컴퓨터네트워크개론

과제 #2: UDP 환경에서의 RDT 구현

김영훈 교수

1. 목표

- UDP 소켓 프로그래밍을 사용하여 신뢰할 수 있는 데이터 전송(RDT) 프로토콜을 개발합니다.
    - RDT(Reliable Data Transfer Protocol)을 단계별로 구현하게 됩니다. 이 과제에서는 파이프라이닝이 포함된 1.0, 2.2, 3.0 및 3.0의 네 가지 RDT 버전을 사용합니다.

2. 개발 환경

- 채점 시 사용하는 시스템 환경은 아래와 같습니다.
    - Sender & Receiver
        - x86-64 CPU
        - 가상 머신(우분투 18.04)
        - Python 3.9
        - Sender와 Receiver 가상 머신이 브리지 네트워크 구성으로 연결 (안내 ppt 확인 요).
- 가급적 개발 및 테스트 환경을 위 환경과 동일하게 맞추어 주시고, 실제 사용하는 환경이 다른 경우 보고서에 명시해주세요. 만약 보고서에 환경을 명시하지 않은 경우 채점 환경에서 동작하지 않았을 경우 감점에 어떠한 이의도 제기하실 수 없습니다.

3. 구현 세부 정보

- 상위 버전의 RDT가 이전 버전을 기반으로 구축되므로 각 RDT 버전을 순서대로 구현하세요.
    - 공통 기능
        - 기본 절차
            - 일대일 상황만 가정하며, 다중 사용자 사례는 고려하지 않습니다.
            - [Receiver] Receiver가 소켓을 열고 **포트 10090에** 바인딩하여 Sender의 전

송을 기다립니다.

- [Sender] Sender는 Receiver의 IP와 포트 번호로 Receiver를 연결할 소켓을 만듭니다. 사용자 입력으로 지정된 파일을 전송합니다. 파일 전송이 완료되면 연결을 닫습니다.
- [Receiver] 파일 전송이 완료되면 Receiver도 연결을 닫습니다.

◆ 매개변수

- [Sender] Sender 프로그램은 실행 시 다음 네 가지 매개변수를 받습니다.

  〈receiver's IP address〉〈window size〉〈source file name〉〈log file name〉

- [Receiver] Receiver 프로그램은 실행 시 다음 두 가지 매개변수를 받습니다.

  〈result file name〉〈log file name〉

- 실행 시 매개변수가 충분히 제공되지 않으면 프로그램을 종료합니다.

◆ 패킷

- 각 RDT 버전에 따라 불안정한 상황에 적절히 대처할 수 있도록 사용자 지정 패킷 헤더를 설계하세요.
- 패킷이 헤더를 위한 충분한 공간을 확보할 수 있도록 각 페이로드의 크기가 너무 크지 않도록 하는 것을 추천합니다.
- **비트 오류와 패킷 손실은 Sender 측에서만 발생한다고** 가정합니다. ('4. 평가 시나리오' 부분 참조).

◆ 로그 파일

- Sender와 Receiver모두 로그 파일을 작성해야 합니다.
- *logHandler.py* 와 *logHandler_fct.py* 가 제공됩니다. 편의에 따라 'logHandler class' 또는 'log handling function' 중 하나를 사용하세요.

◆ 라이브러리 사용

- 비표준 라이브러리는 어떠한 경우에도 사용하실 수 없습니다.
- 표준 라이브러리라고 하더라도 HTTP를 직접 처리해주는 경우를 포함하여 과제 해결에 직접적으로 영향을 미치는 경우 사용하실 수 없습니다.
- 허용된 모듈 예: sys, os, threading, time, socket
- 위에 '가능한 모듈 예시'로 언급되지 않은 모듈을 사용하실 경우에는 구글 스

프레드시트에 문의를 남겨 주세요. 제출 이후에는 허용되지 않은 모듈 사용으로 인한 감점에 대해 이의 제기를 받지 않습니다.

