hw2 과제 보고서

20기 인턴 송수민

- 목차

1. 물리 계층 (Physical Layer)
- 정의
비트를 전기적/광학적/무선 신호로 변환해 물리 매체를 통해 전송하는 계층.
하드웨어적 특성 규정 (케이블, 커넥터, 전압, 주파수 등).

- 작동 원리
0 → 낮은 전압, 1 → 높은 전압과 같이 신호를 매핑.
동기화(클럭 신호)에 따라 비트를 정확히 구분.

- 역할
단순히 데이터가 "흐르는 통로" 제공 (신뢰성 X).

- 하는 일
전송 매체 규격 정의 (UTP, STP, 광섬유, 무선).

- 전송 방식 정의:
단방향(Simplex), 반이중(Half-duplex), 전이중(Full-duplex).
변조/복조 (디지털 ↔ 아날로그 신호 변환).
전송 속도(Bandwidth) 및 전력 규정.

- 관련 장비
리피터(신호 증폭), 허브(단순 브로드캐스트), 케이블, 모뎀.

- 프로토콜/표준
RS-232, DSL, ISDN, IEEE 802.11(무선 LAN의 물리 계층 부분).

2. 데이터 링크 계층 (Data Link Layer)
- 정의

같은 네트워크 구간에서 오류 없는 프레임(Frame) 전송 보장.

- 작동 원리
물리 계층에서 올라온 비트 스트림을 프레임 단위로 묶음.
MAC 주소를 기반으로 수신 대상 식별.

- 역할
신뢰성 있는 점대점(point-to-point) 연결 제공.

- 하는 일
프레임화(Framing): 시작/끝 구분.
오류 검출/수정: CRC, 패리티 비트.
흐름 제어: 송신 속도 조절.
MAC(Media Access Control): 여러 장치가 공유 매체 접근 방법 규정.
LLC(Logical Link Control): 상위 계층에 데이터 전달, 오류 제어.

- 전송 단위
프레임(Frame).

- 주소
물리적 주소(MAC Address, 48bit).

- 관련 장비
스위치, 브리지.

- 프로토콜/표준
Ethernet(IEEE 802.3), Wi-Fi(IEEE 802.11), PPP, HDLC.

3. 네트워크 계층 (Network Layer)
- 정의

서로 다른 네트워크 간 패킷(Packet)을 최종 목적지까지 전달.

- 작동 원리
IP 주소를 이용한 논리적 경로 지정.
라우터가 최적 경로를 계산 후 전달.

- 역할
경로 선택(Routing), 논리적 주소 지정, 패킷 전달.

- 하는 일
라우팅(Routing): RIP, OSPF, BGP.

- 논리 주소 지정: IPv4/IPv6.
패킷 단편화/재조립: MTU 초과 시 분할.

- 혼잡 제어: 네트워크 부하 관리.

- 전송 단위
패킷(Packet).

- 주소
논리 주소(IP Address).

- 관련 장비
라우터, L3 스위치.

- 프로토콜/표준
IP(v4/v6), ICMP(진단/에러 메시지), ARP(IP↔MAC 변환).

4. 전송 계층 (Transport Layer)

- 정의
양 끝단(End-to-End) 간 신뢰성 있는 데이터 전송 보장.

- 작동 원리
TCP: 연결 지향, 신뢰성 보장.
UDP: 비연결성, 빠르지만 신뢰성 낮음.

- 역할
데이터 분할, 오류/흐름 제어, 프로세스 간 통신 제공.

- 하는 일
세그먼트화(Segmentation).
연결 설정/종료 (3-Way, 4-Way Handshake).
오류 제어 (ACK, 재전송).
흐름 제어 (Sliding Window).
혼잡 제어 (Slow Start, Congestion Avoidance).
포트 번호 할당 (Well-known Ports: 80, 443 등).

- 전송 단위
세그먼트(Segment, TCP) / 데이터그램(Datagram, UDP).

- 주소
포트 번호 (16bit).

- 관련 장비
게이트웨이(Transport Gateway).

- 프로토콜/표준
TCP, UDP, SCTP.

5. 세션 계층 (Session Layer)
정의
응용 간 통신 세션(Session)을 설정/관리/종료.

- 작동 원리
대화 제어(Dialog Control) 기능 제공.
동기점(Checkpoint) 설정 가능.

- 역할
연결 유지 및 중단 시 재개 지원.

- 하는 일
세션 생성/유지/종료.
대화 순서 제어 (토큰 관리).
동기화 기능 (중단 후 이어받기).

