

문서 이력				
Version	작성일	변경사유	작성자	변경 내용 요약
1.0	2023.11.08	작성 완료	고영배 외	작성 완료

## 악조건에서의 5G-NR-V2X 서비스 통신 성능검증 모듈

### -설계서 및 사용설명서-

구분	소속	성명	날짜	서명
작성자	아주대학교	고영배 외	2021.10.19	

# 목차

1. 개요 .....	1
2. 시스템 구조 및 설계 .....	2
A. 시스템 개요 .....	2
B. 시스템 주요 기능 .....	3
C. 시스템 구조 .....	5
D. 실행 환경 변수 .....	7
3. 상세 설계 및 소스코드 설명 .....	13
A. select_window.py .....	13
B. packet_header_struct.py .....	15
C. sender_window.py .....	17
D. receiver_window.py .....	31
E. Tmap.html .....	63
4. 성능 검증 툴 사용 설명서 .....	66
A. 성능 검증 툴 실행 전 기기 간 연결 .....	66
B. Select Window .....	69
C. Sender Window .....	70
D. Receiver Window .....	72

# 1.개요

본 문서는 악조건에서의 5G-NR-V2X 서비스 통신 성능검증 모듈 용역으로 개발한 소프트웨어의 설계서 및 사용 설명서이다. 본 용역은 전자통신연구원에서 개발한 5G-NR-V2X 의 실 도로주행 시험 상황에서 통신 성능을 검증하기 위한 소프트웨어를 개발하는 것으로 자세한 요구사항은 다음과 같다.

- FNC-001: 악조건 event 반영
- FNC-002: 통신장치 연결 설정
- FNC-003: 실시간+비실시간 서비스 통신 성능 분석
- QUR-001: 시스템 하자 보수 관리

본 소프트웨어는 송신용 통신 장치를 위한 Sender 소프트웨어와 수신용 통신 장치를 위한 Receiver 소프트웨어로 구성된다. 편의를 위하여 최종 소프트웨어는 두 소프트웨어를 통합한 형태로 개발되었으며, 소프트웨어 구동 시 선택창을 통하여 기능을 선택할 수 있도록 개발했다.

본 문서의 구성은 다음과 같다. 2 장은 5G-NR-V2X 하드웨어와 소프트웨어 간 관계 및 이를 제어하기 위한 소프트웨어의 시스템 구조를 설명한다. 3 장은 소프트웨어 설계를 자세히 기술한다. 4 장은 소프트웨어 사용 방법 및 사용 예시를 기술한다.

## 2. 시스템 구조 및 설계

### A. 시스템 개요

본 용역에서 성능검증을 수행할 서비스는 씨쓰루 (See-Through) 서비스로, 앞차의 영상을 뒤차로 실시간 전송하여 앞차로 인해 가려진 전방 상황을 파악할 수 있도록 하는 응용 서비스이다.

통신 모듈은 Sender와 Receiver로 나뉘어 구성되며 각각 다른 차량에 탑재된다. 각 통신 모듈은 5G-NR 통신 기술을 이용하여 무선 통신을 수행한다. 각 통신 모듈은 이더넷 케이블을 통해서 데스크탑에 연결된 형태이며, Sender의 데스크탑에는 카메라 장치가 부착되어, 실시간으로 촬영되는 영상을 수집한다. 수집된 영상은 이더넷 케이블을 통해서 5G-NR 통신 장치로 전달되며, 5G-NR 무선 통신을 통해서 Receiver의 5G-NR 통신 장치로 전달된다. 최종적으로 Receiver의 데스크톱에 영상 데이터가 수신된다.

본 용역의 목표는 이러한 서비스 과정에서 Sender의 데스크톱과 Receiver의 데스크톱의 End-to-End 성능을 측정하는 소프트웨어를 개발하는 것으로, 이를 위해서 Sender와 Receiver측 데스크탑에서 동작하는 소프트웨어를 각각 설계한다. Sender 소프트웨어의 핵심 기능은 영상을 실시간으로 송신하는 기능이고 Receiver 소프트웨어의 핵심 기능은 영상을 수신하는 기능, 실시간 성능 측정 기능, 악조건 탐지 기능, 성능 지표 기록 기능이다.

## B. 시스템 주요 기능

5G-NR-V2X 디바이스의 통신 성능을 실시간으로 평가하기 위해서 사용한 지표는 Packet Delivery Ratio (PDR), Latency, Throughput이다. 모든 성능지표는 실시간으로 측정하였으며, 통신 장치의 기능을 제어할 수 없는 상황으로 인해서 데스크탑에서 End-to-End 방식으로 측정되었다.

PDR은 무선 데이터가 성공적으로 전달된 비율을 실시간으로 나타내는 지표로 영상 스트리밍 데이터에 포함된 Sequence Header를 이용하여 측정하였다. PDR은 매 1초마다 측정되며 측정 방식은 다음과 같다.

$$PDR = \frac{Packet\_Count}{Seq\_Max - Seq\_Min}$$

매초 측정되는 PDR은 그 시간동안 수신된 영상 패킷의 Sequence Header의 최대값과 최소값의 차이로 그 시간동안 수신된 영상 패킷의 수를 나누는 방식으로 측정하였다. 이 방식은 Packet Sequence가 섞여서 들어오는 상황에서 측정 오차가 발생할 수 있지만, 이는 가까운 시간 내의 성능에 반영되어 모두 보상되기 때문에 긴 시간의 성능을 측정하는 본 시스템에서 충분히 활용 가능할 것으로 생각된다.

Latency는 Round-Trip-Time (RTT)를 측정하는 방식으로 측정하였다. 본래는 데스크톱과 5G-NR-V2X 디바이스간 유선 연결에서 발생하는 지연시간을 제외하고 측정해야 하지만, 5G-NR-V2X 디바이스에서 해당 측정을 위한 기능을 제공하지 못하여 이를 포함한 전체 End-to-End latency를 측정하였다. 측정 방법

은 Receiver에서 Sender로 Round-Trip 메시지를 송신하여 RTT를 측정한 후 Sender delay를 빼고, 2로 나눈 값을 활용했다. Latency 또한 매 1초마다 측정하였다.

$$latency = \frac{RTT - C_{delay}}{2}$$

Throughput은 데스크탑에서 5G-NR-V2X 통신 장치와 연결된 이더넷 인터페이스의 단위 시간당 수신된 통신량을 측정하였다.

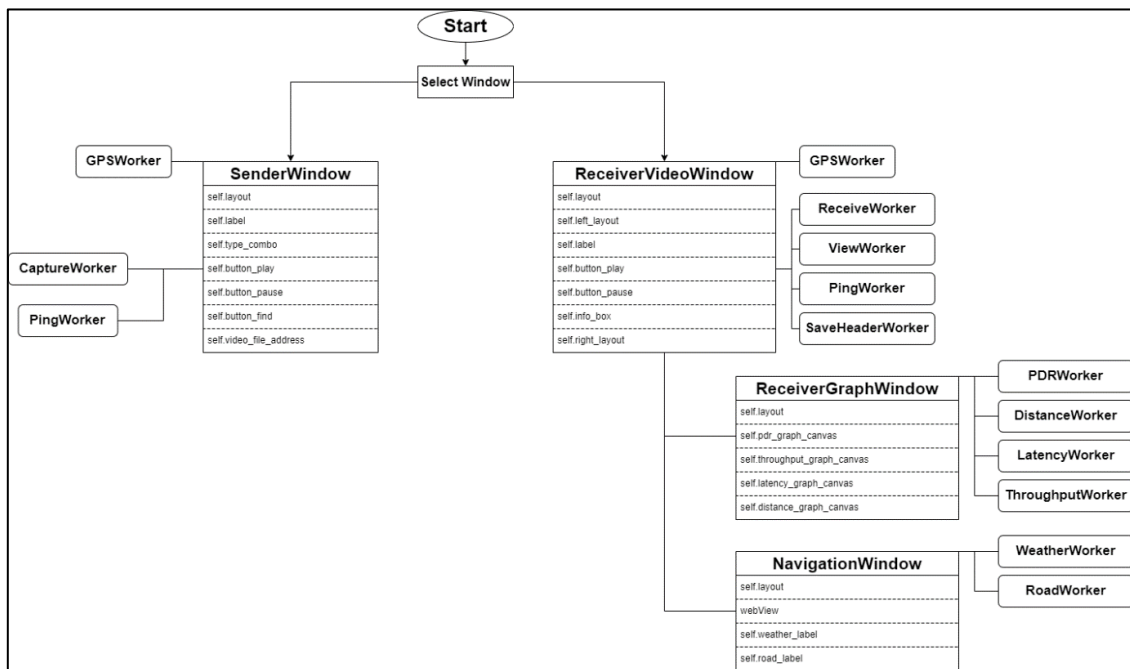
자세한 성능 지표의 측정 방식은 3장 소스코드 구조 및 알고리즘 설명에서 소스코드와 함께 기술한다.

