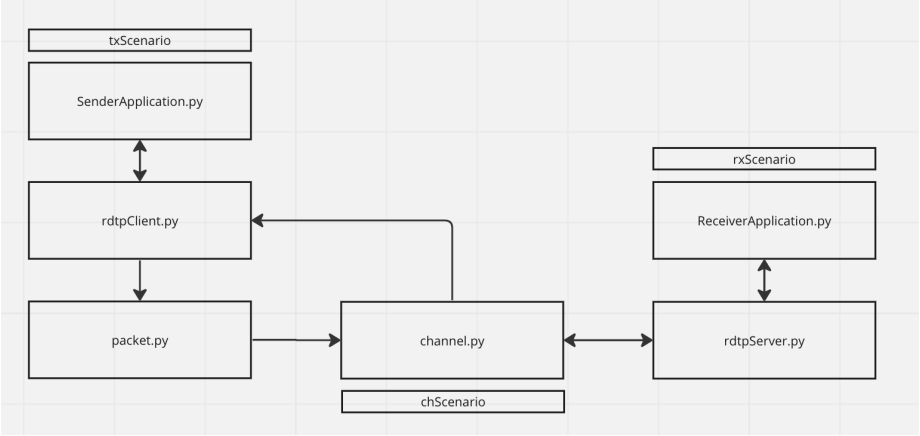
Computer Networks: Reliable Datagram Transfer Protocol

이준희

2023.05.28

1 Overview

RDTP를 구현하기 위해 Applicatoin Layer와 Transport Layer 그리고 단순한 Network Layer를 구현하였다.
과제의 조건을 달성하기 위해 제시된 Scenario를 Configuration File에 정의한 후 각기 객체에서 필요한 부분을 Retrieve하도록 구성되었다.
구성된 파일들은 다음과 같은 관계를 지닌다. 표현의 편의를 위해 ConfigurationFile(Scenario 파일)은 표현하지 않았다.



2 Modules and Relationship

각 파일의 역학과 중요 함수의 역학을 소개하고자한다

2.1 ConfigReader.py

ConfigReader.py는 미리 주어진 RDTP.conf파일을 읽어와 변수 이름을 Key로 값을 value로 가지는 Dictionary 형태를 반환한다.

Configuration이 필요한 곳에서 자유롭게 호출할 수 있다.

2.2 channel.py

Packet이 지나가는 Network Layer를 모사하는 파일이다.
chScenario.txt파일에 서술된 규칙에 따라서 통신에 영향을 끼치는 Network의 다양한 상황을 구현한다.

다음과 같은 상황을 구현할 수 있다

`Latency, Loss, Congestion(Small, Big)`

다음 코드를 통해 Raw한 Bytes데이터를 가져오게 된다.

```
bytesAddressPair = UDPChannelSocket.recvfrom(bufferSize)
```

이후 Bytes의 Header부분을 가져와 Packet이 가야할 Address와 Port를 확인한다

```
rxIP = msg[0:9].decode("utf-8")
rxPort = msg[9:13].decode("utf-8")
txIp = msg[13:22].decode("utf-8")
txPort = msg[22:26].decode("utf-8")
```

다음 코드는 주어진 channel scenario에 따라 적절한 상황 묘사를 하는 부분이다.

```
if channelBehavior == "N":
    time.sleep(t)
elif channelBehavior == "L":
    loss = True
elif channelBehavior == "C":
    time.sleep(t + D)
elif channelBehavior == "c":
    time.sleep(t + d)

if not loss:
    print(f"from {rxIP}:{rxPort} to {txIp}:{txPort}")
    data = msg
    UDPChannelSocket.sendto(data, (rxIP, int(rxPort)))

if count != "INF":
    count -= 1
```

2.3 senderApplication.py

senderApplication은 주어진 Scenario에 따라 Dummy Data를 생성한 후 하위 Layer에 Push하게 된다.

