데이터 통신 과제
제출일 : 2020-05-29
201602022 양승훈

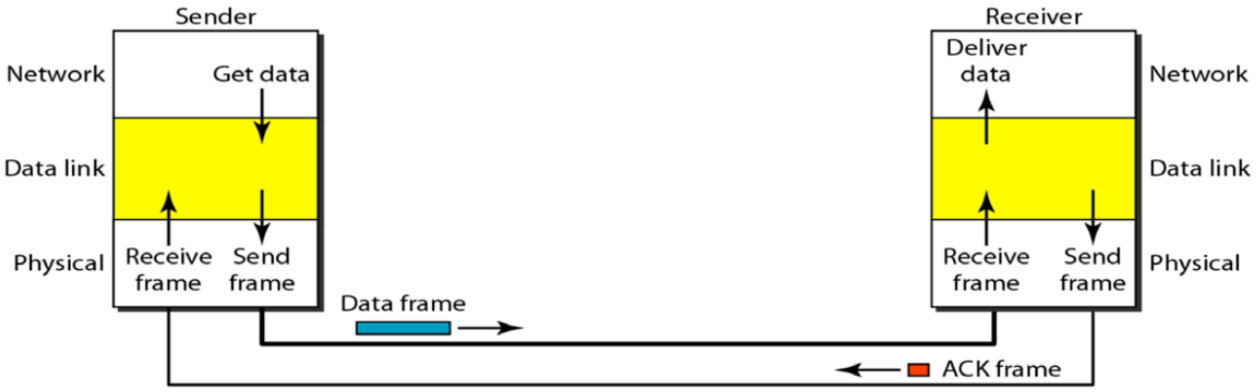
Stop & Wait Protocol

실습 개요

실습 목적

Stop & Wait 프로토콜은 흐름제어만 있는 프로토콜로, ACK를 통해 전송의 흐름을 제어한다. 이번 실습에서는 두 PC간의 직접적인 1:1 통신(2개의 가상머신을 LAN Segment를 통해 연결)으로 실습해야 하지만, PC가 부족하고, 사용중인 랩탑의 저장공간도 부족한 문제로 하나의 가상머신 상에서 2개 프로세스간의 통신으로 대체하여 실습한다.

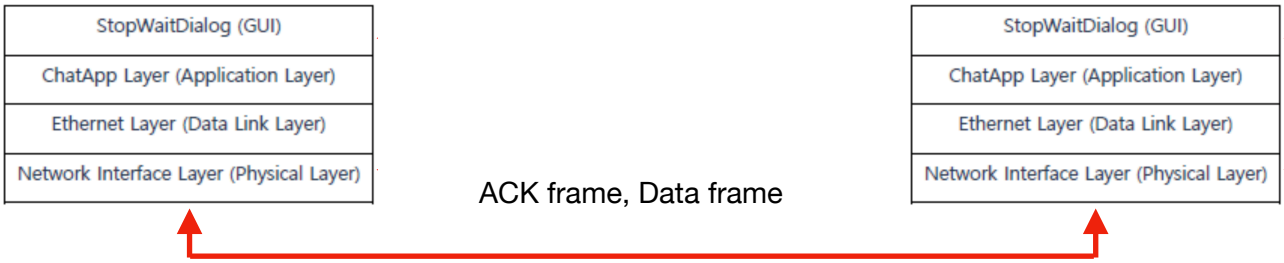
실습 시나리오



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. 이번 실습은 저번 실습 Simplest에 ACK을 주고받는 과정이 더해졌고, 10바이트 단위로 단편화 하여 전송하는 과정이 추가되었다.이후 각 레이어를 올라가며 데이터 프레임으로부터 헤더가 분리되어 최종적으로 송신측에서 보낸 데이터가 최상위 레이어에 전달된다.

