Socket Programming Project #3

TCP/UDP 소켓 프로그래밍을 활용한 채팅 Application 개발
3단계 : 주변 device 인식 가능(ARP Table Scanning)

2019134006 김준섭
2019161011 박수연
tekcos

학정번호-분반 : CSI4106.02-00

목차

Ⅰ - 개요

Python을 이용, TCP, UDP 소켓프로그래밍을 활용한 채팅 application 개발을 목표로 프로젝트를 수행했다. 프로젝트 목표 3단계에 해당하는 주변 device 인식을 구현하였다.

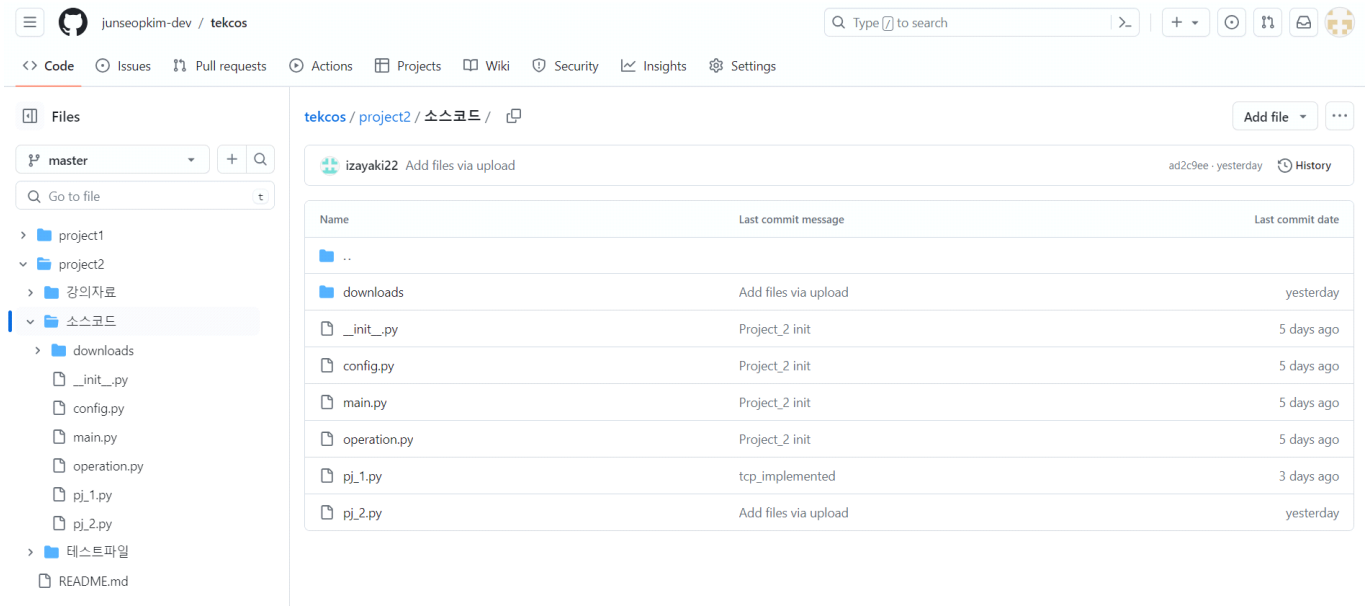
Ⅱ 프로젝트 진행 상황

1. 현황 요약

단계	핵심 구현 기능 목표	완료 여부 (마감 일자)
1	문자열 전송 기능	완료 (2023.10.30)
2	파일 전송 기능	완료 (2023.11.21)
3	주변 device 인식 가능	완료 (2023.12.20)

2. 역할 분담
   a. 김준섭 : Git 관리, 코드 수정 및 테스트, 보고서 작성
   b. 박수연 : Git 관리, 코드 작성 및 테스트, 스크린샷 캡처, 보고서 검토

3. 협업을 위한 Github repository 관리
   a. Github 주소 : https://github.com/junseopkim-dev/tekcos



Ⅲ 사전지식

1. 네트워크 계층구조

유지관리 및 시스템 업데이트를 쉽게 하고자 네트워크 시스템은 계층구조를 이룬다.

크게 2가지의 모델이 존재한다 : Internet Protocol Stack, ISO/OSI Reference Model

a. Internet Protocol Stack : 5개의 계층으로 구성

(1) Application : Supporting network applications.
ex) FTP, SMTP, HTTP

(2) Transport : Process-Process data transfer.
ex) **TCP, UDP**

(3) Network : Routing of datagrams from source to destination.
ex) IP, Routing Protocols

(4) Link : Data transfer between neighboring entwork elements.
ex) Ethernet, 802.111(Wifi), PPP

(5) Physical : Move the individual bits within the frame from one node to the next.

b. ISO/OSI Reference Model : 7개의 계층으로 구성

(1) Application : Supporting network applications.
ex) FTP, SMTP, HTTP

(2) Presentation : Allow applications to interpret meaning of data

(3) Session : Synchronization, checkpointing, recovery of data exchange

(4) Transport : Process-Process data transfer.
ex) **TCP, UDP**

(5) Network : Routing of datagrams from source to destination.
ex) IP, Routing Protocols

(6) Link : Data transfer between neighboring entwork elements.
ex) Ethernet, 802.111(Wifi), PPP

(7) Physical : Move the individual bits within the frame from one node to the next.

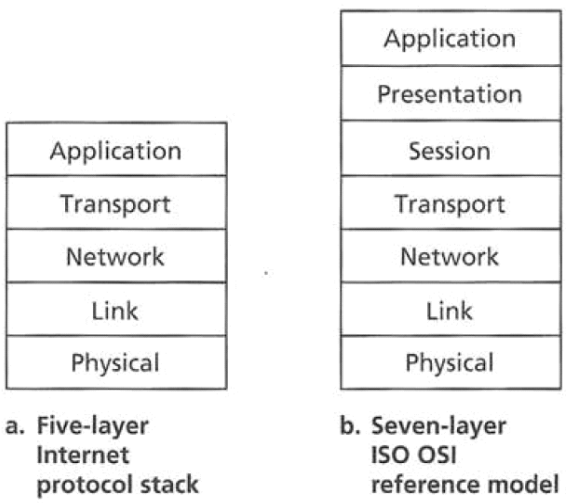


Figure 1.23 ♦ The Internet protocol stack (a) and OSI reference model (b)

2. Transport Layer Protocol : TCP & UDP

a. TCP (Transmission Control Protocol)

(1) 특징 :

(가) reliable (rdt 1.0, 2.0, 2.1, 2.2, 3.0 등)

(나) in-order delivery

(다) 연결 지향적 프로토콜 (3-way handshake)

(2) 장단점

(가) 장점 : rdt 3.0을 통해 높은 신뢰성을 보장한다.

(나) 단점 : UDP보다 속도가 느리다.

b. UDP (User Datagram Protocol)

(1) 특징 :

(가) unreliable

(나) unordered delivery

(다) 비연결형 프로토콜

(2) 장단점

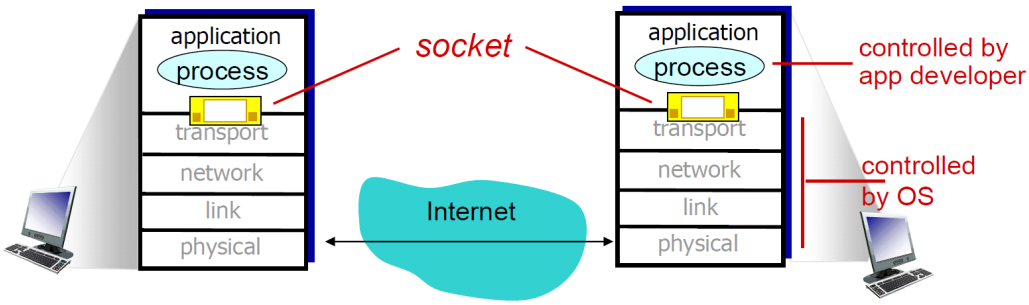
(가) 장점 : 빠른 속도로 전송 가능하다.