See-Through 서비스를 위해서는 실시간 영상 스트리밍 기술이 필요하다. 스트리밍 기술은 Progressive Download 방식, RTSP (Real Time Streaming Protocol) 방식, Adaptive Streaming 방식으로 구분 가능하다. Progressive Download 방식은 동영상을 FTP 방식으로 전송하고, 실시간으로 동영상을 다운로드 받으면서 재생하는 방식이다. RTSP 방식은 이와 다르게 전체 동영상이 아닌 시청중인 프레임만 전송하는 방식이다. Adaptive Streaming 방식은 두 기술을 결합한 형태로 동영상을 짧은 시간으로 잘라서 시청중인 프레임 부근의 짧은 동영상 파일을 전달하는 방식이다. 각 방식들은 응용 서비스에 따라서 장단점이 존재하는데, 본 서비스는 최단 지연시간이 가장 중요한 요소이기 때문에 RTSP 방식을 차용하였다.

개발된 5G-NR-V2X 디바이스는 약 15~20 Mbps의 Data Rate를 지원하기 때

문에 비압축 영상을 스트리밍할 경우 약 360x360 해상도의 영상을 실시간 전달 가능하다. 또한 통신 장치의 최대 MTU는 1306 Bytes로 한 패킷에 영상 한 프레임의 한 줄을 담아서 보내는 방식으로 설계하였다.

## C. 시스템 구조



본 소프트웨어는 위 사진의 구조로 되어 있으며 사용자는 Sender와 Receiver 중 자신이 원하는 기능을 선택한다.

Sender 선택 시 SenderWindow가 열리며 자신의 GPS 좌표 정보를 갱신하는 GPSWorker가 시작된다. Sender 측 주요 기능을 시작하는 self.button\_play 버튼을 누르면 CaptureWorker와 PingWorker가 시작된다. 이에 해당하는 기능은 영상 데이터 수집 및 송신, Latency 패킷 답변 기능으로 self.button\_pause를 누를 때까지 지속된다.

Receiver 선택 시 ReceiverWindow 창이 열리며 Sender 때처럼 GPSWorker가 실행되고 자신의 GPS 좌표 정보를 갱신한다. ReceiverWindow 창이 열린 후 ReceiverGraphWindow와 NavigationWindow 창이 열린다. self.button\_play 버튼을 누르면 패킷을 수신하는 ReceiveWorker, 수신된 영상 데이터를 재생하는 ViewWorker, Latency 계산을 위해 패킷을 전송하는 PingWorker, 성능 지표 값과 패킷 헤더를 저장하는 SaveHeaderWorker가 진행된다. ReceiverGraphWindow 창이 열리면 자동으로 성능 지표 값들을 계산하는 PDRWorker, DistanceWorker, LatencyWorker, ThroughputWorker가 실행된다. NavigationWindow 창이 열리는 경우 WeatherWorker와 RoadWorker가 자동으로 실행된다.



## D. 실행 환경 변수

- sender\_window.py

```
13 # PyQt Windows Size
14 WIN_SIZE_H = 800 # Height
15 WIN_SIZE_W = 600 # Width
16
17 # GPS Sensor Variable
18 SER_PORT = 'COM5' # Serial Port
19 SER_BAUD = 9600 # Serial Baud Rate
20
21 # 5G NR Device Connection Variable
22 DEVICE_ADDR = '192.168.1.11'
23 DEVICE_PORT = 47347
24 SOCKET_SEND_DELAY = 0
25
26 # Video Data Size
27 SENDER_FRAME_MSEC = 40 # Only 'int' value & Milliseconds ( 60 Frame -> 1000 milliseconds / 60 frames = 16.66666....)
28 SEND_FRAME_WIDTH = 300
29 SEND_FRAME_HEIGHT = 300
30
31 # RTT Variable
32 RTT_TIMER = 0
33
34 # Packet Variable
35 WS_REQ = b"\xf1\xf1\x00\x01\x00\x00\x00\x00\x00\x14\x97\x00\x00\x00\x00"
36 WS_RESP_MAGIC_NUM = b'\xf1\xf2'
37 PING_INDICATOR = b'\x03\x02'
```

### ■ PyQt 그래픽 크기 변수

➤ WIN\_SIZE\_H : Sender Window 창 높이

➤ WIN\_SIZE\_W : Sender Window 창 넓이

### ■ GPS 센서 연결 변수

➤ SER\_PORT : GPS 센서와 serial 통신을 위한 포트 이름

➤ SER\_BAUD : GPS 센서와 serial 통신을 위한 포트 번호

### ■ 5G-NR 장비 Network IP/Port 변수

➤ DEVICE\_ADDR : 5G-NR 장비와 노트북 사이 통신을 위한 IP 주소

- DEVICE\_PORT : 5G-NR 장비와 노트북 사이 통신을 위한 포트 주소
- SOCKET\_SEND\_DELAY : 패킷 전송 후 다음 패킷 전송 전 대기 시간

#### ■ 비디오 Data 크기 변수

- SENDER\_FRAME\_MSEC : 카메라 센서를 통해 촬영하는 영상의 프레임
- SEND\_FRAME\_WIDTH : 카메라 센서를 통해 얻은 영상 데이터의 한 프레임 사이즈를 변경하기 위한 변경 후 프레임 넓이
- SEND\_FRAME\_HEIGHT : 카메라 센서를 통해 얻은 영상 데이터의 한 프레임 사이즈를 변경하기 위한 변경 후 프레임 높이

#### ■ RTT(Round Trip Time) 변수

- RTT\_TIMER : Latency 성능 측정을 위한 패킷 전송 시 다음 패킷을 전송할 때까지 대기하는 시간

#### ■ 패킷 변수

- WS\_REQ : 컴퓨터가 5G-NR 장비와의 연결을 위해 5G-NR 장비에 연결 요청 패킷
- WS\_RESP\_MAGIC\_NUM : WS\_REQ를 받은 5G-NR 장비가 연결을 수락할 시 컴퓨터가 받는 패킷
- PING\_INDICATOR : 현재 수신한 패킷이 Latency 측정을 위한 패킷인지 확인하기 위한 값

