데이터 통신 00분반

Stop & wait

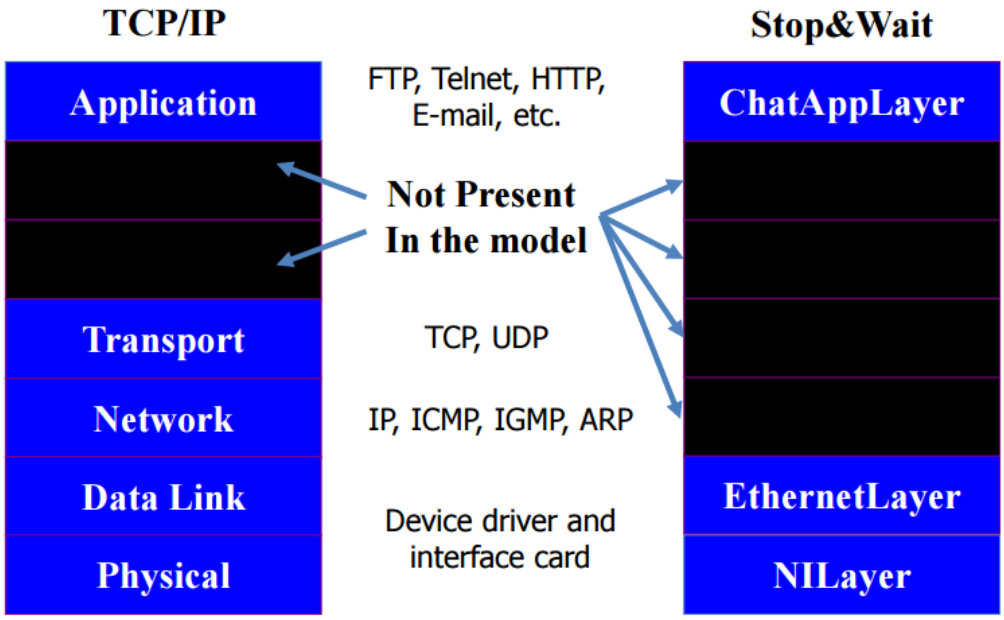
201402447 한원희

실습 개요

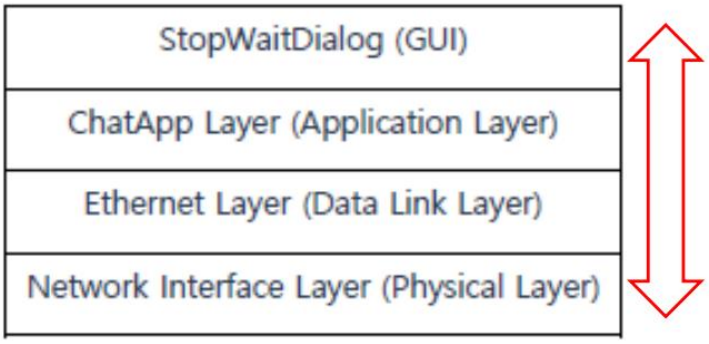
전송계층에서는 데이터 전송 보장을 위해 다양한 방식의 프로토콜을 사용하고 있습니다. 컴퓨터 네트워크 설정에서 재전송을 기반으로 하는 신뢰적인 데이터 전송 프로토콜 중 하나가 ARQ(Automatic Repeat Request) 프로토콜입니다. Stope & Wait 프로토콜은 ARQ 방식의 일종입니다. 이 방식에서는 송신측 A가 B에게 1개의 프레임을 송신하게 되면 B는 해당 프레임의 에러 유무를 판단하여 A에게 ACK를 보내주게 되는 프로토콜을 실습해 봅니다.

프로토콜 스택

› Protocol Stack



프로토콜 스택은 다음과 같은 모습을 하고 있습니다. Stop & Wait에서 ChatAppLayer, EthernetLayer, StopWaitdlg를 수정하는 과제입니다.



다음과 같은 모습으로 스택이 연결되어 있습니다. 맨 위 계층은 StopWaitDialog으로 서로의 계층을 연결해 주는 역할을 수행합니다.

