데이터 통신 과제
제출일 : 2020-05-08
201602022 양승훈

Simplest Protocol

실습 개요

실습 목적

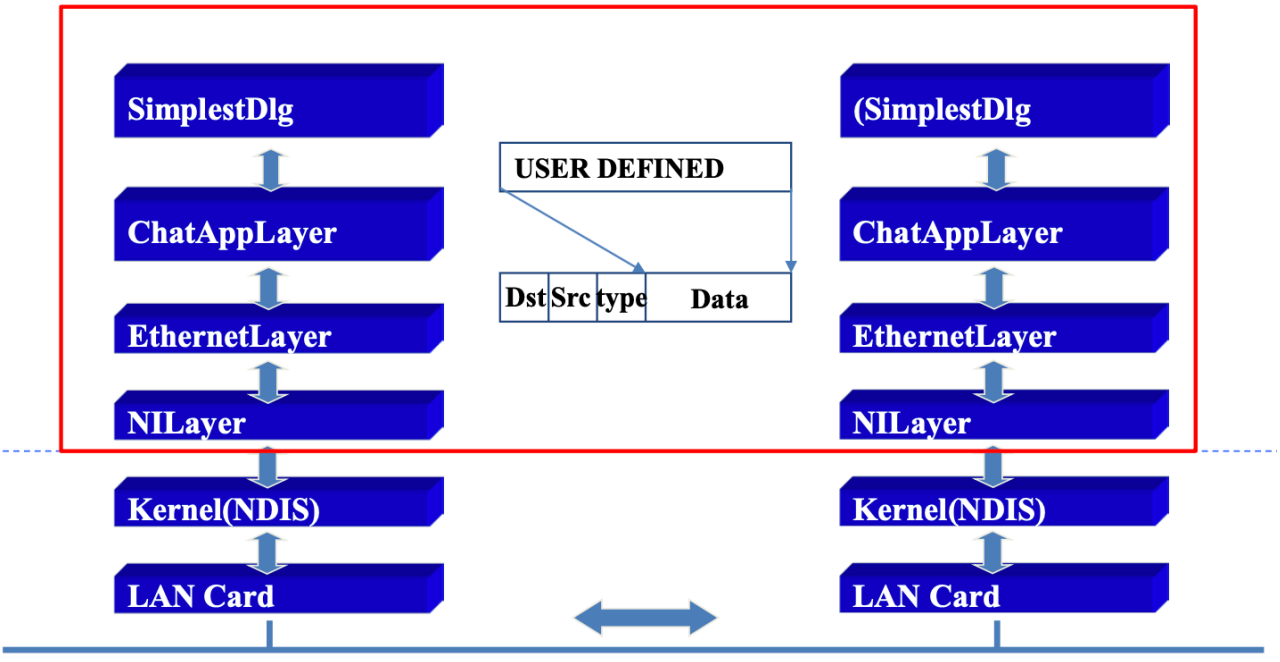
Simplest Protocol은 흐름제어와 에러제어(Data-link Control)가 없는 프로토콜로, 노이즈가 없는 이상적인 채널에서 문제없이 작동하는 프로토콜이다. 이번 실습에서는 Simplest Protocol을 구현해 두 PC간의 직접적인 1:1 통신(2개의 가상머신을 LAN Segment를 통해 연결)을 실습한다.

실습 시나리오

두대의 PC(가상머신)을 직접연결(LAN Segment)해 실습을 진행한다. 송신측에서 수신측으로 메시지를 전송하면 전송된 데이터는 최상위 레이어로부터 각 레이어를 내려가며 데이터에 헤더가 붙여져 데이터 프레임이 만들어지고, 이 데이터 프레임을 수신측에서 받으면 받은 데이터가 자신에게 온 것인지를 확인하고 수신하거나 버린다. 이후 각 레이어를 올라가며 데이터 프레임으로부터 헤더가 분리되어 최종적으로 송신측에서 보낸 데이터가 최상위 레이어에 전달된다.