- receiver\_window.py

```

28 # PyQt Windows Size
29 monitor_size_width = get_monitors()[0].width
30 monitor_size_height = get_monitors()[0].height
31 BLANK_SPACE = 15
32 GRAPH_WIN_SIZE_W = monitor_size_width - BLANK_SPACE*2
33 GRAPH_WIN_SIZE_H = int(monitor_size_height / 2) - BLANK_SPACE
34 VIDEO_WIN_SIZE_W = int(monitor_size_width / 2) - BLANK_SPACE
35 VIDEO_WIN_SIZE_H = int(monitor_size_height / 2) - BLANK_SPACE*3
36 NAVIGATION_WIN_SIZE_W = int(monitor_size_width / 2) - BLANK_SPACE
37 NAVIGATION_WIN_SIZE_H = int(monitor_size_height / 2) - BLANK_SPACE*3
38
39 # Socket Value
40 DEVICE_ADDR = '192.168.1.11'
41 DEVICE_PORT = 47347
42
43 # Packet Value
44 MAX_PACKET_SIZE = 1502
45 MAX_PAYLOAD_SIZE = 1300
46
47 # Camera Capture Size
48 RECV_FRAME_WIDTH = 300
49 RECV_FRAME_HEIGHT = 300
50
51 # GPS Sensor Variable
52 SER_PORT = 'COM5' # Serial Port
53 SER_BAUD = 9600 # Serial Baud Rate
54
55 # Log Cycle
56 HEADER_LOG_CYCLE = 60 # Seconds
57
58 # RTT Variable
59 RTT_TIMER = 1
60
61 # Packet Variable
62 WS_REQ = b"\xf1\xf1\x00\x01\x00\x00\x00\x00\x00\x00\x14\x97\x00\x00\x00\x00"
63 WS_RESP_MAGIC_NUM = b'\xf1\xf2'
64 RX_MAGIC_NUM = b'\xf3\xf2'
65 VIDEO_DATA_INDICATOR = b'\x03\x01'
66 PING_INDICATOR = b'\x03\x02'
67
68 # Graph Data Variable
69 NET_IF = "이더넷"
70
71 # Navigation HTML File Path
72 HTML_FILE_PATH = './resource/Tmap.html'
73
74 # Bad Condition Data Variable
75 WEATHER_API_URL = 'http://apis.data.go.kr/1360000/VilageFcstInfoService_2.0/getUltraSrtNcst'
76 WEATHER_API_SERVICE_KEY = 'QEPmmbFk9szqqPD8q9+s2ezo0oY7VcAt4Rt1QPseyZ5LQucie5H90jnJj/G04H1141QrmGwQxhCF9FG'
77 WEATHER_CONDITION_WAIT_TIMER = 5
78 WEATHER_CONDITION_ERROR_RESEND_TIMER = 10
79 WEATHER_CONDITION_RESEND_TIMER = 600
80 ROAD_API_URL = 'https://apis.openapi.sk.com/tmap/traffic'
81 ROAD_API_SERVICE_KEY = 'fOsIyENUef8Arejv1q6DU4p66e0sMRjB5kiI22do'
82 ROAD_CONDITION_WAIT_TIMER = 5
83 ROAD_CONDITION_RESEND_TIMER = 120

```

## ■ PyQt 그래픽 크기 변수

- monitor\_size\_width : 주 모니터 넓이 값
- monitor\_size\_height : 주 모니터 높이 값
- BLANK\_SPACE : Receiver Window와 모니터 화면 사이의 빈 공간 두께
- GRAPH\_WIN\_SIZE\_W : 성능 그래프 창 넓이 값

- GRAPH\_WIN\_SIZE\_H : 성능 그래프 창 높이 값
- VIDEO\_WIN\_SIZE\_W : 비디오 창 넓이 값
- VIDEO\_WIN\_SIZE\_H : 비디오 창 높이 값
- NAVIGATION\_WIN\_SIZE\_W : 네비게이션 창 넓이 값
- NAVIGATION\_WIN\_SIZE\_H : 네비게이션 창 높이 값

#### ■ 5G-NR 장비 Network IP/Port 변수

- DEVICE\_ADDR : 5G-NR 장비와 노트북 사이 통신을 위한 IP 주소
- DEVICE\_PORT : 5G-NR 장비와 노트북 사이 통신을 위한 포트 주소

#### ■ 패킷 변수

- RECV\_FRAME\_WIDTH : 수신하는 영상 데이터 프레임 넓이
- RECV\_FRAME\_HEIGHT : 수신하는 영상 데이터 프레임 높이

#### ■ GPS 센서 연결 변수

- SER\_PORT : GPS 센서와 serial 통신을 위한 포트 이름
- SER\_BAUD : GPS 센서와 serial 통신을 위한 포트 번호

#### ■ Log 주기 변수

- HEADER\_LOG\_CYCLE : csv 형태의 log를 저장하는 주기(단위 : 초)

#### ■ RTT(Round Trip Time) 변수

- RTT\_TIMER : RTT기반 Latency 측정을 위해 RTT 패킷을 송신하는 주기  
(단위 : 초)

#### ■ 패킷 변수

- WS\_REQ : 컴퓨터가 5G-NR 장비와의 연결을 위해 5G-NR 장비에 연결 요청 패킷
- WS\_RESP\_MAGIC\_NUM : WS\_REQ를 받은 5G-NR 장비가 연결을 수락할 시 컴퓨터가 받는 패킷
- RX\_MAGIC\_NUM : 현재 수신한 패킷이 성능 측정 혹은 데이터 전송을 위한 패킷인지 확인하기 위한 값
- VIDEO\_DATA\_INDICATOR : 현재 수신한 패킷이 영상 데이터 전송을 위한 패킷인지 확인하기 위한 값
- PING\_INDICATOR : 현재 수신한 패킷이 Latency 측정을 위한 패킷인지 확인하기 위한 값

#### ■ 그래프 데이터 변수

- NET\_IF : Throughput 데이터를 얻기 위하여 컴퓨터와 5G-NR 장비 연결 사이 컴퓨터의 물리적 이더넷 포트 이름

#### ■ Navigation HTML 파일 경로 변수

- HTML\_FILE\_PATH : T MAP API를 활용하기 위한 HTML 파일의 위치

■ 악조건 데이터 변수

- WEATHER\_API\_URL : 공공 데이터 포털 날씨 API를 요청할 때 쓰이는 URL
- WEATHER\_API\_SERVICE\_KEY : 공공 데이터 포털 날씨 API 사용 시 필요한 KEY
- WEATHER\_CONDITION\_WAIT\_TIMER : 공공 데이터 포털 날씨 API 요청 후 API 수신까지 기다리는 시간
- WEATHER\_CONDITION\_ERROR\_RESEND\_TIMER : 공공 데이터 포털 날씨 API 수신 시 ERROR 인 경우 다시 요청할 때까지 기다리는 시간
- WEATHER\_CONDITION\_RESEND\_TIMER : 공공 데이터 포털 날씨 API 수신 후 다시 요청할 때까지 기다리는 시간
- ROAD\_API\_URL : T MAP 도로 혼잡도 API를 요청할 때 쓰이는 URL
- ROAD\_API\_SERVICE\_KEY : T MAP 도로 혼잡도 API 사용 시 필요한 KEY
- ROAD\_CONDITION\_WAIT\_TIMER : T MAP 도로 혼잡도 API 수신까지 기다리는 시간
- ROAD\_CONDITION\_RESEND\_TIMER : T MAP 도로 혼잡도 API 수신 후 다시 요청할 때까지 기다리는 시간

### 3. 상세 설계 및 소스코드 설명

#### A. select\_window.py

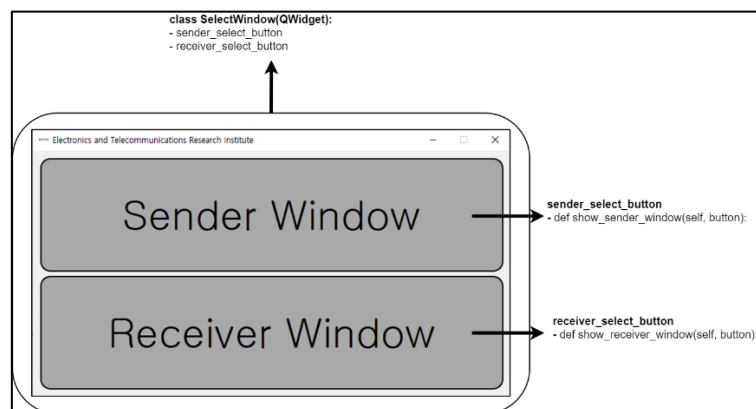
##### i. 역할

- 프로그램 실행 시 나오는 Select Window의 UI 설정
- Receiver와 Sender 중 원하는 기능을 선택 가능

##### ii. 필요 라이브러리

- os
- sys
- PyQt5
- sender\_window
- receiver\_window

##### iii. 코드 설명



- class SelectWindow(QWidget):
  - select\_window.py의 13~ 76번째 줄
  - 17~22번째 줄 : Select Window의 창 제목, 창 아이콘 사진, 창 크기 설정 코드
  - 24~49번째 줄 : Sender Window를 열기 위한 버튼 및 버튼 디자인 코드
  - 51~76번째 줄 : Receiver Window를 열기 위한 버튼 및 버튼 디자인 코드
- def show\_sender\_window(self, button):
  - select\_window.py의 78~85번째 줄
  - Sender Window를 열고 Select Window 창 종료
  - 'sender\_select\_button'을 클릭 시 작동
- def show\_receiver\_window(self, button):
  - select\_window.py의 87~94번째 줄
  - Receiver Window를 열고 Select Window 창 종료
  - 'receiver\_select\_button'을 클릭 시 작동