프로토콜 스택

구조 및 각 레이어의 역할



StopWaitDlg 레이어는 GUI역할을 한다. 송/수신 주소를 설정, 메시지를 작성해서 전송할 수 있고, 송수신된 메시지를 표시해준다. 사용자가 메시지를 송신하면 StopWaitDlg 레이어에서 데이터는 아래 레이어인 ChatAppLayer로 전달되고, 이를 전달받은 ChatAppLayer는 totlen, type, unused 정보, 4바이트를 헤더로 붙인 데이터 프레임을 EthernetLayer로 보낸다. 이 때 10바이트 단위로 단편화하여 보내며 보내는 부분에 따라 type을 달리하여 보낸다. 또 ackChk과 waitACK함수를

이용해서 ACK의 수신을 통한 흐름제어를 구현한다. EthernetLayer는 도착지, 출발지주소와 타입정보, 14바이트를 추가로 붙여서 만들어진 데이터 프레임을 NILayer로 보낸다. 여기서 type부분을 통해 일반 데이터인지, ACK인지, 브로드캐스트인지를 나타내준다. NILayer는 수신측으로 데이터프레임을 전송해준다. 수신측에서는 각 레이어들을 올라가면서 송신측의 해당 레이어에서 더해졌던 헤더들을 제거, 최종적으로 StopWaitDlg 레이어에서는 송신측에서 사용자가 송신한 메시지가 표시되게 된다.

구현 설명

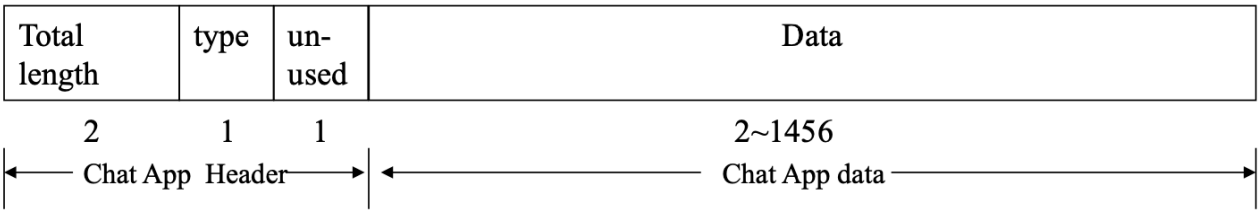
StopWaitDlg.java

```
55    public static void main(String[] args) {
56        m_LayerMgr.AddLayer(new NILayer("NI"));
57        m_LayerMgr.AddLayer(new EthernetLayer("Ethernet"));
58        m_LayerMgr.AddLayer(new ChatAppLayer("ChatApp"));
59        m_LayerMgr.AddLayer(new StopWaitDlg("GUI"));
60
61        m_LayerMgr.ConnectLayers(" NI ( *Ethernet ( *ChatApp ( *GUI ) ) )");
62    }
```

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

ChatAppLayer.java

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



ChatAppLayer의 헤더는 크기를 나타낼 totlen 2바이트와 단편화 정보를 나타낼 type 1바이트, 사용되지 않는 1바이트를 초함해 총 4바이트로 만들어진다.

```
62    private void waitACK() { //ACK 체크
63        while (ackChk.size() <= 0) {                                    //일정시간 대기하며 ack를 체크함
64            try {
65                Thread.sleep(10);
66            } catch (InterruptedException e) {
67                e.printStackTrace();
68            }
69        }
70        ackChk.remove(0);
71    }
```

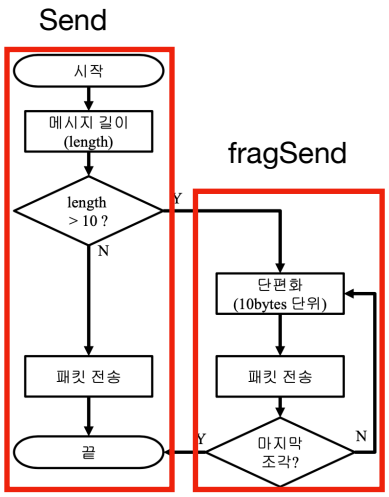
ChatAppLayer에서는 boolean의 리스트 ackChk를 가지고있다. ackChk의 사이즈가 0보다 작다면, 즉 ACK가 없다면 대기하도록 하는 함수다.

```
107   public boolean Send(byte[] input, int length) {
108       byte[] bytes;
109       m_sHeader.capp_totlen = intToByte2(length);
110       m_sHeader.capp_type = (byte) (0x00);
111
112       waitACK();
113       if(length > 10) {                                               //10보다 크면 나눠서 보내야하므로 fragsend
114           fragSend(input, length);
115       }
116       else {                                                          //그렇지 않다면 그냥 보낸다.
117           bytes = objToByte(m_sHeader, input, input.length);
118           this.GetUnderLayer().Send(bytes, bytes.length);
119       }
```

```
120         return true;
121     }
```