- **P1: RDT 1.0**
  - 기본 채널이 완벽하게 신뢰할 수 있다고 가정합니다.
  - Receiver가 Sender로부터 지정된 파일을 받는 프로그램을 만드세요.
  - 헤더는 고려하지 않습니다.
  - window size는 1로 고정되어 있습니다.

- **P2: RDT 2.2**
  - RDT 1.0의 모든 기능이 포함되어야 합니다.
  - 기본 채널에서 비트 오류가 발생할 수 있다고 가정합니다.
  - **UDP가 계산하는** 방식으로 **Check Sum 값을** 계산하여 수신된 패킷이 손상되었는지 확인합니다.
  - **NAK-free 프로토콜을** 구현하여 정상 패킷과 손상된 패킷을 모두 올바르게 처리합니다.
  - window size는 1로 고정되어 있습니다.

- **P3: RDT 3.0**
  - RDT 2.2의 모든 기능이 포함되어야 합니다.
  - 기본 채널이 패킷 손실을 일으킬 수 있다고 가정합니다.
  - Sender가 **0.01초** 동안 ACK를 수신하지 못하면 패킷 손실로 간주하고 데이터를 재전송합니다.
  - 타이머를 위한 추가 thread를 생성하는 대신 파이썬 소켓 라이브러리의 'setimeout' 메서드를 사용하는 것을 추천합니다. (다음 페이지 참조: https://docs.python.org/3/library/socket.html#socket.socket.settimeout)
  - Sender와 Receiver 모두 중복된 패킷을 처리해야 합니다.
  - window size는 1로 고정되어 있습니다.

- **P4: RDT 3.1**
  - RDT 3.0의 모든 기능이 포함되어야 합니다.
  - RDT 3.0과 동일한 상황이지만 이 버전에서는 파이프라이닝 기술을 사용하여 링크 활용도를 확장한다고 가정합니다.
  - 이제 window size가 달라집니다. Sender는 ACK를 기다리지 않고 window size에 맞게 여러 패킷을 전송할 수 있습니다. 다시 말해, window size는 Sender가 전송할 수 있는 unacknowledged 패킷의 최대 개수를 나타냅니다.
  - 패킷 손실 복구를 위해 **'Selective Repeat'** 전략을 사용합니다. 교안에 나온 Selective Repeat 동작 방식을 참고하세요. ([recv_base -window, recv_base -1] 이면 ack, [recv_base, recv_base + window -1] 이면 ack 및 buffer(out order) 혹은 deliver (in order) 등.)
  - Receiver 또한 window size에 맞춘 버퍼가 필요하며, Sender에 입력 받은 window size를 Receiver에 공유하여 Sender와 Reciever간의 window size sync를 맞추도록 구현하세요. Receiver가 Sender에게 window size를 공유하는 소캣의 경우 reliable하다 가정하여도 됩니다. (비트 오류와 패킷 손실이 없다고 가정하고 소켓 라이브러리의 기본 전송 기능을 사용)
  - Window size는 최소 5에서 최대 10 사이의 값을 테스트 할 예정이며, **Selective Response 딜레마에 빠지지 않는, 최소 Sequence number space를 사용하도록 구현하세요.** (힌트 : SR은 window size가 sequence number의 개수의 x배 이하여야한다: x배 사용)
  - 이 과제에서는 Sender가 윈도우 크기만큼 많은 패킷을 전송하기 전에 하나의 ACK도 도착하지 않도록 RTT(왕복 시간)가 항상 충분히 크다고 가정하면 됩니다.
- 요약 표

	RDT 1.0	RDT 2.2	RDT 3.0	RDT 3.0 + 파이프라이닝
소켓	O	O	O	O
헤더	X	O	O	O
타이머	X	X	O	O
파이프라이닝	X	X	X	O

4. 평가 시나리오(총 100점)

