]

선택된 어뎁터로부터 프레임 전송 및 수신을 담당한다.

선택된 어뎁터의 IP 주소와 MAC 주소를 가져온다.

ARP 흐름도 (순서도 넣기) 작성

ARP data flow를 정리하여 순서도를 그려보면 다음과 같다.


Source NIC 설정하여 Source MAC 주소 설정 (Source Ethernet Layer에 전달)
Destination IP 수동 입력 (Source IP Layer에 전달)
Source Application Layer에서 ARP 요청 함수 실행
Source IP Layer로 보내기
ARP Layer로 보내기
ARP cache table 조회
Source IP address가 Source ARP cache table에 있나요?
No
Source IP Layer에 Source IP address 요청, Source ARP cache table 업데이트
Yes
Source MAC address가 Source ARP cache table에 있나요?
No
Source Ethernet Layer에 Source MAC address 요청, Source ARP cache table 업데이트
Yes
ARP request 패킷 작성 (Destination MAC address는 헤더는 broadcast, 프레임은 0으로 채우기
Source Ethernet Layer로 ARP request 패킷 전송
Source NI Layer로 ARP request 패킷 전송
Source NI Layer에서 pcap 라이브러리를 이용하여 패킷 전송
Destination NI Layer에서 패킷 수신
Destination Ethernet Layer로 패킷 전달
Ethernet 헤더의 Destination MAC address가 broadcast인가요?
No
Discard
Yes
Destination IP Layer로 전송
Destination IP address가 자신의 IP address와 같나요?
No
Discard
Yes
Destination ARP Layer로 전송
ARP cache table 조회
Destination MAC address가 Destination ARP cache table에 있나요?
No
Destination Ethernet Layer에 Destination MAC address 요청, Destination ARP cache table 업데이트
Yes
ARP reply 패킷 작성 (Destination MAC address에 자신의 MAC address를 작성)
Destination Ethernet Layer로 ARP reply 패킷 전송
Destination NI Layer로 ARP reply 패킷 전송
Destination NI Layer에서 pcap 라이브러리를 이용하여 ARP reply 패킷 전송
Source NI Layer에서 ARP reply 패킷 수신
Source Ethernet Layer로 ARP reply 패킷 전송
Source MAC address가 내 MAC address와 같나요?
No
discard
Yes
Source ARP Layer로 전송
Source ARP cache table 업데이트

실습 시나리오(유즈 케이스 다이어그램) 작성

1. 두 대의 PC에서 각각 프로그램을 실행한다.
2. 두 대의 PC는 같은 네트워크로 연결한다. (같은 WIFI)
3. ARP Request 및 Reply를 받을 NIC를 선택 및 개방한다.
4. 출발지 호스트에서 전송 버튼을 누른다.
    - ARP cache table에 IP address, MAC address, status를 확인한다.
5. ARP Request 패킷을 작성한다.
    - Hardware Type은 1
    - Protocol Type은 0x0800(IP)
    - Opcode를 1(ARP Request)
    - Sender영역에 자신의 MAC 주소와 IP 주소 설정한다.
    - Target영역에 Hardware는 0으로 protocol address는 목적지 IP주소로 설정한다.
6. Ethernet헤더를 작성한다.
    - Ethernet Header의 Destination Address를 Broadcast 주소로 채운다.
    - Ethernet Header의 Source Address를 자신의 MAC 주소로 채운다.
    - Ethernet Type을 0x0806(ARP)로 설정한다.
7. 레이어 아키텍처에 의해서 Encapsulated Packet(==Ethernet frame)이 생성된다.
8. NI Layer의 pcap라이브러리로 작성된 코드로 packet은 네트워크로 전송된다.
9. OnTimer를 설정(180초)한다.
10. 목적지 호스트에서 패킷(Request)을 수신한다.
    - Ethernet Layer에서 목적지 MAC 주소가 broadcast인지 확인한다.
        - 아니라면, discard(버림)
    - 맞으면, 레이어 아키텍처에 의해서 Demultiplexing을 통해 header를 제외한 data부분을 상위 레이어(IP Layer)로 전달한다.
        - Header는 매 레이어마다 receive 함수에서 frame에 대해 나에게 온 것이 맞는 지 검사할 때 쓰인다.
    - IP Layer에서 수신된 payload의 destination IP address가 자신의 IP 주소와 같은 지 확인한다.
      - 아니라면, discard(버림)
    - 맞으면, ARP Reply 패킷 작성한다.
        - Opcode를 2(ARP Reply)
        - sender영역과 target영역을 swap
        - source MAC address에 자신의 MAC 주소를 넣어준다.
11. 이더넷 헤더를 작성한다.
    - Ethernet Header의 Destination Address를 출발지의 MAC 주소로 채운다.
    - Ethernet Header의 Source Address를 자신의 MAC 주소로 채운다.
    - Ethernet Type을 0x0806(ARP)로 설정한다.
12. 레이어 아키텍처에 의해서 Encapsulated Packet(==Ethernet frame)이 생성된다.
13. NI Layer의 pcap라이브러리로 작성된 코드로 packet은 네트워크로 전송된다.
14. 출발지 호스트에서 패킷(Reply)을 수신한다.
    - Ethernet Layer에서 목적지 MAC 주소가 자신의 MAC 주소와 같은 지 확인한다.
        - 다르다면, discard(버림)

- 같다면, 레이어 아키텍처에 의해서 Demultiplexing을 통해 header를 제외한 data부분을 ARP 레이어로 전송한다.

15. ARP 레이어에서 ARP cache table을 업데이트한다.
16. 업데이트한 ARP cache table을 APP 계층에서 확인하고 UI를 업데이트한다.

실습 시나리오 시퀀스 다이어그램 (보강 필요)

```
sequenceDiagram
    participant PC1 as 출발지 호스트
    participant Net as 네트워크
    participant PC2 as 목적지 호스트

    PC1->>+Net: ARP Request 전송 (Ethernet Header: Broadcast, ARP: 자신의 MAC/IP, 목적지 IP)
    Note over PC1,Net: Ethernet Frame 생성
    Net-->>+PC2: ARP Request 수신
    PC2->>+Net: ARP Reply 전송 (Ethernet Header: 출발지 MAC, ARP: 자신의 MAC/IP, 출발지 MAC/IP)
    Note over PC2,Net: Ethernet Frame 생성
    Net-->>+PC1: ARP Reply 수신
    PC1->>PC1: ARP Cache Table 업데이트
    PC1->>PC1: UI 업데이트
```

코드 구현 설명

> 💡 전체 코드는 깃허브에 있습니다

[App Layer]

[dlg]

UI에서 네트워크 어댑터를 선택하면 돌아가는 코드

```
// 네트워크 어뎁터 선택 UI
void CARPDlg::OnCbnSelchangeComboAdapter()
{
  // MAC 주소, IPv4, IPv6 주소를 저장할 CString 변수 선언
  CString MAC, IPV4, IPV6;
  // 현재 선택된 네트워크 어댑터의 MAC 주소 반환 함수(NI Layer 참조)
  unsigned char* macaddr = m_NILayer->SetAdapter(m_ComboxAdapter.GetCurSel());
  // MAC 주소를 반환받지 못한 경우
  if (macaddr == nullptr) {
    // MAC 주소에 기본 텍스트 설정
    MAC = DEFAULT_EDIT_TEXT;
  }
  // MAC 주소를 반환 받은 경우
  else {
    // MAC 주소 설정
    MAC.Format(_T("%hhx:%hhx:%hhx:%hhx:%hhx:%hhx"), macaddr[0], macaddr[1], macaddr[2], macaddr[3], macaddr[4], macaddr[5]);
    // 출발지 MAC 주소를 선택한 MAC 주소로 채움
    m_EtherLayer->SetSourceAddress(macaddr);
    // 선택한 어댑터의 IP주소를 가져옴
    m_NILayer->GetIPAddress(IPV4, IPV6);
  }
  // MAC 주소를 UI에 표시
  m_editSrcHwAddr.SetWindowTextW(MAC);
  // IPv4 주소를 UI에 표시
  m_SrcIPADDRESS.SetWindowTextW(IPV4);
}
```

++ 보고서용 스크린샷

```
348   // 네트워크 어댑터 선택 UI
349  void CARPDlg::OnCbnSelchangeComboAdapter()
350  {
351      // MAC 주소, IPv4, IPv6 주소를 저장할 CString 변수 선언
352      CString MAC, IPV4, IPV6;
353      // 현재 선택된 네트워크 어댑터의 MAC 주소 반환 함수(NI Layer 참조)
354      unsigned char* macaddr = m_NILayer->SetAdapter(m_ComboxAdapter.GetCurSel());
355      // MAC 주소를 반환받지 못한 경우
356      if (macaddr == nullptr) {
357          // MAC 주소에 기본 텍스트 설정
358          MAC = DEFAULT_EDIT_TEXT;
359      }
360      // MAC 주소를 반환 받은 경우
361      else {
362          // MAC 주소 설정
363          MAC.Format(_T("%hhx:%hhx:%hhx:%hhx:%hhx:%hhx"), macaddr[0], macaddr[1], macaddr[2], macaddr[3], macaddr[4], macaddr[5]);
364          // 출발지 MAC 주소를 선택한 MAC 주소로 채움
365          m_EtherLayer->SetSourceAddress(macaddr);
366          // 선택한 어댑터의 IP주소를 가져옴
367          m_NILayer->GetIPAddress(IPV4, IPV6);
368      }
369      // MAC 주소를 UI에 표시
370      m_editSrcHwAddr.SetWindowTextW(MAC);
371      // IPv4 주소를 UI에 표시
372      m_SrcIPADDRESS.SetWindowTextW(IPV4);
373  }
```