(나) 단점 : 신뢰성을 보장하지 못한다.

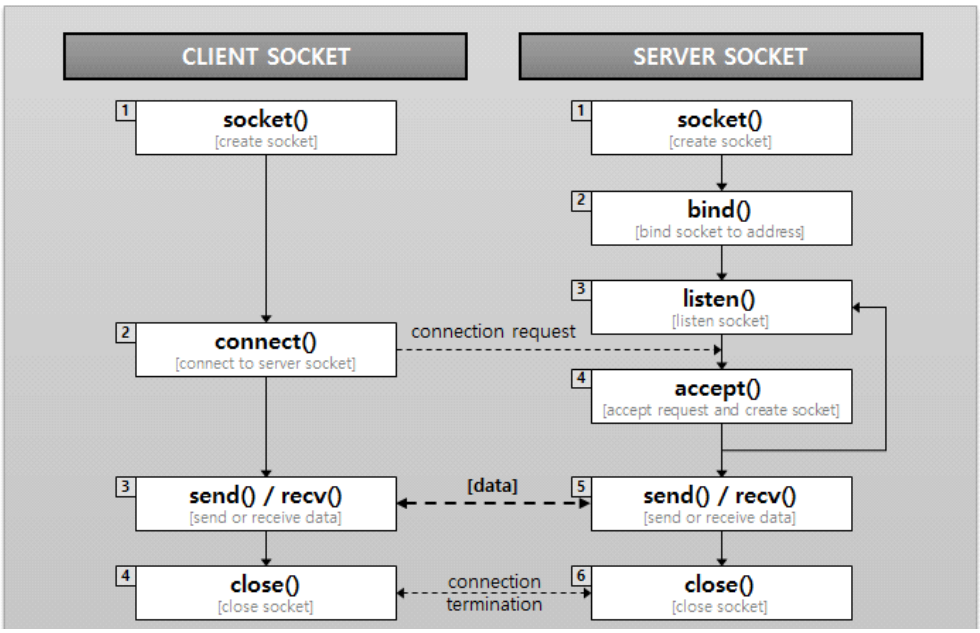
3. Socket

a. A process sends messages into, and receives messages from, the network through a software interface called a **socket**.

b. 프로세스는 집, 소켓은 문에 비유할 수 있다.

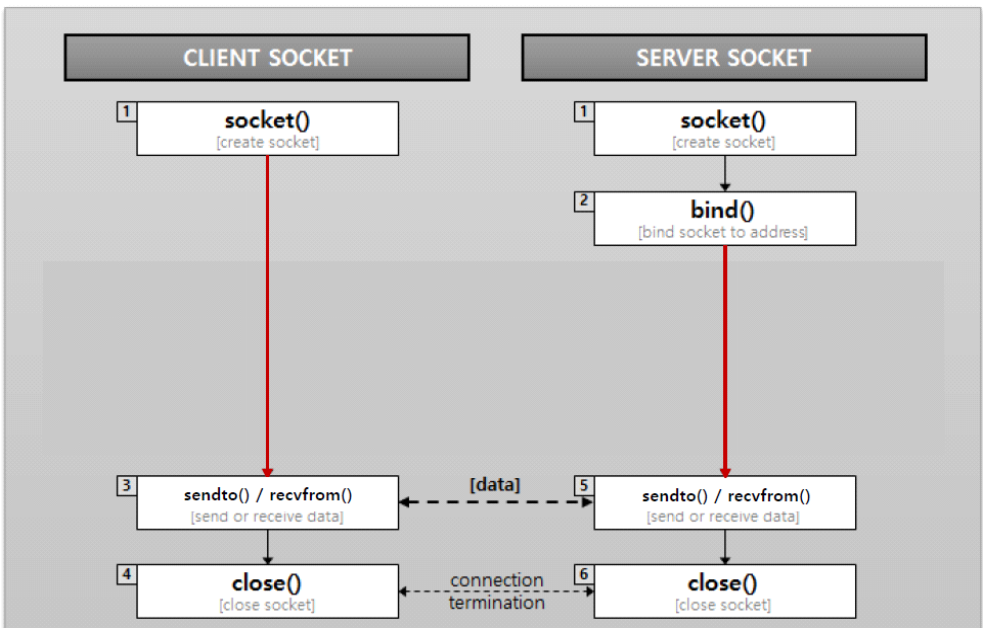


c. Socket Programming with TCP



(1) TCP는 연결 지향성 프로토콜이므로, 데이터 패킷을 주고받기 전에 서로를 연결하는 절차가 필요하다. (3-way handshake)
(2) 서로 데이터를 주고받기 전 서버는 listen()을 통해 클라이언트로부터의 연결요청이 오기를 대기한다.
(3) 서버가 listening하는 사이, 클라이언트는 connect()를 통해 서버에게 연결요청을 보낸다.
(4) 클라이언트로부터 연결요청이 온 경우, 서버는 accept()를 통해 연결을 허가하는 절차를 거친다.
(5) 서버-클라이언트 간 연결이 수립된 이후, send()/recv()를 통해 서로 데이터를 주고받을 수 있게 된다.

d. Socket Programming with UDP



(1) UDP는 TCP와는 달리 비연결형 프로토콜로, 이 때문에 TCP와는 달리 클라이언트의 connect(). 서버의 listen(), accept()가 존재하지 않는다.
(2) 서버-클라이언트 간 sendto(), recvfrom()을 통해 data를 주고받는다.

4. Reliable Data Transfer

줄여서 RDT라고 부르며, 데이터를 전송하는 과정에서 손실, 중복, 오류 등이 발생할 수 있는 불안정한 네트워크 환경에서도 안정적으로 신뢰성있는 데이터 전송을 가능케 하는 기술