그리고 채팅창의 기본 설정을 해주거나 MAC 주소를 받아오거나 수신 메시지를 표시하거나 하는 역할을 해줍니다.

ChatAppLayer는 채팅창에서 send가 눌려 메시지가 들어오면 그 메시지를 확인해주어서 EthernetLayer로 메시지의 크기에 따라 조건에 맞게 보내주거나 아니면 들어온 메시지를 확인해 StopWait에 보내주는 역할을 수행합니다.

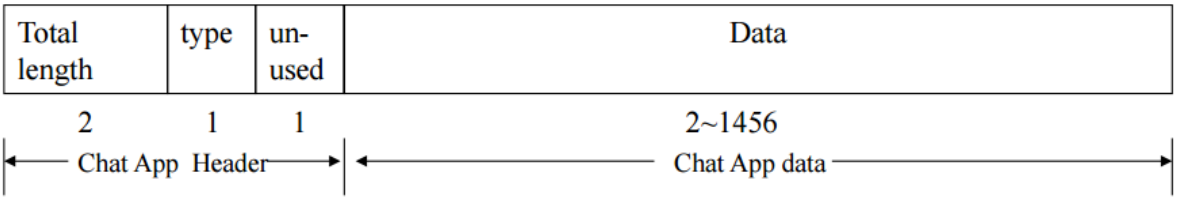
EthernetLayer는 이제 Network Interface Layer에서 외부에서 들어온 데이터를 확인해 ChatAppLayer에 보내주거나 아니면 자신의 프로그램에서 보낼 데이터를 확인해 Network Interface에 보내주는 역할을 수행합니다.

구현 설명

```
public static void main(String[] args) {
    m_LayerMgr.AddLayer(new NILayer("NI"));
    m_LayerMgr.AddLayer(new EthernetLayer("Ethernet"));
    m_LayerMgr.AddLayer(new ChatAppLayer("ChatApp"));
    m_LayerMgr.AddLayer(new StopWaitDlg("GUI"));

    m_LayerMgr.ConnectLayers( " NI ( *Ethernet ( *ChatApp ( *GUI ) ) )" );
}
```

StopWaitDlg의 main부분입니다. 저번 과제와 같이 서로의 Layer를 연결해 줍니다.

```
private class _CHAT_APP {
    byte[] capp_totlen;
    byte capp_type;
    byte capp_unused;
    byte[] capp_data;

    public _CHAT_APP() {
        this.capp_totlen = new byte[2];
        this.capp_type = 0x00;
        this.capp_unused = 0x00;
        this.capp_data = null;
    }
}
```



ChatAppLayer의 헤더입니다. Total length와 type이 눈에 띕니다.

```
public boolean Send(byte[] input, int length) {
    byte[] bytes;
    m_sHeader.capp_totlen = intToByte2(length);
    m_sHeader.capp_type = (byte) (0x00);

    if (length > 10) {  // 배열이 10보다 클경우
        fragSend(input, length);
    } else {    // 배열이 10보다 작을경우
        bytes = this.objToByte(m_sHeader, input, length);
        this.GetUnderLayer().Send(bytes, length + 4);
    }

    return true;
}
```

ChatApp의 Send입니다. 채팅창에서 send가 눌렸을 때 데이터를 받아와 검사합니다. 메시지의 크기가 10보다 클 경우엔 데이터를 잘라서 보내야 하므로 fragsend를 실행해줍니다. 만약 메시지가 10보다 작을 경우에는 데이터에 objToByte 메소드를 통해 데이터에 헤더를 붙여서 EthernetLayer에 보내줍니다.

```
private void fragSend(byte[] input, int length) {
    byte[] bytes = new byte[10];
    int i = 0;
    m_sHeader.capp_totlen = intToByte2(length);
    m_sHeader.capp_type = (byte) (0x01);

    // 첫번째 전송
    byte[] tempbytel = new byte[10];

    System.arraycopy(input, 0, tempbytel, 0, 10);
    bytes = objToByte(m_sHeader, tempbytel, 10);
    this.GetUnderLayer().Send(bytes, bytes.length);
    int maxLen = length / 10;
        /*과제 */
    for(i = 0; i<=maxLen; i++){
        if(i == maxLen){
```

ChatApp의 fragSend입니다.