UI에서 select(reselect) 누르면 돌아가는 코드

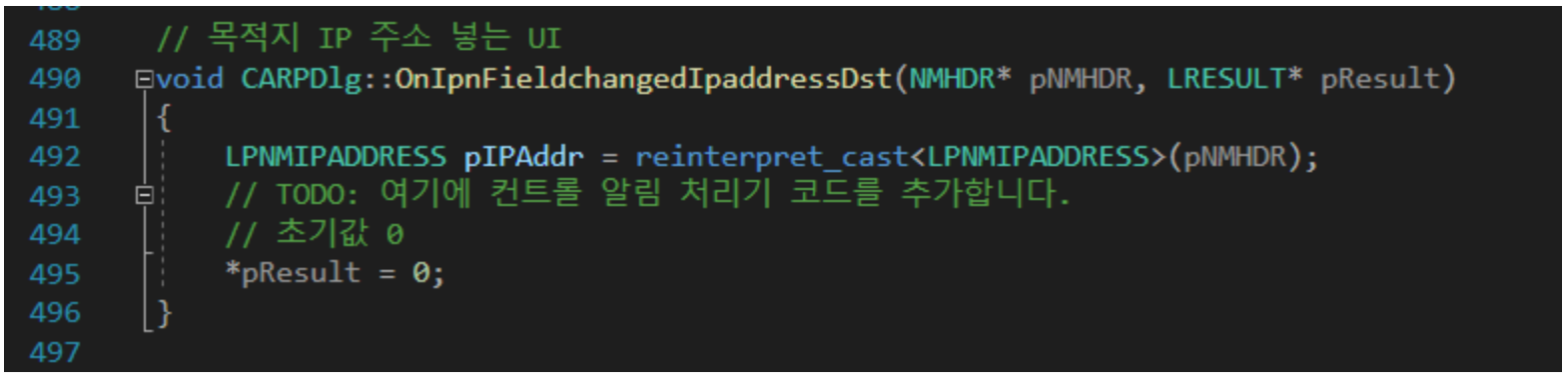
```
// 네트워크 어댑터 정보 확정 버튼
void CARPDlg::OnBnClickedButtonSelect()
{
  // MAC 주소와 IP 주소를 저장할 CString 변수 선언
  CString MAC, IP;
  // 현재 UI(app계층)에서 MAC 주소와 IP 주소를 가져옴
  m_editSrcHwAddr.GetWindowTextW(MAC);
  m_SrcIPADDRESS.GetWindowTextW(IP);

  // 네트워크 어댑터 선택 콤보 박스가 활성화되어 있는 경우
  if (m_ComboxAdapter.IsWindowEnabled()) {

    // MAC 주소와 IP 주소가 유효한 경우
    if (MAC != DEFAULT_EDIT_TEXT && IP != "0.0.0.0") {

      // 네트워크 어댑터 선택 콤보 박스와 IP 주소 입력 필드를 비활성화
      // 대상 IP 주소 입력 필드를 활성화
      m_ComboxAdapter.EnableWindow(FALSE);
      m_SrcIPADDRESS.EnableWindow(FALSE);
      m_DstIPADDRESS.EnableWindow(TRUE);

      // NI Layer에서 패킷 수신 상태 변경 (가능<-불가능)
      m_NILayer->Receiveflip();

      // 자신의 MAC 주소와 IP 주소 설정
      m_ARPLayer->setmyAddr(MAC, IP);

      // 버튼의 텍스트를 ReSelect로 변경
      CDialog::SetDlgItemTextW(IDC_BUTTON_SELECT, _T("ReSelect"));

      // 1초마다 타이머 실행
      SetTimer(1, 1000, NULL);

      // 데이터 수신을 위한 스레드 실행
      AfxBeginThread(m_NILayer->ThreadFunction_RECEIVE, m_NILayer);
    }
    // MAC 주소나 IP 주소가 유효하지 않은 경우
    else {
      // 다른 어댑터를 선택하라는 메시지 출력
      AfxMessageBox(_T("Select other Adapter"));
    }
  }
  // 네트워크 어댑터 선택 콤보 박스가 비활성화되어 있는 경우
  else {
    // 대상 IP 주소 입력 필드를 비활성화
    m_DstIPADDRESS.EnableWindow(FALSE);
    // IP 주소 입력 필드와 네트워크 어댑터 선택 콤보 박스를 활성화
    m_SrcIPADDRESS.EnableWindow(TRUE);
    m_ComboxAdapter.EnableWindow(TRUE);

    // 버튼의 텍스트를 Select로 변경
    CDialog::SetDlgItemTextW(IDC_BUTTON_SELECT, _T("Select"));
```

```
    // 타이머 종료
    KillTimer(1);

    // NI Layer에서 패킷 수신 상태 변경 (가능->불가능)
    m_NILayer->Receiveflip();
  }
}
```

```
// 네트워크 어댑터 정보 확정 버튼
void CARPDlg::OnBnClickedButtonSelect()
{
    // MAC 주소와 IP 주소를 저장할 CString 변수 선언
    CString MAC, IP;
    // 현재 UI(app계층)에서 MAC 주소와 IP 주소를 가져옴
    m_editSrcHwAddr.GetWindowTextW(MAC);
    m_SrcIPADDRESS.GetWindowTextW(IP);

    // 네트워크 어댑터 선택 콤보 박스가 활성화되어 있는 경우
    if (m_ComboxAdapter.IsWindowEnabled()) {

        // MAC 주소와 IP 주소가 유효한 경우
        if (MAC != DEFAULT_EDIT_TEXT && IP != "0.0.0.0") {

            // 네트워크 어댑터 선택 콤보 박스와 IP 주소 입력 필드를 비활성화
            // 대상 IP 주소 입력 필드를 활성화
            m_ComboxAdapter.EnableWindow(FALSE);
            m_SrcIPADDRESS.EnableWindow(FALSE);
            m_DstIPADDRESS.EnableWindow(TRUE);

            // NI Layer에서 패킷 수신 상태 변경 (가능<-불가능)
            m_NILayer->Receiveflip();

            // 자신의 MAC 주소와 IP 주소 설정
            m_ARPLayer->setmyAddr(MAC, IP);

            // 버튼의 텍스트를 ReSelect로 변경
            CDialog::SetDlgItemTextW(IDC_BUTTON_SELECT, _T("ReSelect"));

            // 1초마다 타이머 실행
            SetTimer(1, 1000, NULL);

            // 데이터 수신을 위한 스레드 실행
            AfxBeginThread(m_NILayer->ThreadFunction_RECEIVE, m_NILayer);
        }
    // 네트워크 어댑터 선택 콤보 박스가 비활성화되어 있는 경우
    else {
        // 대상 IP 주소 입력 필드를 비활성화
        m_DstIPADDRESS.EnableWindow(FALSE);
        // IP 주소 입력 필드와 네트워크 어댑터 선택 콤보 박스를 활성화
        m_SrcIPADDRESS.EnableWindow(TRUE);
        m_ComboxAdapter.EnableWindow(TRUE);

        // 버튼의 텍스트를 Select로 변경
        CDialog::SetDlgItemTextW(IDC_BUTTON_SELECT, _T("Select"));

        // 타이머 종료
        KillTimer(1);

        // NI Layer에서 패킷 수신 상태 변경 (가능->불가능)
        m_NILayer->Receiveflip();
    }
}
```

UI에서 목적지 IP 주소 입력하는 UI 코드

```
// 목적지 IP 주소 넣는 UI
void CARPDlg::OnIpnFieldchangedIpaddressDst(NMHDR* pNMHDR, LRESULT* pResult)
{
  LPNMIPADDRESS pIPAddr = reinterpret_cast<LPNMIPADDRESS>(pNMHDR);
  // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
  // 초기값 0
  *pResult = 0;
}
```



UI에서 ARP 요청 날리는 버튼

```
// ARP 요청 보내는 버튼
void CARPDlg::OnBnClickedButtonSendArp()
{
  // 출발지 IP 주소와 목적지 IP 주소를 저장할 배열 선언
  unsigned char srcip[IP_ADDR_SIZE] = {0,}, dstip[IP_ADDR_SIZE] = {0,};

  // 현재 UI(app 계층)에서 소스 IP 주소와 목적지 IP 주소를 가져옴
  m_SrcIPADDRESS.GetAddress(srcip[0], srcip[1], srcip[2], srcip[3]);
  m_DstIPADDRESS.GetAddress(dstip[0], dstip[1], dstip[2], dstip[3]);

  // 목적지 IP 주소 입력 필드가 활성화
  // 네트워크 어댑터 선택 콤보 박스가 비활성화면 (== 테스트 조건)
  if (m_DstIPADDRESS.IsWindowEnabled() && !m_ComboxAdapter.IsWindowEnabled()) {

    // 출발지 IP 주소와 목적지 IP 주소 설정
    // IP Layer에 전달
    m_IPLayer->SetSourceAddress(srcip);
    m_IPLayer->SetDestinAddress(dstip);

    // 소스 IP 주소와 목적지 IP 주소가 같다면
    if (memcmp(srcip, dstip, IP_ADDR_SIZE)==0) {

      // 오류 메시지를 표시하고 함수 종료
      AfxMessageBox(_T("Fail : Invalid Address"));
      return;
    }


    // 목적지 IP 주소의 각 바이트를 합한 값이 0이거나 255 * 4이면
    int check = 0;
    for (int i = 0; i < IP_ADDR_SIZE; i++) {
      check += dstip[i];
    }

    if (check == 0 || check == 255 * 4) {

      // 오류 메시지를 표시하고 함수를 종료
      AfxMessageBox(_T("Fail : Invalid Address"));
      return;
    }

    // 하위 레이어(여기서는 IP Layer)로 ARP 요청 전달
    mp_UnderLayer->Send((unsigned char*)"dummy Data", 11);
  }
  // 네트워크 어댑터가 설정되지 않았다면
  else {
    // 오류 메시지를 표시하고 함수 종료
    AfxMessageBox(_T("Fail : Set Adapter first"));
    return;
```

```
  }
}
```

```
436   // ARP 요청 보내는 버튼
437   void CARPDlg::OnBnClickedButtonSendArp()
438   {
439       // 출발지 IP 주소와 목적지 IP 주소를 저장할 배열 선언
440       unsigned char srcip[IP_ADDR_SIZE] = {0,}, dstip[IP_ADDR_SIZE] = {0,};
441
442       // 현재 UI(app 계층)에서 소스 IP 주소와 목적지 IP 주소를 가져옴
443       m_SrcIPADDRESS.GetAddress(srcip[0], srcip[1], srcip[2], srcip[3]);
444       m_DstIPADDRESS.GetAddress(dstip[0], dstip[1], dstip[2], dstip[3]);
445
446       // 목적지 IP 주소 입력 필드가 활성화
447       // 네트워크 어댑터 선택 콤보 박스가 비활성화면 (== 테스트 조건)
448       if (m_DstIPADDRESS.IsWindowEnabled() && !m_ComboxAdapter.IsWindowEnabled()) {
449
450           // 출발지 IP 주소와 목적지 IP 주소 설정
451           // IP Layer에 전달
452           m_IPLayer->SetSourceAddress(srcip);
453           m_IPLayer->SetDestinAddress(dstip);
454
455           // 소스 IP 주소와 목적지 IP 주소가 같다면
456           if (memcmp(srcip, dstip, IP_ADDR_SIZE)==0) {
457
458               // 오류 메시지를 표시하고 함수 종료
459               AfxMessageBox(_T("Fail : Invalid Address"));
460               return;
461           }
462
463
464           // 목적지 IP 주소의 각 바이트를 합한 값이 0이거나 255 * 4이면
465           int check = 0;
466           for (int i = 0; i < IP_ADDR_SIZE; i++) {
467               check += dstip[i];
468           }
469
470           if (check == 0 || check == 255 * 4) {
471
472               // 오류 메시지를 표시하고 함수를 종료
473               AfxMessageBox(_T("Fail : Invalid Address"));
474               return;
475           }
476
477           // 하위 레이어(여기서는 IP Layer)로 ARP 요청 전달
478           mp_UnderLayer->Send((unsigned char*)"dummy Data", 11);
479       }
480       // 네트워크 어댑터가 설정되지 않았다면
481       else {
482           // 오류 메시지를 표시하고 함수 종료
483           AfxMessageBox(_T("Fail : Set Adapter first"));
484           return;
485       }
486   }
```