프로토콜 스택

구조 및 각 레이어의 역할



SimplestDlg 레이어는 GUI역할을 한다. 송/수신 주소를 설정, 메시지를 작성해서 전송할 수 있고, 송수신된 메시지를 표시해준다. 사용자가 메시지를 송신하면 SimplestDlg 레이어에서 데이터는 아래 레이어인 ChatAppLayer로 전달되고, 이를 전달받은 ChatAppLayer는 totlen, type, unused 정보, 4바이트를 헤더로 붙인 데이터 프레임을 EthernetLayer로 보낸다. EthernetLayer는 도착지, 출발지주소와 타입정보, 14바이트를 추가로 붙여서 만들어진 데이터 프레임을 NILayer로 보낸다. NILayer는 수신측으로 데이터프레임을 전송해준다. 수신측에서는 각 레이어들을 올라가면서 송신측의 해당 레이어에서 더해졌던 헤더들을 제거, 최종적으로 SimplestDlg 레이어에서는 송신측에서 사용자가 송신한 메시지가 표시되게 된다.

구현 설명

SimplestDlg.java

```
53    public static void main(String[] args) {
54
55        m_LayerMgr.AddLayer(new NILayer("NI"));
56        m_LayerMgr.AddLayer(new EthernetLayer("Ethernet"));
57        m_LayerMgr.AddLayer(new ChatAppLayer("ChatApp"));
58        m_LayerMgr.AddLayer(new SimplestDlg("GUI"));
59
60        m_LayerMgr.ConnectLayers(" NI ( *Ethernet ( *ChatApp ( *GUI ) ) )");
61    }
```

최상위 GUI(SimplestDlg)부터 최하위 NI(NILayer)까지 각 레이어들을 main에서 연결해주었다.

```
103           if (e.getSource() == Chat_send_Button) {
104               if (Setting_Button.getText() == "Reset") {
105                   String input = ChattingWrite.getText();
106                   ChattingWrite.setText("");
107                   ChattingArea.append("[SEND] : " + input + "\n");
108                   byte[] bytes = input.getBytes();
109                   m_LayerMgr.GetLayer("ChatApp").Send(bytes, bytes.length);
110               }
111               else {
112                   JOptionPane.showMessageDialog(null, "주소 설정 오류");
113               }
114           }
```

Chat_send_Button이 눌렸을 때 Setting_Button의 text가 Reset이면 주소가 설정되었다는 뜻이다. 따라서 ChattingWrite에 적힌 텍스트를 바이트로 만들어 아래 레이어인 ChatAppLayer로 전송한다. Text를 전송했으므로 ChattingWrite의 내용을 지워주고, ChattingArea에 보낸 메시지를 표시해준다. 만약 Setting_Button의 text가 Reset이 아니라면 주소 설정에 문제가 있는것이므로 오류메시지를 표시한다.

```
259   public boolean Receive(byte[] input) {
260       if (input != null) {
261           byte[] data = input;
262           Text = new String(data);
263           ChattingArea.append("[RECV] : " + Text + "\n");
264           return false;
265       }
266       return false;
267   }
```

SimplestDlg의 Receive함수는 최종적으로 전달받은 데이터는 헤더가 모두 제거된 상태이다. 따라서 input을 문자열로 바꿔서 ChattingArea에 받은 메시지를 표시해준다.

ChatAppLayer.java

```
17    private class _CHAT_APP {
18        byte capp_type;
19        byte capp_unused;
20        byte[] capp_totlen;
21        byte[] capp_data;
22
23        public _CHAT_APP() {
24            this.capp_type = 0x00;
25            this.capp_unused = 0x00;
26            this.capp_totlen = new byte[2];
27            this.capp_data = null;
28        }
29    }
30
31    _CHAT_APP m_sHeader = new _CHAT_APP();
```

Total length	type	un-used	Data

ChatAppLayer에서는 _CHAT_APP이라는 헤더를 사용한다. 데이터에 헤더 4바이트를 붙여 데이터프레임을 만든다.

```
49    public byte[] ObjToByte(_CHAT_APP Header, byte[] input, int length) {
50        byte[] buf = new byte[length + 4];
51
52        buf[0] = Header.capp_totlen[0];
53        buf[1] = Header.capp_totlen[1];
54        buf[2] = Header.capp_type;
55        buf[3] = Header.capp_unused;
56
57        for (int i = 0; i < length; i++)
58            buf[4 + i] = input[i];
59
60        return buf;
61    }
```