메시지의 크기가 10보다 크다면 실행하는 메소드입니다. 헤더의 타입에 데이터는 3가지로 나눌 수 있습니다. 1은 첫번째 데이터를 의미합니다. 2는

중간의 데이터를 의미합니다. 3은 마지막 데이터를 의미합니다. 1, 2의 데이터의 크기는 10으로 보내주고 마지막 3은 10으로 나눈 나머지의 크기입니다.

```
for(i = 0; i<=maxLen; i++){
    if(i == maxLen){
        m_sHeader.capp_type = (byte) (0x03);
        m_sHeader.capp_totlen = intToByte2(length % 10);

        byte[] tempbyte = new byte[length%10];

        System.arraycopy(input, i*10, tempbyte, 0, length % 10);
        byte[] data = objToByte(m_sHeader, tempbyte, length % 10);
        this.GetUnderLayer().Send(data, data.length);
    }
```

maxLen은 마지막 단편화일 때 입니다.

```
}
else if (i != 0) {

    m_sHeader.capp_type = (byte) (0x02);
    m_sHeader.capp_totlen = intToByte2(10);
    byte[] tempbyte = new byte[10];

    System.arraycopy(input, i*10, tempbyte, 0, 10);
    byte[] data = objToByte(m_sHeader, tempbyte, 10);
    this.GetUnderLayer().Send(data, data.length);
    waitACK();
}
```

마지막 단편화가 아니고 i가 0, 즉, 첫번째 단편화가 아니라면 중간 데이터의 단편화를 진행해 줍니다. Type은 2입니다.

```
public synchronized boolean Receive(byte[] input) {
    byte[] data;
    int tempType = 0;

    if (input == null) {
        ackChk.add(true);
        return true;
    }

    tempType |= (byte) (input[2] & 0xFF);
```

ChatApp의 Receive 메소드입니다. Input[2]에는 현재 데이터의 type이 들어가 있습니다. 위의 데이터 형태를 보면 나옵니다.

```
if(tempType == 0) {
    data = RemoveCappHeader(input, input.length);
    this.GetUpperLayer(0).Receive(data);
    return true;
}
```

데이터의 타입이 0 이라면 데이터의 크기가 10보다 작을 때 입니다. 더 이상 다른 데이터가 없으므로 바로 헤더를 없애준 뒤 StopWaitdlg로 보내줍니다.

```
else{
    if(tempType == 1){
        fragBytes = new byte[byte2ToInt(input[0],input[1])];
        fragCount = 1;

        data = RemoveCappHeader(input, input.length);

        for(int i = 0; i < 10; i++){
            fragBytes[i] = data[i];
        }
    }
    else if(tempType == 2){
        data = RemoveCappHeader(input, input.length);

        for(int i = 0; i < 10; i++){
            fragBytes[fragCount*10 + i] = data[i];
        }

        fragCount += 1;
    }
```

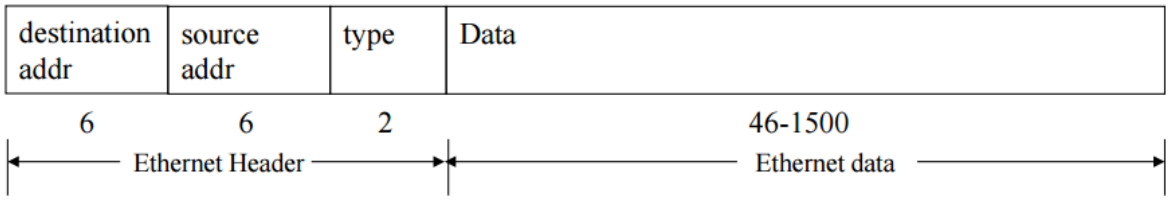
데이터가 10보다 클 때 이용되는 메소드입니다. fragBytes와 fragCount는 전역변수로 설정되어 있습니다. 첫번째 데이터가 들어온다면 크기를 헤더의 0, 1에 들어있는 총 길이를 받아 배열의 크기를 설정해준뒤 fragCount를 1로 설정해줍니다. fragCount는 현재 데이터 frag를 받은 숫자라고 볼 수 있습니다. 타입 2를 거칠 때 마다 fragCount를 하나씩 올려주고 fragBytes에 데이터를 계속해서 이어 붙여줍니다.

```
else if(tempType == 3){
    data = RemoveCappHeader(input, input.lengt
    int totlen = byte2ToInt(input[0],input[1])

    for(int i = 0; i < totlen; i++){
        fragBytes[fragCount*10 + i] = data[i];
    }
    this.GetUpperLayer(0).Receive(fragBytes);
}

is.GetUnderLayer().Send(null, 0); // ack 송신
turn true;
```