[IP Layer]

[IPLayer.cpp]

```
#include "pch.h"
#include "stdafx.h"
#include "IPLayer.h"



// 생성자
CIPLayer::CIPLayer(char* pName) : CBaseLayer(pName){

    // 헤더 정보 초기화
    ResetHeader();

    ////////////////////////////////////
    ////////////////////////////////////
    // 굳이 IP 헤더를 다 선언할 필요는 없지만, 객체 지향으로 코드를 짜기 위함
    // 나중에 참조할 일이 있지 않을까..
    ////////////////////////////////////
    ////////////////////////////////////
```

```
    // IP 버전과 헤더 길이를 설정
    // 여기서는 IP 버전 4와 헤더 길이 20바이트를 의미하는 0x45를 설정
    m_sHeader.ver_hlegnth = 0x45;

    // 서비스 타입을 설정
    // 여기서는 기본값인 0x00을 설정
    m_sHeader.tos = 0x00;

    // 전체 패킷 길이를 설정
    // 여기서는 최대값인 0xffff를 설정
    m_sHeader.tlength = 0xffff;

    // 패킷 식별자를 설정
    // 여기서는 0x0000으로 설정
    m_sHeader.id = 0x0000;

    // 플래그와 프래그먼트 오프셋을 설정
    // 여기서는 0x00으로 설정합니다.
    m_sHeader.offset = 0x00;

    // 프레임 생존 시간을 설정
    // 여기서는 최대값인 0xff를 설정
    m_sHeader.ttl = 0xff;

    // 프로토콜 타입을 설정
    // 여기서는 TCP를 의미하는 0x06을 설정
    m_sHeader.ptype = 0x06;

    // 체크섬을 설정
    // 초기값으로 0x0000을 설정
    m_sHeader.checksum = 0x0000;
}

// 소멸자
CIPLayer::~CIPLayer(){
}

// IP 주소 초기화 함수
void CIPLayer::ResetHeader(){

    // 출발지 IP 주소 0으로 초기화
    memset(m_sHeader.ip_srcaddr, 0, IP_ADDR_SIZE);
    // 목적지 IP 주소 0으로 초기화
    memset(m_sHeader.ip_dstaddr, 0, IP_ADDR_SIZE);
    // 데이터 필드 0으로 초기화
    memset(m_sHeader.ip_data, 0, IP_MAX_DATA_SIZE);
}

// 출발지 IP 주소 반환 함수
unsigned char* CIPLayer::GetSourceAddress(){
    return m_sHeader.ip_srcaddr;
}

// 목적지 IP 주소 반환 함수
unsigned char* CIPLayer::GetDestinAddress() {
    return m_sHeader.ip_dstaddr;
}

// 출발지 IP 주소 설정 함수
void CIPLayer::SetSourceAddress(unsigned char* pAddress){
    memcpy(m_sHeader.ip_srcaddr, pAddress, IP_ADDR_SIZE);
}

// 목적지 IP 주소 설정 함수
void CIPLayer::SetDestinAddress(unsigned char* pAddress){
    memcpy(m_sHeader.ip_dstaddr, pAddress, IP_ADDR_SIZE);
}


//UpperLayer = AppLayer, UnderLayer = ARPLayer?
//
// IP 계층 패킷 전송 함수
BOOL CIPLayer::Send(unsigned char* ppayload, int nlength){

    // argument로 받은 payload를 IP data field에 복사
    memcpy(m_sHeader.ip_data, ppayload, nlength);

    // 패킷 전송 성공 여부 저장 변수 선언
    BOOL bSuccess = FALSE;

    // 하위 레이어로 패킷 전송 && 결과는 bSuccess에 저장
    bSuccess = this->GetUnderLayer()->Send((unsigned char*)&m_sHeader, IP_HEADER_SIZE + nlength);

    // 패킷 전송 성공 여부 반환
```

```
    return bSuccess;
}

// IP 계층 패킷 수신 함수
BOOL CIPLayer::Receive(unsigned char* ppayload){

    // 패킷 수신 여부 저장 변수 선언
    BOOL bSuccess = FALSE;

    // argument로 받은 payload를 IP헤더 구조체로 타입 캐스팅
    PIP_HEADER pFrame = (PIP_HEADER)ppayload;

    // 수신한 패킷의 목적지 IP 주소와 현재 레이어의 출발지 IP 주소가 같다면
    if(memcmp(pFrame->ip_dstaddr, m_sHeader.ip_srcaddr, sizeof(m_sHeader.ip_srcaddr)) == 0){

        // 상위 레이어로 패킷의 데이터를 전달 && 결과를 bSuccess에 저장
        bSuccess = mp_aUpperLayer[0]->Receive(pFrame->ip_data);
    }

    // 패킷 수신 성공 여부 반환
    return bSuccess;
}
```

```
// 생성자
CIPLayer::CIPLayer(char* pName) : CBaseLayer(pName){

    // 헤더 정보 초기화
    ResetHeader();

    ////////////////////////////////////
    ////////////////////////////////////
    // 굳이 IP 헤더를 다 선언할 필요는 없지만, 객체 지향으로 코드를 짜기 위함
    // 나중에 참조할 일이 있지 않을까..
    ////////////////////////////////////
    ////////////////////////////////////

    // IP 버전과 헤더 길이를 설정
    // 여기서는 IP 버전 4와 헤더 길이 20바이트를 의미하는 0x45를 설정
    m_sHeader.ver_hlegnth = 0x45;

    // 서비스 타입을 설정
    // 여기서는 기본값인 0x00을 설정
    m_sHeader.tos = 0x00;

    // 전체 패킷 길이를 설정
    // 여기서는 최대값인 0xffff를 설정
    m_sHeader.tlength = 0xffff;

    // 패킷 식별자를 설정
    // 여기서는 0x0000으로 설정
    m_sHeader.id = 0x0000;

    // 플래그와 프래그먼트 오프셋을 설정
    // 여기서는 0x00으로 설정합니다.
    m_sHeader.offset = 0x00;

    // 프레임 생존 시간을 설정
    // 여기서는 최대값인 0xff를 설정
    m_sHeader.ttl = 0xff;

    // 프로토콜 타입을 설정
    // 여기서는 TCP를 의미하는 0x06을 설정
    m_sHeader.ptype = 0x06;

    // 체크섬을 설정
    // 초기값으로 0x0000을 설정
    m_sHeader.checksum = 0x0000;
}

// 소멸자
CIPLayer::~CIPLayer(){
}
```

```cpp
// IP 주소 초기화 함수
void CIPLayer::ResetHeader(){

    // 출발지 IP 주소 0으로 초기화
    memset(m_sHeader.ip_srcaddr, 0, IP_ADDR_SIZE);
    // 목적지 IP 주소 0으로 초기화
    memset(m_sHeader.ip_dstaddr, 0, IP_ADDR_SIZE);
    // 데이터 필드 0으로 초기화
    memset(m_sHeader.ip_data, 0, IP_MAX_DATA_SIZE);
}

// 출발지 IP 주소 반환 함수
unsigned char* CIPLayer::GetSourceAddress(){
    return m_sHeader.ip_srcaddr;
}

// 목적지 IP 주소 반환 함수
unsigned char* CIPLayer::GetDestinAddress() {
    return m_sHeader.ip_dstaddr;
}

// 출발지 IP 주소 설정 함수
void CIPLayer::SetSourceAddress(unsigned char* pAddress){
    memcpy(m_sHeader.ip_srcaddr, pAddress, IP_ADDR_SIZE);
}

// 목적지 IP 주소 설정 함수
void CIPLayer::SetDestinAddress(unsigned char* pAddress){
    memcpy(m_sHeader.ip_dstaddr, pAddress, IP_ADDR_SIZE);
}
```

```cpp
//UpperLayer = AppLayer, UnderLayer = ARPLayer?
//
// IP 계층 패킷 전송 함수
BOOL CIPLayer::Send(unsigned char* ppayload, int nlength){

    // argument로 받은 payload를 IP data field에 복사
    memcpy(m_sHeader.ip_data, ppayload, nlength);

    // 패킷 전송 성공 여부 저장 변수 선언
    BOOL bSuccess = FALSE;

    // 하위 레이어로 패킷 전송 && 결과는 bSuccess에 저장
    bSuccess = this->GetUnderLayer()->Send((unsigned char*)&m_sHeader, IP_HEADER_SIZE + nlength);

    // 패킷 전송 성공 여부 반환
    return bSuccess;
}

// IP 계층 패킷 수신 함수
BOOL CIPLayer::Receive(unsigned char* ppayload){

    // 패킷 수신 여부 저장 변수 선언
    BOOL bSuccess = FALSE;

    // argument로 받은 payload를 IP헤더 구조체로 타입 캐스팅
    PIP_HEADER pFrame = (PIP_HEADER)ppayload;

    // 수신한 패킷의 목적지 IP 주소와 현재 레이어의 출발지 IP 주소가 같다면
    if(memcmp(pFrame->ip_dstaddr, m_sHeader.ip_srcaddr, sizeof(m_sHeader.ip_srcaddr)) == 0){

        // 상위 레이어로 패킷의 데이터를 전달 && 결과를 bSuccess에 저장
        bSuccess = mp_aUpperLayer[0]->Receive(pFrame->ip_data);
    }

    // 패킷 수신 성공 여부 반환
    return bSuccess;
}
```

[ARP Layer]

