컴퓨터 네트워크 과제: ARP 기능 탑재 채팅 프로그램

2분반 5조

📋 목차

---

1. 프로젝트 개요

1.1 목적

기존 이더넷 기반 채팅 프로그램에 **IP 계층**과 **ARP 계층**을 추가하여 실제 네트워크 프로토콜 스택의 동작 원리를 이해하고 구현합니다.

1.2 개발 환경

- **언어:** Java 21
- **라이브러리:** jNetPcap 2.3.1 (패킷 캡처 및 전송)
- **GUI:** Swing
- **OS:** macOS / Linux
- **IDE:** VS Code

1.3 주요 기능

- ✅ **IP 계층 구현** (IPv4 헤더 20바이트)
- ✅ **ARP 프로토콜 구현** (Request, Reply, Gratuitous ARP, Proxy ARP)
- ✅ **이더넷 역다중화** (EtherType 기반 계층 분리)
- ✅ **ARP 캐시 관리** (IP-MAC 매핑 테이블)
- ✅ **GUI 기반 실시간 ARP 테이블 표시**

---

2. 구현 요구사항

2.1 필수 구현 항목

✅ IP/ARP 계층 추가 및 계층 연결

- IPLayer: IPv4 패킷 생성 및 파싱 (20바이트 헤더)

- ARPLayer: ARP 프로토콜 처리 (28바이트 ARP 패킷)
- 기존 계층과의 연결: ChatApp → IP → Ethernet/ARP → Physical

✅ **ARP 기능 구현**

1. **ARP Request/Reply:** IP 주소 → MAC 주소 변환
2. **Gratuitous ARP:** 자신의 IP를 네트워크에 공지
3. **Proxy ARP:** 다른 호스트 대신 ARP 응답

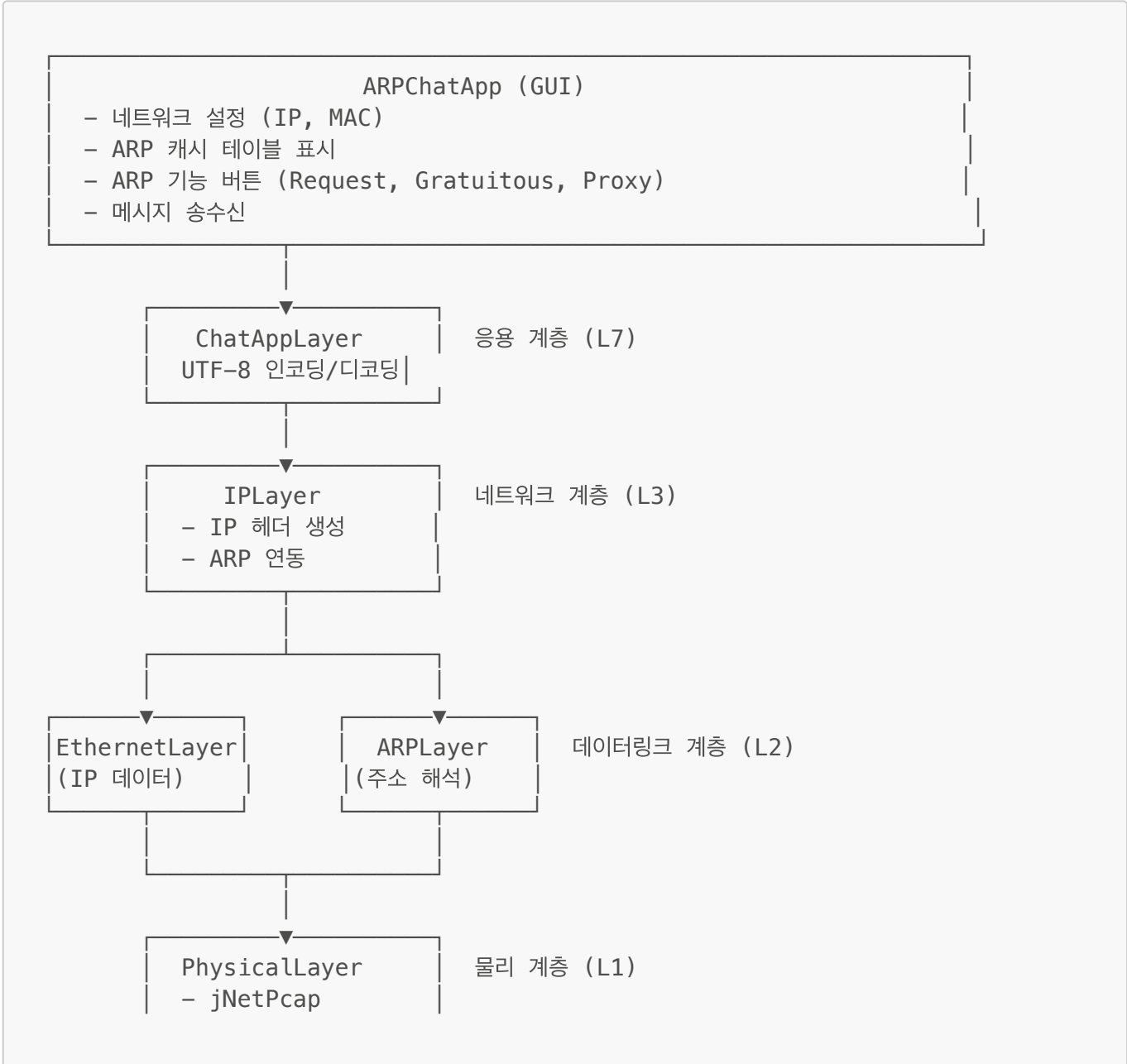
✅ **이더넷 역다중화**

- EtherType 0x0800 (IPv4) → IPLayer
- EtherType 0x0806 (ARP) → ARPLayer

---

3. 계층 구조 및 설계

3.1 전체 계층 구조



```
│  - 패킷 송수신          │
└─────────────────────────┘
```

3.2 계층 간 인터페이스

BaseLayer 인터페이스:

```java
public interface BaseLayer {
    String GetLayerName();
    BaseLayer GetUnderLayer();
    BaseLayer GetUpperLayer(int index);
    void SetUnderLayer(BaseLayer layer);
    void SetUpperLayer(BaseLayer layer);
    boolean Send(byte[] input, int length);
    boolean Receive(byte[] input);
}
```