## B. packet\_header\_struct.py

### i. 역할

- 영상 데이터 패킷 전송 시 포함되는 header 값
- 패킷 전송 시 각 필드에 값들이 입력되어 전송

### ii. 필요 라이브러리

- socket
- scapy

```
from socket import *  
from scapy.all import *
```

### iii. 코드 설명

- class V2X\_TxPDU(Packet):
  - packet\_header\_struct.py의 6~27번째 줄
  - Header 명 : V2X\_TxPDU
  - Header 필드 값 및 선언
- class DB\_V2X(Packet):
  - packet\_header\_struct.py의 30~47번째 줄
  - Header 명 : DB\_V2X

- Header 필드 값 및 선언
- 현재 ulPayloadLength에는 자신의 위도 값,  
ulPayloadCrc32에는 자신의 경도 값을 저장
- header 필드 값이 달라지는 경우 위 코드를 수정 혹은 추가하여 필드  
값 수정 및 추가

```
class V2X_TxPDU(Packet):
    name = "V2X_TxPDU"
    fields_desc = [
        ShortField("magic_num", 0),
        ShortField("ver", 0),
        IntField("psid", 0),
        XByteField("e_v2x_comm_type", 0),
        XByteField("e_payload_type", 0),
        XByteField("elements_indicator", 0),
        XByteField("tx_power", 0),
        XByteField("e_signer_id", 0),
        XByteField("e_priority", 0),
        XByteField("channel_load", 0),
        XByteField("reserved1", 0),
        LongField("expiry_time", 0),
        IntField("transmitter_profile_id", 0),
        IntField("peer_l2id", 0),
        IntField("reserved2", 0),
        LongField("reserved3", 0),
        IntField("crc", 0),
        ShortField("length", 0)
    ]
```

```
class DB_V2X(Packet):
    name = "DB_V2X"
    fields_desc = [
        IntField("eDeviceType", 1),
        IntField("eTeleCommType", 1),
        IntField("unDeviceId", 1),
        LongField("ulTimeStamp", 1),
        IntField("eServiceId", 1),
        IntField("eActionType", 1),
        IntField("eRegionId", 1),
        IntField("ePayloadType", 1),
        IntField("eCommId", 1),
        ShortField("usDbVer", 1),
        ShortField("usHwVer", 1),
        ShortField("usSwVer", 1),
        IntField("ulPayloadLength", 1),
        IntField("ulPayloadCrc32", 1)
    ]
```

## C. sender\_window.py

### i. 역할

- Select Window에서 Sender Window 선택 시 나오는 Select Window를 구성하는 UI와 데이터 송신과 관련된 주요 기능

### ii. 필요 라이브러리 및 파일

- cv2
- numpy
- serial
- scapy
- PyQt5
- pygrabber
- packet\_header\_struct.py

```
1  import cv2
2  import numpy
3  import serial
4  from socket import *
5  from scapy.all import *
6  from PyQt5.QtGui import *
7  from PyQt5.QtCore import *
8  from PyQt5.QtWidgets import *
9  import packet_header_struct
10 from pygrabber.dshow_graph import FilterGraph
```

### iii. 코드 설명

- 환경 변수 (13~37번째 줄)(환경에 따라 변경 필요)

```
13 # PyQt Windows Size
14 WIN_SIZE_H = 800 # Height
15 WIN_SIZE_W = 600 # Width
16
17 # GPS Sensor Variable
18 SER_PORT = 'COM5' # Serial Port
19 SER_BAUD = 9600 # Serial Baud Rate
20
21 # 5G NR Device Connection Variable
22 DEVICE_ADDR = '192.168.1.11'
23 DEVICE_PORT = 47347
24 SOCKET_SEND_DELAY = 0
25
26 # Video Data Size
27 SENDER_FRAME_MSEC = 40 # Only 'int' value & Milliseconds ( 60 Frame -> 1000 milliseconds / 60 frames = 16.66666....)
28 SEND_FRAME_WIDTH = 300
29 SEND_FRAME_HEIGHT = 300
30
31 # RTT Variable
32 RTT_TIMER = 0
33
34 # Packet Variable
35 WS_REQ = b"\xf1\xf1\x00\x01\x00\x00\x00\x00\x00\x14\x97\x00\x00\x00\x00"
36 WS_RESP_MAGIC_NUM = b"\xf1\xf2"
37 PING_INDICATOR = b"\x03\x02"
```

- WIN\_SIZE\_H : Sender Window 창 높이
- WIN\_SIZE\_W : Sender Window 창 넓이
- SER\_PORT : GPS 센서와 serial 통신을 위한 포트 이름
- SER\_BAUD : GPS 센서와 serial 통신을 위한 포트 번호
- DEVICE\_ADDR : 5G-NR 장비와 노트북 사이 통신을 위한 IP 주소
- DEVICE\_PORT : 5G-NR 장비와 노트북 사이 통신을 위한 포트 주소
- SOCKET\_SEND\_DELAY : 패킷 전송 후 기기 과부하 방지를 위해 다음 패킷 전송 전 대기 시간
- SENDER\_FRAME\_MSEC : 카메라 센서를 통해 촬영하는 영상의 프레임

- SEND\_FRAME\_WIDTH : 카메라 센서를 통해 얻은 영상 데이터의 한 프레임 사이즈를 변경하기 위한 변경 후 프레임 넓이
  - SEND\_FRAME\_HEIGHT : 카메라 센서를 통해 얻은 영상 데이터의 한 프레임 사이즈를 변경하기 위한 변경 후 프레임 높이
  - RTT\_TIMER : Latency 성능 측정을 위한 패킷 전송 시 다음 패킷을 전송할 때까지 대기하는 시간
  - WS\_REQ : 컴퓨터가 5G-NR 장비와의 연결을 위해 5G-NR 장비에 연결 요청 패킷
  - WS\_RESP\_MAGIC\_NUM : WS\_REQ를 받은 5G-NR 장비가 연결을 수락할 시 컴퓨터가 받는 패킷
  - PING\_INDICATOR : 현재 수신한 패킷이 Latency 측정을 위한 패킷인지 확인하기 위한 값
- 전역 변수(기능 작동을 위해 사용)

```

40  pkt_seq_num = 0
41  latitude = 12.0
42  longitude = 34.0
43  camera_list = {}

```

- pkt\_seq\_num : 패킷 전송 시 수신 측에서 몇 번째 전송 패킷인지 알기 위해 필요한 변수
- latitude : GPS 센서로 측정되는 본인의 위도 값

- longitude : GPS 센서로 측정되는 본인의 경도 값
- camera\_list : 현재 컴퓨터에 연결된 카메라들 정보가 저장되는 배열

- def resource\_path(relative\_path):

```
45  def resource_path(relative_path):
46      base_path = getattr(sys, '_MEIPASS', os.path.dirname(os.path.abspath(__file__)))
47      return os.path.join(base_path, relative_path)
```

- 외부 파일을 사용하기 위한 함수로 relative\_path에 들어있는 파일 경로를 참고해 파일을 호출

- def rescale\_frame(original\_frame, width, height):

```
49  def rescale_frame(original_frame, width, height):
50      return cv2.resize(original_frame, (width, height), interpolation=cv2.INTER_AREA)
```