ChatAppLayer의 ObjToByte함수는 상위 레이어 SimplestDlg로부터 받은 데이터와 헤더를 이용해서 데이터프레임을 만들어낸다. 데이터의 앞에 4개의 헤더를 붙여주고 반환해준다.

```
70    public byte[] RemoveCappHeader(byte[] input, int length) {
71        byte[] buf = new byte[4];
72        byte[] decodedBytes = new byte[length - 4];
73        for(int i = 0; i < 4; i++) {
74            buf[i] = input[i];
75        }
76        for(int i = 0; i < input.length - 4; i ++) {
77            decodedBytes[i] = input[i + 4];
78        }
79        return decodedBytes;
80    }
```

ChatAppLayer의 RemoveCappHeader함수는 하위 레이어 EthernetLayer로부터 받은 데이터의 앞 4바이트를 제거하고, 뒷부분의 데이터를 반환해준다.

```
63    public boolean Send(byte[] input, int length) {
64        byte[] encodedBytes = this.ObjToByte(this.m_sHeader, input, length);
65        debugMSG("인코딩된 바이트스트림 : " + new String(encodedBytes));
66        this.GetUnderLayer().Send(encodedBytes, length + 4);
67        return true;
68    }
```

Send함수는 상위 레이어 SimplestDlg로부터 받은 데이터를 ObjToByte를 통해 캡슐화 하여 하위 레이어인 EthernetLayer로 보내준다.

```
82    public synchronized boolean Receive(byte[] input) {
83        byte[] data;
84        data = RemoveCappHeader(input, input.length);
85        this.GetUpperLayer(0).Receive(data);
86        // 주소설정
87        return true;
88    }
```

Receive함수는 하위 레이어 EthernetLayer로부터 받은 데이터를 RemoveCappHeader를 통해 헤더를 제거하여 상위 레이어인 SimplestDlg로 보내준다.

EthernetLayer.java

```java
    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_HEADER {
        _ETHERNET_ADDR enet_dstaddr;
        _ETHERNET_ADDR enet_srcaddr;
        byte[] enet_type;
        byte[] enet_data;

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

    _ETHERNET_HEADER m_sHeader = new _ETHERNET_HEADER();
```

destination addr	source addr	type	Data

EthernetLayer에서는 주소를 나타낼 _ETHERNET_ADDR두개, 각각 출발지와 도착지 정보를 포함하는 _ETHERNET_HEADER 헤더를 사용한다. 출발지 6바이트, 도착지6바이트와 데이터의 타입을 나타낼 enet_type을 포함해 데이터에 헤더 14바이트를 붙여 데이터 프레임을 만든다.

```java
    public byte[] ObjToByte(_ETHERNET_HEADER Header, byte[] input, int length) {
        byte[] buf = new byte[length + 14];

        buf[0] = Header.enet_dstaddr.addr[0];
        buf[1] = Header.enet_dstaddr.addr[1];
        buf[2] = Header.enet_dstaddr.addr[2];
        buf[3] = Header.enet_dstaddr.addr[3];
        buf[4] = Header.enet_dstaddr.addr[4];
        buf[5] = Header.enet_dstaddr.addr[5];

        buf[6] = Header.enet_srcaddr.addr[0];
        buf[7] = Header.enet_srcaddr.addr[1];
        buf[8] = Header.enet_srcaddr.addr[2];
        buf[9] = Header.enet_srcaddr.addr[3];
        buf[10] = Header.enet_srcaddr.addr[4];
        buf[11] = Header.enet_srcaddr.addr[5];

        buf[12] = Header.enet_type[0];
        buf[13] = Header.enet_type[1];

        for (int i = 0; i < length; i++)
            buf[14 + i] = input[i];

        return buf;
    }
```

EthernetLayer의 ObjToByte함수는 상위 레이어 ChatAppLayer로부터 받은 데이터와 헤더를 이용해서 데이터프레임을 만들어낸다. 데이터의 앞에 도착지, 출발지주소와 타입을 나타낼 14바이트의 해더를 붙여주고 반환해준다.

```
118    public byte[] RemoveEtherHeader(byte[] input, int length) {
119        byte[] data = new byte[length - 14];
120        for (int i = 0; i < length - 14; i++)
121            data[i] = input[14 + i];
122        return data;
123    }
```

EthernetLayer의 RemoveEtherHeader함수는 하위 레이어 NILayer로부터 받은 데이터의 앞 14바이트를 제거하고, 뒷부분의 데이터를 반환해준다.

```
109    public boolean Send(byte[] input, int length) {
110        byte[] type = {0x06, 0x08};
111        this.SetEnetType(type);
112        byte[] bytes = ObjToByte(m_sHeader, input, length);
113        this.GetUnderLayer().Send(bytes, length + 14);
114
115        return false;
116    }
```