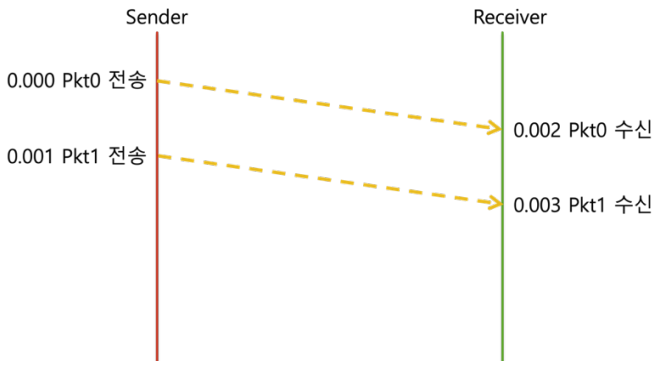
- 전송 및 로깅 기능을 위한 코드가 제공됩니다.
  - 테스트 목적으로 다음 파일을 수정해도 되나, TA는 평가를 위해 원본 파일을 사용한다는 점에 유의하세요.
  - *PASender.py*
    - *config.txt에서* 값을 편집하여 비트 오류 및 패킷 손실의 다양한 상황을 시뮬레이션 할 수 있습니다.
    - **Sender 측에서는** 이 'PASender'를 사용하여 손실 환경을 만듭니다.
    - **Receiver측에서는** 비트 오류와 패킷 손실이 없다고 가정하고 소켓 라이브러리의 기본 전송 기능을 사용합니다.
  - *logHander.py* (또는 *logHandler_fct.py*)
    - 전체 패킷 처리 과정을 기록하는 파일을 생성해줍니다.
    - TA는 로그 파일을 보고 프로그램을 평가하므로 오타가 있으면 해당 시나리오에서 불합격 처리됩니다. 가능하면 미리 정의된 변수와 함수를 사용하세요.
    - 'class' 와 'function' 중 어떤 로깅 유형을 선택할지 명시하세요.
  - make_test_file.py
    - 전송할 파일을 임의로 만들어 테스트 해볼 수 있는 make_test_file.py가 제공됩니다.
    - 이를 활용하여 원하는 크기의 테스트 파일을 만들어 실험해볼 수 있습니다.

- Sender와 Receiver양쪽에서 로그 파일을 어떻게 작성해야 하는지에 대한 이해를 돕기 위해 다음 예시가 제공됩니다.
  - 예시로 제공된 것으로, 아래 로그에 있는 시간은 정확하지 않을 수 있습니다.

- **P1: RDT 1.0(20점)**
  - Sender가 Receiver에게 작은 파일(≤ 1KB)을 전송합니다.
  - 비트 오류와 패킷 손실이 모두 존재하지 않는다고 가정합니다.
    (loss_rate = 0, corrupt_rate = 0)
  - 시나리오 예시

0.000 pkt: 0 | Send DATA 0.001 pkt: 0 | Send DATA File transfer is finished.
Sender측 로그

    - ◆ RDT 1.0에서는 Sender 쪽만 로깅하는 것으로 충분합니다.