- 프레임(original\_frame)의 사이즈를 폭(width)\*높이(height)로 변경하는 함수

- def find\_camera\_list():

```
52  def find_camera_list():
53      global camera_list
54      camera_list = {}
55      devices = FilterGraph().get_input_devices()
56      for device_index, device_name in enumerate(devices):
57          camera_list[device_index] = device_name
```

- 컴퓨터에 현재 연결된 카메라 목록을 읽어오고 카메라 정보를 camera\_list 전역 변수에 저장하는 함수

- def send\_5g(send\_sock, video\_data):

```
59 def send_5g(send_sock, video_data):
60     global pkt_seq_num
61     global latitude
62     global longitude
63     pkt_seq_num_temp = pkt_seq_num.to_bytes(4, byteorder='big')
64     send_data = b'\x03\x01' + pkt_seq_num_temp + video_data
65     pkt_seq_num = (pkt_seq_num + 1) % 1000000
66
67     db_v2x_tmp_p = packet_header_struct.DB_V2X(
68         eDeviceType=htonl(0x0001),
69         eTeleCommType=htonl(0x0002),
70         unDeviceId=0x0000,
71         ullTimeStamp=0x0000,
72         eServiceId=htonl(0x0005),
73         eActionType=htonl(0x0001),
74         eRegionId=htonl(0x0004),
75         ePayloadType=htonl(0x000b),
76         eCommId=htonl(0x0001),
77         usDbVer=0x0001,
78         usHwVer=0x0111,
79         usSwVer=0x0001,
80         ulPayloadLength=int(latitude * 1000000),
81         ulPayloadCrc32=int(longitude * 1000000),
82     )
83
84     v2x_tx_pdu_p = packet_header_struct.V2X_TxPDU(
85         magic_num=htons(0xf2f2),
86         ver=0x0001,
87         psid=5271,
88         e_v2x_comm_type=0,
89         e_payload_type=4,
90         elements_indicator=0,
91         tx_power=20,
92         e_signer_id=0,
93         e_priority=0,
94         channel_load=0,
95         reserved1=0,
96         expiry_time=0,
97         transmitter_profile_id=100,
98         peer_l2id=0,
99         reserved2=0,
100        reserved3=0,
101        crc=0,
102        length=len(bytes(db_v2x_tmp_p)) + len(send_data)
103    )
104
105    serialized = bytes(v2x_tx_pdu_p) + bytes(db_v2x_tmp_p) + send_data
106    try:
107        send_sock.send(serialized)
108    except:
109        print(traceback.format_exc())
110    if SOCKET_SEND_DELAY != 0:
111        time.sleep(SOCKET_SEND_DELAY)
```

- 컴퓨터에서 5G\_NR 장비로 패킷을 송신할 때 사용하는 함수
- 전역 변수 pkt\_seq\_num, latitude, longitude의 값을 불러와서 전송 패킷에 첨부
- 다음에 전송되는 패킷을 위하여 전역 변수 pkt\_seq\_num는 사용 후 값이 1이 증가하며 1000000이상이 되면 0부터 다시 시작한다.
- 전역 변수 latitude와 longitude는 각각 1000000이 곱해진 뒤 헤더 DB\_V2X의 ulPayloadLength와 ulPayloadCrc32에 저장됨
- 헤더 DB\_V2X와 V2X\_TxPDU 외 전송 데이터 부분은 **b'Wx03Wx01' + 현재 패킷 순번 + 비디오 데이터**의 형태로 이루어짐(64번째 줄)
- 최종 전송 패킷은 **DB\_V2X 헤더 + V2X\_TxPDU 헤더 + 전송 데이터**의 형태로 serialized에 저장되어 5G\_NR 장비에 전송됨
- 필요 시 SOCKET\_SEND\_DELAY에 저장된 시간만큼 다음 패킷 전송 전 대기



- class SenderWindow(QWidget):

```
260 class SenderWindow(QWidget):
261     def __init__(self):
262         super().__init__()
263
264         while True:
265             try:
266                 self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
267                 self.sock.connect((DEVICE_ADDR, DEVICE_PORT))
268                 break
269             except:
270                 print(traceback.format_exc())
271 self.sock.send(WS_REQ)
272 try:
273     ws_resp = self.sock.recv(1024)
274     if ws_resp[0:2] != WS_RESP_MAGIC_NUM:
275         print("Fail")
276 except:
277     print(traceback.format_exc())
278
279 # UI declaration #
280 # Video Screen Area
281 self.label = QLabel()
282 self.label.setAlignment(Qt.AlignCenter)
283 self.label.setPixmap(QPixmap(resource_path('resource/stop_icons.png')))
284 # Play video & send video
285 self.button_play = QPushButton("Play & Send")
286 self.button_play.clicked.connect(self.play_send_video)
287 # Stop play & send video
288 self.button_pause = QPushButton("Pause")
289 self.button_pause.clicked.connect(self.pause_video)
290 self.button_pause.setDisabled(True)
291 # Reset Camera list
292 self.button_find = QPushButton("Find Camera")
293 self.button_find.clicked.connect(self.find_camera)
294 # Video file path
295 self.video_file_address = QLineEdit()
296 self.video_file_address.setPlaceholderText("Saved Video File Path(If send video file)")
297 # Type of transmission data(Camera / Video)
298 self.type_combo = QComboBox(self)
299 find_camera_list()
300 for i in camera_list:
301     self.type_combo.addItem(camera_list[i])
302 self.type_combo.addItem("Saved Video")
303
304 # UI Arrangement #
305 self.layout = QVBoxLayout()
306 self.layout.addWidget(self.label)
307 self.layout.addWidget(self.video_file_address)
308 self.layout.addWidget(self.type_combo)
309 self.layout.addWidget(self.button_play)
310 self.layout.addWidget(self.button_pause)
311 self.layout.addWidget(self.button_find)
312
313 # Final UI Layout Arrangement #
314 self.setLayout(self.layout)
315 self.setWindowTitle("Sender Window")
316 self.setFixedSize(QSize(WIN_SIZE_H, WIN_SIZE_W))
317
318 # GPS thread
319 self.gps_worker_th = GPSWorker()
320 self.gps_worker_th.start()
```

```

323 def play_send_video(self):
324     # Define OpenCV by type of transmission data(Camera / Video) #
325     if self.type_combo.currentText() == "Saved Video":
326         self.send_data_type = self.video_file_address.text()
327         try:
328             self.video_cap = cv2.VideoCapture(self.send_data_type)
329         except:
330             print(traceback.format_exc())
331             return
332     else:
333         for i in camera_list:
334             if self.type_combo.currentText() == camera_list[i]:
335                 try:
336                     self.video_cap = cv2.VideoCapture(cv2.CAP_DSHOW+i)
337                     break
338                 except Exception as e:
339                     print(traceback.format_exc())
340                     return
341
342     # Start Capture Thread
343     self.cap_th = CaptureWorker(self.sock, self.video_cap, self.label)
344     self.ping_th = PingWorker(self.sock)
345     self.cap_th.start()
346     self.ping_th.start()
347     self.button_play.setDisabled(True)
348     self.button_pause.setDisabled(False)
349
350 def pause_video(self):
351     self.cap_th.stop()
352     self.ping_th.stop()
353     self.button_play.setDisabled(False)
354     self.button_pause.setDisabled(True)
355
356 def find_camera(self):
357     find_camera_list()
358     self.type_combo.clear()
359     for i in camera_list:
360         self.type_combo.addItem(camera_list[i])
361     self.type_combo.addItem("Saved Video")
362
363
364 def closeEvent(self, event):
365     event.accept()
366