a. rdt1.0

(1) 특징 : 통신이 완전히 신뢰할 수 있는 경우, Bit error, loss of packet 없음

(2) FSM

a. rdt1.0: sending side

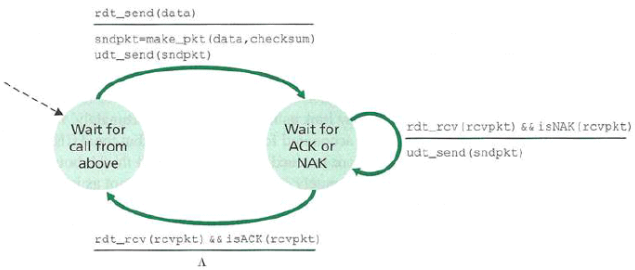
b. rdt1.0: receiving side

Figure 3.9 ♦ rdt1.0 – A protocol for a completely reliable channel

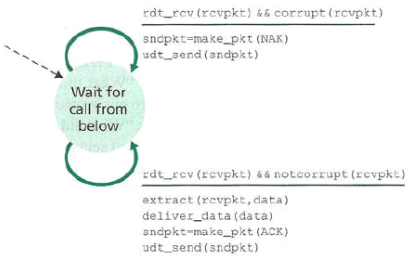
b. rdt2.0

(1) 특징 : 패킷이 전송되는 과정에서 bit error가 발생할 수 있는 경우 보완

(2) FSM



a. rdt2.0: sending side



b. rdt2.0: receiving side

Figure 3.10 ♦ rdt2.0 – A protocol for a channel with bit errors

c. rdt2.1

(1) 특징 : rdt 2.0에서 ACK/NAK에 손상이 발생할 경우 보완

(2) FSM

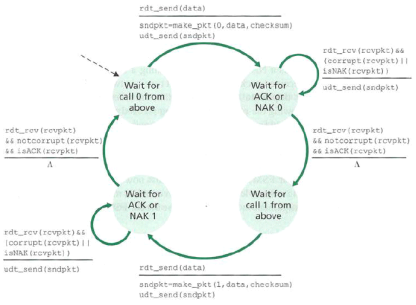


Figure 3.11 ♦ rdt2.1 sender

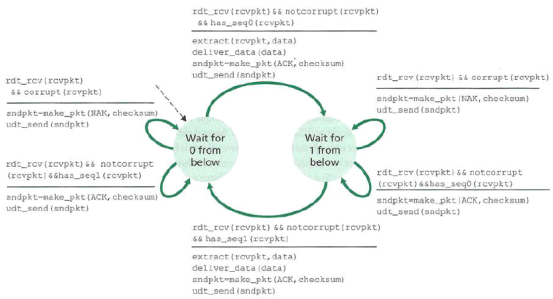


Figure 3.12 ♦ rdt2.1 receiver

d. rdt2.2

(1) 특징 : rdt 2.1과 달리 ACK만을 사용

(2) FSM

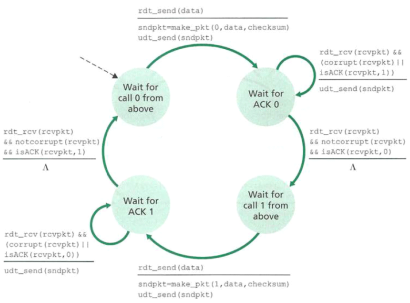


Figure 3.13 ♦ rdt2.2 sender

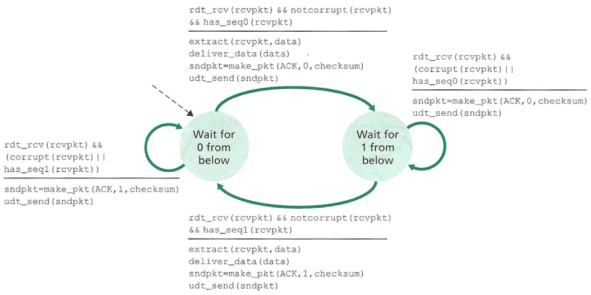


Figure 3.14 ♦ rdt2.2 receiver

e. rdt3.0

(1) 특징 : 패킷 에러 뿐만 아니라 소실(loss)이 발생할 경우 보완

(2) FSM

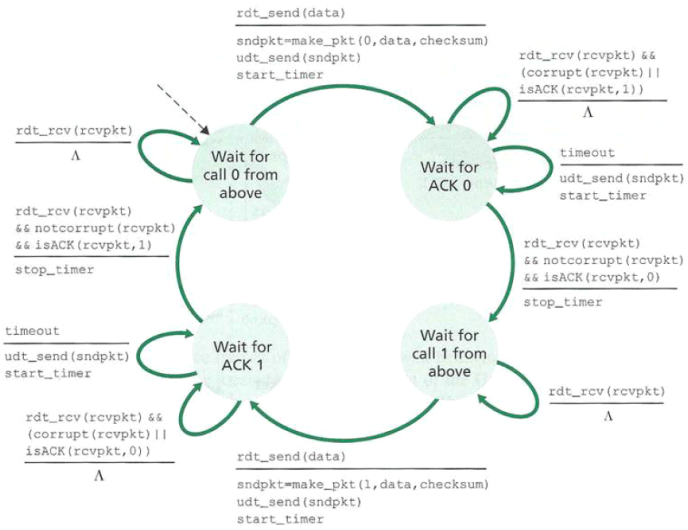


Figure 3.15 ♦ rdt3.0 sender

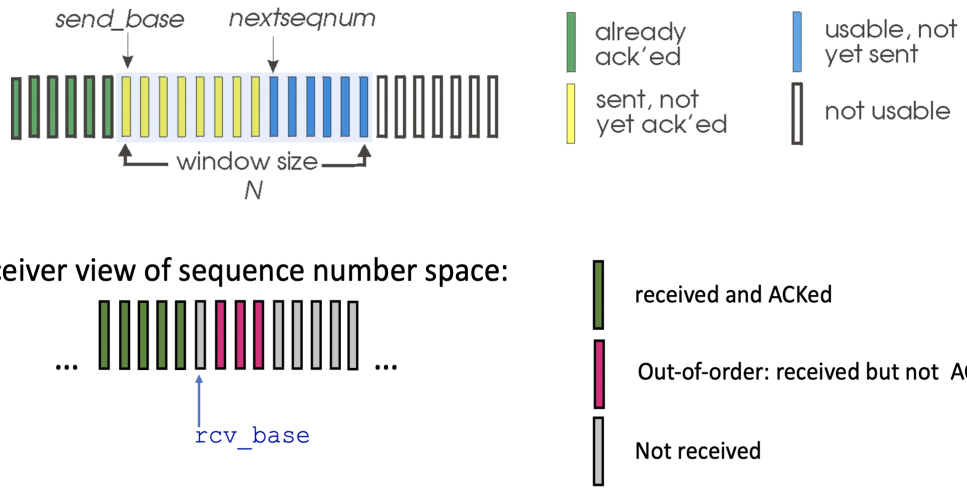
5. Pipelining

패킷을 보낸 후 응답이 올 때까지 기다리는 stop and wait 방식 특징상 느린 속도의 단점을 극복하고자 패킷을 보낸 뒤 기다리지 않고 바로 다음 패킷을 보내는 방식. 크게 2가지 방법이 존재한다.

* Window size는 (사용하는 sequence number /2) 이하여야 한다.

a. Go-Back N (GBN)

(1) 수신자가 어떤 패킷을 받지 못하면 그 패킷부터 모든 패킷을 다시 보내는 방법이다.



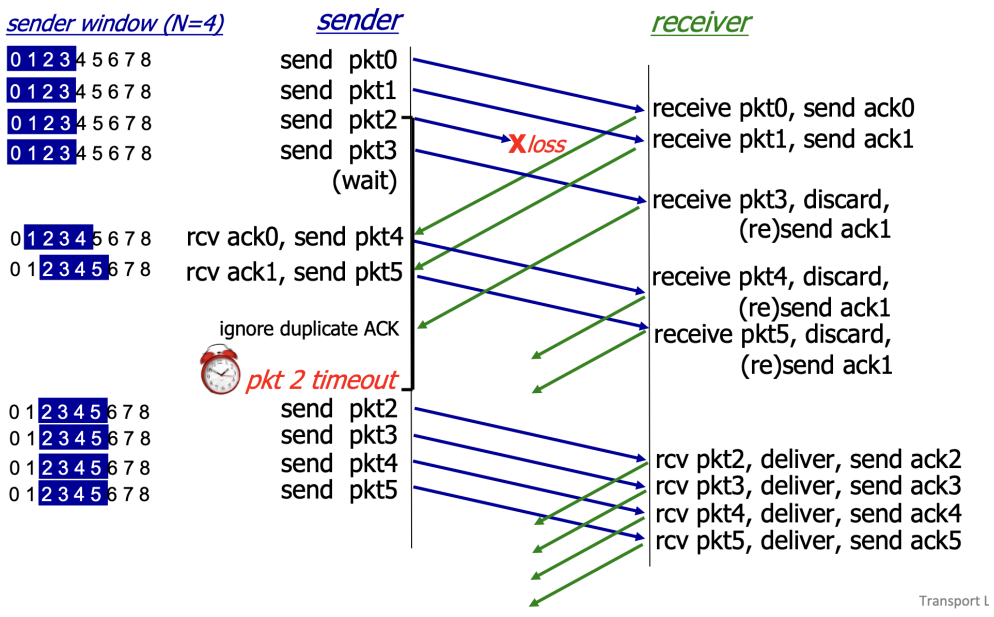
(2) 송신자는 ACK를 받지 않은 상태로 동시에 최대로 보낼 수 있는 패킷의 수를 정하는데, 이를 window size라고 하며, 현재 window 위치를 설정해 window size만큼의 범위를 설정한다.