다음은 Scenario에 따라 만들어진 Dummy Data Segment를 모두 Push될 때까지 시도하는 코드이다
하위 Layer의 WindowSize상황에 따라 실패될 때 마다 계속 시도하게 된다.

```
while len(filesList) > 0:
    data = filesList[0]
    result = client.sendData(data)
    if not result:
        # PUSH 실패 -> Delay 후 다시 시도
        time.sleep(0.5)
    else:
        filesList.pop(0)
        if len(delayList) > 0:
            sleepTime = delayList[0]
            delayList.pop(0)
            time.sleep(sleepTime)
```

모든 파일이 하위 Layer로 Push된 후 하위 Layer의 파일 전송 상황을 Check한다
하위 Layer에서 파일 전송이 모두 완료되었음이 확인되면 Exit()한다.

```
while True:
    result = client.isOver()
    if result:
        print("sending done!")
        break
```

2.4 rdtpClient.py

rdtpClient.py는 상위 Layer에서 Push된 데이터를 직접 송신하는 파일이다.
상위 Layer와는 파일의 Push성공 여부 그리고 모든 Segment 전송 여부를 교환하고
하위 Layer와는 Packet을 만들어 보내고 다시 받는 일을 한다.

다음과 같은 함수가 있다.

`def __init()`: 필요한 변수를 초기화하며, `Timer`와 , `Listener`를 비동기화된 `Process`로 독립실행, `Socket`을 열고, `Hand-Shake`를 진행한다.

`def __timer()`: 특정한 시간마다 반복적으로 `ACK`되지 못한 `Packet`이 있는지 확인하고 필요시 새롭게 `Packet`을 보낸다.
+ 전체 `Process`의 실행동안 실행 중이며, 등록된 `Packet`이 없다면 아무것도 하지 않는다.

`def __innerSend()`: 반복적으로 보내야하는 `Packet`이 있을 때 전송하는 역할을 하고 필요한 변수를 `Update`한다.
+ `__timer()`에 `Packet`을 등록한다.

def __listener(): Channel로 부터 들어오는 data를 분석하여 필요한 Process를(Hand-Shake확인, 연결 해제, ACK) 시행한다.

def sendData(): 상위 Layer가 호출하는 함수. PUSH받은 data를 Window상황, HandShake가 된 상태인지를 고려해 반환을 결정한다.

def isOver(): 상위 Layer가 호출하는 함수. 모든 데이터 전송이 끝나고 연결 해제가 되었는지 확인하는 함수.

2.5 packet.py

packet.py는 rdtpClient가 보내야하는 segment단위로 생성된다. segment data에 Header를 씌우고 전송한 후 삭제된다.

다음과 같은 함수가 있다.

def __init__(self, UDPClientSocket, bytesToSend, size, mSeqNum, rxIP, rxPort, myIP, myPort):
 -> rdtpClient.py로 부터받은 UDPClientSocket을 준비하고. __send()를 호출한다.

def makePkt(self, bytesToSend, size, rxIP, rxPort, myIP, myPort):
 -> bytesToSend의 data에 주어진 size, rxIP, rxPort, myIP, myPort의 데이터를 Header로 새긴다.

def __send(self, bytesToSend, size, rxIP, rxPort, myIP, myPort):
 -> makePkt()을 통해 만들어진 Packet을 Channel로 전송한다.

2.6 receiverApplication.py

receiverApplication.py는 하위 Layer를 인스턴스화 시키고,
Scenario에 따라 주어진 시간의 Term을 두고 하위 Layer에 도착한 Data를 읽어들인다.

다음과 같은 코드로 하위 Layer로부터 Data를 읽어들이고, rxScenario에 따라 Term을 둔다. 모든 파일을 받고 연결 해제가 될 때까지 Scenario는 반복된다.

```
while True:
    result = server.retrieveData() # Data 가져오기
    if result == "NONE":
        print(f"receiver application sleeps for {rxScenario[i]}")
        time.sleep(rxScenario[i])
        i += 1
        if i >= len(rxScenario):
            i = 0
    elif result == "KILL":
```

```
        print("rx done")
        break
```