타입 3는 마지막 데이터일 때 입니다. 위와 같이 데이터를 fragBytes에 이어 붙여주고 마지막 데이터이니 끝나면 데이터를 위 레이어에 보내줍니다.

```
private class _ETHERNET_ADDR {
    private byte[] addr = new byte[6];

    public _ETHERNET_ADDR() {
        this.addr[0] = (byte) 0x00;
        this.addr[1] = (byte) 0x00;
        this.addr[2] = (byte) 0x00;
        this.addr[3] = (byte) 0x00;
        this.addr[4] = (byte) 0x00;
        this.addr[5] = (byte) 0x00;

    }
}

private class _ETHERNET_Frame {
    _ETHERNET_ADDR enet_dstaddr;
    _ETHERNET_ADDR enet_srcaddr;
    byte[] enet_type;
    byte[] enet_data;

    public _ETHERNET_Frame() {
        this.enet_dstaddr = new _ETHERNET_ADDR();
        this.enet_srcaddr = new _ETHERNET_ADDR();
        this.enet_type = new byte[2];
        this.enet_data = null;
    }
}
```

EternetLayer의 헤더입니다. 위는 dst주소와 src주소를 사용할 때 크기이고 밑은 헤더의 크기입니다.



헤더의 크기는 14입니다.

```
public boolean Send(byte[] input, int length) {
    if (input == null && length == 0) // ack
        m_sHeader.enet_type = intToByte2(2);
    else if (isBroadcast(m_sHeader.enet_dstaddr.addr)) // broadcast
        m_sHeader.enet_type = intToByte2(0xff);
    else // nomal
        m_sHeader.enet_type = intToByte2(1);

    byte[] data = ObjToByte(m_sHeader, input, length);
    this.GetUnderLayer().Send(data, data.length);

    return true;
}
```

send입니다. 받은 데이터를 Network Interface에 보내는 역할을 수행합니다.

만약 일반 데이터일 경우 타입은 1입니다.

```
public synchronized boolean Receive(byte[] input) {
    byte[] data = {};
    byte[] temp_src = m_sHeader.enet_srcaddr.addr;
    int temp_type = byte2ToInt(input[12], input[13]);
    byte[] temp_dst = m_sHeader.enet_dstaddr.addr;

    if(temp_type == 0){
        this.GetUpperLayer(0).Receive(); // ack 송신
        return true;
    }
    if(temp_type == 1){
        // broadcast 인경우
        if(isBroadcast(input)){
            data = RemoveEthernetHeader(input, input.length);
            this.GetUpperLayer(0).Receive(data);
            return false;
```

NI에서 받은 데이터를 검사하는 메소드입니다.

```
// 일반 데이터일때
if(temp_type == 1){
    // broadcast 인 경우
    if(isBroadcast(input)){
        data = RemoveEthernetHeader(input, input.length);
        this.GetUpperLayer(0).Receive(data);
        return true;
    }

    // 목적지 주소가 자신인지
    if(isMyPacket(input)){
        return true;
    }

    // 내가 전송한 데이터인지
    if(chkAddr(input)){
        data = RemoveEthernetHeader(input, input.length);
        this.GetUpperLayer(0).Receive(data);
        return true;
    }
```

데이터의 타입을 검사해주어 일반 데이터일 때 각각의 타입을 검사해줍니다.