Send함수는 type을 0x06, 0x08로 지정, 상위 레이어 ChatAppLayer로부터 받은 데이터와 함께 ObjToByte를 통해 캡슐화 하여 하위 레이어인 EthernetLayer로 보내준다. 0x06, 0x08으로 타입을 지정한 이유는 mac address라는 타입을 명시해주기 위함이다.

```
156    public boolean Receive(byte[] input) {
157        byte[] data;
158        boolean MyPacket, Mine, Broadcast;
159        MyPacket = IsItMyPacket(input);
160
161        if (MyPacket == true){
162            //내가 만든 패킷이면 수신하지 않음.
163            return false;
164        }else {
165            Broadcast = IsItBroadcast(input);
166            if (Broadcast == false) {
167                //브로드 캐스팅도 아니면서,
168                Mine = IsItMine(input);
169                if (Mine == false){
170                    // 목적지가 자신이 아니면 수신하지 않음.
171                    return false;
172                }
173            }
174        }
175        data = RemoveEtherHeader(input, input.length);
176        this.GetUpperLayer(0).Receive(data);
177
178        return true;
179    }
```

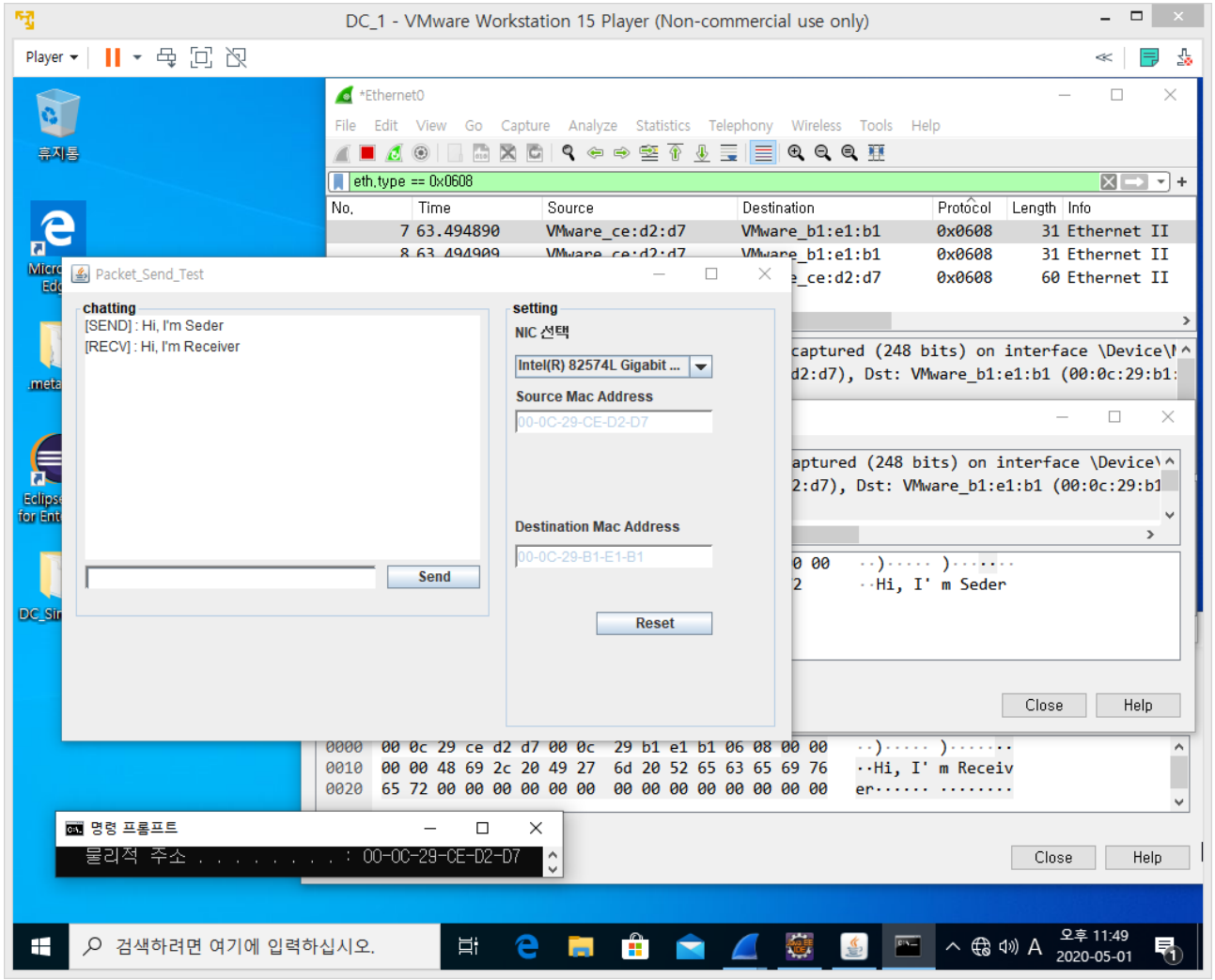
Receive함수는 하위 레이어 NILayer로부터 받은 데이터를 RemoveEtherHeader를 통해 헤더를 제거하여 상위 레이어인 SimplestDlg로 보내준다. 전달받은 데이터가 수신자가 만든 패킷이 아니면서 BroadCast이거나 자신에게 온 패킷일때만 데이터를 취한다.