(3) 송신자는 보낼 패킷들이 들어있는 버퍼의 시작부터 window의 범위를 설정하고, sequence number가 그 window의 범위 안에 들어 있는 패킷을 모조리 보낸다.

(4) 수신자는 패킷들을 연속해서 받고, 각각의 패킷을 받을 때마다 sequence number를 확인하고, ACK를 보낸다.

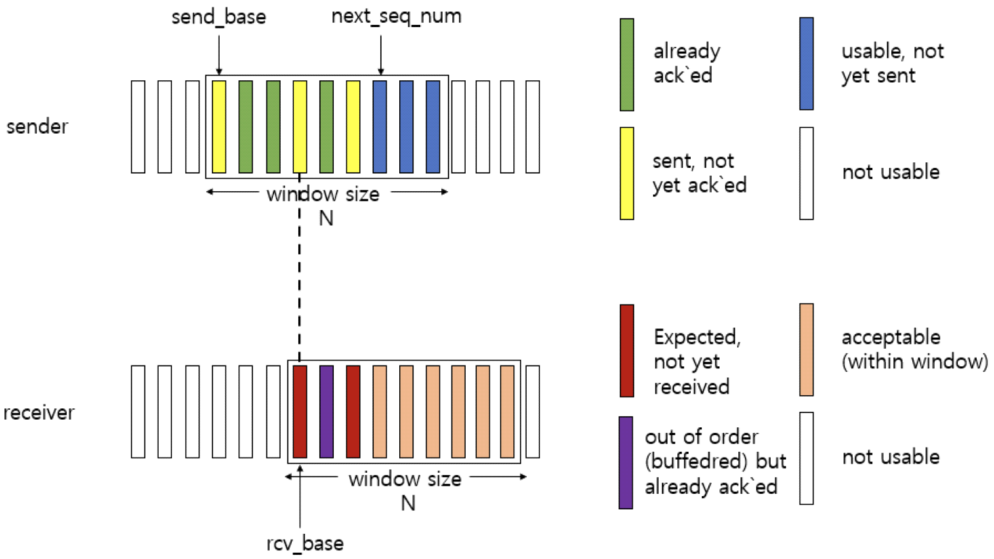
(5) 이 때, 송신자가 보낸 패킷들 중에서 일부가 유실될 수 있다. 그러면 수신자가 받은 패킷들의 sequence number에 빈틈이 생기게 된다. 이 때, 빈틈이 생기기 전에 정상적으로(연속적으로) 받은 패킷의 마지막 번호를 지속적으로 ACK에 실어서 보낸다.

(6) 송신자는 내가 보낸 패킷의 ACK가 정상적으로 왔다면, 그 패킷을 처리됐다고 기록한 후, window를 오른쪽으로 한 칸 밀어서 하나의 패킷을 더 보낼 수 있는 상태로 만든다.



b. Selective Repeat (SR)

(1) 수신자가 받은 각각의 패킷에 대해 ACK를 보내는 방식이다.



(2) 송신자는 GBN 방식과 마찬가지로 window를 가지고 있으며, window 범위 안에 들어오는 패킷들을 다 보내고 나면, 각각의 패킷에 대해 ACK을 기다린다.

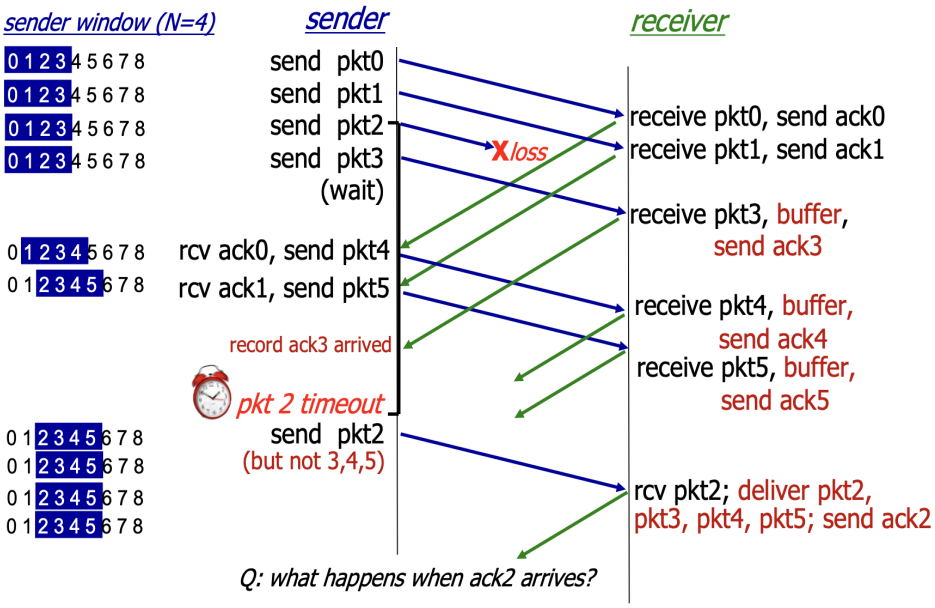
(3) 각각의 패킷에 대해 타이머를 가지고 있고, 일정 시간이 지나도 ACK 가 도착하지 않으면 timeout을 발생시켜 해당 패킷만을 다시 보낸다.

(4) 특정 패킷에 대해 ACK가 들어오면 그 패킷을 완료되었다고 기록한다. 송신자가 보낸 패킷들 중 아직 ACK되지 않은 패킷들 중 가장 번호가 작은 패킷이 ACK되면 window를 한 칸 이동시킨다.

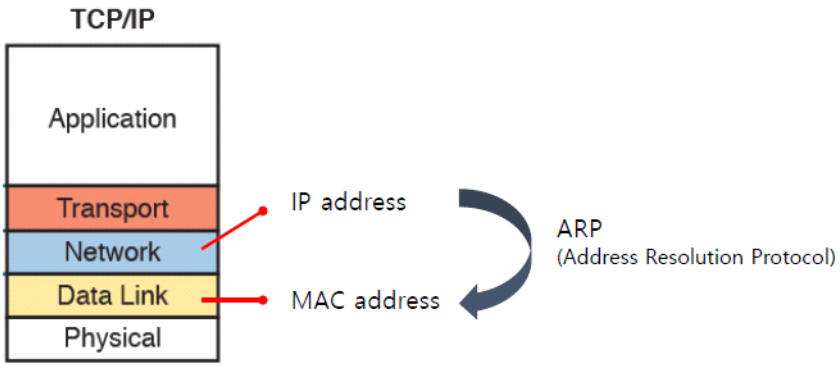
(5) 수신자는 window와 버퍼를 가지고 있다. 어떤 패킷을 받았을 때 이 패킷이 순서(rcv_base)에 어긋난다면 일단 버퍼에 저장해 둔다.

(6) 순서에 맞는 패킷이 들어오면 버퍼에 있는 패킷들까지 모조리 ACK을 한꺼번에 보낸다. 그리고 ACK를 보낸 패킷 수만큼 window를 이동한다.