```

- Sender Window의 기본적인 UI와 UI 기능을 선언하는 QWidget
- Sender Window가 열리기 전 5G\_NR 장비에게 **WS\_REQ** 패킷을 전송하고 5G\_NR 장비로부터 **WS\_RESP\_MAGIC\_NUM** 을 수신하는 과정을 통해 컴퓨터와 5G\_NR 장비 사이 간 socket을 형성 (264~277번째 줄)
- 279~316번째 줄은 Sender Window UI 디자인 설정 및 각 부분에 대한 기능 설정 코드로 크게 6개로 이루어져 있다.
  - label : 현재 전송되는 영상 데이터가 재생되는 구역
  - video\_file\_address : 저장된 영상 데이터를 보내는 경우 영상 데이터가 저장된 경로를 입력하는 구역
  - type\_combo : 저장된 영상 데이터와 실시간 영상 데이터 중 보낼 데이터를 선택 가능한 선택 박스로 현재 연결된 카메라 목록 중 사용하고자 하는 카메라 혹은 저장된 영상 중 선택
  - button\_play : 영상 데이터 전송을 시작하는 버튼으로 **play\_send\_video** 함수와 연동됨
  - button\_pause : 영상 데이터 전송을 중지하는 버튼으로 **pause\_video** 함수와 연동됨
  - button\_find : 컴퓨터에 연결된 카메라 목록을 다시 검색하여 type\_combo에 갱신하는 버튼으로 **find\_camera** 함수와 연동
- 318~320번째 줄은 GPS 센서를 통해 현재 자신의 위치를 측정하는

GPSWorker(Qthread)를 호출 및 실행하는 부분

■ def play\_send\_video(self):

```
323     def play_send_video(self):
324         # Define OpenCV by type of transmission data(Camera / Video) #
325         if self.type_combo.currentText() == "Saved Video":
326             self.send_data_type = self.video_file_address.text()
327             try:
328                 self.video_cap = cv2.VideoCapture(self.send_data_type)
329             except:
330                 print(traceback.format_exc())
331                 return
332         else:
333             for i in camera_list:
334                 if self.type_combo.currentText() == camera_list[i]:
335                     try:
336                         self.video_cap = cv2.VideoCapture(cv2.CAP_DSHOW+i)
337                         break
338                     except Exception as e:
339                         print(traceback.format_exc())
340                         return
341
342         # Start Capture Thread
343         self.cap_th = CaptureWorker(self.sock, self.video_cap, self.label)
344         self.ping_th = PingWorker(self.sock)
345         self.cap_th.start()
346         self.ping_th.start()
347         self.button_play.setDisabled(True)
348         self.button_pause.setDisabled(False)
```

- type\_combo에서 선택한 영상 전송 매체에 따라 전송할 영상 데이터를 설정  
(324~340번째 줄)
- 영상 데이터의 프레임을 불러와 **label**에 표현 및 **send\_5g** 함수를 통해 packet을 전송하는 **CaptureWorker**를 실행
- Latency 측정을 위한 **PingWorker**를 실행

- 활성화된 button\_play를 비활성화
- 비활성화된 button\_pause를 활성화

■ def pause\_video (self):

```
350     def pause_video(self):
351         self.cap_th.stop()
352         self.ping_th.stop()
353         self.button_play.setDisabled(False)
354         self.button_pause.setDisabled(True)
```

- CaptureWorker와 PingWorker를 정지
- 비활성화된 button\_play를 활성화
- 활성화된 button\_pause를 비활성화

■ def find\_camera(self):

```
356     def find_camera(self):
357         find_camera_list()
358         self.type_combo.clear()
359         for i in camera_list:
360             self.type_combo.addItem(camera_list[i])
361         self.type_combo.addItem("Saved Video")
```

- 컴퓨터에 현재 연결된 카메라 목록을 갱신하면서  
type\_combo에 있는 값들도 변경

- class GPSWorker(QThread):

```
114 class GPSWorker(QThread):
115     def __init__(self):
116         super().__init__()
117
118         self.trig = True
119         while True:
120             try:
121                 self.ser = serial.Serial(SER_PORT, SER_BAUD)
122                 break
123             except:
124                 print(traceback.format_exc())
125
126     def run(self):
127         global latitude
128         global longitude
129
130         while self.trig:
131             if self.ser.readable(): # Read data only readable chance
132                 ser_resp = self.ser.readline() # Read GPS data from GPS sensor
133                 resp_data = ser_resp.decode().split(sep=',') # Decode and Split data to use
134                 if resp_data[0] == '$GPGLL': # Read only GPGLL data
135                     # Calculate GPS data ( DMS -> Degree )
136                     # latitude
137                     lat_val1 = math.trunc(float(resp_data[1]) / 100)
138                     lat_val2 = float(resp_data[1]) % 100
139                     lat_val2 = lat_val2 / 60
140                     if (latitude != lat_val1 + lat_val2):
141                         latitude = lat_val1 + lat_val2
142                     # longitude
143                     long_val1 = math.trunc(float(resp_data[3]) / 100)
144                     long_val2 = float(resp_data[3]) % 100
145                     long_val2 = long_val2 / 60
146                     if (longitude != long_val1 + long_val2):
147                         longitude = long_val1 + long_val2
148                 self.ser.close()
149
150     def stop(self):
151         self.trig = False
152         self.quit()
153         self.wait(100)
```

- GPS 센서로부터 받아온 데이터를 변환 뒤 전역 변수 latitude, longitude에 저장하는 역할
- 환경 변수 SER\_PORT와 SER\_BAUD를 사용하여 GPS 센서와 serial 통신 기반 연결 진행

- GPS 센서를 통해 받아오는 데이터는 DMS(Degrees Minutes Seconds)로 Degree(위도, 경도)로 변경

- class CaptureWorker(QThread):

```

156 class CaptureWorker(QThread):
157     def __init__(self, sock, cap, label):
158         super().__init__()
159         global pkt_seq_num
160         pkt_seq_num = 0
161         self.video_cap = cap
162         self.video_label = label
163         self.sock = sock
164         self.trig = True
165
166     def run(self):
167         # 2023.06.08 frame 100 msec
168         while self.trig:
169             cv2.waitKey(SENDER_FRAME_MSEC)
170             ret, frame = self.video_cap.read()
171             if ret:
172                 try:
173                     np_frame = numpy.asarray(rescale_frame(frame, SEND_FRAME_WIDTH, SEND_FRAME_HEIGHT))
174                 except:
175                     print(traceback.format_exc())
176                 try:
177                     # Update video label
178                     frame = cv2.cvtColor(np_frame, cv2.COLOR_BGR2RGB)
179                     image = QImage(frame, frame.shape[1], frame.shape[0], QImage.Format_RGB888)
180                     pixmap = QPixmap.fromImage(image)
181                     pixmap = pixmap.scaled(self.video_label.width(), self.video_label.height(), Qt.KeepAspectRatio)
182                     self.video_label.setPixmap(pixmap)
183                 except:
184                     print(traceback.format_exc())
185                 try:
186                     for i in range(SEND_FRAME_HEIGHT):
187                         data = np_frame[i].flatten().tobytes()
188                         line_num = struct.pack(">h", i)
189                         send_5g(self.sock, line_num + data)
190                 except:
191                     print(traceback.format_exc())
192             self.video_label.setPixmap(QPixmap(resource_path('resource/stop_icons.png')))
193             self.video_cap.release()
194
195     def stop(self):
196         self.trig = False
197         self.quit()
198         self.wait(100)

```

- button\_play 버튼을 누르면 실행되는 QThread로 영상 데이터를 label에 표현 및 영상 데이터 프레임을 나눠 패킷에 첨부해 전송하는 역할
- rescale\_frame 함수를 통한 영상 데이터 프레임 사이즈를 변경
- 영상 데이터 프레임은 배열 형태로 배열을 순서대로 나눈 뒤 send\_5g 함수를 통해 패킷을 전송
- 전송 시 프레임 조립을 위한 몇 번째 데이터인지 정보를 같이 전송