- 전송 단위
데이터(Data).

- 관련 장비
게이트웨이, 프록시.

- 프로토콜/표준
NetBIOS, RPC, PPTP, SOCKS.

6. 표현 계층 (Presentation Layer)
- 정의
데이터의 형식, 부호, 인코딩, 암호화를 관리.

- 작동 원리

응용 프로그램이 이해할 수 있도록 변환.

- 역할
데이터 번역기 역할.

- 하는 일
문자 코드 변환 (ASCII ↔ EBCDIC, UTF-8 ↔ UTF-16).
데이터 압축/해제 (JPEG, MPEG, MP3).
암호화/복호화 (TLS, SSL).
구문 변환 (JSON ↔ XML).

- 전송 단위
데이터(Data).

- 관련 장비
게이트웨이.

- 프로토콜/표준
TLS/SSL, MPEG, JPEG, GIF.

7. 응용 계층 (Application Layer)
- 정의
최종 사용자와 직접 맞닿는 계층, 네트워크 서비스 제공.

- 작동 원리
응용 프로그램이 네트워크 기능을 사용할 수 있도록 인터페이스 제공.

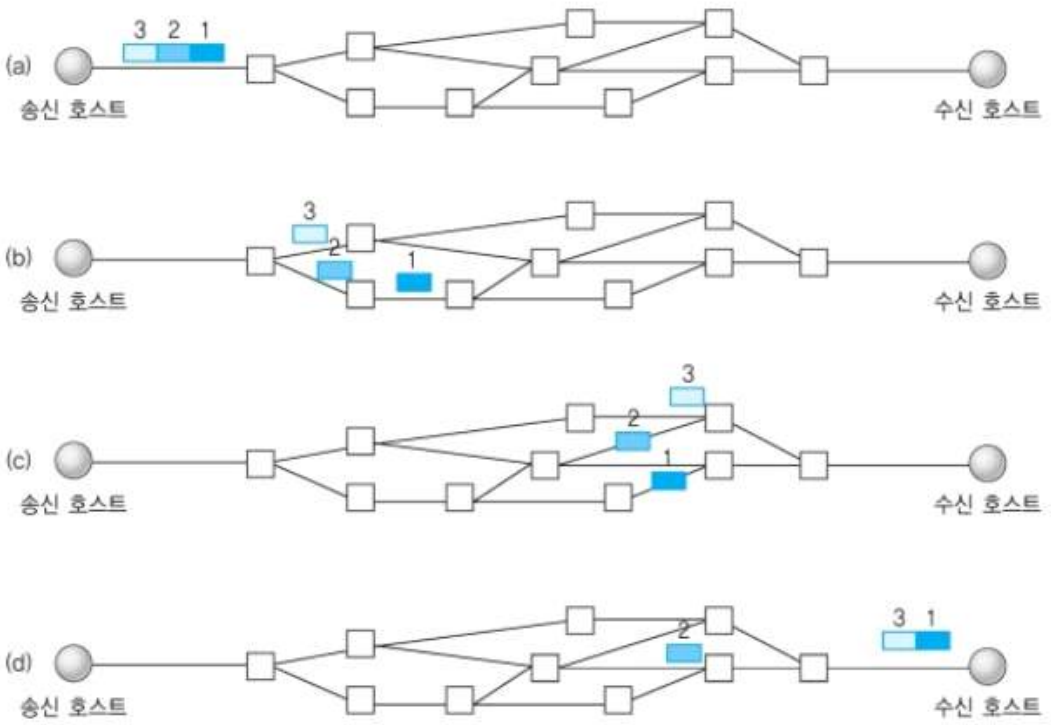
- 역할
실제 응용 서비스 수행 (웹, 메일, 파일 전송 등).

- 하는 일
데이터 생성 및 요청/응답 처리.
사용자 인터페이스 제공.
파일 전송, 이메일, 웹 서비스 지원.

- 전송 단위
데이터(Data).

- 프로토콜/표준
HTTP/HTTPS, FTP, SMTP, POP3, IMAP, DNS, SNMP, DHCP.

8. UDP 통신



1) 개요
- UDP(User Datagram Protocol, 사용자 데이터그램 프로토콜)는 포트 번호를 이용해 데이터를 적절한 애플리케이션으로 전달하는 전송계

층 프로토콜이다.
- 비연결 지향적(Non-connection-oriented) 프로토콜로, 데이터 전송 시 별도의 연결 설정 과정이나 재전송, 혼잡 제어, 윈도우 제어 등을 수행하지 않는다.
- 데이터 순서가 바뀌더라도 이를 재정렬하지 않으며, 단순히 애플리케이션에 데이터를 전달하는 것만이 목적이다.