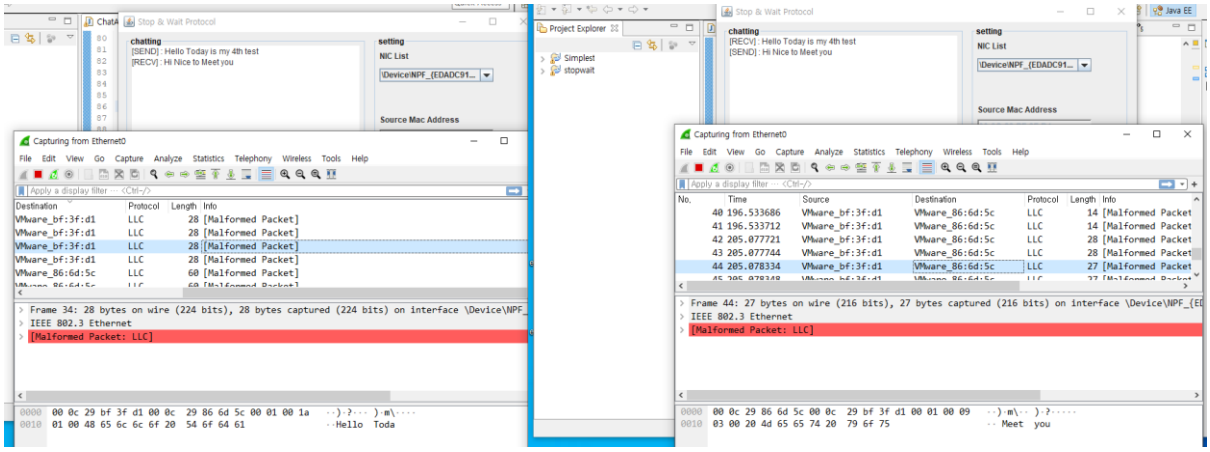
```
private boolean isBroadcast(byte[] bytes) {
    for(int i = 0; i< 6; i++)
        if (bytes[i] != (byte) 0xff)
            return false;
    return (bytes[12] == (byte) 0xff && bytes[13] == (byte) 0xff);
}

private boolean isMyPacket(byte[] input){
    for(int i = 0; i < 6; i++)
        if(m_sHeader.enet_srcaddr.addr[i] != input[6 + i])
            return false;
    return true;
}

private boolean chkAddr(byte[] input) {
    byte[] temp = m_sHeader.enet_srcaddr.addr;
    for(int i = 0; i< 6; i++)
        if(m_sHeader.enet_srcaddr.addr[i] != input[i])
            return false;
    return true;
}
```

데이터를 검사해주는 메소드입니다. Broadcast는 0xff로 검사해주고, MyPacket은 src 주소로 자신의 src주소와 검사해줍니다. 자기 자신이 데이터를 자기 자신에게 보냈다는 말입니다.