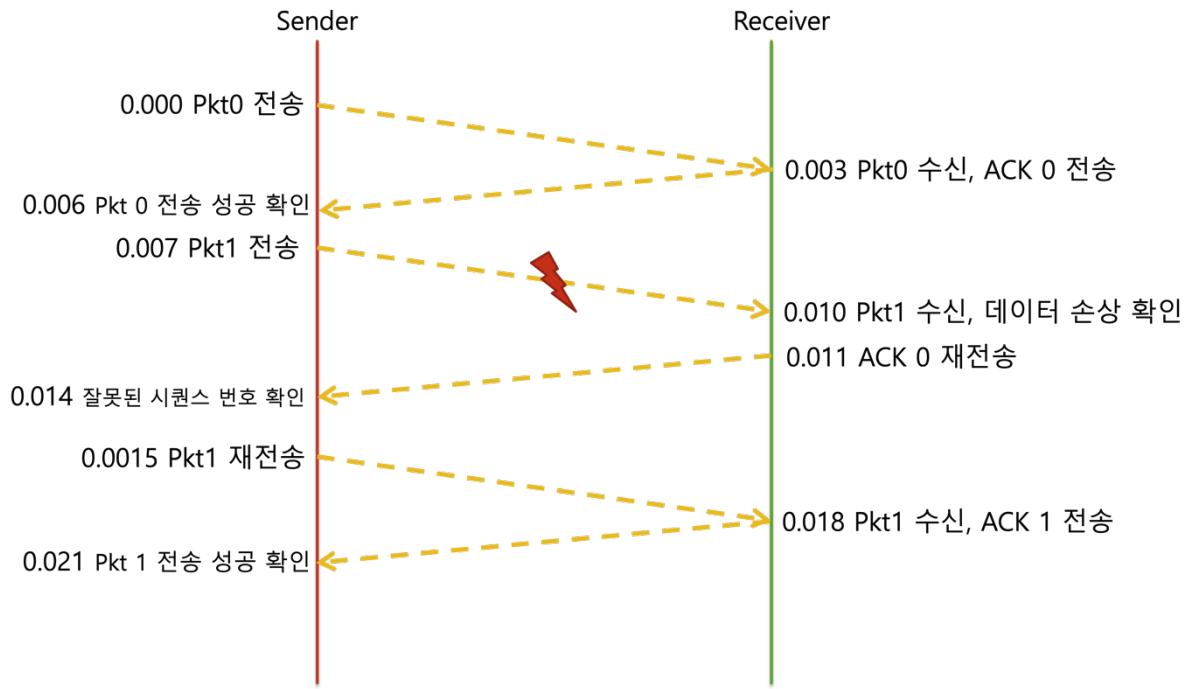
- **P2: RDT 2.2(20점)**
  - Sender가 Receiver에게 중간 크기의 파일(≤ 1MB)을 전송합니다.
  - 패킷 손상 처리, NAK 방지를 수행해야 합니다.
  - 시나리오 예시

```
0.000 pkt: 0 | Send DATA
0.006 pkt: 0 | Sent Successfully
0.007 pkt: 1 | Send DATA
0.014 pkt: 1 | Wrong Sequence Number
0.015 pkt: 1 | Send DATA Again
0.021 pkt: 1 | Sent Successfully

File transfer is finished.
```

sender 측

```
0.003 ACK: 0 | Send ACK
0.010 ACK: 1 | DATA Corrupted
0.011 ACK: 0 | Send ACK Again
0.018 ACK: 1 | Send ACK

File transfer is finished.
```