3.3 계층 연결 코드 (ARPChatApp.java)

```java
// 계층 생성
chatLayer = new ChatAppLayer(...);
ipLayer = new IPLayer();
arpLayer = new ARPLayer();
ethernetLayer = new EthernetLayer();
physicalLayer = new PhysicalLayer();

// 계층 연결
// ChatApp ↔ IP
chatLayer.SetUnderLayer(ipLayer);
ipLayer.SetUpperLayer(chatLayer);

// IP ↔ Ethernet (데이터 전송)
ipLayer.SetUnderLayer(ethernetLayer);
ethernetLayer.SetUpperLayer(ipLayer);

// ARP ↔ Ethernet (주소 해석)
arpLayer.SetUnderLayer(ethernetLayer);
ethernetLayer.SetUpperLayer(arpLayer);

// Ethernet ↔ Physical
ethernetLayer.SetUnderLayer(physicalLayer);
physicalLayer.SetUpperLayer(ethernetLayer);

// IP와 ARP 연결 (IP가 ARP 사용)
ipLayer.setArpLayer(arpLayer);
```

---

4. 이더넷 역다중화 구현

4.1 역다중화 개념

역다중화(Demultiplexing):
이더넷 프레임의 **EtherType** 필드를 읽어서 상위 계층 프로토콜을 판별하고, 적절한 계층으로 데이터를 전달하는 과정입니다.

```
Ethernet Frame:
┌──────────┬──────────┬──────────┬──────────┐
│ Dst MAC  │ Src MAC  │ EtherType│ Payload  │
│ 6 bytes  │ 6 bytes  │ 2 bytes  │ 46-1500  │
└──────────┴──────────┴──────────┴──────────┘
                        │
                        ├─ 0x0800 → IPv4 → IPLayer
                        ├─ 0x0806 → ARP  → ARPLayer
                        └─ 기타   → ChatAppLayer (하위 호환)
```

4.2 코드 구현 (EthernetLayer.java)

파일 위치: `src/main/java/com/demo/EthernetLayer.java` (라인 200-230)

```
/**
 * 이더넷 역다중화 구현
 *
 * EtherType 값에 따라 상위 계층 선택:
 * - 0x0800 (IPv4) → IPLayer
 * - 0x0806 (ARP)  → ARPLayer
 * - 기타          → ChatAppLayer (레거시 지원)
 */
@Override
public boolean Receive(byte[] input) {
    // ... (필터링 생략) ...

    // 4. EtherType 추출 (12-13번째 바이트)
    int receivedEtherType = ((input[12] & 0xFF) << 8) | (input[13] &
0xFF);

    // 5. 페이로드 추출 (14번째 바이트부터)
    byte[] payload = Arrays.copyOfRange(input, 14, input.length);

    // 6. 디버그 로그
    System.out.println("[Ethernet] 수신 - EtherType: 0x" +
                       String.format("%04X", receivedEtherType) +
                       ", 페이로드 길이: " + payload.length + "바이트");

    // 7. 이더넷 역다중화: EtherType에 따라 상위 계층 선택
    boolean delivered = false;
    for (BaseLayer upper : uppers) {
        // IPLayer는 0x0800만 처리
        if (receivedEtherType == ETHERTYPE_IP && upper instanceof IPLayer)
{
```

```
                upper.Receive(payload);
                delivered = true;
            }
            // ARPLayer는 0x0806만 처리
            else if (receivedEtherType == ETHERTYPE_ARP && upper instanceof
ARPLayer) {
                upper.Receive(payload);
                delivered = true;
            }
            // 기타 상위 계층 (ChatAppLayer 등 - 하위 호환성)
            else if (!(upper instanceof IPLayer) && !(upper instanceof
ARPLayer)) {
                upper.Receive(payload);
                delivered = true;
            }
        }

        if (!delivered) {
            System.out.println("[Ethernet] 경고: 처리할 상위 계층 없음");
        }

        return delivered;
    }
```

4.3 EtherType 상수 정의

```
// EthernetLayer.java 라인 30-31
private static final int ETHERTYPE_IP = 0x0800;   // IPv4
private static final int ETHERTYPE_ARP = 0x0806;  // ARP
```

4.4 동작 원리

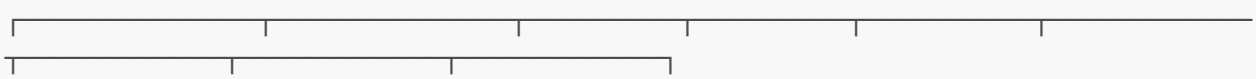
1. **패킷 수신:** PhysicalLayer에서 이더넷 프레임 수신
2. **EtherType 추출:** 12-13번째 바이트에서 2바이트 읽기
3. **타입 판별:**
    - `0x0800` → IP 프로토콜 → IPLayer.Receive() 호출
    - `0x0806` → ARP 프로토콜 → ARPLayer.Receive() 호출
4. **페이로드 전달:** 이더넷 헤더(14바이트) 제거 후 상위 계층으로 전달

---

5. ARP 기능 구현

5.1 ARP 패킷 구조

ARP 패킷 (28바이트):



```
|Hardware Type |Protocol Type |HW Len   |Proto Len|Operation |Sender HW
|Sender Proto|Target HW   |Target Proto|
|   2 bytes    |   2 bytes    | 1 byte  | 1 byte  | 2 bytes  |  6 bytes
|  4 bytes   |  6 bytes   |  4 bytes   |
├──────────────┼──────────────┼──────────┼─────────┼──────────┼──────────────
┼──────────────┼──────────────┼──────────┤
|   0x0001     |   0x0800     |    6     |    4    | 1 or 2   | Sender MAC
| Sender IP  | Target MAC | Target IP  |
|  (Ethernet)  |   (IPv4)     |          |         |(REQ/REPLY)|
|              |              |          |
└──────────────┴──────────────┴──────────┴─────────┴──────────┴──────────────
┴──────────────┴──────────────┴──────────┘
```