(7) window의 범위보다 이전에 속하는 패킷이 도달한다면, ACK에 오류가 생겨 재전송된 패킷이라 간주하고 ACK만 보내고 별다른 처리는 하지 않는다.



6. IP Address and MAC Address



a. IP Address
   (1) Internet Protocol Address
   (2) IPv4는 콜론(.)으로 구분된 32비트 주소, IPv6는 16진수로 구성된 128비트 주소
   (3) Network layer에서의 주소
   (4) Host가 움직이면(다른 LAN으로 이동시) 변경됨

b. MAC Address
   (1) Media Access Control Address
   (2) 하이픈(-)혹은 콜론(:)으로 구분된 48비트 주소. 2개의 16진수가 6개로 구성됨.
   (3) Link layer에서의 주소
   (4) Network Interface에 할당된 고유 식별 주소
   (5) 다른 LAN으로 이동해도 변경되지 않음

7. Address Resolution Protocol(ARP)

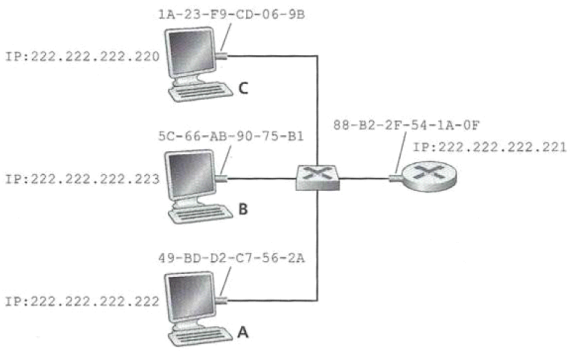


Figure 6.17 ♦ Each interface on a LAN has an IP address and a MAC address

Neighbour	Linklayer Address	Expire (O)	Expire (I) Netif
10.0.0.1	0:c:29:12:34:56	2m25s	2m25s
10.0.0.41	0:c:29:65:43:21	39s	39s
10.0.0.47	0:c:00:00:00:01	1m14s	1m14s
10.0.0.53	ab:cd:ef:81:6:72	2m32s	2m32s
10.0.0.58	fe:dc:ba:11:e2:6e	1m51s	1m51s

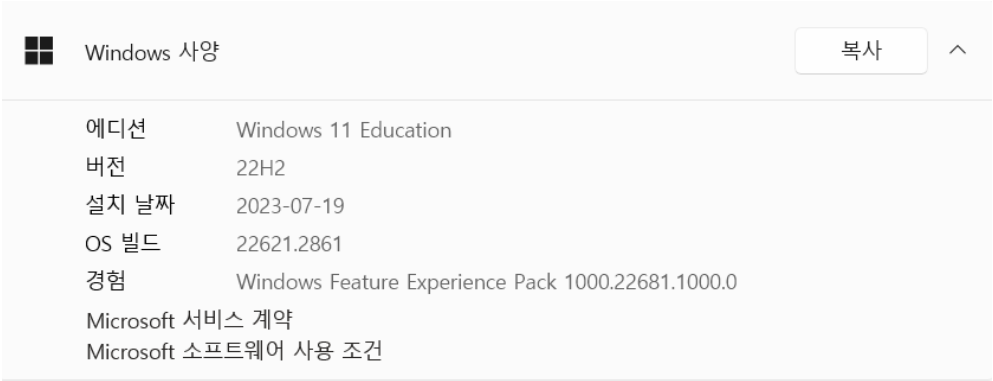
ARP Table

a. 목적 : 동일한 LAN에 있는 host간 datagram 전송시 destination의 IP 주소를 MAC address로 변환하는 프로토콜
   IP 주소를 MAC 주소와 매칭
b. Host는 destination node의 IP주소와 MAC 주소가 담긴 frame을 보낸다.
c. ARP Table을 통해 IP 주소와 MAC 주소를 알맞게 대응시킨다.
d. ARP Table
   (1) IP주소와 MAC 주소를 일대일 대응시킨 목록이 저장된 테이블.
   (2) Host와 Router는 ARP Table을 가지고 있다.
   (3) 만약 ARP Table이 destination node의 entry를 갖고 있지 않을 경우, Broadcast를 수행한다.

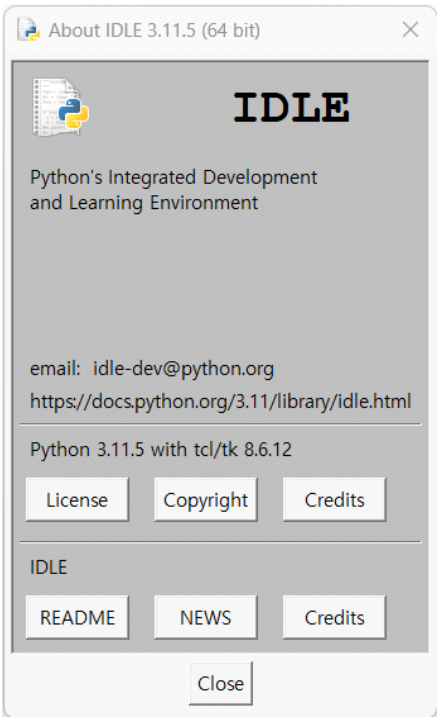
Ⅳ 구현

1. 구현 목표 : 파이썬의 외부 라이브러리 netifaces, psutil, scapy를 활용해 동일 subnet 내 기기들의 IP Address 확인

2. 구현 환경
   a. OS 정보
      (1) 에디션 : Windows 11 Education
      (2) 버전 : 22H2
      (3) 설치 날짜 : 2023-07-19
      (4) OS 빌드 : 22621.2861
      (5) 경험 : Windows Feature Experience Pack 1000.22681.1000.0



   b. Python 버전 : 3.11.5



3. 구현 코드 : pj_1.py, pj_2.py는 이전과 동일, pj_3.py에서 변경이 이루어졌다.
   a. pj_3.py : 외부 라이브러리 netifaces, psutil, scapy를 이용해 주변 기기 IP 주소 확인 기능을 구현
      (1) 전체 코드

```
from scapy.all import srp,Ether,ARP,conf
import netifaces,psutil
from threading import Thread


class ARPTable:
    def __init__(self)->None:
        self.ARP_table =list()
        self.interface =None

    def get_ARP_table(self,interface:str,ips:str)->int:
        # interface: 네트워크 인터페이스의 이름 ex) en0, wl0, 이더넷 등
        # ips: 탐색할 ip의 범위 ex) 192.168.0.1/24는 192.168.0.0 ~ 192.168.0.255까지 256개 ip
범위를 탐색
        # interface와 ips는 ARP scanning 창으로부터 사용자의 입력값을 받아서 설정됨

        self.ARP_table =list()
        self.interface =interface

        # todo: scapy의 all verbose를 show하도록 설정하고,
        # todo: scapy의 srp를 사용해 ARP response를 get

        conf.verb =1

        ans,_=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ips),iface=interface,timeout=2,inter =0.1)

        for snd,rcv in ans:
            # todo: arp response (ans)로부터 ip address와 mac address를 get
            ip_addr =rcv[ARP].psrc
            mac_addr =rcv[Ether].hwsrc
            self.ARP_table.append((ip_addr,mac_addr))

    def default_ip_nif(self):
        # gateway의 IP address와 네트워크 어댑터(네트워크 인터페이스)의 이름을 가져온다.
        default =netifaces.gateways()['default']
        gateway_ip,nif =default[list(default.keys())[0]]
        if nif[0]=='{':
            # windows
            ip_info =netifaces.ifaddresses(nif)
            ip_addr =ip_info[netifaces.AF_INET][0]['addr']
            ps_if_addrs =psutil.net_if_addrs()
            exit_point =False
            for key in ps_if_addrs:
                for adress_family in ps_if_addrs[key]:
                    if adress_family.family ==netifaces.AF_INET:
                        if adress_family.address ==ip_addr:
                            nif =key
                            exit_point =True
                            break
                    if exit_point:
                        break

        return gateway_ip,nif
```

b. 세부 설명

(1) 함수 __init__(self)