- UDP의 헤더는 송신지 포트, 목적지 포트, 데이터그램 길이, 체크섬으로 구성되어 있으며, TCP 헤더의 20바이트에 비해 8바이트로 매우 간단하다.
- 오버헤드가 적기 때문에 TCP보다 통신 효율이 높고, 실시간 전송이나 고속 전송이 필요한 환경에서 유리하다.
- '데이터그램'이란 UDP 헤더와 실제 데이터를 합친 단위를 의미하며, TCP의 세그먼트와 대응되는 개념이다.

2) 특징
- 비연결형 서비스로 데이터그램 방식을 제공한다.
 : 데이터의 전송 순서가 바뀔 수 있다.

- 데이터의 수신 여부를 확인하지 않는다.
 : TCP의 3-way handshaking과는 다른 과정

- 신뢰성이 낮다.
 : 흐름 제어가 없어서 제대로 전송되었는지, 오류가 없는지 확인할 수 없다.

- 1:1 & 1:N & N:N 통신이 가능하다.

위와 같은 특징들로 인해서 신뢰성보다는 연속성 있는 전송이 필요할 때 사용하는 프로토콜로 실시간 서비스에 자주 사용된다.

3) 헤더 구성

UDP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Length																Checksum														

UDP는 데이터 전송 자체에만 초점을 맞추고 설계되었으므로 헤더에 들어 있는게 없다.

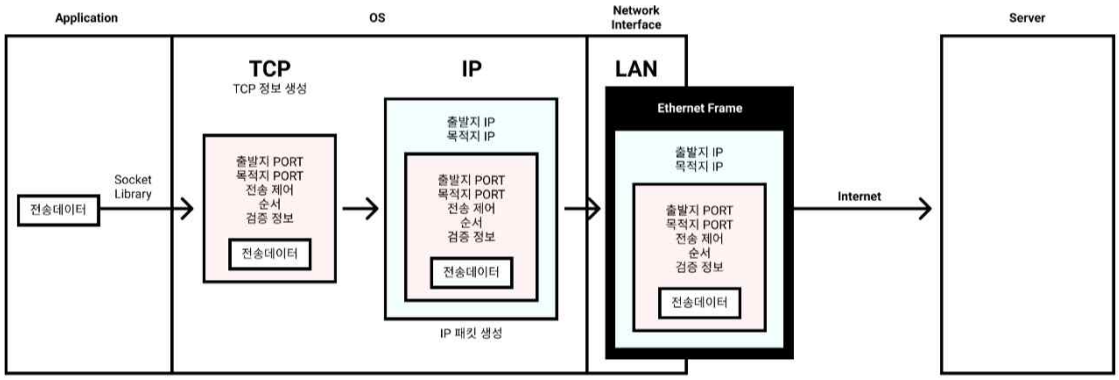
① Sorec Port : 데이터를 생성한 애플리케이션에서 사용하는 포트 번호
② Destinaton Port : 목적지 애플리케이션이 사용하는 포트 번호
③ checksum : 중복 검사의 한 형태로, 오류 정정을 통해 공간(전자 통신)이나 시간(기억 장치) 속에서 송신된 자료의 무결성을 보호하는 단순한 방법. (TCP의 체크섬과는 다르게 UDP의 체크섬은 사용해도 되고 안해도 되는 옵션)

8. TCP

1) 개요

- TCP(Transmission Control Protocol)는 IP 규칙으로만 통신하기에 부족하거나 불안정하던 여러 단점들(패킷 순서가 이상하거나 패킷이 유실)을 커버해, 패킷 전송을 제어하여 신뢰성을 보증하는 프로토콜이다.
- TCP 규칙에 써 있는 대로 올바르게 도착했는지 정확히 누구에게 전달되야 하는지 하나하나 따진다. 그래서 은행 업무나 메일과 같은 반드시 수신자가 정보를 받아야 하는 신뢰성 있는 통신이 필요할 때 사용된다.

2) 전송 데이터가 포장되는 과정



① 전송데이터를 TCP 포장한다.
② 포장한 전송데이터를 IP 포장한다
③ 포장한 전송데이터를 이더넷 포장한다
④ 인터넷을 통해 상대 컴퓨터 서버에 도달하여 포장된걸 하나씩 풀며 전송데이터를 받게 된다.