[ARPLayer.cpp]

```cpp
#include "pch.h"
#include "ARPLayer.h"


// IP 주소와 MAC 주소를 받아서 ARP 노드를 생성
CARPLayer::_ARP_NODE::_ARP_NODE(unsigned char* cipaddr, unsigned char* cenetaddr, unsigned char bincomplete = false) {
```

```
  // IP(프로토콜)주소 저장
  memcpy(prot_addr, cipaddr, IP_ADDR_SIZE);
  // MAC(하드웨어)주소 저장
  memcpy(hard_addr, cenetaddr, MAC_ADDR_SIZE);
  // status는 incomplete로 저장
  status = bincomplete;
  // ARP 테이블에 항목을 추가한 시간 저장
  spanTime = CTime::GetCurrentTime();
}

// IP 주소와 MAC 주소를 0으로 초기화한 ARP 노드를 생성
CARPLayer::_ARP_NODE::_ARP_NODE(unsigned int ipaddr = 0, unsigned int enetaddr = 0, unsigned char incomplete = false) {
  // IP 주소 초기화
  memset(prot_addr, ipaddr, IP_ADDR_SIZE);
  // MAC 주소 초기화
  memset(hard_addr, enetaddr, MAC_ADDR_SIZE);
  // status는 incomplete로 초기화
  status = incomplete;
  //ARP 테이블에 항목을 추가한 시간 저장
  spanTime = CTime::GetCurrentTime();
}

// 다른 ARP 노드 객체로부터 정보를 복사하여 ARP 노드를 생성
CARPLayer::_ARP_NODE::_ARP_NODE(const struct _ARP_NODE& ot) {
  // IP 주소 복사
  memcpy(prot_addr, ot.prot_addr, IP_ADDR_SIZE);
  //MAC 주소 복사
  memcpy(hard_addr, ot.hard_addr, MAC_ADDR_SIZE);
  // 상태 복사
  status = ot.status;
  //ARP 테이블에 항목을 추가한 시간 복사
  spanTime = ot.spanTime;
}


// ARP 헤더 초기화 inline 함수
inline void CARPLayer::ResetHeader() {
  m_sHeader = ARP_HEADER();
}


// ARP 계층 패킷 수신 함수
BOOL CARPLayer::Receive(unsigned char* ppayload) {
  PARP_HEADER arp_data = (PARP_HEADER)ppayload;

  // ARP Request 패킷을 받으면 ARP Reply 패킷을 생성하여 응답
  switch (arp_data->op) {
  case ARP_OP_REQUEST:
    // 목적지 IP 주소가 자신의 IP 주소와 같다면
    if (memcmp(arp_data->prot_dstaddr, myip, IP_ADDR_SIZE) == 0)
      // 자신의 MAC 주소를 목적지 MAC 주소로 설정합니다.
      memcpy(arp_data->hard_dstaddr, mymac, MAC_ADDR_SIZE);
    else
    // 목적지 IP 주소가 자신의 IP와 다르다면
      // ARP 테이블에서 해당 IP 주소를 찾아 MAC 주소를 목적지 MAC 주소로 설정
      for (auto& node : m_arpTable) {
        if (node == arp_data->prot_dstaddr) {
          memcpy(arp_data->hard_dstaddr, node.hard_addr, MAC_ADDR_SIZE);
          node.spanTime = CTime::GetCurrentTime();
          break;
        }
      }
    // opcode를 ARP Reply(0x0200)로 설정
    arp_data->op = ARP_OP_REPLY;
    // 출발지 주소와 목적지 주소를 swap
    swapaddr(arp_data->hard_srcaddr, arp_data->hard_dstaddr, MAC_ADDR_SIZE);
    swapaddr(arp_data->prot_srcaddr, arp_data->prot_dstaddr, IP_ADDR_SIZE);

    // ARP Reply 패킷을 하위 레이어(Ethernet Layer)로 전송
    return mp_UnderLayer->Send((unsigned char*)arp_data, ARP_HEADER_SIZE);
    break;
  //ARP Reply 패킷을 받으면 ARP 테이블 업데이트
  case ARP_OP_REPLY:
    for (auto& node : m_arpTable) {
      if (node == arp_data->prot_srcaddr) {
        memcpy(node.hard_addr, arp_data->hard_srcaddr, MAC_ADDR_SIZE);
        node.status = true;
        node.spanTime = CTime::GetCurrentTime();
        break;
      }
    }
    break;
  // ReRequest 받은 경우
  case ARP_OP_RREQUEST:
```

```
    // 아직 구현 안함
    break;
  // ReReply 받은 경우
  case ARP_OP_RREPLY:
    // 아직 구현 안함
    break;
  default:
    // 알 수 없는 opcode가 들어온 경우 에러 발생
    throw("unknown Opcode Error");
    return false;
  }

  // 패킷 수신 성공값 반환
  return true;
}

// ARP 계층 패킷 전송 함수
BOOL CARPLayer::Send(unsigned char* ppayload, int nlength) {

  // 전달받은 payload를 IP헤더 타입으로 캐스팅하여 ip data 확보
  PIP_HEADER ip_data = (PIP_HEADER)ppayload;

  // 이더넷 레이어 참조
  CEthernetLayer* m_ether = (CEthernetLayer*)mp_UnderLayer;

  // broadcast 주소 설정
  unsigned char broadcastAddr[MAC_ADDR_SIZE];
  memset(broadcastAddr, 255, MAC_ADDR_SIZE);
  m_ether->SetDestinAddress(broadcastAddr);

  // 새로운 ARP 노드 생성
  ARP_NODE newNode(ip_data->dstaddr, broadcastAddr);

  // opcode를 ARP request(0x0100)로 설정
  setOpcode(ARP_OP_REQUEST);

  int idx = inCache(ip_data->dstaddr);
  // 주어진 주소가 ARP cache table에 있는 지 확인
  if (idx != -1) {

    // 주소가 캐시에 있고 상태가 FALSE인 경우
    if (m_arpTable[idx].status == FALSE) {

      // 메시지를 출력하고 true를 반환
      AfxMessageBox(_T("Already In Cache!"));
      return true;
    }

    // 주소가 캐시에 있지만 상태가 TRUE인 경우
    else {
      // 캐시를 새 노드로 업데이트
      m_arpTable[idx] = newNode;
    }
  }
  // 주소가 캐시에 없는 경우
  else {
    // 새 노드를 캐시에 추가
    m_arpTable.push_back(newNode);
  }

  // 출발지 주소와 목적지 주소 설정
  setSrcAddr(m_ether->GetSourceAddress(), ip_data->srcaddr);
  setDstAddr(broadcastAddr, ip_data->dstaddr);

  // 패킷을 하위 레이어(여기서는 Ethernet Layer)로 전송
  return ((CEthernetLayer*)mp_UnderLayer)->Send((unsigned char*)&m_sHeader, ARP_HEADER_SIZE);

  // 패킷 전송 성공값 반환
  return true;
}

// IP 주소가 ARP cache table에 있는 지 확인하는 함수
int CARPLayer::inCache(const unsigned char* ipaddr) {
  int res = -1;
  for (int i = 0; i < m_arpTable.size(); i++) {
    if (m_arpTable[i] == ipaddr) {
      res = i;
      break;
    }
  }
  return res;
}

// ARP Layer의 MAC, IP 주소 설정 함수
```

```
void CARPLayer::setmyAddr(CString MAC, CString IP) {
  StrToaddr(ARP_IP_TYPE, myip, IP);
  StrToaddr(ARP_MAC_TYPE, mymac, MAC);
}

// ARP 헤더의 하드웨어 타입과 프로토콜 타입을 설정하는 함수
void CARPLayer::setType(const unsigned short htype, const unsigned short ptype) {

  //ARP 패킷에서 사용되는 하드웨어 및 프로토콜 유형 설정
  m_sHeader.hard_type = htype;
  m_sHeader.prot_type = ptype;

  // 하드웨어 타입에 따라 하드웨어 주소의 길이를 설정
  switch (m_sHeader.hard_type) {
  case ARP_MAC_TYPE:
    m_sHeader.hard_length = MAC_ADDR_SIZE;
    break;
  default:
    throw("Hardware Type Error!");
  }

  // 프로토콜 타입에 따라 프로토콜 주소의 길이를 설정
  switch (m_sHeader.prot_type) {
  case ARP_IP_TYPE:
    m_sHeader.prot_length = IP_ADDR_SIZE;
    break;
  default:
    throw("Protocol Type Error!");
  }
}

// ARP 헤더의 opcode를 설정하는 함수
void CARPLayer::setOpcode(const unsigned short opcode) {
  // 유효한 opcode 범위 내라면 ARP 헤더의 opcode를 설정
  if (opcode >= ARP_OP_REQUEST && opcode <= ARP_OP_RREPLY) {
    m_sHeader.op = opcode;
  }
  else
    throw("Operator code Error!");
}

// ARP 헤더의 출발지 주소를 설정하는 함수
void CARPLayer::setSrcAddr(const unsigned char enetAddr[], const unsigned char ipAddr[]) {
  memcpy(m_sHeader.hard_srcaddr, enetAddr, MAC_ADDR_SIZE);
  memcpy(m_sHeader.prot_srcaddr, ipAddr, IP_ADDR_SIZE);
}

// ARP 헤더의 목적지 주소를 설정하는 함수
void CARPLayer::setDstAddr(const unsigned char enetAddr[], const unsigned char ipAddr[]) {
  memcpy(m_sHeader.hard_dstaddr, enetAddr, MAC_ADDR_SIZE);
  memcpy(m_sHeader.prot_dstaddr, ipAddr, IP_ADDR_SIZE);
}

// 인자로 받은 주소 swap 함수
void CARPLayer::swapaddr(unsigned char lAddr[], unsigned char rAddr[], const unsigned char size) {
  unsigned char tempAddr[MAC_ADDR_SIZE] = { 0, };

  memcpy(tempAddr, lAddr, size);
  memcpy(lAddr, rAddr, size);
  memcpy(rAddr, tempAddr, size);
}

// 생성자
CARPLayer::CARPLayer(char* pName)
  : CBaseLayer(pName)
{
  // ARP 헤더 초기화
  ResetHeader();
  // ARP 헤더의 하드웨어 타입, 프로토콜 타입 설정
  setType(ARP_MAC_TYPE, ARP_IP_TYPE);
  // ip, mac 주소 초기화
  memset(myip, 0, IP_ADDR_SIZE);
  memset(mymac, 0, MAC_ADDR_SIZE);
}

// 소멸자
CARPLayer::~CARPLayer() {

}

// 아래의 연산자 오버로딩들은 ARP 노드 간의 비교를 수행
// IP 주소를 기준으로 비교
bool CARPLayer::_ARP_NODE::operator==(const unsigned char* ipaddr) {
  return memcmp(prot_addr, ipaddr, IP_ADDR_SIZE) == 0;
```

```
}
bool CARPLayer::_ARP_NODE::operator==(const struct _ARP_NODE& ot) {
  return *this == ot.prot_addr;