receiver 측



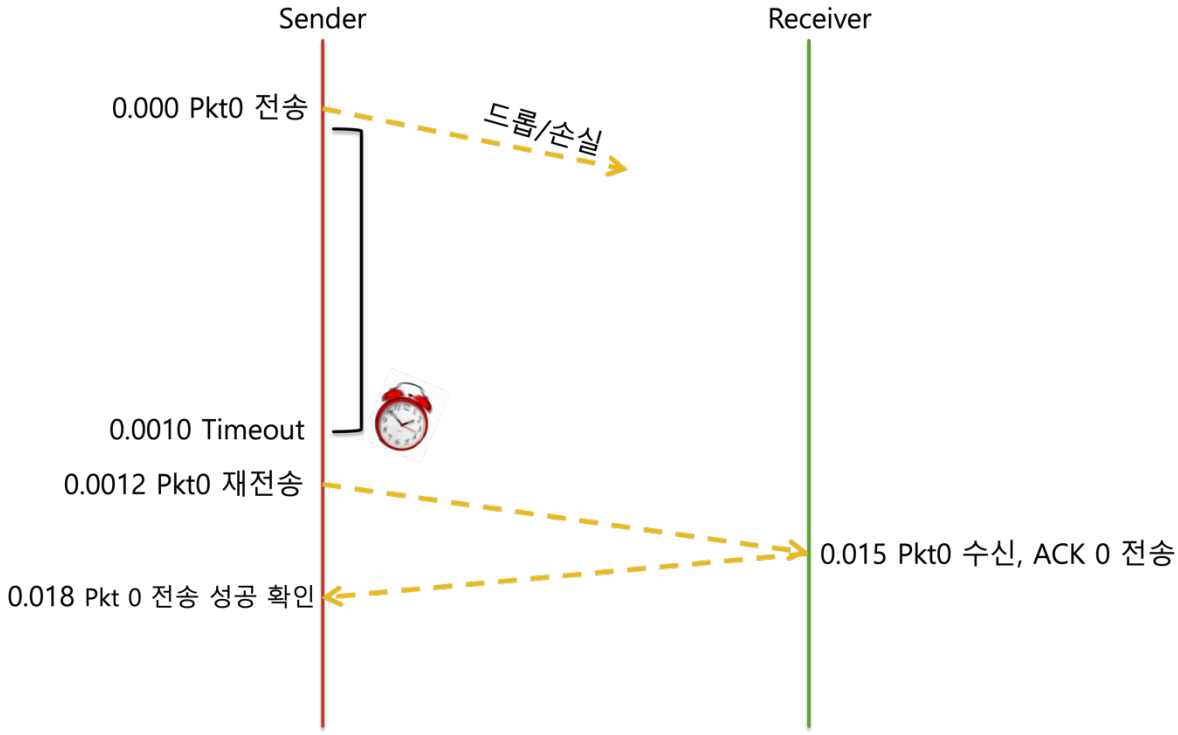
- **P3: RDT 3.0 (20pts)**
  - Sender가 Receiver에게 대용량 파일(≤ 50MB)을 전송합니다.
  - 패킷 드롭 처리를 수행해야 합니다.
  - 시나리오 예시

```
0.000 pkt: 0 | Send DATA
0.010 pkt: 0 | TIMEOUT
0.012 pkt: 0 | Send DATA Again
0.018 pkt: 0 | Sent Successfully

File transfer is finished.
```

sender side

```
0.015 ACK: 0 | Send ACK

File transfer is finished.
```

receiver side



- **P4: RDT 3.0 + 파이프라이닝 (20점)**
  - Sender가 Receiver에게 대용량 파일(≤ 50MB)을 전송합니다.
  - 파이프라이닝 및 Selective Repeat를 수행해야 합니다.
  - 시나리오 예시
    - ◆ 다음 예의 window size는 4이며, 0~5개의 pkts를 전송합니다.

0.000 pkt: 0 | Send DATA
0.001 pkt: 1 | Send DATA
0.002 pkt: 2 | Send DATA
0.003 pkt: 3 | Send DATA
0.004 pkt: 0 | Sent Successfully
0.005 pkt: 4 | Send DATA
0.005 pkt: 1 | Sent Successfully
0.007 pkt: 3 | Wrong Sequence
0.007 pkt: 3 | Sent Successfully: Mark
0.007 pkt: 5 | Send DATA
0.009 pkt: 4 | Wrong Sequence
0.009 pkt: 4 | Sent Successfully: Mark
0.011 pkt: 5 | Wrong Sequence
0.011 pkt: 5 | Sent Successfully: Mark
0.012 pkt: 2 | TIMEOUT
0.013 pkt: 2 | Send DATA Again
0.017 pkt: 2 | Sent Successfully