3) TCP 통신 확인

TCP는 신뢰성 있는 통신 프로토콜이다. 데이터를 단순히 보내는 것이 아니라, 목적지에 안전하게 도착했는지 확인하고, 전송 후에도 제대로 도착했는지 검증한다. 쉽게 말해, 굉장히 “친절하게 확인”하는 프로토콜이다.

- 통신 시작과 종료 과정:

TCP는 통신을 시작할 때, 그리고 종료할 때 서로 준비가 되었는지 확인한다.
패킷을 보내기 전에 순서를 정하고, 서로 확인한 뒤 실제 데이터를 주고받는다.

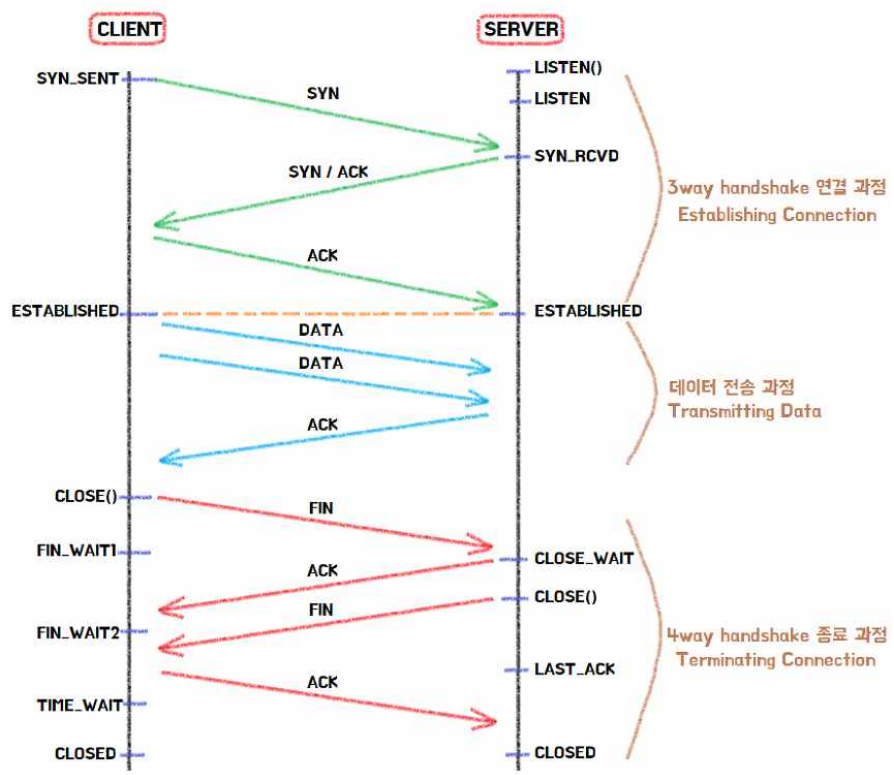
- 핸드쉐이크(Handshake):

3 Way Handshake: 통신을 시작할 때 클라이언트와 서버가 서로 준비되었는지 확인하는 과정

4 Way Handshake: 통신을 종료할 때 서로 연결을 안전하게 끊는 과정

요약하면, 3 Way는 시작, 4 Way는 종료이며, 둘 다 “상호 인증 과정”이다.

4) 패킷의 역할:
TCP 패킷 안에는 SYN, ACK, FIN 등 인증용 플래그가 들어 있다.
예를 들어, SYN과 ACK는 “나 준비됐어요”, FIN은 “이제 끝낼게요”라는 의미로, 패킷의 순서와 전달 여부를 검증한다.



(1) 3-way handshake (TCP 연결 수립)
TCP는 신뢰성을 위해 연결을 먼저 설정한다.

① 클라이언트 → 서버 : SYN
클라이언트가 서버와 연결을 시작하겠다고 신호(SYN)를 보낸다. 이때 seq 번호가 랜덤으로 생성된다.

② 서버 → 클라이언트 : SYN + ACK
서버는 클라이언트의 요청을 받고 연결을 승인하며, 클라이언트 seq 번호를 확인 하여 ACK로 응답한다.

③ 클라이언트 → 서버 : ACK
클라이언트가 서버의 SYN에 응답하여 연결을 확정한다.

(2) 데이터 통신 과정
TCP 연결이 설정되면 데이터를 주고받을 수 있다.