Send함수는 아래 레이어로 input을 전달하는 역할을 한다. 그런데 이번 stop and wait protocol에서는 10바이트 단위로 데이터를 단편화 시킬것이므로 여러가지 조건이 추가된다. 일단 waitAck을 통해 Ack을 받은 후에 다시 전송할 수 있도록 해서 흐름 제어를 구현해준다. 만약 전송할 데이터의 크기가 10보다 크다면 이는 단편화해서 전송해야하므로 fragSend함수를 호출해주고, 그렇지 않다면 헤더를 붙여 아래 레이어로 보내준다. 단편화 되지 않았을 때의 타입은 0x00이된다.

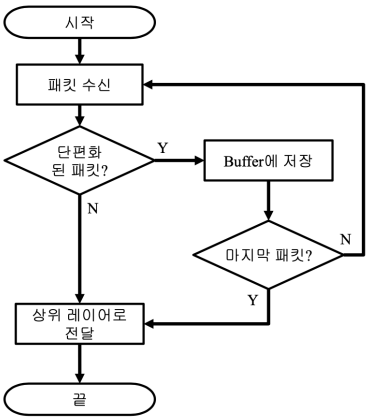
```
73      private void fragSend(byte[] input, int length) {
74          byte[] bytes = new byte[10];
75          int i = 0;
76          m_sHeader.capp_totlen = intToByte2(length);                    //첫 헤더엔 전체 크기?
77          m_sHeader.capp_type = (byte) (0x01);
78
79          // 첫번째 전송
80          System.arraycopy(input, 0, bytes, 0, 10);                      //첫번째 10바이트를 전송
81          bytes = objToByte(m_sHeader, bytes, 10);
82          this.GetUnderLayer().Send(bytes, bytes.length);
83
84          int maxLen = length / 10;
85          m_sHeader.capp_totlen = intToByte2(10);                        //중간 부분은 모두 크기가 10으로 잘려있다.
86          m_sHeader.capp_type = (byte)(0x02);                            //중간 부분은 타입이 0x02이다.
87          for(i = 1; i < maxLen; i++) {
88              waitACK();
89              if(i + 1 == maxLen && length % 10 == 0)                    //마지막 데이터인 경우 타입을 바꿔준다.
90                  m_sHeader.capp_type = (byte)(0x03);
91              System.arraycopy(input, 10 * i, bytes, 0, 10);             //이후 자리부터 10개를 bytes에 복사
92              bytes = objToByte(m_sHeader, bytes, 10);
93              this.GetUnderLayer().Send(bytes, bytes.length);
94          }
95
96          if (length % 10 != 0) {                                        //나머지 부분, 즉 마지막 부분
97              waitACK();
98              m_sHeader.capp_totlen = intToByte2(length % 10);
99              m_sHeader.capp_type = (byte) (0x03);
100             bytes = new byte[length % 10];                             //나머지 부분의 크기에 맞는 바이트 배열 선언
101             System.arraycopy(input, length - (length % 10), bytes, 0, length % 10);     //이후 자리부터 나머지 수만큼 복사
102             bytes = objToByte(m_sHeader, bytes, bytes.length);
103             this.GetUnderLayer().Send(bytes, bytes.length);
104         }
105     }
```