}
bool CARPLayer::_ARP_NODE::operator<(const unsigned char* ipaddr) {
  return memcmp(prot_addr, ipaddr, IP_ADDR_SIZE) == -1;
}
bool CARPLayer::_ARP_NODE::operator<(const struct _ARP_NODE& ot) {
  return *this < ot.prot_addr;
}
bool CARPLayer::_ARP_NODE::operator>(const unsigned char* ipaddr) {
  return memcmp(prot_addr, ipaddr, IP_ADDR_SIZE) == 1;
}
bool CARPLayer::_ARP_NODE::operator>(const struct _ARP_NODE& ot) {
  return *this > ot.prot_addr;
}

// ARP 헤더 초기화
CARPLayer::_ARP_HEADER::_ARP_HEADER() {
  hard_type = prot_type = 0x0000;
  hard_length = prot_length = 0x00;
  op = 0x0000;
  memset(hard_srcaddr, 0x00, MAC_ADDR_SIZE);
  memset(prot_srcaddr, 0x00, IP_ADDR_SIZE);
  memset(hard_dstaddr, 0x00, MAC_ADDR_SIZE);
  memset(prot_dstaddr, 0x00, IP_ADDR_SIZE);
};

// 인자로 받은 ARP 헤더에서 정보를 복사하여 ARP 헤더 생성
CARPLayer::_ARP_HEADER::_ARP_HEADER(const struct _ARP_HEADER& ot)
  :hard_type(ot.hard_type), prot_type(ot.prot_type),
  hard_length(ot.hard_length), prot_length(ot.prot_length),
  op(ot.op)
{
  memcpy(hard_srcaddr, ot.hard_srcaddr, MAC_ADDR_SIZE);
  memcpy(hard_dstaddr, ot.hard_dstaddr, MAC_ADDR_SIZE);
  memcpy(prot_srcaddr, ot.prot_srcaddr, IP_ADDR_SIZE);
  memcpy(prot_dstaddr, ot.prot_dstaddr, IP_ADDR_SIZE);
}

// 주소를 문자열로 변환하는 함수
void addrToStr(unsigned short type, CString& dst, unsigned char* src) {
  switch (type) {
  // 주소 타입이 IP인 경우
  case ARP_IP_TYPE:
    // IP 주소 형식(0.0.0.0)으로 문자열 생성
    dst.Format(_T("%hhu.%hhu.%hhu.%hhu"),
      src[0], src[1], src[2], src[3]);
    break;
  // 주소 타입이 MAC인 경우
  case ARP_MAC_TYPE:
    // MAC 주소 형식(00:00 : 00 : 00 : 00 : 00)으로 문자열 생성
    dst.Format(_T("%02hhx:%02hhx:%02hhx:%02hhx:%02hhx:%02hhx"),
      src[0], src[1], src[2], src[3], src[4], src[5]);
    break;
  default:
    break;
  }
}

// 문자열을 주소로 포맷팅하는 함수
void StrToaddr(unsigned short type, unsigned char* dst, CString& src) {
  switch (type) {
  // 주소 타입이 IP인 경우
  case ARP_IP_TYPE:
    // IP 주소 형식(0.0.0.0)의 문자열을 IP 주소로 변환
    swscanf_s(src, _T("%hhu.%hhu.%hhu.%hhu"),
      &dst[0], &dst[1], &dst[2], &dst[3]);
    break;
  // 주소 타입이 MAC인 경우
  case ARP_MAC_TYPE:
    // MAC 주소 형식(00:00 : 00 : 00 : 00 : 00)의 문자열을 MAC 주소로 변환
    swscanf_s(src, _T("%hhx:%hhx:%hhx:%hhx:%hhx:%hhx"),
      &dst[0], &dst[1], &dst[2], &dst[3], &dst[4], &dst[5]);
    break;
  default:
    break;
  }
}

// ARP cache table 업데이트 함수
void CARPLayer::updateTable() {
  CTime cur = CTime::GetCurrentTime();
  for (int i = 0; i < m_arpTable.size(); i++) {
```

```
    if (m_arpTable[i].status == ARP_TIME_OUT) continue;

    if ((cur - m_arpTable[i].spanTime) > (m_arpTable[i].status == TRUE ? 600 : 180)) {
      m_arpTable[i].status = ARP_TIME_OUT;
    }
  }
}

// 인자로 IP 주소를 가진 항목을 ARP cache table에서 삭제하는 함수
void CARPLayer::deleteItem(CString IP) {
  auto k = m_arpTable.begin();
  unsigned char addr[IP_ADDR_SIZE] = { 0, };
  StrToaddr(ARP_IP_TYPE, addr, IP);

  for (; k != m_arpTable.end(); k++) {
    if (*k == addr) {
      break;
    }
  }

  m_arpTable.erase(k);
}

// 현재 ARP cache table 반환 함수
std::vector<CARPLayer::ARP_NODE> CARPLayer::getTable() {
  return m_arpTable;
}

// ARP cache table 비우는 함수
void CARPLayer::clearTable() {
  m_arpTable.clear();
}
```

```cpp
#include "pch.h"
#include "ARPLayer.h"


// IP 주소와 MAC 주소를 받아서 ARP 노드를 생성
CARPLayer::_ARP_NODE::_ARP_NODE(unsigned char* cipaddr, unsigned char* cenetaddr, unsigned char bincomplete = false) {
    // IP(프로토콜)주소 저장
    memcpy(prot_addr, cipaddr, IP_ADDR_SIZE);
    // MAC(하드웨어)주소 저장
    memcpy(hard_addr, cenetaddr, MAC_ADDR_SIZE);
    // status는 incomplete로 저장
    status = bincomplete;
    // ARP 테이블에 항목을 추가한 시간 저장
    spanTime = CTime::GetCurrentTime();
}

// IP 주소와 MAC 주소를 0으로 초기화한 ARP 노드를 생성
CARPLayer::_ARP_NODE::_ARP_NODE(unsigned int ipaddr = 0, unsigned int enetaddr = 0, unsigned char incomplete = false) {
    // IP 주소 초기화
    memset(prot_addr, ipaddr, IP_ADDR_SIZE);
    // MAC 주소 초기화
    memset(hard_addr, enetaddr, MAC_ADDR_SIZE);
    // status는 incomplete로 초기화
    status = incomplete;
    //ARP 테이블에 항목을 추가한 시간 저장
    spanTime = CTime::GetCurrentTime();
}

// 다른 ARP 노드 객체로부터 정보를 복사하여 ARP 노드를 생성
CARPLayer::_ARP_NODE::_ARP_NODE(const struct _ARP_NODE& ot) {
    // IP 주소 복사
    memcpy(prot_addr, ot.prot_addr, IP_ADDR_SIZE);
    //MAC 주소 복사
    memcpy(hard_addr, ot.hard_addr, MAC_ADDR_SIZE);
    // 상태 복사
    status = ot.status;
    //ARP 테이블에 항목을 추가한 시간 복사
    spanTime = ot.spanTime;
}


// ARP 헤더 초기화 inline 함수
inline void CARPLayer::ResetHeader() {
    m_sHeader = ARP_HEADER();
}
```

```cpp
// ARP 계층 패킷 수신 함수
BOOL CARPLayer::Receive(unsigned char* ppayload) {
    PARP_HEADER arp_data = (PARP_HEADER)ppayload;

    // ARP Request 패킷을 받으면 ARP Reply 패킷을 생성하여 응답
    switch (arp_data->op) {
    case ARP_OP_REQUEST:
        // 목적지 IP 주소가 자신의 IP 주소와 같다면
        if (memcmp(arp_data->prot_dstaddr, myip, IP_ADDR_SIZE) == 0)
            // 자신의 MAC 주소를 목적지 MAC 주소로 설정합니다.
            memcpy(arp_data->hard_dstaddr, mymac, MAC_ADDR_SIZE);
        else
        // 목적지 IP 주소가 자신의 IP와 다르다면
            // ARP 테이블에서 해당 IP 주소를 찾아 MAC 주소를 목적지 MAC 주소로 설정
            for (auto& node : m_arpTable) {
                if (node == arp_data->prot_dstaddr) {
                    memcpy(arp_data->hard_dstaddr, node.hard_addr, MAC_ADDR_SIZE);
                    node.spanTime = CTime::GetCurrentTime();
                    break;
                }
            }
        // opcode를 ARP Reply(0x0200)로 설정
        arp_data->op = ARP_OP_REPLY;
        // 출발지 주소와 목적지 주소를 swap
        swapaddr(arp_data->hard_srcaddr, arp_data->hard_dstaddr, MAC_ADDR_SIZE);
        swapaddr(arp_data->prot_srcaddr, arp_data->prot_dstaddr, IP_ADDR_SIZE);

        // ARP Reply 패킷을 하위 레이어(Ethernet Layer)로 전송
        return mp_UnderLayer->Send((unsigned char*)arp_data, ARP_HEADER_SIZE);
        break;
    //ARP Reply 패킷을 받으면 ARP 테이블 업데이트
    case ARP_OP_REPLY:
        for (auto& node : m_arpTable) {
            if (node == arp_data->prot_srcaddr) {
                memcpy(node.hard_addr, arp_data->hard_srcaddr, MAC_ADDR_SIZE);
                node.status = true;
                node.spanTime = CTime::GetCurrentTime();
                break;
            }
        }
        break;
    // ReRequest 받은 경우
    case ARP_OP_RREQUEST:
        // 아직 구현 안함
        break;
    // ReReply 받은 경우
    case ARP_OP_RREPLY:
        // 아직 구현 안함
        break;
    default:
        // 알 수 없는 opcode가 들어온 경우 에러 발생
        throw("unknown Opcode Error");
        return false;
    }

    // 패킷 수신 성공값 반환
    return true;
}
```

```
108 BOOL CARPLayer::Send(unsigned char* ppayload, int nlength) {
109
110     // 전달받은 payload를 IP헤더 타입으로 캐스팅하여 ip data 확보
111     PIP_HEADER ip_data = (PIP_HEADER)ppayload;
112
113     // 이더넷 레이어 참조
114     CEthernetLayer* m_ether = (CEthernetLayer*)mp_UnderLayer;
115
116     // broadcast 주소 설정
117     unsigned char broadcastAddr[MAC_ADDR_SIZE];
118     memset(broadcastAddr, 255, MAC_ADDR_SIZE);
119     m_ether->SetDestinAddress(broadcastAddr);
120
121     // 새로운 ARP 노드 생성