① 클라이언트 → 서버 : PSH + ACK
클라이언트가 데이터를 서버로 즉시 전달하고, 이전 seq 번호에 대한 확인(ACK)을 보낸다.

② 서버 → 클라이언트 : ACK
서버가 데이터를 받았음을 확인한다.

(3) 4-way handshake (TCP 연결 종료)
TCP 연결 종료 시에는 4단계 과정으로 안전하게 종료한다.

① 서버 → 클라이언트 : FIN + ACK
서버가 연결 종료 의사를 전달한다.

② 클라이언트 → 서버 : FIN + ACK
클라이언트도 종료 신호와 확인 응답을 보낸다.

③ 서버 → 클라이언트 : ACK
최종 확인을 하고 연결이 종료된다.

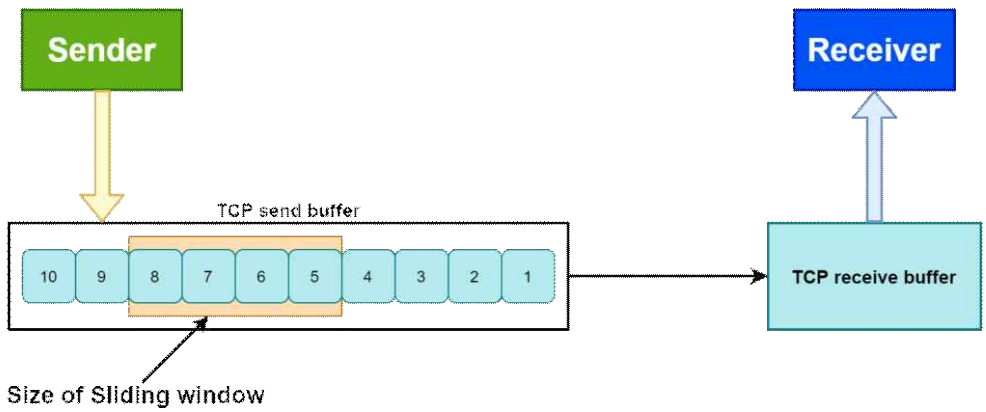
플래그	의미
SYN	접속 요청: TCP 연결을 시작할 때 가장 먼저 보내는 패킷. “나 연결할래요”라고 알리는 신호.
ACK	수신 확인: 상대방이 보낸 패킷을 잘 받았음을 알려주는 패킷. 다른 플래그와 함께 나타나기도 함.
PSH	즉시 전송 요청: 데이터를 지연시키지 않고 바로 목적지로 보내라는 의미.
FIN	접속 종료: 현재 연결을 끝내고 싶을 때 보내는 패킷. “이제 연결 끊을게요” 신호.

5) TCP의 전송 제어 기법

TCP(Transmission Control Protocol)는 원활한 통신을 위해 전송 흐름을 제어하는 기능을 프로토콜 자체에 포함한다. 만약 TCP가 없었다면 개발자가 일일이 데이터를 어떤 단위로 보낼 것인지 정의하고, 패킷이 유실되면 어떤 예외 처리를 해야 하는지까지 신경 써야 한다. 덕분에 우리는 온전히 상위 계층의 동작에만 집중할 수 있다.

(1) 흐름 제어(Flow Control)

- 수신자가 처리할 수 있는 데이터 속도가 다르기 때문에, 송신 측은 수신 측의 데이터 처리 속도를 파악하고 얼마나 빠르게 어느 정도의 데이터를 전송할지 제어한다.
- 슬라이딩 윈도우(Sliding Window) 방식을 사용하며, Window라는 데이터를 담는 공간을 동적으로 조절하여 전송 데이터량을 조절한다.