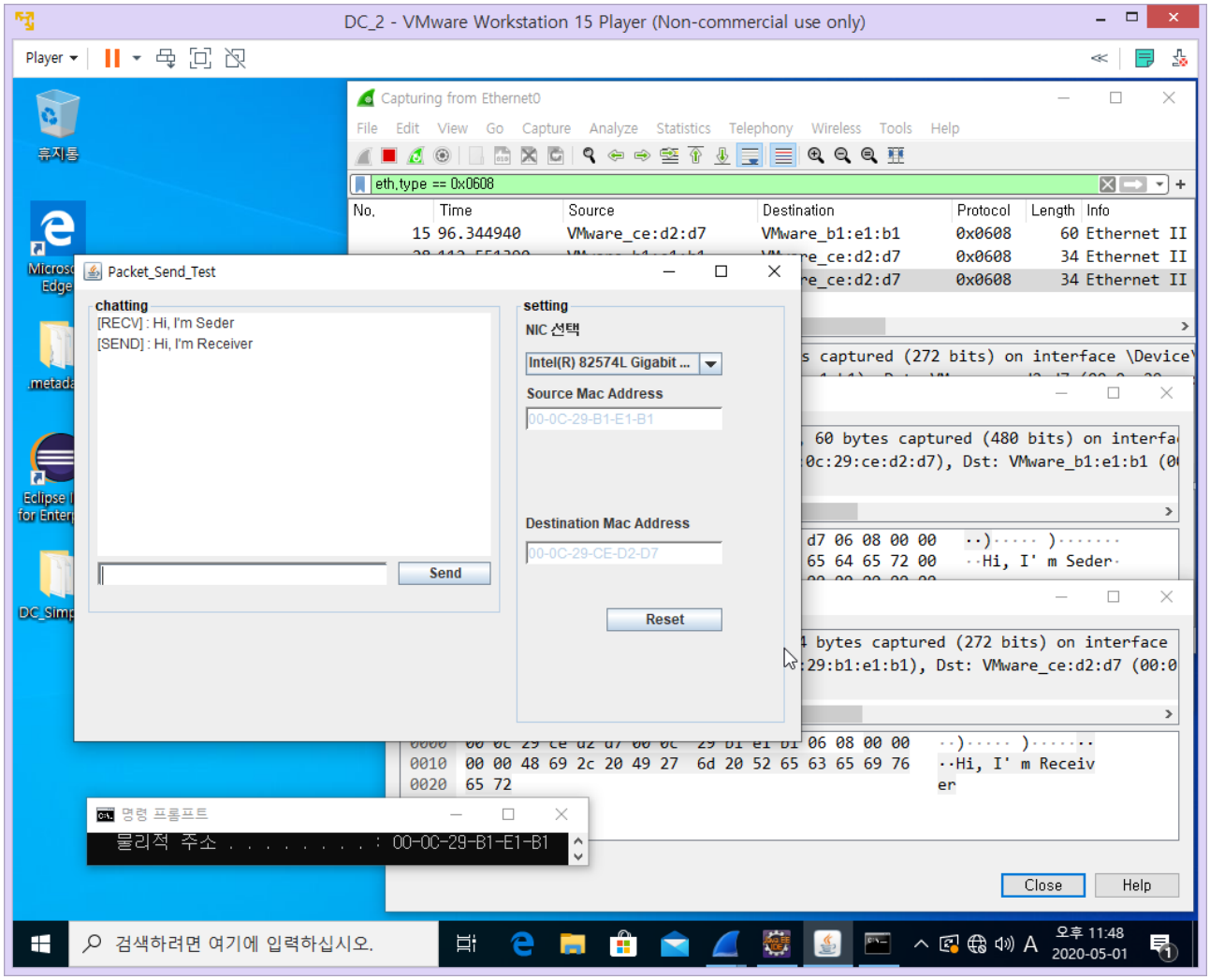
```
125    public boolean IsItMyPacket(byte[] input) {
126        for (int i = 0; i < 6; i++) {
127            if (m_sHeader.enet_srcaddr.addr[i] == input[6 + i])
128                continue;
129            else
130                return false;
131        }
132        return true;
133    }
134
```