122     ARP_NODE newNode(ip_data->dstaddr, broadcastAddr);
123
124     // opcode를 ARP request(0x0100)로 설정
125     setOpcode(ARP_OP_REQUEST);
126
127     int idx = inCache(ip_data->dstaddr);
128     // 주어진 주소가 ARP cache table에 있는 지 확인
129     if (idx != -1) {
130
131         // 주소가 캐시에 있고 상태가 FALSE인 경우
132         if (m_arpTable[idx].status == FALSE) {
133
134             // 메시지를 출력하고 true를 반환
135             AfxMessageBox(_T("Already In Cache!"));
136             return true;
137         }
138
139         // 주소가 캐시에 있지만 상태가 TRUE인 경우
140         else {
141             // 캐시를 새 노드로 업데이트
142             m_arpTable[idx] = newNode;
143         }
144     }
145     // 주소가 캐시에 없는 경우
146     else {
147         // 새 노드를 캐시에 추가
148         m_arpTable.push_back(newNode);
149     }
150
151     // 출발지 주소와 목적지 주소 설정
152     setSrcAddr(m_ether->GetSourceAddress(), ip_data->srcaddr);
153     setDstAddr(broadcastAddr, ip_data->dstaddr);
154
155     // 패킷을 하위 레이어(여기서는 Ethernet Layer)로 전송
156     return ((CEthernetLayer*)mp_UnderLayer)->Send((unsigned char*)&m_sHeader, ARP_HEADER_SIZE);
157
158     // 패킷 전송 성공값 반환
159     return true;
160 }
```

```
// ARP Layer의 MAC, IP 주소 설정 함수
void CARPLayer::setmyAddr(CString MAC, CString IP) {
    StrToaddr(ARP_IP_TYPE, myip, IP);
    StrToaddr(ARP_MAC_TYPE, mymac, MAC);
}

// ARP 헤더의 하드웨어 타입과 프로토콜 타입을 설정하는 함수
void CARPLayer::setType(const unsigned short htype, const unsigned short ptype) {

    //ARP 패킷에서 사용되는 하드웨어 및 프로토콜 유형 설정
    m_sHeader.hard_type = htype;
    m_sHeader.prot_type = ptype;

    // 하드웨어 타입에 따라 하드웨어 주소의 길이를 설정
    switch (m_sHeader.hard_type) {
    case ARP_MAC_TYPE:
        m_sHeader.hard_length = MAC_ADDR_SIZE;
        break;
    default:
        throw("Hardware Type Error!");
    }

    // 프로토콜 타입에 따라 프로토콜 주소의 길이를 설정
    switch (m_sHeader.prot_type) {
    case ARP_IP_TYPE:
        m_sHeader.prot_length = IP_ADDR_SIZE;
        break;
    default:
        throw("Protocol Type Error!");
    }
}

// ARP 헤더의 opcode를 설정하는 함수
void CARPLayer::setOpcode(const unsigned short opcode) {
    // 유효한 opcode 범위 내라면 ARP 헤더의 opcode를 설정
    if (opcode >= ARP_OP_REQUEST && opcode <= ARP_OP_RREPLY) {
        m_sHeader.op = opcode;
    }
    else
        throw("Operator code Error!");
}

// ARP 헤더의 출발지 주소를 설정하는 함수
void CARPLayer::setSrcAddr(const unsigned char enetAddr[], const unsigned char ipAddr[]) {
    memcpy(m_sHeader.hard_srcaddr, enetAddr, MAC_ADDR_SIZE);
    memcpy(m_sHeader.prot_srcaddr, ipAddr, IP_ADDR_SIZE);
}
```

```
// ARP 헤더의 목적지 주소를 설정하는 함수
void CARPLayer::setDstAddr(const unsigned char enetAddr[], const unsigned char ipAddr[]) {
    memcpy(m_sHeader.hard_dstaddr, enetAddr, MAC_ADDR_SIZE);
    memcpy(m_sHeader.prot_dstaddr, ipAddr, IP_ADDR_SIZE);
}

// 인자로 받은 주소 swap 함수
void CARPLayer::swapaddr(unsigned char lAddr[], unsigned char rAddr[], const unsigned char size) {
    unsigned char tempAddr[MAC_ADDR_SIZE] = { 0, };

    memcpy(tempAddr, lAddr, size);
    memcpy(lAddr, rAddr, size);
    memcpy(rAddr, tempAddr, size);
}

// 생성자
CARPLayer::CARPLayer(char* pName)
    : CBaseLayer(pName)
{
    // ARP 헤더 초기화
    ResetHeader();
    // ARP 헤더의 하드웨어 타입, 프로토콜 타입 설정
    setType(ARP_MAC_TYPE, ARP_IP_TYPE);
    // ip, mac 주소 초기화
    memset(myip, 0, IP_ADDR_SIZE);
    memset(mymac, 0, MAC_ADDR_SIZE);
}

// 소멸자
CARPLayer::~CARPLayer() {

}
```

```cpp

// 아래의 연산자 오버로딩들은 ARP 노드 간의 비교를 수행
// IP 주소를 기준으로 비교
bool CARPLayer::_ARP_NODE::operator==(const unsigned char* ipaddr) {
    return memcmp(prot_addr, ipaddr, IP_ADDR_SIZE) == 0;
}
bool CARPLayer::_ARP_NODE::operator==(const struct _ARP_NODE& ot) {
    return *this == ot.prot_addr;
}
bool CARPLayer::_ARP_NODE::operator<(const unsigned char* ipaddr) {
    return memcmp(prot_addr, ipaddr, IP_ADDR_SIZE) == -1;
}
bool CARPLayer::_ARP_NODE::operator<(const struct _ARP_NODE& ot) {
    return *this < ot.prot_addr;
}
bool CARPLayer::_ARP_NODE::operator>(const unsigned char* ipaddr) {
    return memcmp(prot_addr, ipaddr, IP_ADDR_SIZE) == 1;
}
bool CARPLayer::_ARP_NODE::operator>(const struct _ARP_NODE& ot) {
    return *this > ot.prot_addr;
}

// ARP 헤더 초기화
CARPLayer::_ARP_HEADER::_ARP_HEADER() {
    hard_type = prot_type = 0x0000;
    hard_length = prot_length = 0x00;
    op = 0x0000;
    memset(hard_srcaddr, 0x00, MAC_ADDR_SIZE);
    memset(prot_srcaddr, 0x00, IP_ADDR_SIZE);
    memset(hard_dstaddr, 0x00, MAC_ADDR_SIZE);
    memset(prot_dstaddr, 0x00, IP_ADDR_SIZE);
};

// 인자로 받은 ARP 헤더에서 정보를 복사하여 ARP 헤더 생성
CARPLayer::_ARP_HEADER::_ARP_HEADER(const struct _ARP_HEADER& ot)
    :hard_type(ot.hard_type), prot_type(ot.prot_type),
    hard_length(ot.hard_length), prot_length(ot.prot_length),
    op(ot.op)
{
    memcpy(hard_srcaddr, ot.hard_srcaddr, MAC_ADDR_SIZE);
    memcpy(hard_dstaddr, ot.hard_dstaddr, MAC_ADDR_SIZE);
    memcpy(prot_srcaddr, ot.prot_srcaddr, IP_ADDR_SIZE);
    memcpy(prot_dstaddr, ot.prot_dstaddr, IP_ADDR_SIZE);
}
```

```cpp
void addrToStr(unsigned short type, CString& dst, unsigned char* src) {
    switch (type) {
    // 주소 타입이 IP인 경우
    case ARP_IP_TYPE:
        // IP 주소 형식(0.0.0.0)으로 문자열 생성
        dst.Format(_T("%hhu.%hhu.%hhu.%hhu"),
            src[0], src[1], src[2], src[3]);
        break;
    // 주소 타입이 MAC인 경우
    case ARP_MAC_TYPE:
        // MAC 주소 형식(00:00 : 00 : 00 : 00 : 00)으로 문자열 생성
        dst.Format(_T("%02hhx:%02hhx:%02hhx:%02hhx:%02hhx:%02hhx"),
            src[0], src[1], src[2], src[3], src[4], src[5]);
        break;
    default:
        break;
    }
}

// 문자열을 주소로 포맷팅하는 함수
void StrToaddr(unsigned short type, unsigned char* dst, CString& src) {
    switch (type) {
    // 주소 타입이 IP인 경우
    case ARP_IP_TYPE:
        // IP 주소 형식(0.0.0.0)의 문자열을 IP 주소로 변환
        swscanf_s(src, _T("%hhu.%hhu.%hhu.%hhu"),
            &dst[0], &dst[1], &dst[2], &dst[3]);
        break;
    // 주소 타입이 MAC인 경우
    case ARP_MAC_TYPE:
        // MAC 주소 형식(00:00 : 00 : 00 : 00 : 00)의 문자열을 MAC 주소로 변환
        swscanf_s(src, _T("%hhx:%hhx:%hhx:%hhx:%hhx:%hhx"),
            &dst[0], &dst[1], &dst[2], &dst[3], &dst[4], &dst[5]);
        break;
    default:
        break;
    }
}

// ARP cache table 업데이트 함수
void CARPLayer::updateTable() {
    CTime cur = CTime::GetCurrentTime();
    for (int i = 0; i < m_arpTable.size(); i++) {
        if (m_arpTable[i].status == ARP_TIME_OUT) continue;

        if ((cur - m_arpTable[i].spanTime) > (m_arpTable[i].status == TRUE ? 600 : 180)) {
            m_arpTable[i].status = ARP_TIME_OUT;
        }
    }
}
```

```cpp
// 인자로 IP 주소를 가진 항목을 ARP cache table에서 삭제하는 함수
void CARPLayer::deleteItem(CString IP) {
    auto k = m_arpTable.begin();
    unsigned char addr[IP_ADDR_SIZE] = { 0, };
    StrToaddr(ARP_IP_TYPE, addr, IP);

    for (; k != m_arpTable.end(); k++) {
        if (*k == addr) {
            break;
        }
    }

    m_arpTable.erase(k);
}

// 현재 ARP cache table 반환 함수
std::vector<CARPLayer::ARP_NODE> CARPLayer::getTable() {
    return m_arpTable;
}

// ARP cache table 비우는 함수
void CARPLayer::clearTable() {
    m_arpTable.clear();
}
```