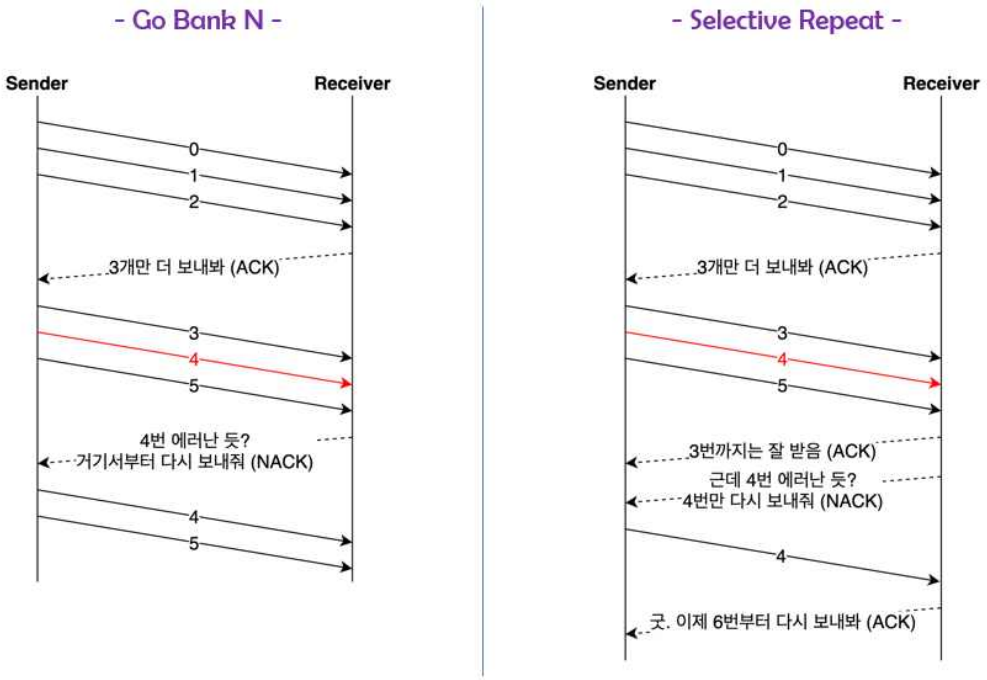
(2) 오류 제어(Error Control)

- 통신 도중 데이터가 유실되거나 잘못된 데이터가 수신되었을 경우 대처한다.

- Go-Back-N 기법과 Selective Repeat(선택적 재전송) 기법을 사용한다.

  - Go-Back-N 기법: 어느 데이터부터 오류가 발생했는지 파악하고, 그 부분부터 순서대로 다시 전송하여 제어한다.

  - Selective Repeat 기법: 에러가 발생한 데이터만 재전송하고, 이전에 받은 순서 가 잘못된 데이터 버퍼를 재정렬하여 제어한다.



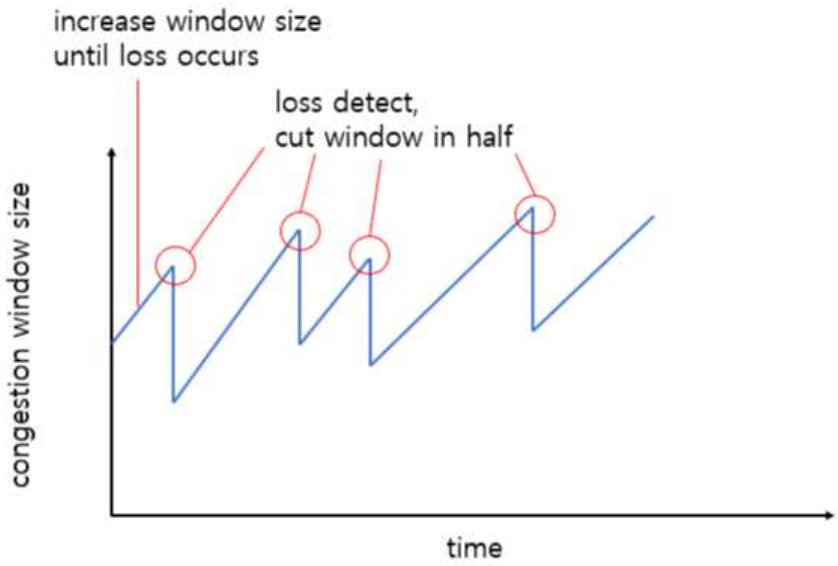
(3) 혼잡 제어(Congestion Control)

- 네트워크가 불안정하여 데이터가 원활히 통신되지 않으면 재전송이 반복되며, 과도한 재전송은 네트워크 붕괴를 초래할 수 있다.

- 따라서 네트워크 혼잡 상태가 감지되면 송신 측의 전송 데이터 크기를 조절하여 전송량을 제어한다.

- TCP에는 Tahoe, Reno, New Reno, Cubic, Elastic-TCP 등 다양한

혼잡 제어 기법이 존재한다.



6) 헤더 구성

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

TCP는 오래전에 설계되었고, 다양한 기능이 포함되어 있어 헤더가 거의 꽉 차 있다. 각 필드의 역할은 다음과 같다.