- class PingWorker(QThread):

```
203 class PingWorker(QThread):
204     def __init__(self, sock):
205         super().__init__()
206         self.sock = sock
207         send_ping_length = 14
208         v2x_tx_pdu_p = packet_header_struct.V2X_TxPDU(
209             magic_num=htons(0xf2f2),
210             ver=0x0001,
211             psid=5271,
212             e_v2x_comm_type=0,
213             e_payload_type=4,
214             elements_indicator=0,
215             tx_power=20,
216             e_signer_id=0,
217             e_priority=0,
218             channel_load=0,
219             reserved1=0,
220             expiry_time=0,
221             transmitter_profile_id=100,
222             peer_l2id=0,
223             reserved2=0,
224             reserved3=0,
225             crc=0,
226             length=send_ping_length
227         )
228
229         self.header = bytes(v2x_tx_pdu_p)
230
231         self.trig = True
232
233     def run(self):
234         while self.trig:
235             # RTT 패킷 수신
236             try:
237                 packet = self.sock.recv(1024)
238
239                 if packet[-6:][:2] == PING_INDICATOR :
240                     # Delay calculate time data
241                     recv_time = datetime.now().strftime("%S%f")
242                     byte_rt = int(recv_time).to_bytes(length=4, byteorder="big", signed=False)
243                     send_time = datetime.now().strftime("%S%f")
244                     byte_st = int(send_time).to_bytes(length=4, byteorder="big", signed=False)
245                     # RTT packet delivery
246                     payload_data = b'\x03\x02' + packet[-4:] + byte_rt + byte_st
247
248                     send_data = self.header + payload_data
249                     self.sock.send(send_data)
250                     time.sleep(RTT_TIMER)
251             except:
252                 print(traceback.format_exc())
253
254     def stop(self):
255         self.trig = False
256         self.quit()
257         self.wait(100)
```



- button\_play 버튼을 누르면 실행되는 QThread로 RTT 기반 Latency 측정을 위해 RTT 패킷을 수신 및 답장을 전송하는 역할
- 작동하는 동안 패킷을 지속적으로 수신하며 **PING\_INDICATOR**를 통해 현재 수신한 패킷이 Latency와 관련된 패킷인지 확인
- Latency 관련 패킷을 수신하는 경우 패킷 수신 시간과 패킷 송신 시간을 기록하여 패킷에 첨부 후 Receiver 측에 전송
- 필요 시 **RTT\_TIMER**에 저장된 시간만큼 다음 패킷 전송 전 대기

## D. receiver\_window.py

- i. 역할
  - Select Window에서 Receiver Window 선택 시 나오는 Receiver Window를 구성하는 UI와 데이터 송신과 관련된 주요 기능
- ii. 필요 라이브러리 및 파일
  - Os
  - cv2
  - csv
  - json
  - time

- math
- numpy
- serial
- pickle
- struct
- psutil
- requests
- haversine
- datetime
- socket
- scapy
- PyQt5
- collections
- screeninfo
- matplotlib
- PyQtWebEngine
- packet\_header\_struct.py

```
1  ✓ import os
2      import cv2
3      import csv
4      import json
5      import time
6      import math
7      import numpy
8      import serial
9      import pickle
10     import struct
11     import psutil
12     import requests
13     import haversine
14     import datetime as dt
15     import packet_header_struct
16     from socket import *
17     from scapy.all import *
18     from PyQt5.QtGui import *
19     from PyQt5.QtCore import *
20     from PyQt5.QtWidgets import *
21     from collections import deque
22     from screeninfo import get_monitors
23     from matplotlib.figure import Figure
24     from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
25     from PyQt5.QtWebEngineWidgets import QWebEngineView
26     from PyQt5.QtWebEngineWidgets import QWebEngineScript
```

### iii. 코드 설명

- 환경 변수 (13~37번째 줄)(환경에 따라 변경 필요)

```
28 # PyQt Windows Size
29 monitor_size_width = get_monitors()[0].width
30 monitor_size_height = get_monitors()[0].height
31 BLANK_SPACE = 15
32 GRAPH_WIN_SIZE_W = monitor_size_width - BLANK_SPACE*2
33 GRAPH_WIN_SIZE_H = int(monitor_size_height / 2) - BLANK_SPACE
34 VIDEO_WIN_SIZE_W = int(monitor_size_width / 2) - BLANK_SPACE
35 VIDEO_WIN_SIZE_H = int(monitor_size_height / 2) - BLANK_SPACE*3
36 NAVIGATION_WIN_SIZE_W = int(monitor_size_width / 2) - BLANK_SPACE
37 NAVIGATION_WIN_SIZE_H = int(monitor_size_height / 2) - BLANK_SPACE*3
38
39 # Socket Value
40 DEVICE_ADDR = '192.168.1.11'
41 DEVICE_PORT = 47347
42
43 # Packet Value
44 MAX_PACKET_SIZE = 1502
45 MAX_PAYLOAD_SIZE = 1300
46
47 # Camera Capture Size
48 RECV_FRAME_WIDTH = 300
49 RECV_FRAME_HEIGHT = 300
50
51 # GPS Sensor Variable
52 SER_PORT = 'COM5' # Serial Port
53 SER_BAUD = 9600 # Serial Baud Rate
54
55 # Log Cycle
56 HEADER_LOG_CYCLE = 60 # Seconds
57
58 # RTT Variable
59 RTT_TIMER = 1
60
61 # Packet Variable
62 WS_REQ = b"\xf1\xf1\x00\x01\x00\x00\x00\x00\x00\x00\x14\x97\x00\x00\x00"
63 WS_RESP_MAGIC_NUM = b'\xf1\xf2'
64 RX_MAGIC_NUM = b'\xf3\xf2'
65 VIDEO_DATA_INDICATOR = b'\x03\x01'
66 PING_INDICATOR = b'\x03\x02'
67
68 # Graph Data Variable
69 NET_IF = "이더넷"
70
71 # Navigation HTML File Path
72 HTML_FILE_PATH = './resource/Tmap.html'
73
74 # Bad Condition Data Variable
75 WEATHER_API_URL = 'http://apis.data.go.kr/1360000/VilageFcstInfoService_2.0/getUltraSrtNcst'
76 WEATHER_API_SERVICE_KEY = 'QEPmvbFk9szqqPD8q9+s2ezo0OoY7VcAt4Rt1QPseyZ5LQucie5H90jnJj/GO4H1I41QrmGWQxhCF9FG'
77 WEATHER_CONDITION_WAIT_TIMER = 5
78 WEATHER_CONDITION_ERROR_RESEND_TIMER = 10
79 WEATHER_CONDITION_RESEND_TIMER = 600
80 ROAD_API_URL = 'https://apis.openapi.sk.com/tmap/traffic'
81 ROAD_API_SERVICE_KEY = 'f0sIyENUef8ArejvlqGDU4p66e0sMRjB5kII22do'
82 ROAD_CONDITION_WAIT_TIMER = 5
83 ROAD_CONDITION_RESEND_TIMER = 120
```

- monitor\_size\_width : 주 모니터 넓이 값
- monitor\_size\_height : 주 모니터 높이 값
- BLANK\_SPACE : Receiver Window와 모니터 화면 사이의 빈 공간 두께
- GRAPH\_WIN\_SIZE\_W : 성능 그래프 창 넓이 값
- GRAPH\_WIN\_SIZE\_H : 성능 그래프 창 높이 값
- VIDEO\_WIN\_SIZE\_W : 비디오 창 넓이 값
- VIDEO\_WIN\_SIZE\_H : 비디오 창 높이 값
- NAVIGATION\_WIN\_SIZE\_W : 네비게이션 창 넓이 값
- NAVIGATION\_WIN\_SIZE\_H : 네비게이션 창 높이 값
- DEVICE\_ADDR : 5G-NR 장비와 노트북 사이 통신을 위한 IP 주소
- DEVICE\_PORT : 5G-NR 장비와 노트북 사이 통신을 위한 포트 주소
- RECV\_FRAME\_WIDTH : 수신하는 영상 데이터 프레임 넓이
- RECV\_FRAME\_HEIGHT : 수신하는 영상 데이터 프레임 높이
- SER\_PORT : GPS 센서와 serial 통신을 위한 포트 이름
- SER\_BAUD : GPS 센서와 serial 통신을 위한 포트 번호
- HEADER\_LOG\_CYCLE : csv 형태의 log를 저장하는 주기(단위 : 초)
- RTT\_TIMER : RTT기반 Latency 측정을 위해 RTT 패킷을 송신하는 주기