5.2 ARP Request 구현

파일 위치: `src/main/java/com/demo/ARPLayer.java` (라인 145-175)

```java
/**
 * ARP Request 전송
 *
 * 목적: "IP 주소 X.X.X.X의 MAC 주소를 아는 사람?"
 * 전송: 브로드캐스트 (FF:FF:FF:FF:FF:FF)
 *
 * @param targetIp 조회할 IP 주소 (4바이트)
 */
public void sendArpRequest(byte[] targetIp) {
    System.out.println("[ARP] Request 전송: " + formatIp(targetIp));

    // ARP 패킷 생성 (28바이트)
    byte[] arpPacket = new byte[28];
    ByteBuffer buffer = ByteBuffer.wrap(arpPacket);

    // Hardware Type: Ethernet (0x0001)
    buffer.putShort((short) HARDWARE_TYPE_ETHERNET);

    // Protocol Type: IPv4 (0x0800)
    buffer.putShort((short) PROTOCOL_TYPE_IP);

    // Hardware Address Length: 6 (MAC 주소)
    buffer.put((byte) HARDWARE_LEN);

    // Protocol Address Length: 4 (IP 주소)
    buffer.put((byte) PROTOCOL_LEN);

    // Operation: Request (1)
    buffer.putShort((short) OPERATION_REQUEST);

    // Sender Hardware Address: 내 MAC 주소
    buffer.put(myMac);

    // Sender Protocol Address: 내 IP 주소
```

```
        buffer.put(myIp);

        // Target Hardware Address: 00:00:00:00:00:00 (모름)
        buffer.put(new byte[6]);

        // Target Protocol Address: 조회할 IP 주소
        buffer.put(targetIp);

        // 브로드캐스트로 전송
        byte[] broadcast = new byte[]{
            (byte)0xFF, (byte)0xFF, (byte)0xFF,
            (byte)0xFF, (byte)0xFF, (byte)0xFF
        };

        // EthernetLayer를 통해 전송
        if (underLayer != null) {
            ((EthernetLayer) underLayer).setDstMac(broadcast);
            ((EthernetLayer) underLayer).setEtherType(0x0806); // ARP
            underLayer.Send(arpPacket, arpPacket.length);
        }
    }
```

5.3 ARP Reply 구현

파일 위치: `src/main/java/com/demo/ARPLayer.java` (라인 180-220)

```
    /**
     * ARP Reply 전송
     *
     * 목적: "IP 주소 X.X.X.X는 MAC 주소 YY:YY:YY:YY:YY:YY입니다"
     * 전송: 유니캐스트 (요청자 MAC 주소)
     *
     * @param targetIp 요청자 IP 주소
     * @param targetMac 요청자 MAC 주소
     */
    public void sendArpReply(byte[] targetIp, byte[] targetMac) {
        System.out.println("[ARP] Reply 전송: " + formatIp(myIp) +
                           " → " + formatMac(targetMac));

        // ARP 패킷 생성 (28바이트)
        byte[] arpPacket = new byte[28];
        ByteBuffer buffer = ByteBuffer.wrap(arpPacket);

        buffer.putShort((short) HARDWARE_TYPE_ETHERNET);
        buffer.putShort((short) PROTOCOL_TYPE_IP);
        buffer.put((byte) HARDWARE_LEN);
        buffer.put((byte) PROTOCOL_LEN);

        // Operation: Reply (2)
        buffer.putShort((short) OPERATION_REPLY);
```

```
        // Sender: 나
        buffer.put(myMac);
        buffer.put(myIp);

        // Target: 요청자
        buffer.put(targetMac);
        buffer.put(targetIp);

        // 유니캐스트로 전송
        if (underLayer != null) {
            ((EthernetLayer) underLayer).setDstMac(targetMac);
            ((EthernetLayer) underLayer).setEtherType(0x0806);
            underLayer.Send(arpPacket, arpPacket.length);
        }
    }
```

5.4 ARP 수신 및 캐시 업데이트

파일 위치: `src/main/java/com/demo/ARPLayer.java` (라인 280-360)

```
    /**
     * ARP 패킷 수신 처리
     *
     * 동작:
     * 1. ARP 패킷 파싱
     * 2. Sender IP-MAC을 ARP 캐시에 저장 (학습)
     * 3. Operation 타입 확인 (Request or Reply)
     * 4. Request이고 Target IP가 나면 Reply 전송
     */
    @Override
    public boolean Receive(byte[] input) {
        if (input == null || input.length < 28) {
            return false;
        }

        ByteBuffer buffer = ByteBuffer.wrap(input);

        // 헤더 파싱
        int hwType = buffer.getShort() & 0xFFFF;
        int protoType = buffer.getShort() & 0xFFFF;
        buffer.get(); // hwLen (사용하지 않음)
        buffer.get(); // protoLen (사용하지 않음)
        int operation = buffer.getShort() & 0xFFFF;

        // Sender 정보 추출
        byte[] senderMac = new byte[6];
        buffer.get(senderMac);

        byte[] senderIp = new byte[4];
        buffer.get(senderIp);
```

```
        // Target 정보 추출
        byte[] targetMac = new byte[6];
        buffer.get(targetMac);

        byte[] targetIp = new byte[4];
        buffer.get(targetIp);

        // ARP 캐시에 Sender 정보 저장 (학습)
        String senderIpStr = formatIp(senderIp);
        arpCache.put(senderIpStr, Arrays.copyOf(senderMac, 6));
        System.out.println("[ARP] 캐시 업데이트: " + senderIpStr +
                           " → " + formatMac(senderMac));

        // Operation 처리
        if (operation == OPERATION_REQUEST) {
            // Target IP가 나인지 확인
            if (Arrays.equals(targetIp, myIp)) {
                System.out.println("[ARP] Request 수신 - Reply 전송");
                sendArpReply(senderIp, senderMac);
            }
            // Proxy ARP 처리
            else if (proxyArpEnabled) {
                String targetIpStr = formatIp(targetIp);
                byte[] proxyMac = proxyTable.get(targetIpStr);
                if (proxyMac != null) {
                    System.out.println("[ARP] Proxy ARP 응답: " + targetIpStr);
                    // Proxy MAC으로 응답
                    sendArpReply(senderIp, senderMac);
                }
            }
        }
        else if (operation == OPERATION_REPLY) {
            System.out.println("[ARP] Reply 수신: " + senderIpStr);
            // 캐시 업데이트는 이미 위에서 완료
        }

        return true;
    }
```