함수 __init__

```
def __init__(self)->None:
    self.ARP_table =list()
    self.interface =None
```

설명

요약 :
ARPTable 클래스의 생성자, 객체의 초기 상태 설정,
ARP 테이블 저장하고 네트워크 인터페이스 참조할 준비

```
self.ARP_table =list()
```
ARP_table이라는 빈 리스트를 초기화.
향후 이 리스트에 네트워크 장치들의 IP주소와 MAC 주소 저장

```
self.interface =None
```
'interface'라는 변수를 None으로 초기화
향후 특정 네트워크 인터페이스 참조에 사용

(2) 함수 get_ARP_table(self,interface:str,ips:str)

함수 get_ARP_table(self,interface:str,ips:str)

```
def get_ARP_table(self,interface:str,ips:str)->int:

    self.ARP_table =list()
    self.interface =interface

    conf.verb =1

    ans,_=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ips),iface=interface,timeout=2,inter =0.1)

    for snd,rcv in ans:
        ip_addr =rcv[ARP].psrc
        mac_addr =rcv[Ether].hwsrc
        self.ARP_table.append((ip_addr,mac_addr))
```

설명

요약 :
네트워크 스캔을 수행하여 각 장치의 IP와 MAC 주소 찾아내고, 이 정보를 ARP_table 리스트에 저장

```
self.ARP_table =list()
```
새로운 스캔 시작 전 이전 데이터 지우기 위해 ARP_table 리스트 비움

```
self.interface =interface
```
클래스 인스턴스 변수 interface를 사용자가 입력한 인터페이스 이름으로 설정

```
conf.verb =1
```
Scapy 라이브러리의 전역설정 변경, 작업 중 발생하는 모든 세부 정보 출력
scapy의 all verbose를 show하도록 설정

```
ans,_=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ips),iface=interface,timeout=2,inter =0.1)
```
Scapy의 'srp' 함수를 사용해 ARP 요청 전송.
dst="ff:ff:ff:ff:ff:ff = 브로드캐스트
pdst=ips : 스캔할 IP 범위 설정, iface : 사용할 네트워크 인터페이스 설정,
timeout : 응답 대기 시간 설정, inter : 요청 간의 간격 설정

```
for snd,rcv in ans:
```
srp 함수로부터 받은 응답 ans 순회.

```
    ip_addr =rcv[ARP].psrc
```
수신된 패킷에서 ARP 프로토콜을 사용해 소스 IP 주소 추출

```
    mac_addr =rcv[Ether].hwsrc
```
수신된 패킷에서 Ethernet 레이어를 사용해 소스 MAC 주소를 추출

```
    self.ARP_table.append((ip_addr,mac_addr))
```
추출한 IP 주소와 MAC 주소를 ARP_table 리스트에 추가

(3) 함수 default_ip_nif(self)

함수 default_ip_nif(self)

```
    def default_ip_nif(self):
        # gateway의 IP address와 네트워크 어댑터(네트워크 인터페이스)의 이름을 가져온다.
        default =netifaces.gateways()['default']
        gateway_ip,nif =default[list(default.keys())[0]]
        if nif[0]=='{':
            # windows
            ip_info =netifaces.ifaddresses(nif)
            ip_addr =ip_info[netifaces.AF_INET][0]['addr']
            ps_if_addrs =psutil.net_if_addrs()
            exit_point =False
            for key in ps_if_addrs:
                for adress_family in ps_if_addrs[key]:
                    if adress_family.family ==netifaces.AF_INET:
                        if adress_family.address ==ip_addr:
                            nif =key
                            exit_point =True
                            break
                    if exit_point:
                        break

        return gateway_ip,nif
```

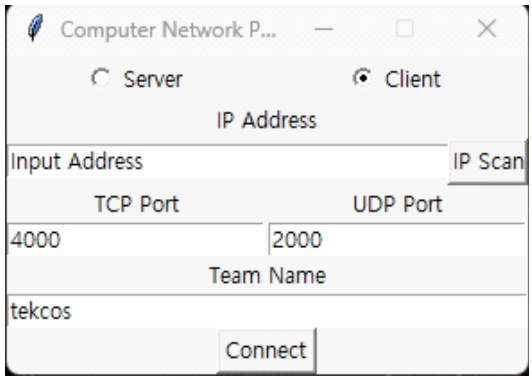
설명

요약 : 전송할 데이터를 패킷으로 변환

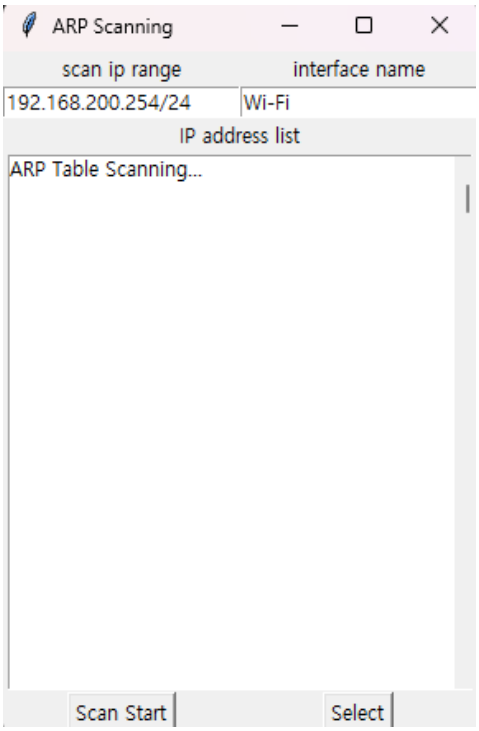
```
default =netifaces.gateways()['default']
        netifaces 모듈을 사용해 시스템의 기본 게이트웨이 정보 가져옴
gateway_ip,nif =default[list(default.keys())[0]]
        기본 게이트웨이의 IP주소와 해당 게이트웨이를 사용하는 네트워크 인터페이스 이름 추출

if nif[0]=='{':
        네트워크 인터페이스의 이름이 중괄호'{'로 시작하는 지 확인,
        (Windows 시스템에서 네트워크 인터페이스 특징)
    ip_info =netifaces.ifaddresses(nif)
        해당 네트워크 인터페이스에 할당된 IP 주소 정보를 가져옴
    ip_addr =ip_info[netifaces.AF_INET][0]['addr']
        IPv4 주소정보(AF_INET)를 사용해 실제 IP 주소 추출
    ps_if_addrs =psutil.net_if_addrs()
        psutill 모듈을 사용해 시스템의 모든 네트워크 인터페이스와 그 주소들 가져옴
    exit_point =False
        반복문 탈출을 위한 플래그 변수 초기화

for key in ps_if_addrs:
        모든 네트워크 인터페이스 순회하는 반복문
    for adress_family in ps_if_addrs[key]:
        특정 네트워크 인터페이스에 대한 주소 정보 순회
        if adress_family.family ==netifaces.AF_INET:
        주소 유형이 IPv4인지 확인
            if adress_family.address ==ip_addr:
        주소가 앞서 추출한 IP주소와 일치하는 지 확인
                nif =key
        일치하는 주소를 찾으면, 네트워크 인터페이스의 이름 업데이트
                exit_point =True
                break
        플래그를 설정하여 반복문 탈출
            if exit_point:
                break
        플래그를 체크하여 외부 반복문도 탈출

return gateway_ip,nif
        추출한 게이트웨이 IP주소와 네트워크 인터페이스 이름 반환
```