fragSend함수는 단편화되어 데이터를 전달해야하는 경우에 호출된다. 단편화 될 데이터는 3가지 부분으로 나눌 수 있다. 맨 앞부분 0x01, 중간에 들어갈 데이터들은 0x02, 마지막 부분 0x03이 있다. 일단 첫번째 10바이트를 따서 0x01타입으로 아랫레이어에 전달해준다. send함수에서 이미 waitAck이 불렸으므로 그냥 보내면 된다. 첫번째 단편화데이터프레임에서의 사이즈 정보 totlen은 전체 데이터의 사이즈 정보를 갖는다. 이후부터는 중간부분을 10바이트씩 나누어(for문 반복) 0x02타입으로 보내준다. 보내기 전에 waitAck을 호출해 흐름을 제어해준다. 중간 데이터들은 모두 10바이트로 나눴기 때문에 totlen으로 10을 갖는다. 만약 데이터길이가 10으로 나누어 떨어진다면 마지막 단편화 데이터도 0x02타입으로 보내질 수 있으므로 반복문 안에서 조건문을 통해 이런 상황을 방지해준다. 이후 마지막 남은 단편화 데이터는 waitAck후에 남은 크기만큼 0x03타입으로, totlen으로 남은 데이터 크기만큼으로 설정해서 보내준다. receive함수는 아랫 레이어로부터 null을 받으면 ack을 받은것이므로 ackChk에

```
123     public synchronized boolean Receive(byte[] input) {
124         byte[] data, tempBytes;
125         int tempType = 0;
126
127         if (input == null) {                                           //null을 받으면 ack이다
128             ackChk.add(true);
129             return true;
130         }
131
132         tempType |= (byte) (input[2] & 0xFF);                          //헤더의 2번인덱스, 타입
133
134         if(tempType == 0) {                                            //타입이 0이라면 단편화 되지 않고 한번에 받아온 프레임
135             data = RemoveCappHeader(input, input.length);              //헤더를 제거하고
136             this.GetUpperLayer(0).Receive(data);                       //윗 레이어로 전달
137         }
138         else{
139             if(tempType == 1) {
140                 int size = byte2ToInt(input[0], input[1]);             //헤더의 0, 1번 인덱스, 크기
141                 fragBytes = new byte[size];                            //크기만큼의 fragBytes배열을 만들어줌
142                 fragCount = 1;
143                 tempBytes = RemoveCappHeader(input, input.length);     //헤더가 제거된것을 tempbytes로 저장
144                 System.arraycopy(tempBytes, 0, fragBytes, 0, 10);      //fragBytes의 첫 위치에 tempBytes를 넣어줌
145             }
146             else {
147                 tempBytes = RemoveCappHeader(input, input.length);     //헤더 제거된것을 tempbytes로 저장
148                 System.arraycopy(tempBytes, 0, fragBytes, (fragCount++) * 10, byte2ToInt(input[0], input[1]));     //fragbytes의 알맞은 위치에 tempbytes를 추가
149                 if(tempType == 3) {
```

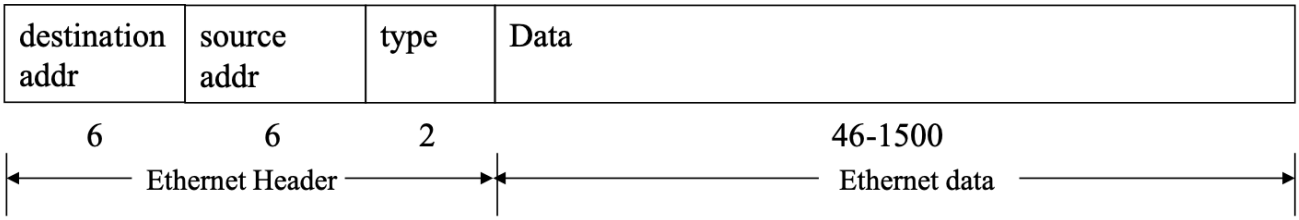
```
150                 this.GetUpperLayer(0).Receive(fragBytes);          //마지막 단편화 데이터였다면 모두 모인것이므로 윗 레이어로 전달
151             }
152         }
153     }
154     this.GetUnderLayer().Send(null, 0);                            //ack송신
155     return true;
156 }
```



true를 추가해준다. 그럼 ack을 받았으므로 waitAck에서 잠자던 스레드들이 깨어나게 된다. 만약 받은 헤더의 타입이 0이라면 단편화되지 않고 전달된 데이터이다. 헤더만 제거해서 윗 레이어로 전달해주면 된다. 타입이 1이면 첫번째 단편화 데이터이다. 첫 단편화 데이터의 totlen은 전체 데이터의 크기를 나타내므로 이를 이용해 전체 데이터가 합쳐질 공간 fragBytes를 확보해준다. 첫번째 단편화 데이터를 수신한것이므로 fragCount도 1로만들어준다. 그리고 해당 단편화 데이터에서 헤더를 제거해 fragBytes에 넣어준다. 첫번째 단편화 데이터가 아닌 데이터는 계속 받아오면서 헤더를 제거해 fragBytes의 뒤에 붙여준다. 만약 마지막 단편화 데이터가 들어오게되면 모두 수신한것이므로 fragBytes를 상위 레이어로 전달해준다. 이렇게 모두 수신이 되면 정상적으로 수신했다는 의미로 ACK을 보내준다.