5.5 Gratuitous ARP 구현

파일 위치: `src/main/java/com/demo/ARPLayer.java` (라인 225-250)

```
    /**
     * Gratuitous ARP 전송
     *
     * 목적:
     * 1. 네트워크 진입 시 자신의 IP-MAC 매핑 공지
     * 2. IP 주소 충돌 감지
     * 3. 다른 호스트의 ARP 캐시 업데이트
     *
```

```
     * 특징: Sender IP = Target IP (자신의 IP)
     */
    public void sendGratuitousArp() {
        System.out.println("[ARP] Gratuitous ARP 전송");

        byte[] arpPacket = new byte[28];
        ByteBuffer buffer = ByteBuffer.wrap(arpPacket);

        buffer.putShort((short) HARDWARE_TYPE_ETHERNET);
        buffer.putShort((short) PROTOCOL_TYPE_IP);
        buffer.put((byte) HARDWARE_LEN);
        buffer.put((byte) PROTOCOL_LEN);
        buffer.putShort((short) OPERATION_REQUEST); // Request

        // Sender: 나
        buffer.put(myMac);
        buffer.put(myIp);

        // Target: 나 (Gratuitous의 특징!)
        buffer.put(new byte[6]); // Target MAC은 00:00:00:00:00:00
        buffer.put(myIp);        // Target IP = Sender IP

        // 브로드캐스트로 전송
        byte[] broadcast = new byte[]{
            (byte)0xFF, (byte)0xFF, (byte)0xFF,
            (byte)0xFF, (byte)0xFF, (byte)0xFF
        };

        if (underLayer != null) {
            ((EthernetLayer) underLayer).setDstMac(broadcast);
            ((EthernetLayer) underLayer).setEtherType(0x0806);
            underLayer.Send(arpPacket, arpPacket.length);
        }
    }
```

5.6 Proxy ARP 구현

파일 위치: `src/main/java/com/demo/ARPLayer.java` (라인 90-100, 330-350)

```
    /**
     * Proxy ARP 엔트리 추가
     *
     * 다른 호스트(예: 192.168.0.200)를 대신하여 ARP 응답
     *
     * @param ip Proxy할 IP 주소 문자열
     * @param mac Proxy MAC 주소 (6바이트)
     */
    public void addProxyArpEntry(String ip, byte[] mac) {
        if (mac != null && mac.length >= 6) {
            byte[] macCopy = Arrays.copyOf(mac, 6);
            proxyTable.put(ip, macCopy);
```

```
            System.out.println("[Proxy ARP] 추가: " + ip +
                               " → " + formatMac(macCopy));
        }
    }

    // Receive() 메서드 내부 (라인 330-350)
    // Proxy ARP 처리 부분
    else if (proxyArpEnabled) {
        String targetIpStr = formatIp(targetIp);
        byte[] proxyMac = proxyTable.get(targetIpStr);
        if (proxyMac != null) {
            System.out.println("[ARP] Proxy ARP 응답: " + targetIpStr);

            // Proxy용 ARP Reply 생성
            byte[] arpPacket = new byte[28];
            ByteBuffer proxyBuffer = ByteBuffer.wrap(arpPacket);

            proxyBuffer.putShort((short) HARDWARE_TYPE_ETHERNET);
            proxyBuffer.putShort((short) PROTOCOL_TYPE_IP);
            proxyBuffer.put((byte) HARDWARE_LEN);
            proxyBuffer.put((byte) PROTOCOL_LEN);
            proxyBuffer.putShort((short) OPERATION_REPLY);

            // Sender: Proxy 정보 (실제로는 다른 호스트)
            proxyBuffer.put(proxyMac);      // Proxy MAC
            proxyBuffer.put(targetIp);      // Target IP

            // Target: 요청자
            proxyBuffer.put(senderMac);
            proxyBuffer.put(senderIp);

            // 응답 전송
            if (underLayer != null) {
                ((EthernetLayer) underLayer).setDstMac(senderMac);
                ((EthernetLayer) underLayer).setEtherType(0x0806);
                underLayer.Send(arpPacket, arpPacket.length);
            }
        }
    }
```