4. 정상 동작 스크린샷

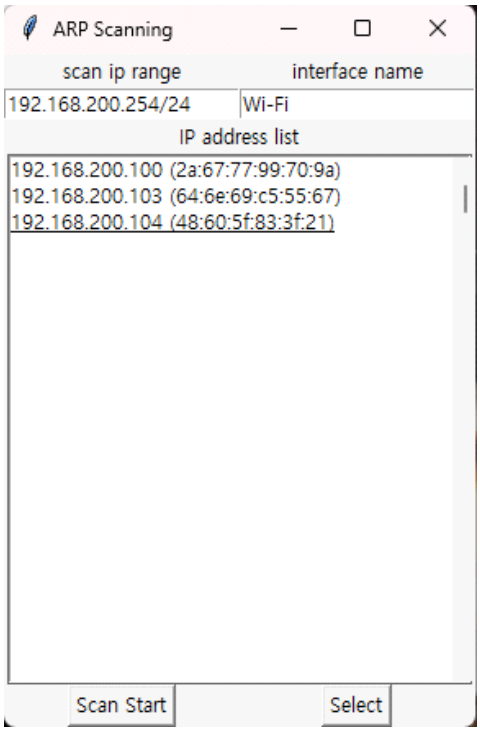
a. Scan 이전



b. Scan 수행 및 ARP Table 출력

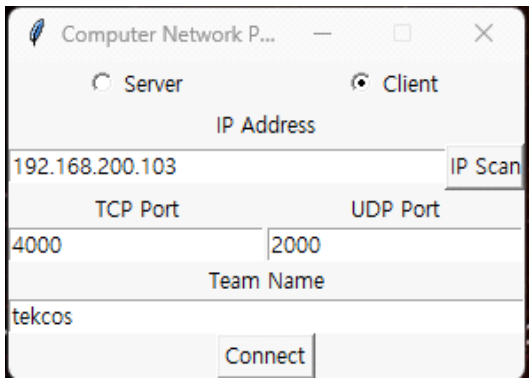


스캔 중

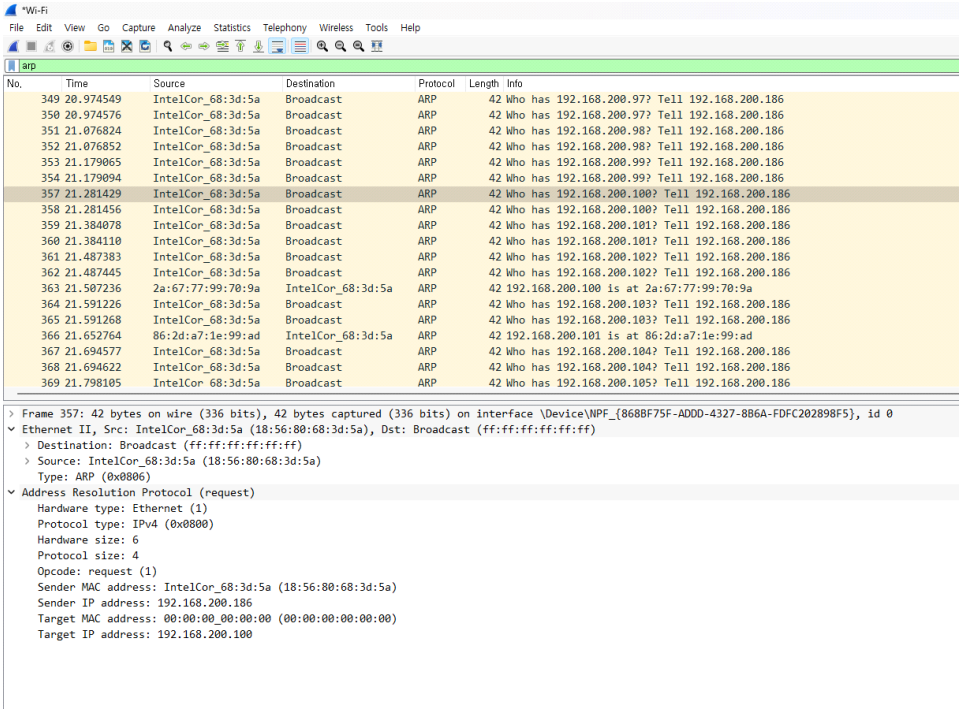


스캔 결과 ARP Table 출력

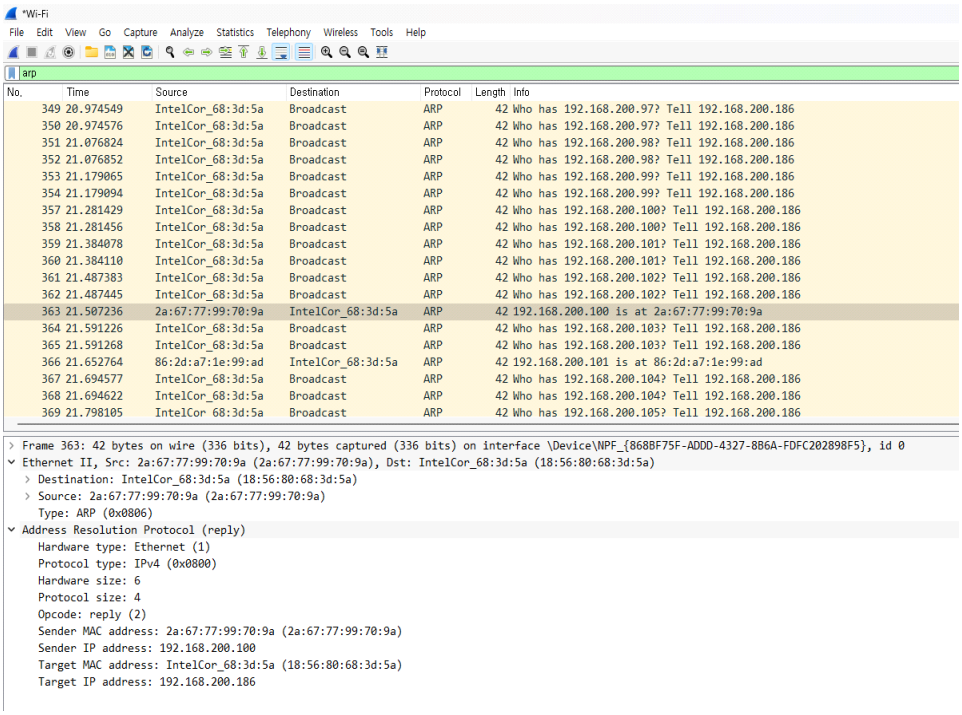
c. 스캔 후



5. Wireshark 사용해 sender(ARP packet 발생시킨 host)와 receiver(ARP packet 수신한 host) 각각의 ARP 패킷 관찰 스크린샷

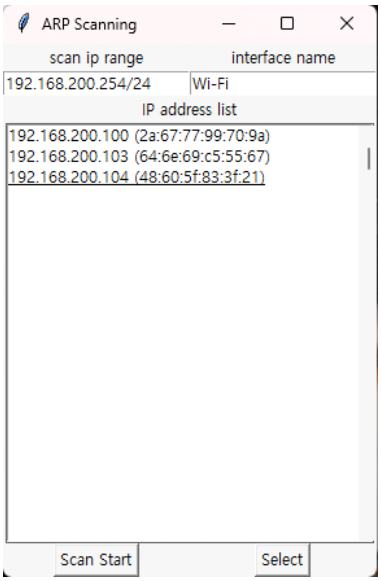


ARP Request

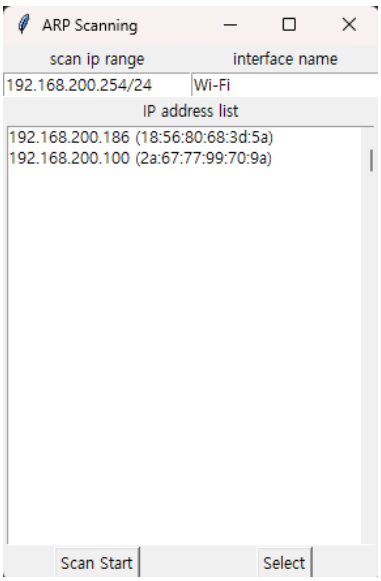


ARP Reply

6. Mobility에 따른 IP Address 및 ARP table 확인
   테더링(핫스팟)을 이용해 mobility로 인한 와이파이의 변화 구현
   a. 장소 A에 있을 때 (와이파이 A에 연결)
      (1) Wireless network interface를 disable 시키기 전
          = 와이파이 A에 연결된 상태



      (2) Wireless network interface를 disable 시킨 후 다시 enable 시켰을 때
          = 여전히 와이파이 A에 연결된 상태



      (3) 설명 : Disable 시키기 전과 Disable 시킨 후 다시 Enable 시킨 후를 비교했을 때, IP Address list의 목록은 조금씩 변화하더라도 서브넷 파트 (192.168.200.xxx)에는 변화가 없음을 관찰할 수 있다. Broadcasting 결과 IP Address list가 변화하더라도 여전히 서브넷 파트는 192.168.200.xxx로 동일하다.

b. 장소 A에서 B로 이동했을 때

(와이파이 A disabled 이후 와이파이 B로 연결)

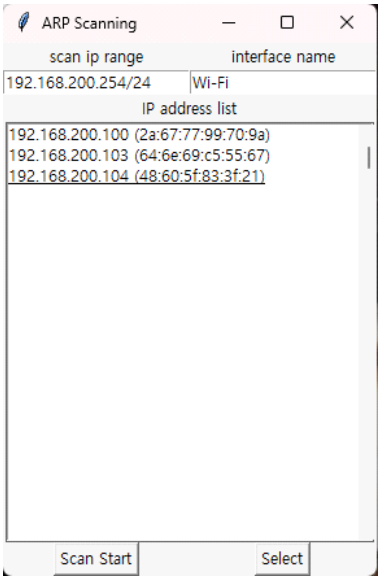
방법 1 : 물리적으로 다른 장소로 이동하여 새로운 공유기에 연결

방법 2 : 하나의 모바일 기기의 테더링을 활성화시켜 공유기로 설정 => 연결된 기기 중 하나의 기기를 모바일 기기와 거리가 멀어지게 하거나, 연결을 끊고 다른 공유기에 연결

본 팀은 이 중 방법2를 채택, 집 와이파이에서 테더링(핫스팟)으로 전환하여 수행하였다.

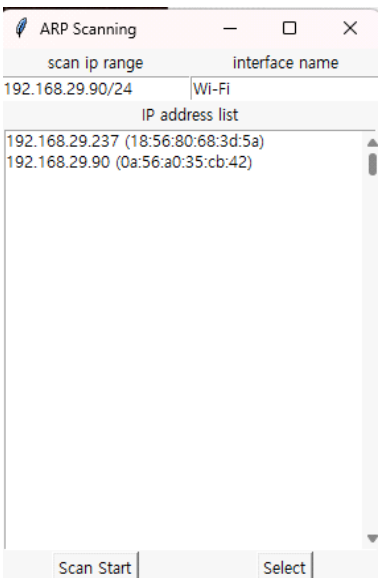
(1) Wireless network interface를 disable 시키기 전

= 와이파이 A에 연결된 상태



(2) 기존의 와이파이 A와의 연결을 해제하고 테더링(핫스팟)을 이용해 와이파이 B에 연결

= 와이파이 B로 변경된 상태



(3) 설명 : 처음 와이파이 A로 연결된 상태일 때에는 서브넷 파트가 192.168.200.xxx였지만, 와이파이 A에서 테더링(핫스팟)을 통해 와이파이 B로 변경되었을 때, 서프넷 파트가 192.168.29.xxx로 변경된 것을 확인할 수 있다. 서브넷 파트가 변화함에 따라 다시 Broadcasting을 수행하여 얻어진 IP Address list 역시 해당 서브넷 파트를 갖는 IP들의 목록으로 변화함을 관찰할 수 있다.