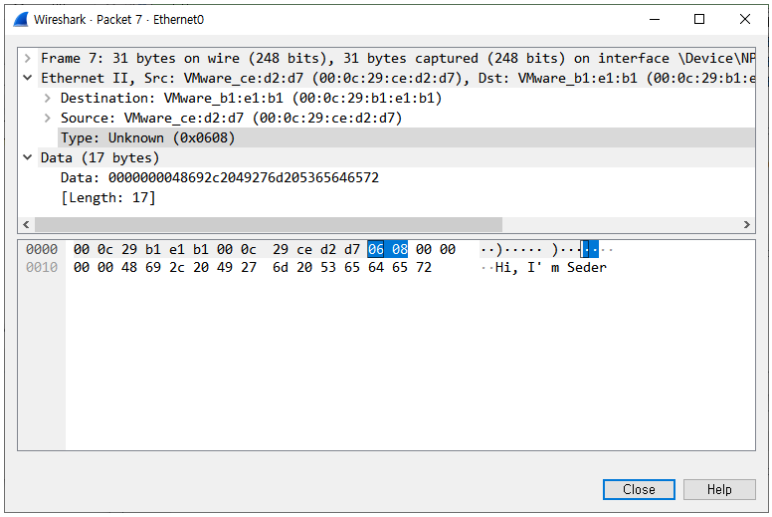
```java
    public boolean IsItMine(byte[] input) {
        for (int i = 0; i < 6; i++) {
            if (m_sHeader.enet_srcaddr.addr[i] == input[i])
                continue;
            else {
                return false;
            }
        }
        return true;
    }

    public boolean IsItBroadcast(byte[] input) {
        for (int i = 0; i < 6; i++) {
            if (input[i] == 0xff) {
                continue;
            } else
                return false;
        }
        return true;
    }
```

IsItMyPacket함수는 출발지 주소와 입력의 6번째 바이트부터 6개의 바이트, 즉 수신된 데이터의 출발지가 같은지를 검사한다. 즉 자신이 보낸 패킷인지를 확인해서 자신이 보낸것이면 true를, 그렇지 않다면 false를 반환한다.
IsItMine함수는 IsItMyPacket과 유사하게 0번째 바이트부터 6개의 바이트, 즉 수신된 데이터의 목적지가 같은지를 검사한다. 즉 자신에게 온 패킷이 맞는지를 확인해서 자신에게 온것이면 true를, 그렇지 않다면 false를 반환한다.
IsItBroadcast함수는 IsItMine함수와 유사하게 0번째 바이트부터 6개의 바이트, 즉 수신된 데이터의 목적지가 0xff인지를 검사한다. Broadcast mac address는 FF:FF:FF:FF:FF:FF이므로 검사를 모두 통과하면 브로드캐스트라는 것이므로 true를, 그렇지 않다면 false를 반환한다.

실행 결과







가상머신 DC_1과 DC_2를 만들었고, LAN Segment로 네트워크 설정을 변경해서 서로 통신에 성공했다. 서로 각자의 MAC Address를 입력해서 연결해주었고, WireShark에 필터로 eth.type == 0x0608로 줘서 주고받은 메시지 패킷도 정상적으로 확인할 수 있었다.