2.7 rdtpServer.py

```
def __init__():
    -> Socket을 초기화하고 __startServer를 비동기 Thread에서 실행시킨다.

def __startServer(self):
    -> 들어오는 Segment를 처리하는 함수이다.
    ...
    -> 다음과 같이 Header를 분리해낸다.
    txIP = msg[13:22].decode("utf-8")
    txPort = msg[22:26].decode("utf-8")
    seqStartNum = msg[26:36].decode("utf-8")
    lastSeqNum = msg[36:46].decode("utf-8")
    ...

    -> 들어온 Data에 대해 들어온 값의 Type에 따라 처리를 다르게 한다
    if body.isalpha() and body.decode("utf-8") == "DONE":
        -> DONE은 모든 파일 전송이 끝났으며, Sender에서 연결을 해제하고
자 할 때를 나타낸다.
        ...
    elif body.isascii() and body.decode("utf-8").__contains__("REDY"):
        -> REDY는 Sender로 부터 들어오는 HandShake 신호이다.
        ...
    else:
        -> 이외 데이터는 실제 보내오는 Data이다.
        ...
        if self.availableWindow < len(body):
            -> 현재 Receiver가 받을 수 있는 Window의 남은 크기를 넘어서
는 Data는 폐기한다.
            ...
        elif int(seqStartNum) == rcvBase and self.availableWindow >= len(body):
            -> 현재 Receiver가 받을 수 있는 Window의 남은 크기보다 작으
며 rcvBase에 맞는, 즉 순서에 맞는 Data는 받는다.
            ...
        else:
            -> 규칙에 맞지 않는 데이터는 폐기한다
            ex) 순서에 맞지 않는 데이터가 이에 해당된다.
            ...
    ...


def retrieveData(self):
    -> 상위 Layer에서 Window에 들어온 Segment를 Read할 수 있는 함수이다.
```

데이터가 있다면 Data자체를
없다면 NONE을
연결이 해제되었다면 KILL를 반환한다.

2.8 statusMonitor.py

statusMonitor.py는 Sender와 Receiver가 필요한 Event가 일어났을 때 마다 현재 자신의 상태를 나타내는 출력을 담고 있는 파일이다.

주어진 Parameter에 따라 Terminal에 출력한다.

```
def receiverStatus(wndwStart=0, wndwEnd=0, LastByteRcvd=0, LastByteRead=0, status=0, pktInWi
    -> receiver의 상태를 표현하는 함수이다


def senderStatus(wndwStart=0, wndwEnd=0, LastByteAcked=0, LastByteSent=0, LastByteWritten=0,
                 AckedNum=0, unAckedNum=0, unSentNum=0, windowSeqList = list()):
    -> receiver의 상태를 표현하는 함수이다
```

다음과 같은 출력물을 가진다
receiver status

```
------------------Receiver Status------------------
unAcked : □ | unSent : □ | acked : ■
LastByteRcvd 1221 | LastByteRead 993 | windowAvailiable 284
status : received packet
Window Status : 993[[1121][1221]]1221
---------------------------------------------------
```

sender status

```
------------------Sender Status------------------
unAcked : □ | unSent : □ | acked : ■
LastByteAcked 277 | LastByteSent 624 | LastByteWritten 724
sending out data at 1685275755.0306401 | size : 100
Window Status : 277[[397][496][624][724]]724
---------------------------------------------------
```

```
------------------Sender Status------------------
unAcked : □ | unSent : □ | acked : ■
LastByteAcked 277 | LastByteSent 624 | LastByteWritten 724
Window Status : 277[[397][496][624][724]]724
---------------------------------------------------
```

2.9 Config Files

configuration files를 간략하게 설명한다.

`txScenario.txt` : `Sender`가 작동할 때 필요한 `Scenario`를 서술한 파일. 과제서에 서술된 규칙을 다른다.

`rxScenario.txt` : `Receiver`가 작동할 때 필요한 `Scenario`를 서술한 파일. 과제서에 서술된 규칙을 다른다.

`chScenario.txt` : `Channel`이 작동할 때 필요한 `Scenario`를 서술한 파일. 과제서에 서술된 규칙을 다른다.

`RDTP.conf` : 각 파일들이 작동할 때 필요한 상수들을 담은 파일. 과제서에 서술된 규칙을 다른다.