5.7 ARP 캐시 관리

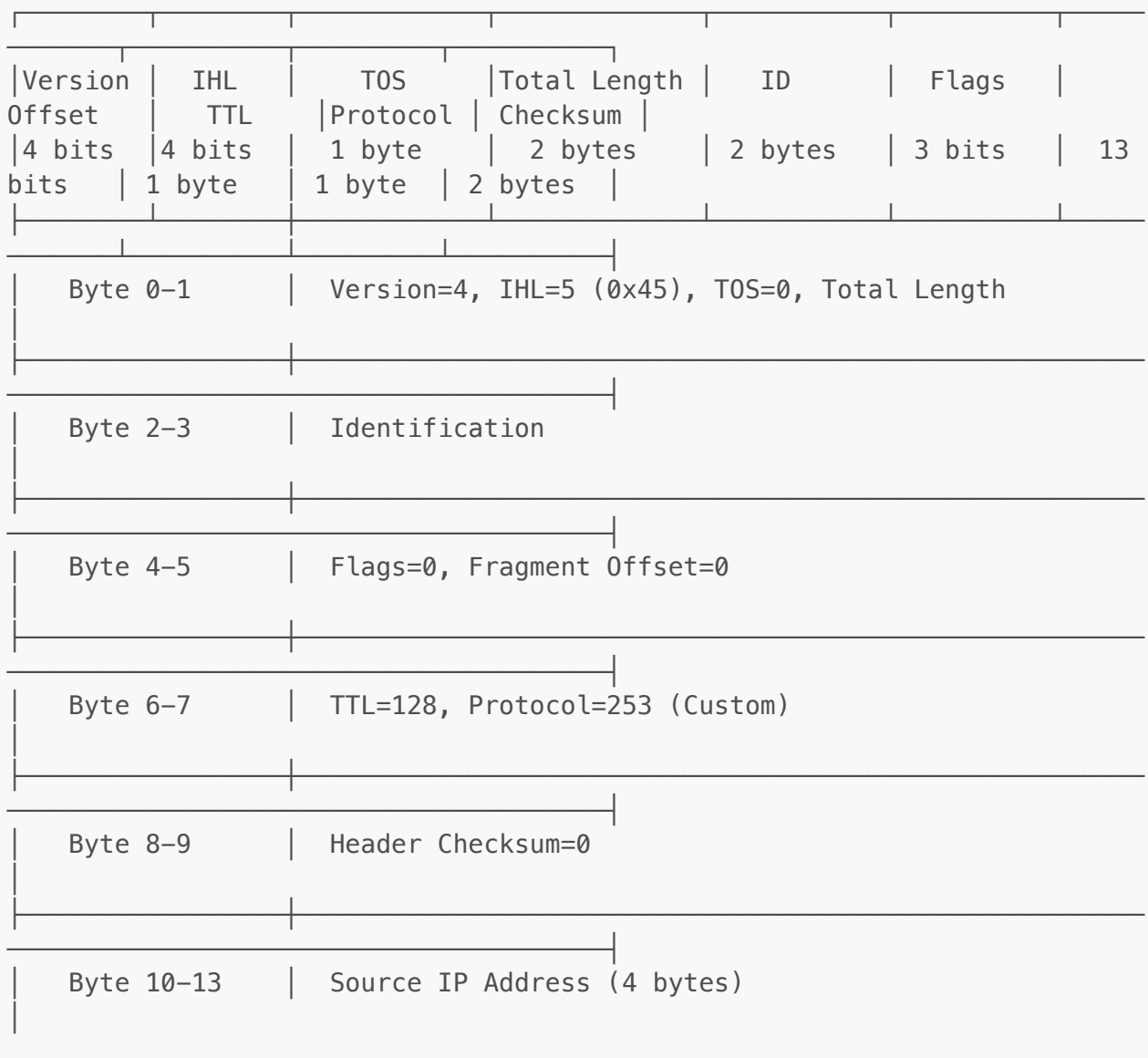
```
    // ARP 캐시 (ConcurrentHashMap 사용 - 멀티스레드 안전)
    private final Map<String, byte[]> arpCache = new ConcurrentHashMap<>();

    /**
     * ARP 캐시 조회
     */
    public byte[] lookupArpCache(String ip) {
        return arpCache.get(ip);
    }
```

```java
/**
 * ARP 캐시 초기화
 */
public void clearArpCache() {
    arpCache.clear();
    System.out.println("[ARP] 캐시 초기화됨");
}

/**
 * ARP 캐시 전체 반환 (GUI 표시용)
 */
public Map<String, byte[]> getArpCache() {
    return new ConcurrentHashMap<>(arpCache);
}
```

---

6. IP 계층 구현

6.1 IP 헤더 구조 (20바이트)



```
├──────────────────┼──────────────────────────────────────────────────────────────┤
│   Byte 14-17     │  Destination IP Address (4 bytes)                            │
├──────────────────┼──────────────────────────────────────────────────────────────┤
│   Byte 18-19     │  (Options - 이 구현에서는 사용하지 않음, IHL=5이므로 옵션 없음)  │
└──────────────────┴──────────────────────────────────────────────────────────────┘
```

6.2 IP 패킷 전송 구현

파일 위치: `src/main/java/com/demo/IPLayer.java` (라인 132-215)

```java
/**
 * IP 패킷 전송
 *
 * 과정:
 * 1. ARP 캐시에서 목적지 IP의 MAC 주소 조회
 * 2. 없으면 ARP Request 전송 후 실패 반환
 * 3. IP 헤더 생성 (20바이트)
 * 4. 페이로드 추가
 * 5. EthernetLayer로 전송
 */
@Override
public boolean Send(byte[] input, int length) {
    if (underLayer == null || arpLayer == null) {
        System.out.println("[IP] 하위 계층 또는 ARP 계층이 설정되지 않음");
        return false;
    }

    // 목적지 IP에 대한 MAC 주소 조회
    String dstIpStr = formatIp(dstIp);
    byte[] dstMac = arpLayer.lookupArpCache(dstIpStr);

    // ARP 캐시에 없으면 ARP Request 전송
    if (dstMac == null) {
        System.out.println("[IP] ARP 캐시에 " + dstIpStr +
                           " 없음 - ARP Request 전송");
        arpLayer.sendArpRequest(dstIp);
        return false;
    }

    System.out.println("[IP] 목적지 MAC 주소 발견: " + formatMac(dstMac));

    // IP 패킷 생성: IP 헤더(20바이트) + 페이로드
    int totalLength = 20 + length;
    byte[] ipPacket = new byte[totalLength];
    ByteBuffer buffer = ByteBuffer.wrap(ipPacket);
```

```
        // ===== IP 헤더 생성 (20바이트) =====

        // Byte 0: Version (4비트) + IHL (4비트) = 0x45
        buffer.put((byte) ((IP_VERSION << 4) | IP_HEADER_LENGTH));

        // Byte 1: Type of Service
        buffer.put((byte) IP_TOS);

        // Byte 2-3: Total Length (헤더 + 페이로드)
        buffer.putShort((short) totalLength);

        // Byte 4-5: Identification (패킷 ID)
        buffer.putShort((short) (ipIdentification++ & 0xFFFF));

        // Byte 6-7: Flags + Fragment Offset
        buffer.putShort((short) 0);

        // Byte 8: Time to Live
        buffer.put((byte) IP_TTL);

        // Byte 9: Protocol (253 = Custom)
        buffer.put((byte) IP_PROTOCOL_CUSTOM);

        // Byte 10-11: Header Checksum (간단한 구현으로 0)
        buffer.putShort((short) 0);

        // Byte 12-15: Source IP Address
        buffer.put(myIp);

        // Byte 16-19: Destination IP Address
        buffer.put(dstIp);

        // ===== 페이로드 복사 =====
        buffer.put(input, 0, length);

        System.out.println("[IP] 패킷 전송: " + formatIp(myIp) +
                           " → " + formatIp(dstIp) +
                           " (길이: " + totalLength + "바이트)");

        // EthernetLayer의 목적지 MAC 설정
        if (underLayer instanceof EthernetLayer) {
            ((EthernetLayer) underLayer).setDstMac(dstMac);
            ((EthernetLayer) underLayer).setEtherType(0x0800); // IPv4
        }

        // 하위 계층(Ethernet)으로 전송
        return underLayer.Send(ipPacket, ipPacket.length);
    }
```