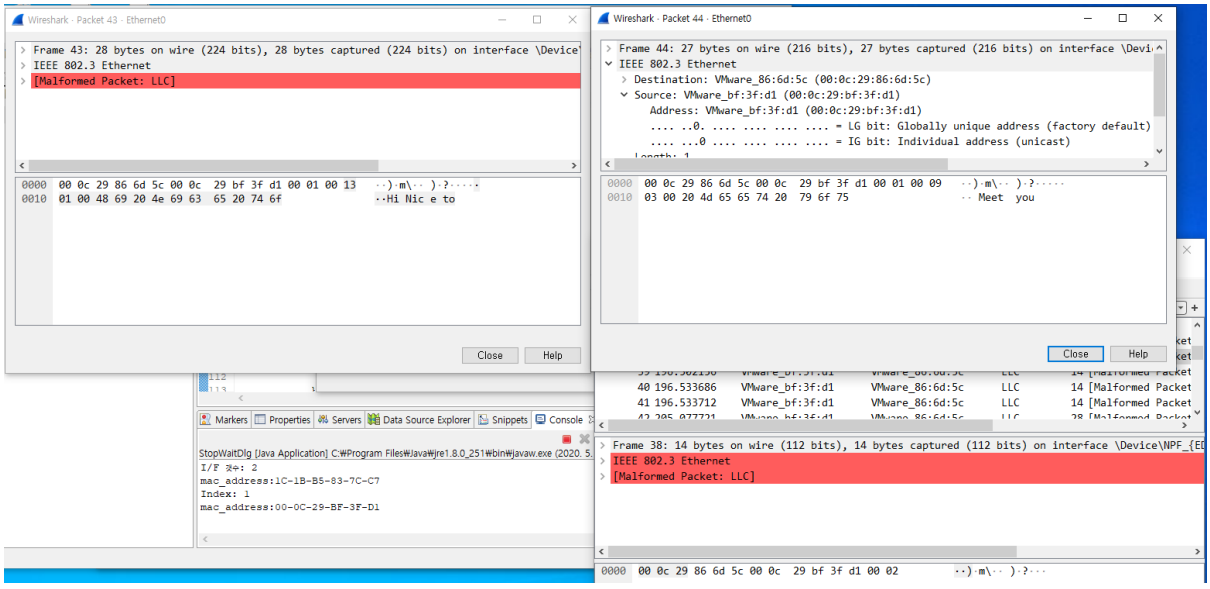
chkAddr는 헤더의 src주소와 받은 데이터의 dst주소입니다. 다른 컴퓨터에서 자신에게 보낸 데이터를 확인하는 메소드입니다.

검사해 준 뒤 맞다면 EthernetLayer에 보내줍니다.

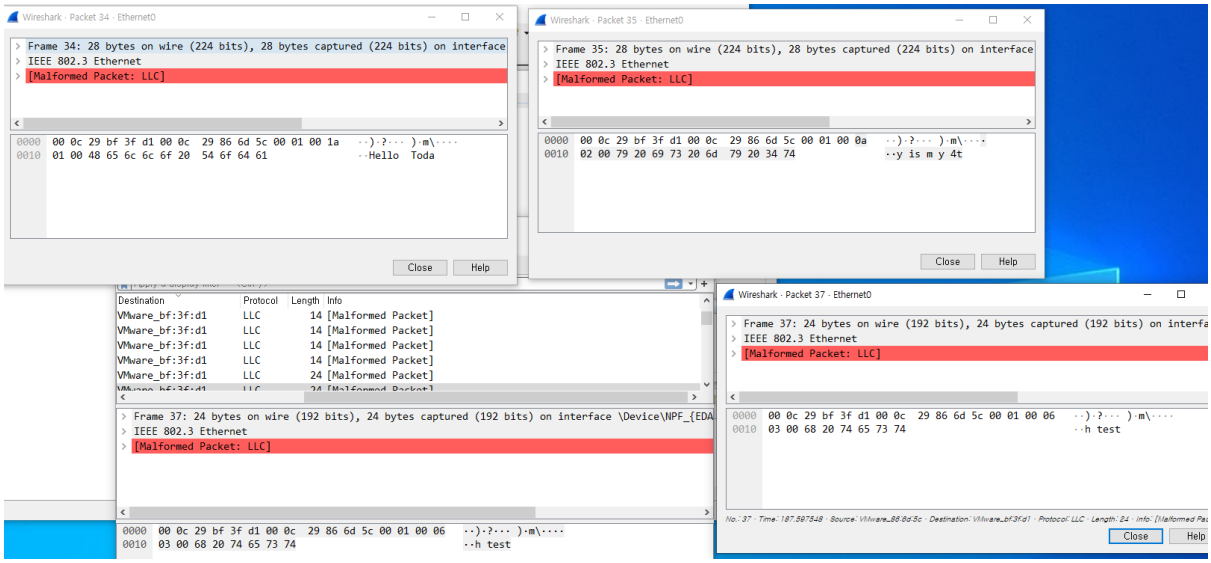
결과 화면



채팅창에 메시지가 표시된 모습입니다.



Hi Nice to Meet you가 보내진 모습입니다.



Hello Today is my 4th test가 보내진 모습입니다.