[Ethernet Layer]

[EthernetLayer.cpp]

```
#include "stdafx.h"
#include "pch.h"
#include "EthernetLayer.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

// 생성자
CEthernetLayer::CEthernetLayer(char* pName)
  : CBaseLayer(pName)
{
  // 이더넷 헤더 초기화
  ResetHeader();
}

// 소멸자
CEthernetLayer::~CEthernetLayer()
{
}

// 이더넷 헤더 초기화 함수
void CEthernetLayer::ResetHeader()
{
  memset(m_sHeader.mac_dstaddr, 0, 6);
  memset(m_sHeader.mac_srcaddr, 0, 6);
  memset(m_sHeader.mac_data, MAC_MAX_DATA_SIZE, 6);
  m_sHeader.mac_type = 0;
}

// 출발지 MAC 주소 반환 함수
unsigned char* CEthernetLayer::GetSourceAddress()
{
  return m_sHeader.mac_srcaddr;
}

// 목적지 MAC 주소 반환 함수
unsigned char* CEthernetLayer::GetDestinAddress()
{

  return m_sHeader.mac_dstaddr;

}

// 출발지 MAC 주소 설정 함수
void CEthernetLayer::SetSourceAddress(unsigned char* pAddress)
{

  // 매개변수로 넘겨받은 source 주소를 Ethernet source주소로 지정
  memcpy(m_sHeader.mac_srcaddr, pAddress, 6);

}

// 목적지 MAC 주소 설정 함수
void CEthernetLayer::SetDestinAddress(unsigned char* pAddress)
{
  // 매개변수로 넘겨받은 destination 주소를 Ethernet dest주소로 지정
  memcpy(m_sHeader.mac_dstaddr, pAddress, 6);
}

// 이더넷 계층 패킷 전송 함수
BOOL CEthernetLayer::Send(unsigned char* ppayload, int nlength)
{
  // 윗 계층에서 받은 Frame 길이만큼을 Ethernet계층의 data로 넣음
  memcpy(m_sHeader.mac_data, ppayload, nlength);
  // 윗 계층에서 받은 type 또한 헤더에 포함
  m_sHeader.mac_type = MAC_ARP_TYPE;

  BOOL bSuccess = FALSE;

  // 이더넷 데이터와 이더넷 헤더의 사이즈를 합한 크기만큼의 이더넷 프레임을 하위 계층으로 전송
  bSuccess = this->GetUnderLayer()->Send((unsigned char*)&m_sHeader, MAC_HEADER_SIZE + nlength);

  return bSuccess;
}

// 이더넷 계층 패킷 수신 함수
BOOL CEthernetLayer::Receive(unsigned char* ppayload)
{
  // 이더넷 헤더를 받아서 이더넷 헤더 구조체로 타입 캐스팅
  PETHERNET_HEADER pFrame = (PETHERNET_HEADER)ppayload;
```

```
  BOOL bSuccess = FALSE;

  // 목적지 주소를 확인
  if(memcmp(pFrame->mac_dstaddr, m_sHeader.mac_srcaddr, sizeof(m_sHeader.mac_srcaddr))==0){//주소 확인

    // enet_type을 기준으로 이더넷 프레임의 데이터를 넘겨줄 레이어 지정

    // IP 타입의 경우, 상위 계층(IP Layer)으로 데이터 전달
    if (pFrame->mac_type == MAC_IP_TYPE)
      bSuccess = mp_aUpperLayer[0]->Receive(pFrame->mac_data);
    // ARP 타입의 경우, 상위 계층(ARP Layer)으로 데이터 전달
    else if(pFrame->mac_type == MAC_ARP_TYPE)
      bSuccess = mp_aUpperLayer[1]->Receive(pFrame->mac_data);
  }

  return bSuccess;
}
```

```
// 생성자
CEthernetLayer::CEthernetLayer(char* pName)
    : CBaseLayer(pName)
{
    // 이더넷 헤더 초기화
    ResetHeader();
}

// 소멸자
CEthernetLayer::~CEthernetLayer()
{
}

// 이더넷 헤더 초기화 함수
void CEthernetLayer::ResetHeader()
{
    memset(m_sHeader.mac_dstaddr, 0, 6);
    memset(m_sHeader.mac_srcaddr, 0, 6);
    memset(m_sHeader.mac_data, MAC_MAX_DATA_SIZE, 6);
    m_sHeader.mac_type = 0;
}

// 출발지 MAC 주소 반환 함수
unsigned char* CEthernetLayer::GetSourceAddress()
{
    return m_sHeader.mac_srcaddr;
}

// 목적지 MAC 주소 반환 함수
unsigned char* CEthernetLayer::GetDestinAddress()
{

    return m_sHeader.mac_dstaddr;

}

// 출발지 MAC 주소 설정 함수
void CEthernetLayer::SetSourceAddress(unsigned char* pAddress)
{

    // 매개변수로 넘겨받은 source 주소를 Ethernet source주소로 지정
    memcpy(m_sHeader.mac_srcaddr, pAddress, 6);

}

// 목적지 MAC 주소 설정 함수
void CEthernetLayer::SetDestinAddress(unsigned char* pAddress)
{
    // 매개변수로 넘겨받은 destination 주소를 Ethernet dest주소로 지정
    memcpy(m_sHeader.mac_dstaddr, pAddress, 6);
}
```

```
64  BOOL CEthernetLayer::Send(unsigned char* ppayload, int nlength)
65  {
66      // 윗 계층에서 받은 Frame 길이만큼을 Ethernet계층의 data로 넣음
67      memcpy(m_sHeader.mac_data, ppayload, nlength);
68      // 윗 계층에서 받은 type 또한 헤더에 포함
69      m_sHeader.mac_type = MAC_ARP_TYPE;
70
71      BOOL bSuccess = FALSE;
72
73      // 이더넷 데이터와 이더넷 헤더의 사이즈를 합한 크기만큼의 이더넷 프레임을 하위 계층으로 전송
74      bSuccess = this->GetUnderLayer()->Send((unsigned char*)&m_sHeader, MAC_HEADER_SIZE + nlength);
75
76      return bSuccess;
77  }
78
79  // 이더넷 계층 패킷 수신 함수
80  BOOL CEthernetLayer::Receive(unsigned char* ppayload)
81  {
82      // 이더넷 헤더를 받아서 이더넷 헤더 구조체로 타입 캐스팅
83      PETHERNET_HEADER pFrame = (PETHERNET_HEADER)ppayload;
84
85      BOOL bSuccess = FALSE;
86
87      // 목적지 주소를 확인
88      if(memcmp(pFrame->mac_dstaddr, m_sHeader.mac_srcaddr, sizeof(m_sHeader.mac_srcaddr))==0){//주소 확인
89
90          // enet_type을 기준으로 이더넷 프레임의 데이터를 넘겨줄 레이어 지정
91
92          // IP 타입의 경우, 상위 계층(IP Layer)으로 데이터 전달
93          if (pFrame->mac_type == MAC_IP_TYPE)
94              bSuccess = mp_aUpperLayer[0]->Receive(pFrame->mac_data);
95          // ARP 타입의 경우, 상위 계층(ARP Layer)으로 데이터 전달
96          else if(pFrame->mac_type == MAC_ARP_TYPE)
97              bSuccess = mp_aUpperLayer[1]->Receive(pFrame->mac_data);
98      }
99
100     return bSuccess;
101 }
102
```