c. **장소 A에 있을 때와 장소 A에서 B로 이동했을 때(테더링으로 와이파이를 변경했을 때)의 IP address 및 ARP Table 변화에 대한 차이점 및 이유**

장소 A에서는 동일한 LAN 상에 있어 IP 서브넷이 유지되나, 장소 A에서 B로 이동하면 다른 LAN으로 옮겨져 IP 서브넷이 변경된다. 네트워크 변경시 DHCP 서버가 동적으로 새로운 IP주소를 할당해 IP 주소가 변경된다. ARP는 IP 주소를 MAC 주소와 매핑하는 프로토콜로, 네트워크 변경 시 새로운 IP와 이에 대응하는 MAC 주소를 찾고자 브로드캐스팅을 수행, 새로운 ARP Table을 생성한다.

V 고찰

1. 외부 라이브러리 netifaces, psutil, scapy를 사용해 ARP Table Scan을 통해 주변 기기의 IP Address 및 MAC Address를 확인하는 프로그램을 구현할 수 있었다.
2. 장소 A에 있을 때는, 즉 같은 와이파이가 유지되고 있는 상황에서는 비록 disable하고 다시 enable하더라도 여전히 Host가 같은 LAN에서 존재하기 때문에, IP는 동일한 서브넷 파트를 갖는다. ARP는 LAN에서 IP에 대응하는 MAC 주소를 동기화하고자 브로드캐스팅을 수행하여 ARP Table을 동기화한다.
3. 반면 장소 A에서 B로 변경될 때, 혹은 와이파이 A에서 테더링을 통해 와이파이 B로 변경할 때, Host가 다른 LAN으로 이동하게 되므로, IP의 서브넷 파트가 변화한다. ARP는 새로운 IP에 대응하는 MAC 주소를 찾기 위해 브로드캐스팅을 수행하며, 새로운 ARP Table을 형성한다.

VI 참고문헌

1. Kurose, J., & Ross, K. (2018, October 23). Computer Networking: A Top-Down Approach, Global Edition. Pearson Higher Ed.
2. 컴퓨터네트워크 강의자료 Chapter 1. Introduction
3. 컴퓨터네트워크 강의자료 Chapter 2. Application Layer (2.1-2.3)
4. 컴퓨터네트워크 강의자료 Chapter 3. Transport Layer (3.1-3.4)
5. 컴퓨터네트워크 강의자료 Chapter 3. Transport Layer (3.5-3.7)
6. 컴퓨터네트워크 강의자료 Chapter 4. The Data Plane (4.1-4.4)
7. 컴퓨터네트워크 강의자료 Chapter 6. The Link Layer and LANs (6.4-6.7)
8. 컴퓨터네트워크 강의자료 231011_Socket_Programming_Project1
9. 컴퓨터네트워크 강의자료 231101_Socket_Programming_Project2
10. 컴퓨터네트워크 강의자료 231129_Socket_Programming_Project3
11. 컴퓨터네트워크 강의자료 Socket_Programming_Project2 추가설명