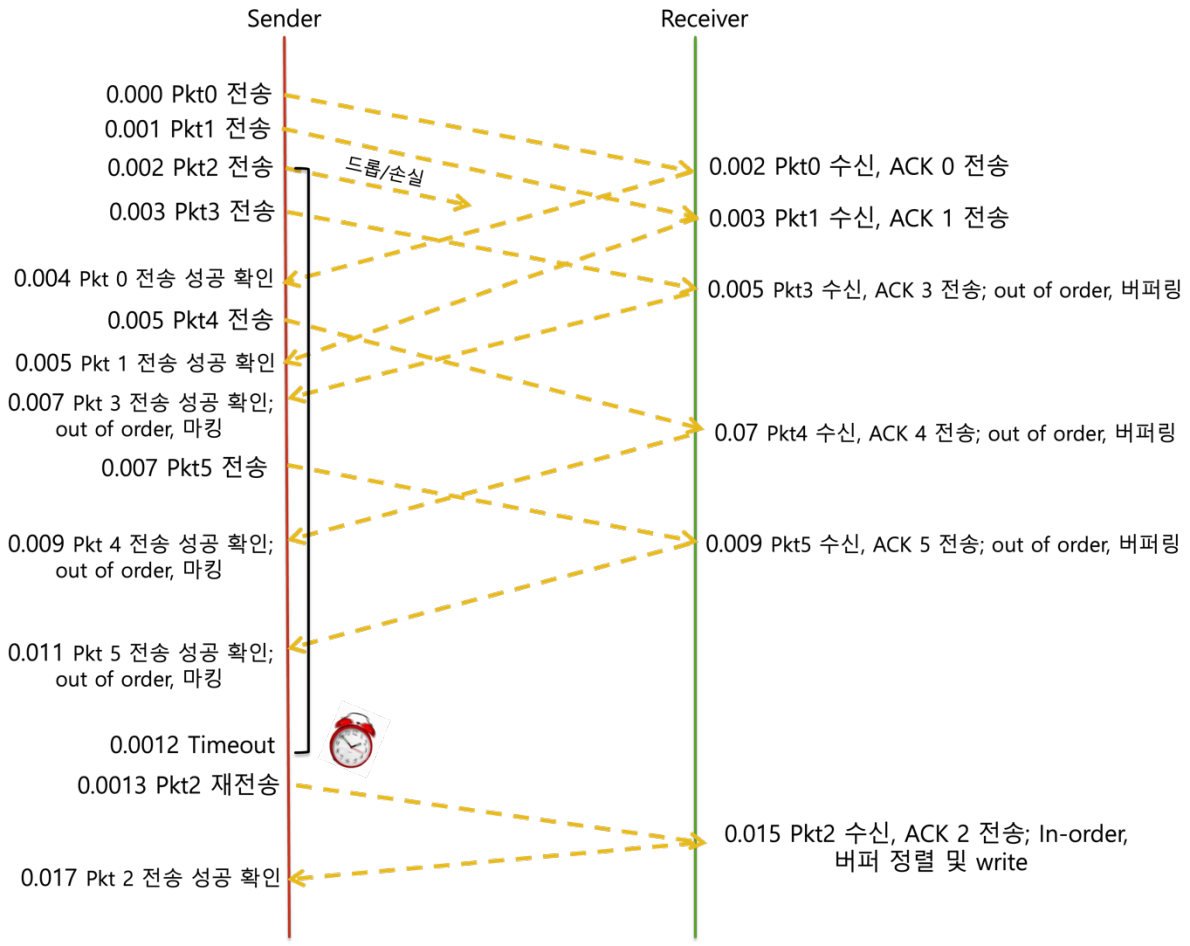
File transfer is finished.

sender side

0.002 ACK: 0 | Send ACK
0.003 ACK: 1 | Send ACK
0.005 ACK: 3 | Wrong Sequence: Buffer
0.005 ACK: 3 | Send ACK
0.007 ACK: 4 | Wrong Sequence: Buffer
0.007 ACK: 4 | Send ACK
0.009 ACK: 5 | Wrong Sequence: Buffer
0.009 ACK: 5 | Send ACK
0.015 ACK: 2 | Send ACK

File transfer is finished.