6.3 IP 패킷 수신 처리

파일 위치: `src/main/java/com/demo/IPLayer.java` (라인 220-310)

```java
/**
 * IP 패킷 수신
 *
 * 과정:
 * 1. IP 헤더 파싱
 * 2. 목적지 IP 필터링 (자신의 IP인 경우만 수락)
 * 3. 페이로드 추출
 * 4. 상위 계층(ChatApp)으로 전달
 */
@Override
public boolean Receive(byte[] input) {
    // 최소 IP 헤더 크기 체크 (20바이트)
    if (input == null || input.length < 20) {
        return false;
    }

    ByteBuffer buffer = ByteBuffer.wrap(input);

    // Version + IHL 파싱
    int versionIhl = buffer.get() & 0xFF;
    int version = (versionIhl >> 4) & 0x0F;
    int ihl = versionIhl & 0x0F;

    // IPv4만 처리
    if (version != IP_VERSION) {
        return false;
    }

    // 헤더 길이 계산 (IHL * 4 바이트)
    int headerLength = ihl * 4;

    if (input.length < headerLength) {
        return false;
    }

    // 나머지 헤더 필드 파싱
    buffer.get();                                       // TOS
    int totalLength = buffer.getShort() & 0xFFFF;       // Total Length
    buffer.getShort();                                  // Identification
    buffer.getShort();                                  // Flags + Offset
    buffer.get();                                       // TTL
    int protocol = buffer.get() & 0xFF;                 // Protocol
    buffer.getShort();                                  // Checksum

    // Source IP
    byte[] srcIp = new byte[4];
    buffer.get(srcIp);

    // Destination IP
    byte[] dstIpReceived = new byte[4];
    buffer.get(dstIpReceived);

    System.out.println("[IP] 패킷 수신: " + formatIp(srcIp) +
```

```
                        " → " + formatIp(dstIpReceived) +
                        " (프로토콜: " + protocol + ")");

    // 목적지 IP 필터링 - 자신의 IP인 경우만 수락
    if (!Arrays.equals(dstIpReceived, myIp)) {
        System.out.println("[IP] 목적지 IP 불일치 - 패킷 드롭");
        return false;
    }

    // 페이로드 추출 (IP 헤더 제거)
    int payloadLength = totalLength - headerLength;
    if (payloadLength <= 0 || headerLength + payloadLength > input.length)
{
        return false;
    }

    byte[] payload = Arrays.copyOfRange(input, headerLength,
                                        headerLength + payloadLength);

    // 상위 계층(ChatApp)으로 전달
    for (BaseLayer upper : uppers) {
        upper.Receive(payload);
    }

    return true;
}
```

---

7. 실행 화면

7.1 프로그램 실행

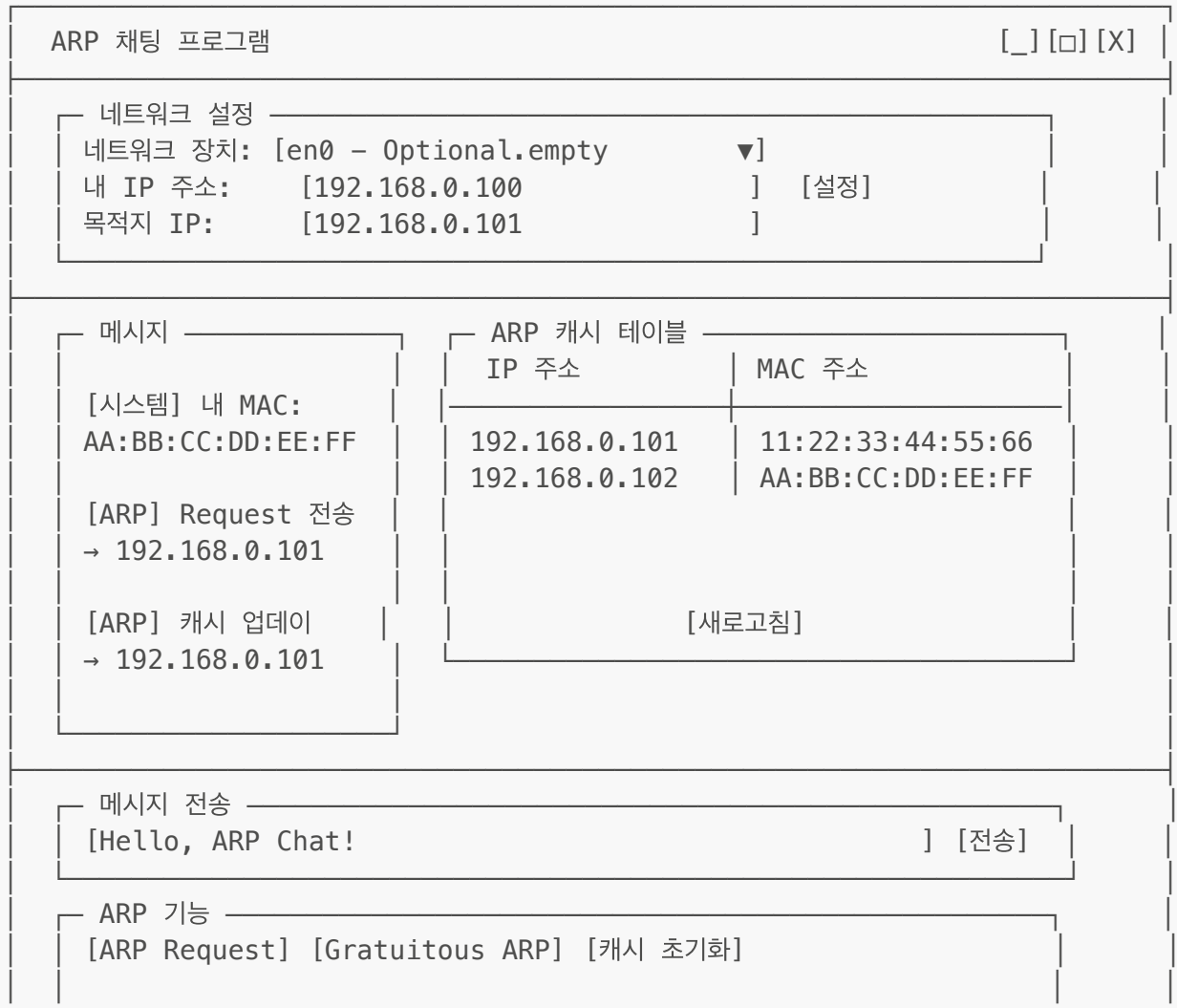
```
git clone https://github.com/qoweh/network_homework.git
cd third
sudo ./run_arp_chat.sh
```