① Source Port: 데이터를 생성한 애플리케이션에서 사용하는 포트 번호다.

② Destination Port: 목적지 애플리케이션에서 사용하는 포트 번호다.

③ Sequence Number 필드: 세그먼트 순서를 맞추기 위해 사용하는 필드다.

④ Acknowledgement Number 필드: 다음 세그먼트 수신 준비 상태를 표시하고, 이미 받은 데이터를 확인하는 역할을 한다.

⑤ Data Offset 필드: TCP 헤더의 크기를 나타내며, 5~15 사이 값으로 532bit(160bit)~1532bit(480bit)를 표현한다.
⑥ Reserved 필드: 차후 사용을 위해 예약된 필드다.
⑦ Control Flags (SYN, ACK, FIN 등): 긴급, 혼잡, 확인, 수신 거부 등 다양한 제어 기능을 수행한다.
⑧ Window Size 필드: 수신자가 한 번에 받을 수 있는 데이터 양을 나타낸다. 송신자는 Window Size만큼 ACK를 기다리지 않고 데이터를 계속 전송할 수 있다.
⑨ Checksum: 세그먼트 내용의 유효성과 손상 여부를 검사한다.

10. UDP 통신 구현

- 코드 분석

1. 헤더 파일

```
#include <QMainWindow>
#include <QUdpSocket>
#include <QDateTime>
#include <QHostAddress>
#include <QNetworkInterface>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_btnConnect_clicked();
    void on_btnSend_clicked();
    void udpDataReceived();

private:
    Ui::MainWindow *ui;

    QUdpSocket *udpSocket;
    QHostAddress remoteAddr;
    quint16 remotePort;
    quint16 localPort;

    QString getLocalIP();
    void displayMessage(const QString &msg, const QString &from);
};
```

QMainWindow: Qt에서 기본 제공하는 메인 윈도우 클래스. 메뉴, 툴바, 상태바 등을 가진 창을 만들 때 사용.

QUdpSocket: UDP 통신을 위해 사용되는 클래스.

QDateTime: 시간과 날짜 정보를 처리할 때 사용.

QHostAddress: IP 주소를 다루는 클래스.

QNetworkInterface: 컴퓨터의 네트워크 인터페이스(로컬 IP 등)를 다룰 때 사용.

버튼 클릭 이벤트로 데이터를 보내고, 도착한 데이터를 받아서 화면에 표시하는 구조로 클래스를 설계하였다.

-mainwiindow.cpp파일

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , udpSocket(new QUdpSocket(this))
{
    ui->setupUi(this);

    // 내 IP 자동 표시
    ui->lineEditLocalIP->setText(getLocalIP());
    ui->textEditRecv->setReadOnly(true);

    connect(udpSocket, &QUdpSocket::readyRead, this, &MainWindow::udpDataReceived);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

클래스가 생성될 때, 먼저 UI와 UDP 소켓 객체를 동적으로 생성하고 필요한 값을 초기화한다. UI는 setupUi()를 통해 화면에 배치하고, 버튼과 텍스트박스 등 위젯들을 사용할 수 있도록 연결한다. 로컬 IP는 getLocalIP()를 호출해 가져와 lineEditLocalIP에 표시하고, 수신 메시지를 보여주는 textEditRecv는 읽기 전용으로 설정해 사용자가 직접 편집하지 못하도록 한다.

UDP 소켓은 readyRead 시그널과 udpDataReceived() 슬롯을 연결하여, 데이터가 도착하면 자동으로 처리되도록 준비한다. 이렇게 생성자에서 초기값을 세팅하고 시그널과 슬롯을 연결함으로써, 프로그램은 버튼 클릭과 UDP 수신 이벤트를 정상적으로 처리할 수 있는 상태가 된다.