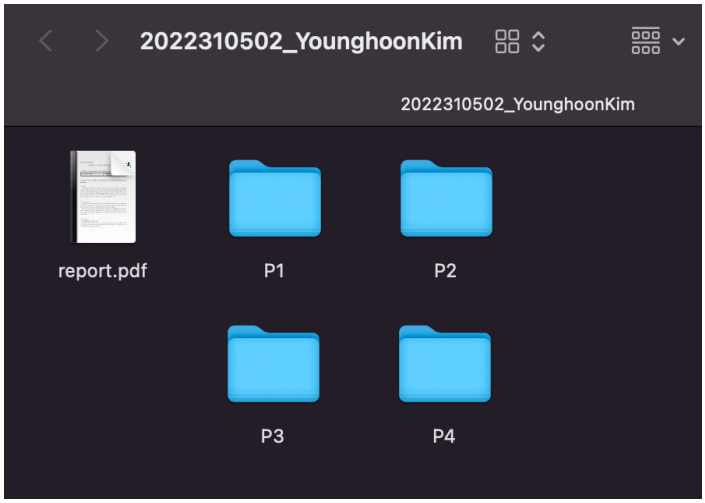
receiver side



- **보고서 (10점)**
  - 보고서 (10점)
    - iCampus에 업로드된 보고서 형식을 사용합니다.
    - 주어진 상황에서 어려움을 극복하기 위해 프로그램을 어떻게 설계했는지 설명하세요. 또한, 프로그램이 각 어려움에 효과적으로 대처하고 있음을 보여줄 수 있는 최소한의 출력 스크린샷(로그 파일, 터미널에 인쇄된 로그 등)을 첨부하세요.
- **형식(10점)**
  - 채점을 자동으로 진행하므로 파일 이름, 확장자, 보고서 형식, 포트 번호 등은 과제에 제시된 규칙을 지켜야 합니다.

5. 기타

- **제출 형식 및 방법**
  - 아이캠퍼스에 작성한 코드들과 보고서를 압축하여 제출합니다.
    - 각 RDT 버전에 대한 폴더를 만들고 각 폴더에 관련 파일을 모두 첨부하세요. (폴더 이름: **P1 / P2 / P3 / P4**)
      - P1~P4 폴더 **각각**에는 하나의 *sender.*py와 하나의 *receiver.py*가 있어야 합니다. TA는 콘솔을 사용하여 각 폴더에서 다른 Sender 및 Receiver파일로 프로그램을 실행합니다.
      - Sender와 Receiver 프로그램이 올바르게 작동하는 한 실행에 필요한 다른 파일은 자유롭게 위치해도 됩니다. (단, 각 폴더 내의 sender.py 혹은 receiver.py를 실행했을 때 적절하게 로드되어야 합니다.)
    - 보고서는 제시된 보고서 양식을 활용하여, <학번>.pdf로 RDT 버전별 폴더가 있는 곳에 두어야 합니다.
    - (1) P1~P4 폴더와 (2) 보고서 파일(pdf)을 "PA2_<학번>.zip"으로 압축한 후 iCampus에 업로드합니다.



- **제출 기한**
  - 마감일은 2022/05/05 23:59:00입니다(23:59:59 아님).
  - 지연 제출은 이틀 동안 허용됩니다.
    - 이 과제의 총 점수는 지연된 날 하루당 25%p씩 감점됩니다.
  - 표절이 확인되면 연관된 모든 학생들에 대해 최소 이번 과제 0점의 조치를 취할 예정입니다. 상황에 따라 F학점 부여 이상의 조치가 있을 수 있습니다.
- **질문**
  - 공지된 Google 스프레드시트 또는 이메일(comnet.at.skku@gmail.com)을 통해 질문할 수 있습니다.
  - 중복된 답변을 방지하려면 이전 질문을 다시 확인하신 후 답변을 남겨 주세요.
  - 5/5일 18:00 이후에는 질문에 답변하기 어려울 수 있습니다.