실행 로그:

```
╔═══════════════════════════════════════════════════════════════════════════
═══════╗
║                ARP 기능이 추가된 패킷 채팅 프로그램 (ARP Chat v2.0)
║
╚═══════════════════════════════════════════════════════════════════════════
═══════╝

[컴파일] 클래스 파일이 없습니다. 컴파일을 시작합니다...
[완료] 컴파일 성공

[시작] ARP 채팅 프로그램을 시작합니다...
```

```
╔═══════════════════════════════════════════════════════════════════════════════════════╗
║                ARP 기능이 추가된 패킷 채팅 프로그램 (ARP Chat v2.0)                   ║
║                                                                                       ║
║   • 계층 구조: ChatApp → IP → Ethernet/ARP → Physical                                 ║
║   • ARP 기능: Request, Reply, Gratuitous ARP, Proxy ARP                               ║
║   • IP 통신: IPv4 기반 메시지 송수신                                                  ║
╚═══════════════════════════════════════════════════════════════════════════════════════╝

사용 가능한 네트워크 장치:
1. en0 - Optional.empty
2. awdl0 - Optional.empty
...
```

7.2 GUI 화면 구성



```
  │   │ [✓] Proxy ARP 활성화                                          │       │
  │   │ Proxy IP:  [192.168.0.200      ]                             │       │
  │   │ Proxy MAC: [AA:BB:CC:DD:EE:FF  ] [Proxy 추가]                │       │
  │   │                                                              │       │
  │   │                              [종료]                          │       │
  │   └──────────────────────────────────────────────────────────────┘       │
  └──────────────────────────────────────────────────────────────────────────┘
```

7.3 주요 화면 설명

📌 화면 1: 초기 실행

- 네트워크 장치 목록 표시
- IP 주소 입력 필드
- "설정" 버튼으로 네트워크 초기화

📌 화면 2: ARP Request 실행

- "ARP Request" 버튼 클릭
- 메시지 창에 "ARP Request 전송" 로그
- ARP Reply 수신 후 캐시 테이블 업데이트

📌 화면 3: ARP 캐시 테이블

- IP-MAC 매핑 실시간 표시
- "새로고침" 버튼으로 수동 업데이트

📌 화면 4: Gratuitous ARP

- "Gratuitous ARP" 버튼 클릭
- 브로드캐스트로 자신의 IP 공지

📌 화면 5: Proxy ARP 설정

- Proxy IP, MAC 입력
- "Proxy 추가" 버튼으로 등록
- 다른 호스트의 ARP Request에 대리 응답

📌 화면 6: 메시지 송수신

- 메시지 입력 후 "전송" 클릭
- IP 계층 → ARP 조회 → 이더넷 전송
- 수신 메시지 표시

---

8. Wireshark 캡처 분석

8.1 Wireshark 필터 설정

```
arp or (ip.addr == 192.168.0.100)
```

8.2 ARP Request 패킷 캡처

```
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: aa:bb:cc:dd:ee:ff, Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination: Broadcast (ff:ff:ff:ff:ff:ff)
    Source: aa:bb:cc:dd:ee:ff
    Type: ARP (0x0806)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: aa:bb:cc:dd:ee:ff
    Sender IP address: 192.168.0.100
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.0.101
```

분석:

- ✅ Destination: 브로드캐스트 (FF:FF:FF:FF:FF:FF)
- ✅ EtherType: 0x0806 (ARP)
- ✅ Opcode: 1 (Request)
- ✅ Target MAC: 00:00:00:00:00:00 (모름)
- ✅ Target IP: 192.168.0.101 (조회 대상)

8.3 ARP Reply 패킷 캡처

```
Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: 11:22:33:44:55:66, Dst: aa:bb:cc:dd:ee:ff
    Destination: aa:bb:cc:dd:ee:ff
    Source: 11:22:33:44:55:66
    Type: ARP (0x0806)
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 11:22:33:44:55:66
    Sender IP address: 192.168.0.101
    Target MAC address: aa:bb:cc:dd:ee:ff
    Target IP address: 192.168.0.100
```

분석:

- ✅ Destination: 유니캐스트 (요청자 MAC)
- ✅ Opcode: 2 (Reply)
- ✅ Sender MAC: 11:22:33:44:55:66 (응답자)
- ✅ Target MAC: aa:bb:cc:dd:ee:ff (요청자)