EthernetLayer.java

```
23  private class _ETHERNET_ADDR {
24      private byte[] addr = new byte[6];
25
26      public _ETHERNET_ADDR() {
27          this.addr[0] = (byte) 0x00;
28          this.addr[1] = (byte) 0x00;
29          this.addr[2] = (byte) 0x00;
30          this.addr[3] = (byte) 0x00;
31          this.addr[4] = (byte) 0x00;
32          this.addr[5] = (byte) 0x00;
33
34      }
35  }
36
37  private class _ETHERNET_Frame {
38      _ETHERNET_ADDR enet_dstaddr;
39      _ETHERNET_ADDR enet_srcaddr;
40      byte[] enet_type;
41      byte[] enet_data;
42
43      public _ETHERNET_Frame() {
44          this.enet_dstaddr = new _ETHERNET_ADDR();
45          this.enet_srcaddr = new _ETHERNET_ADDR();
46          this.enet_type = new byte[2];
47          this.enet_data = null;
48      }
49  }
```



Ethernet layer의 헤더는 목적지 맥주소 6바이트, 출발지 맥주소 6바이트, 그리고 데이터인지, ACK인지를 나타낼 type 2바이트를 가져 총 14바이트의 헤더를 가진다.

```
66  public boolean Send(byte[] input, int length) {
67      if (input == null && length == 0)                          // ack는 input이 없고 length가 0인경우. type을 2로 설정
68          m_sHeader.enet_type = intToByte2(2);
69      else if (isBroadcast(m_sHeader.enet_dstaddr.addr))         // broadcast는 0xff
70          m_sHeader.enet_type = intToByte2(0xff);
71      else                                                       // nomal는 1
72          m_sHeader.enet_type = intToByte2(1);
73
74      byte[] bytes = ObjToByte(m_sHeader, input, length);        //인코딩하여 아랫 레이어로 전달
75      this.GetUnderLayer().Send(bytes, length + 14);
76      return true;
77  }
```

Ethernet레이어에서 input이 없다면 ack이므로 type을 2로, 도착지 주소가 브로드캐스트이면 type을 0xff, 그렇지 않다면 일반 데이터로, type을 1로 설정해준다. 이후엔 그에 따라 input에 헤더를 붙여서 아랫 레이어로 보내준다.

```
86  public synchronized boolean Receive(byte[] input) {
87      byte[] data;
88      byte[] temp_src = m_sHeader.enet_srcaddr.addr;
89      int temp_type = byte2ToInt(input[12], input[13]);
90
91      if(temp_type == 2) {                                    //type이 2이면 ack이다.
92          this.GetUpperLayer(0).Receive(null);
93      }
94      else if(temp_type == 1) {                               //type이 1이면 normal이다.
95          if(chkAddr(input) || isBroadcast(input) || !isMyPacket(input)) {   //자신에게 온것이거나 브로드캐스트이거나 자신이보낸것이 아니라면
96              data = RemoveEthernetHeader(input, input.length);              //헤더를 제거하고
97              this.GetUpperLayer(0).Receive(data);                           //상위 레이어에 전달
98              this.Send(null, 0);                                            //ack전송
99              return true;
100         }
101     }
102     return false;
103 }
```

receive함수는 type이 2이면 ack이므로 상위 레이어에 null로 보내준다. 그렇지 않다면 헤더를 제거해 상위 레이어로 전달해주고 수신했으므로 ack을 송신해준다.

실행 결과

macOS상의 가상머신 Parallels에서 돌아가는 Windows10에 두개의 Stop & Wait Protocol 프로세스를 실행해 서로 각자의 MAC Address(00-1C-42-5A-D4-32, 00-1C-42-5A-D4-33)를 입력해서 연결해주었고, WireShark에 필터로 eth.addrs == 00-1C-42-5A-D4-33로 줘서 주고받은 메시지 패킷도 정상적으로 확인할 수 있었다. 데이터는 타입이 00 01, ACK는 00 02타입으로 나오는것을 확인할 수 있었다.