클래스가 소멸될 때는 ui 객체를 해제하여 메모리를 정리하고, UDP 소켓은 부모 객체가 자동으로 관리하므로 별도로 해제하지 않아도 된다.

```
QString MainWindow::getLocalIP()
{
    QList<QHostAddress> ipList = QNetworkInterface::allAddresses();
    for (auto &ip : ipList) {
        if (ip != QHostAddress::LocalHost && ip.protocol() == QAbstractSocket::IPv4Protocol)
            return ip.toString();
    }
    return "127.0.0.1";
}

void MainWindow::on_btnConnect_clicked()
{
    localPort = ui->lineEditLocalPort->text().toUShort();
    remoteAddr = QHostAddress(ui->lineEditRemoteIP->text());
    remotePort = ui->lineEditRemotePort->text().toUShort();

    if (udpSocket->bind(QHostAddress::Any, localPort)) {
        displayMessage("UDP 소켓 바인드 완료 (포트 " + QString::number(localPort) + ")", "SYSTEM");
    } else {
        displayMessage("UDP 소켓 바인드 실패", "SYSTEM");
    }
}

void MainWindow::on_btnSend_clicked()
{
    QString msg = ui->lineEditSend->text();
    if (msg.isEmpty()) return;

    QByteArray data = msg.toUtf8();
    udpSocket->writeDatagram(data, remoteAddr, remotePort);

    displayMessage(msg, "Me → " + remoteAddr.toString() + ":" + QString::number(remotePort));
}
```

getLocalIP() 함수는 컴퓨터의 모든 네트워크 인터페이스를 탐색하여, 로컬호스트가 아니면서 IPv4 프로토콜인 주소를 찾아 반환한다. 만약 조건에 맞는 주소가 없으면 기본값으로 "127.0.0.1"을 반환하도록 한다. 이렇게 해서 프로그램은 항상 사용 가능한 로컬 IP를 가져오도록 준비된다.

on_btnConnect_clicked() 슬롯은 연결 버튼이 클릭되었을 때 호출된다. 먼저 UI에서 입력된 로컬 포트, 원격 IP, 원격 포트를 가져와 멤버 변수에 저장한다. 이후 UDP 소켓을 bind()하여 로컬

포트를 열고 데이터를 받을 준비를 한다. 바인드에 성공하면 "UDP 소켓 바인드 완료" 메시지를 시스템 메시지 영역에 표시하고, 실패하면 "UDP 소켓 바인드 실패" 메시지를 표시하여 사용자가 상태를 확인할 수 있도록 한다.

on_btnSend_clicked() 슬롯은 메시지 전송 버튼이 클릭될 때 호출된다. 먼저 입력창에서 문자열을 가져오고, 비어 있으면 바로 함수를 종료한다. 유효한 문자열이면 UTF-8 바이트 배열로 변환한 후, writeDatagram()을 통해 원격 주소와 포트로 UDP 메시지를 전송한다. 전송이 끝나면 화면에 "Me → [원격IP:포트]" 형태로 메시지를 표시하여, 사용자가 성공적으로 메시지를 보냈음을 확인하도록 한다.

```
void MainWindow::udpDataReceived()
{
    while (udpSocket->hasPendingDatagrams()) {
        QByteArray datagram;
        datagram.resize(udpSocket->pendingDatagramSize());
        QHostAddress sender;
        quint16 senderPort;

        udpSocket->readDatagram(datagram.data(), datagram.size(), &sender, &senderPort);

        QString msg = QString::fromUtf8(datagram);
        displayMessage(msg, sender.toString() + ":" + QString::number(senderPort));
    }
}

void MainWindow::displayMessage(const QString &msg, const QString &from)
{
    QString timestamp = QDateTime::currentDateTime().toString("[yyyy-MM-dd hh:mm:ss] ");
    ui->textEditRecv->append(timestamp + " [" + from + "]  " + msg);
}
```

udpDataReceived() 함수는 UDP 소켓에 데이터가 도착했을 때 자동으로 호출된다. 먼저 소켓에 대기 중인 데이터그램이 있는지 확인하고, 존재하면 반복문을 통해 모두 처리한다. 각 데이터그램의 크기만큼 바이트 배열을 준비하고, readDatagram()을 호출하여 실제 데이터를 읽어온다. 이때 발신자 주소와 포트도 함께 가져온다. 읽어온

데이터는 UTF-8 문자열로 변환한 후, displayMessage() 함수를 호출하여 화면에 표시한다. 이렇게 함으로써 프로그램은 수신된 모든 메시지를 실시간으로 사용자에게 보여주도록 준비한다.

displayMessage() 함수는 메시지를 화면에 출력할 때 호출된다. 먼저 현재 시간을 [yyyy-MM-dd hh:mm:ss] 형식으로 가져와 타임스탬프를 생성하고, 발신자 정보와 메시지 내용을 조합하여 텍스트박스 끝에 추가한다. 이렇게 하면 수신 메시지 영역에 시간과 발신자가 함께 표시된 로그 형태로 메시지가 쌓이게 되며, 사용자는 언제, 누구에게서 메시지가 왔는지 쉽게 확인할 수 있다.

- 실행 결과