8.4 Gratuitous ARP 패킷 캡처

```
Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: aa:bb:cc:dd:ee:ff, Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination: Broadcast (ff:ff:ff:ff:ff:ff)
    Source: aa:bb:cc:dd:ee:ff
    Type: ARP (0x0806)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: aa:bb:cc:dd:ee:ff
    Sender IP address: 192.168.0.100
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.0.100  ← Sender IP와 동일!
```

분석:

- ✅ Sender IP = Target IP (Gratuitous의 특징)
- ✅ 브로드캐스트로 네트워크 공지
- ✅ IP 주소 충돌 감지 가능

8.5 IP 패킷 캡처 (이더넷 역다중화 확인)

```
Frame 4: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
Ethernet II, Src: aa:bb:cc:dd:ee:ff, Dst: 11:22:33:44:55:66
    Destination: 11:22:33:44:55:66
    Source: aa:bb:cc:dd:ee:ff
    Type: IPv4 (0x0800)  ← 이더넷 역다중화!
Internet Protocol Version 4, Src: 192.168.0.100, Dst: 192.168.0.101
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Total Length: 60
    Identification: 0x0001
    Time to live: 128
    Protocol: Unknown (253)  ← 사용자 정의 프로토콜
    Source: 192.168.0.100
    Destination: 192.168.0.101
Data (40 bytes)
```

분석:

- ✅ EtherType: 0x0800 (IPv4) → IPLayer로 전달
- ✅ IP 헤더: 20바이트 (Version=4, IHL=5)
- ✅ TTL: 128
- ✅ Protocol: 253 (Custom)

8.6 캡처 화면 예시 구성

필수 캡처 항목:

1. **ARP Request** (브로드캐스트)
    - Wireshark 패킷 리스트
    - Ethernet 헤더 펼친 상태
    - ARP 헤더 펼친 상태
    - Hex Dump
2. **ARP Reply** (유니캐스트)
    - Opcode=2 확인
    - Sender/Target 정보
3. **Gratuitous ARP**
    - Sender IP = Target IP 확인
4. **IPv4 Packet**
    - EtherType 0x0800 확인
    - IP 헤더 20바이트 확인
5. **필터 적용 화면**
    - `arp or ip` 필터 표시

---

9. 결론

9.1 구현 완료 항목

✅ IP 계층

- IPv4 헤더 20바이트 정확히 구현
- ARP와 연동하여 IP→MAC 변환
- 목적지 IP 기반 필터링

✅ ARP 계층

- ARP Request/Reply 정상 동작
- ARP 캐시 테이블 관리 (ConcurrentHashMap)
- Gratuitous ARP 구현 (네트워크 진입 공지)
- Proxy ARP 구현 (다른 호스트 대리 응답)

✅ **이더넷 역다중화**

- EtherType 0x0800 → IPLayer
- EtherType 0x0806 → ARPLayer
- instanceof 타입 체크로 계층 분리

✅ **GUI 구현**

- ARP 캐시 테이블 실시간 표시 (JTable)
- ARP 기능 버튼 (Request, Gratuitous, Proxy)
- 네트워크 설정 패널
- 메시지 송수신 인터페이스

9.2 학습 내용

1. **계층 구조의 이해**
    - 각 계층의 역할과 책임
    - 계층 간 인터페이스 설계 (BaseLayer)
    - 데이터 캡슐화/역캡슐화 과정
2. **ARP 프로토콜 동작 원리**
    - IP-MAC 주소 변환 메커니즘
    - 브로드캐스트 vs 유니캐스트
    - ARP 캐시의 중요성
3. **이더넷 역다중화**
    - EtherType 필드의 역할
    - 프로토콜 분리 메커니즘
    - instanceof를 활용한 타입 기반 라우팅
4. **네트워크 프로그래밍**
    - jNetPcap을 이용한 Raw 소켓 프로그래밍
    - ByteBuffer를 이용한 패킷 직렬화/역직렬화
    - 멀티스레드 환경에서의 동기화 (ConcurrentHashMap)

9.3 개선 가능한 부분

1. **IP Checksum 계산**
    - 현재는 0으로 설정
    - RFC 791에 따른 정확한 Checksum 계산 추가 필요
2. **ARP 캐시 타임아웃**
    - 현재는 무한정 저장
    - TTL 기반 자동 삭제 메커니즘 추가
3. **재전송 메커니즘**

- ARP Request 실패 시 재시도
- IP 패킷 전송 실패 시 큐잉 및 재전송

4. **IPv6 지원**
   - 현재는 IPv4만 지원
   - NDP (Neighbor Discovery Protocol) 구현

9.4 참고 문헌

- **RFC 826:** An Ethernet Address Resolution Protocol (ARP)
- **RFC 791:** Internet Protocol (IP)
- **IEEE 802.3:** Ethernet Standard
- **jNetPcap Documentation:** https://github.com/slytechs-repos/jnetpcap-wrapper

---

파일 구조

```
third/
├── lib/
│   └── jnetpcap-wrapper-2.3.1-jdk21.jar
├── src/main/java/com/demo/
│   ├── BaseLayer.java           # 계층 인터페이스
│   ├── PhysicalLayer.java       # 물리 계층 (jNetPcap)
│   ├── EthernetLayer.java       # 이더넷 계층 + 역다중화
│   ├── ARPLayer.java            # ARP 프로토콜 구현
│   ├── IPLayer.java             # IP 계층 (20바이트 헤더)
│   ├── ChatAppLayer.java        # 응용 계층
│   ├── ARPChatApp.java          # GUI 메인 프로그램
│   └── BasicChatApp.java        # 레거시 프로그램
├── run_arp_chat.sh              # 실행 스크립트
├── README.md                    # 프로젝트 설명
├── ARP_README.md                # 상세 문서
├── IMPLEMENTATION_SUMMARY.md    # 구현 요약
└── TESTING_GUIDE.md             # 테스트 가이드
```