[NI Layer]

[NILayer.cpp]

```
// NetworkInterfaceLayer.cpp: implementation of the NetworkInterfaceLayer class.

#include "stdafx.h"
#include "pch.h"
#include "NILayer.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif


// 생성자
CNILayer::CNILayer(char* pName)
  : CBaseLayer(pName), device(NULL) {
  char errbuf[PCAP_ERRBUF_SIZE];
  m_AdapterObject = nullptr;
  canRead = false;
  adapter = nullptr;
  OidData = nullptr;
  memset(data, 0, MAC_MAX_SIZE);
  try {
    // 모든 네트워크 장치를 검색
    if (pcap_findalldevs(&allDevices, errbuf) == -1)
    {
      allDevices = NULL;

      // 장치를 찾는 데 실패하면 예외 발생
      throw(CString(errbuf));
    }
  }
  // 에러 발생 시
  catch (CString errorInfo) {

    // 에러 메시지를 출력하고 함수를 종료
    AfxMessageBox(errorInfo);
    return;
```

```
  }

  try {
    // OidData에 메모리를 할당
    OidData = (PPACKET_OID_DATA)malloc(sizeof(PACKET_OID_DATA));
    if (OidData == nullptr) {
      // 메모리 할당에 실패하면 예외를 발생
      throw(CString("MALLOC FAIL"));
    }
    OidData->Oid = 0x01010101;
    OidData->Length = 6;
    m_AdapterObject = nullptr;
    memset(data, 0, MAC_MAX_SIZE);
  }
  // 에러 발생 시
  catch (CString errorInfo) {
    // 에러 메시지 출력하고 함수 종료
    AfxMessageBox(errorInfo);
    return;
  }
}

// 소멸자
CNILayer::~CNILayer() {
  pcap_if_t* temp = nullptr;

  // allDevices에 할당한 메모리를 해제
  while (allDevices) {
    temp = allDevices;
    allDevices = allDevices->next;
    delete(temp);
  }

  // OidData에 할당한 메모리를 해제
  delete(OidData);
}

// network interface layer에서 패킷 수신 함수
BOOL CNILayer::Receive(unsigned char* pkt) {

  // 패킷이 null인 경우 FALSE 반환 (Host끼리 통신 시 사용)
  if (pkt == nullptr) {
    return FALSE;
  }

  // 상위 계층에서 패킷을 받지 못한 경우 FALSE 반환 (Etherent Layer에서 패킷 받아올 때 사용)
  if (!(mp_aUpperLayer[0]->Receive(pkt))) {
    return FALSE;
  }

  // 수신 성공 시 TRUE 반환
  return TRUE;
}

// network interface layer에서 패킷 전송 함수
BOOL CNILayer::Send(unsigned char* ppayload, int nlength) {

  // 패킷 전송
  if (pcap_sendpacket(m_AdapterObject, ppayload, nlength))
  {
    // 실패 시 에러 메시지 출력
    AfxMessageBox(_T("Fail: Send Packet!"));
    // FALSE 반환
    return FALSE;
  }

  // 전송 성공 시 TRUE 반환
  return TRUE;
}

// 네트워크 어뎁터 목록을 콤보박스에 설정하는 함수
void CNILayer::SetAdapterComboBox(CComboBox& adapterlist) {
  // 모든 네트워크 어뎁터를 순회하며
  for (pcap_if_t* d = allDevices; d; d = d->next) {
    //각 장치의 설명을 콤보 박스에 추가
    adapterlist.AddString(CString(d->description));
  }
}

// 선택한 네트워크 어뎁터를 설정하는 함수
UCHAR* CNILayer::SetAdapter(const int index) {
  char errbuf[PCAP_ERRBUF_SIZE];
  device = allDevices;
```

```
// 이미 열려있는 어뎁터가 있다면 닫음
if (m_AdapterObject != nullptr)
  pcap_close(m_AdapterObject);

// 입력된 인덱스만큼 네트워크 어뎁터 목록을 순회
for (int i = 0; i < index && device; i++) {
  device = device->next;
}
// 해당 장치를 열음
if (device != nullptr)
  m_AdapterObject = pcap_open_live((const char*)device->name, 65536, 0, 1000, errbuf);
// 장치를 열지 못했다면 에러 메시지를 출력하고 nullptr를 반환
if (m_AdapterObject == nullptr)
{
  AfxMessageBox(_T("Fail: Connect Adapter"));
  return nullptr;
}

// 어뎁터의 MAC 주소를 요청
adapter = PacketOpenAdapter(device->name);
PacketRequest(adapter, FALSE, OidData);

// 어뎁터를 닫음
PacketCloseAdapter(adapter);
//AfxBeginThread(ThreadFunction_RECEIVE, this);
```

```
  // MAC 주소 반환
  return (OidData->Data);
}

// 네트워크 장치의 MAC 주소 목록을 가져오는 함수
void CNILayer::GetMacAddressList(CStringArray& adapterlist) {
  for (pcap_if_t* d = allDevices; d; d = d->next) {
    adapterlist.Add(CString(d->description));
  }
}


// 검색
// 선택한 네트워크 어댑터의 IP 주소를 가져오는 함수
// 매개변수로 ipv4 ipv6 값을 저장할 문자열을 받음
// 네트워크 어댑터를 먼저 설정해야 함
void CNILayer::GetIPAddress(CString& ipv4addr, CString& ipv6addr) {

  // IP 주소를 저장하기 위한 문자열 배열을 선언
  char ip[IPV6_ADDR_STR_LEN];
  // 초기값으로 ipv4addr, ipv6addr를 설정
  ipv4addr = DEFAULT_EDIT_TEXT;
  ipv6addr = DEFAULT_EDIT_TEXT;

  // 장치의 모든 주소를 순회
  for (auto addr = device->addresses; addr != nullptr; addr = addr->next)
  {
    // 실제 주소를 저장
    auto realaddr = addr->addr;
    // 주소의 패밀리 타입을 가져옴
    const int sa_family = realaddr->sa_family;
```

```
    // 주소를 문자열로 변환하여 ptr에 저장
    const char* ptr = inet_ntop(sa_family, &realaddr->sa_data[sa_family == AF_INET ? 2 : 6], ip, IPV6_ADDR_STR_LEN);

    // 패밀리 타입에 따라 ipv4addr 또는 ipv6addr에 주소를 저장
    switch (sa_family)
    {
    case AF_INET:
      ipv4addr = ptr;
      break;
    case AF_INET6:
      ipv6addr = ptr;
      break;
    default:
      return;
    }
  }
}

// 패킷 수신하는 스레드 함수
UINT CNILayer::ThreadFunction_RECEIVE(LPVOID pParam) {
  struct pcap_pkthdr* header;
  const u_char* pkt_data;
  CNILayer* pNI = (CNILayer*)pParam;

  int result = 1;
```

```
  while (pNI->canRead)
  {
    result = pcap_next_ex(pNI->m_AdapterObject, &header, &pkt_data);
    if (result == 1) {
      memcpy(pNI->data, pkt_data, MAC_MAX_SIZE);
      pNI->Receive(pNI->data);
    }
  }
  return 0;
}

// NI Layer 패킷 수신 상태 변경 함수
void CNILayer::Receiveflip() {
  canRead = !canRead;
}
```

```
// 생성자
CNILayer::CNILayer(char* pName)
    : CBaseLayer(pName), device(NULL) {
    char errbuf[PCAP_ERRBUF_SIZE];
    m_AdapterObject = nullptr;
    canRead = false;
    adapter = nullptr;
    OidData = nullptr;
    memset(data, 0, MAC_MAX_SIZE);
    try {
        // 모든 네트워크 장치를 검색
        if (pcap_findalldevs(&allDevices, errbuf) == -1)
        {
            allDevices = NULL;

            // 장치를 찾는 데 실패하면 예외 발생
            throw(CString(errbuf));
        }
    }
    // 에러 발생 시
    catch (CString errorInfo) {

        // 에러 메시지를 출력하고 함수를 종료
        AfxMessageBox(errorInfo);
        return;
    }

    try {
        // OidData에 메모리를 할당
        OidData = (PPACKET_OID_DATA)malloc(sizeof(PACKET_OID_DATA));
        if (OidData == nullptr) {
            // 메모리 할당에 실패하면 예외를 발생
            throw(CString("MALLOC FAIL"));
        }
        OidData->Oid = 0x01010101;
        OidData->Length = 6;
        m_AdapterObject = nullptr;
        memset(data, 0, MAC_MAX_SIZE);
    }
    // 에러 발생 시
    catch (CString errorInfo) {
        // 에러 메시지 출력하고 함수 종료
        AfxMessageBox(errorInfo);
        return;
    }
}
```

```
60   // 소멸자
61   CNILayer::~CNILayer() {
62       pcap_if_t* temp = nullptr;
63
64       // allDevices에 할당한 메모리를 해제
65       while (allDevices) {
66           temp = allDevices;
67           allDevices = allDevices->next;
68           delete(temp);
69       }
70
71       // OidData에 할당한 메모리를 해제
72       delete(OidData);
73   }
74
75   // network interface layer에서 패킷 수신 함수
76   BOOL CNILayer::Receive(unsigned char* pkt) {
77
78       // 패킷이 null인 경우 FALSE 반환 (Host끼리 통신 시 사용)
79       if (pkt == nullptr) {
80           return FALSE;
81       }
82
83       // 상위 계층에서 패킷을 받지 못한 경우 FALSE 반환 (Etherent Layer에서 패킷 받아올 때 사용)
84       if (!(mp_aUpperLayer[0]->Receive(pkt))) {
85           return FALSE;
86       }
87
88       // 수신 성공 시 TRUE 반환
89       return TRUE;
90   }
91
92   // network interface layer에서 패킷 전송 함수
93   BOOL CNILayer::Send(unsigned char* ppayload, int nlength) {
94
95       // 패킷 전송
96       if (pcap_sendpacket(m_AdapterObject, ppayload, nlength))
97       {
98           // 실패 시 에러 메시지 출력
99           AfxMessageBox(_T("Fail: Send Packet!"));
100          // FALSE 반환
101          return FALSE;
102      }
103
104      // 전송 성공 시 TRUE 반환
105      return TRUE;
106  }
```

```
116
117  // 선택한 네트워크 어댑터를 설정하는 함수
118  UCHAR* CNILayer::SetAdapter(const int index) {
119      char errbuf[PCAP_ERRBUF_SIZE];
120      device = allDevices;
121
122      // 이미 열려있는 어댑터가 있다면 닫음
123      if (m_AdapterObject != nullptr)
124          pcap_close(m_AdapterObject);
125
126      // 입력된 인덱스만큼 네트워크 어댑터 목록을 순회
127      for (int i = 0; i < index && device; i++) {
128          device = device->next;
129      }
130      // 해당 장치를 열음
131      if (device != nullptr)
132          m_AdapterObject = pcap_open_live((const char*)device->name, 65536, 0, 1000, errbuf);
133      // 장치를 열지 못했다면 에러 메시지를 출력하고 nullptr를 반환
134      if (m_AdapterObject == nullptr)
135      {
136          AfxMessageBox(_T("Fail: Connect Adapter"));
137          return nullptr;
138      }
139
140      // 어댑터의 MAC 주소를 요청
141      adapter = PacketOpenAdapter(device->name);
142      PacketRequest(adapter, FALSE, OidData);
143
144      // 어댑터를 닫음
145      PacketCloseAdapter(adapter);
146      //AfxBeginThread(ThreadFunction_RECEIVE, this);
147
148      // MAC 주소 반환
149      return (OidData->Data);
150  }
151
152  // 네트워크 장치의 MAC 주소 목록을 가져오는 함수
153  void CNILayer::GetMacAddressList(CStringArray& adapterlist) {
154      for (pcap_if_t* d = allDevices; d; d = d->next) {
155          adapterlist.Add(CString(d->description));
156      }
157  }
158
```

```
159
160 // 검색
161 // 선택한 네트워크 어댑터의 IP 주소를 가져오는 함수
162 // 매개변수로 ipv4 ipv6 값을 저장할 문자열을 받음
163 // 네트워크 어댑터를 먼저 설정해야 함
164 void CNILayer::GetIPAddress(CString& ipv4addr, CString& ipv6addr) {
165
166     // IP 주소를 저장하기 위한 문자열 배열을 선언
167     char ip[IPV6_ADDR_STR_LEN];
168     // 초기값으로 ipv4addr, ipv6addr를 설정
169     ipv4addr = DEFAULT_EDIT_TEXT;
170     ipv6addr = DEFAULT_EDIT_TEXT;
171
172     // 장치의 모든 주소를 순회
173     for (auto addr = device->addresses; addr != nullptr; addr = addr->next)
174     {
175         // 실제 주소를 저장
176         auto realaddr = addr->addr;
177         // 주소의 패밀리 타입을 가져옴
178         const int sa_family = realaddr->sa_family;
179
180         // 주소를 문자열로 변환하여 ptr에 저장
181         const char* ptr = inet_ntop(sa_family, &realaddr->sa_data[sa_family == AF_INET ? 2 : 6], ip, IPV6_ADDR_STR_LEN);
182
183         // 패밀리 타입에 따라 ipv4addr 또는 ipv6addr에 주소를 저장
184         switch (sa_family)
185         {
186         case AF_INET:
187             ipv4addr = ptr;
188             break;
189         case AF_INET6:
190             ipv6addr = ptr;
191             break;
192         default:
193             return;
194         }
195     }
196 }
197
```

```
198 // 패킷 수신하는 스레드 함수
199 UINT CNILayer::ThreadFunction_RECEIVE(LPVOID pParam) {
200     struct pcap_pkthdr* header;
201     const u_char* pkt_data;
202     CNILayer* pNI = (CNILayer*)pParam;
203
204     int result = 1;
205
206     while (pNI->canRead)
207     {
208         result = pcap_next_ex(pNI->m_AdapterObject, &header, &pkt_data);
209         if (result == 1) {
210             memcpy(pNI->data, pkt_data, MAC_MAX_SIZE);
211             pNI->Receive(pNI->data);
212         }
213     }
214     return 0;
215 }
216
217 // NI Layer 패킷 수신 상태 변경 함수
218 void CNILayer::Receiveflip() {
219     canRead = !canRead;
220 }
```