(단위 : 초)

- WS\_REQ : 컴퓨터가 5G-NR 장비와의 연결을 위해 5G-NR 장비에 연결 요청 패킷
- WS\_RESP\_MAGIC\_NUM : WS\_REQ를 받은 5G-NR 장비가 연결을 수락할 시 컴퓨터가 받는 패킷
- RX\_MAGIC\_NUM : 현재 수신한 패킷이 성능 측정 혹은 데이터 전송을 위한 패킷인지 확인하기 위한 값
- VIDEO\_DATA\_INDICATOR : 현재 수신한 패킷이 영상 데이터 전송을 위한 패킷인지 확인하기 위한 값
- PING\_INDICATOR : 현재 수신한 패킷이 Latency 측정을 위한 패킷인지 확인하기 위한 값
- NET\_IF : Throughput 데이터를 얻기 위하여 컴퓨터와 5G-NR 장비 연결 사이 컴퓨터의 물리적 이더넷 포트 이름
- HTML\_FILE\_PATH : T MAP API를 활용하기 위한 HTML 파일의 위치
- WEATHER\_API\_URL : 공공 데이터 포털 날씨 API를 요청할 때 쓰이는 URL
- WEATHER\_API\_SERVICE\_KEY : 공공 데이터 포털 날씨 API 사용 시 필요한 KEY

- WEATHER\_CONDITION\_WAIT\_TIMER : 공공 데이터 포털 날씨 API 요청 후 API 수신까지 기다리는 시간
- WEATHER\_CONDITION\_ERROR\_RESEND\_TIMER : 공공 데이터 포털 날씨 API 수신 시 ERROR 인 경우 다시 요청할 때까지 기다리는 시간
- WEATHER\_CONDITION\_RESEND\_TIMER : 공공 데이터 포털 날씨 API 수신 후 다시 요청할 때까지 기다리는 시간
- ROAD\_API\_URL : T MAP 도로 혼잡도 API를 요청할 때 쓰이는 URL
- ROAD\_API\_SERVICE\_KEY : T MAP 도로 혼잡도 API 사용 시 필요한 KEY
- ROAD\_CONDITION\_WAIT\_TIMER : T MAP 도로 혼잡도 API 수신까지 기다리는 시간
- ROAD\_CONDITION\_RESEND\_TIMER : T MAP 도로 혼잡도 API 수신 후 다시 요청할 때까지 기다리는 시간

- 전역 변수

```
87 sender_latitude = 37.570286992195
88 sender_longitude = 126.98361037914
89 latitude = 37.570286992195
90 longitude = 126.98361037914
91 road_condition = 0
92 weather_condition = 0
93 pdr_result = 0.0
94 throughput_result = 0.0
95 latency_result = 0.0
96 distance_result = 0.0
97 result_queue = deque()
98 webView = 0
99 wes_tag = True
```

- sender\_latitude : 데이터 송신자의 GPS 센서 기반 위도 값
- sender\_longitude : 데이터 송신자 GPS 센서 기반 경도 값
- latitude : GPS 센서로 측정되는 본인의 위도 값
- longitude : GPS 센서로 측정되는 본인의 경도 값
- road\_condition : 현재 도로 복잡도 상황
- weather\_condition : 현재 날씨 상황
- pdr\_result : 갱신되는 Packet Delivery Ratio 값이 저장되는 변수
- throughput\_result : 갱신되는 Throughput 값이 저장되는 변수
- latency\_result : 갱신되는 Latency 값이 저장되는 변수
- distance\_result : 갱신되는 Distance 값이 저장되는 변수



- result\_queue : 수신한 패킷의 헤더 및 데이터들이 저장되는 큐
- WebView : 네비게이션이 표시되는 창 변수
- wes\_tag : 5G\_NR 장비와 컴퓨터 사이 연결을 성공하기 위해 반복 연결 시도하는 트리거 값
  - def resource\_path(relative\_path):
- Sender Window의 함수와 동일 함수로 Sender Window 참고
  - class GPSWorker(QThread):
- Sender Window의 QThread와 동일 QThread로 Sender Window 참고

- class SaveHeaderWorker(QThread):

```
class SaveHeaderWorker(QThread):
    info_signal = pyqtSignal(str)

    def __init__(self, info_box, header_q):
        super().__init__()

        self.info_box = info_box
        self.header_q = header_q
        self.trig = True

    def run(self):
        global result_queue
        i = 0

        while self.trig:
            num_header = len(self.header_q)
            if num_header > 0:
                try:
                    now = dt.datetime.now()
                    past = now - dt.timedelta(minutes=1)
                    file_name = ('TETRI GBU 01(RXS)_' + past.strftime('%Y-%m-%d %H:%M') + ".")
                    file_name = file_name + now.strftime('%Y-%m-%d %H:%M') + ". " + str(HEADER_LOG_CYCLE) + "seconds.csv"
                    f = open(file_name, 'w', encoding='utf-8', newline='')
                    wr = csv.writer(f)
                    header_list = ['No.', 'eDeviceType', 'eTeleCommType', 'unDeviceId', 'ulTimeStamp',
                                   'eServiceId', 'eActionType', 'eRegionId', 'ePayloadType', 'eCommId', 'usDbVer',
                                   'usSwVer', 'ulPayloadLength', 'ulPayloadCrc32',
                                   'Road Condition', 'Weather Condition',
                                   'PDR', 'Throughput', 'Latency', 'Distance', 'Mileage']
                    wr.writerow(header_list)

                    i = 0
                    while True:
                        try:
                            header_log = self.header_q.popleft()

                            # DB_V2X (length = 54)
                            eDeviceType = struct.unpack('>I', header_log[0][38:42])[0]
                            eTeleCommType = struct.unpack('>I', header_log[0][42:46])[0]
                            unDeviceId = struct.unpack('>I', header_log[0][46:50])[0]
                            eServiceId = struct.unpack('>I', header_log[0][58:62])[0]
                            eActionType = struct.unpack('>I', header_log[0][62:66])[0]
                            eRegionId = struct.unpack('>I', header_log[0][66:70])[0]
                            ePayloadType = struct.unpack('>I', header_log[0][70:74])[0]
                            eCommId = struct.unpack('>I', header_log[0][74:78])[0]
                            usDbVer = struct.unpack('>H', header_log[0][78:80])[0]
                            usSwVer = struct.unpack('>H', header_log[0][80:82])[0]
                            ulPayloadLength = struct.unpack('>I', header_log[0][82:84])[0]
                            ulPayloadCrc32 = struct.unpack('>I', header_log[0][84:88])[0]

                            # Calculated Results
                            road_condition_log = header_log[1]
                            weather_condition_log = header_log[2]
                            pdr_result_log = header_log[3]
                            throughput_result_log = header_log[4]
                            latency_result_log = header_log[5]
                            distance_result_log = header_log[6]
                            ulTimeStamp = header_log[9]

                            if i == 0:
                                mileage_log = 0
                                before_latitude = header_log[7]
                                before_longitude = header_log[8]
                            else:
                                mileage_log = mileage_log + haversine.haversine((before_latitude, before_longitude),
                                                                                 (header_log[7], header_log[8]), unit='m')
                                before_latitude = header_log[7]
                                before_longitude = header_log[8]

                            log = [
                                i,
                                eDeviceType,
                                eTeleCommType,
                                unDeviceId,
                                ulTimeStamp,
                                eServiceId,
                                eActionType,
                                eRegionId,
                                ePayloadType,
                                eCommId,
                                usDbVer,
                                usSwVer,
                                ulPayloadLength,
                                ulPayloadCrc32,
                                road_condition_log,
                                weather_condition_log,
                                pdr_result_log,
                                throughput_result_log,
                                latency_result_log,
                                distance_result_log,
                                mileage_log
                            ]

                            if now < header_log[9]:
                                break
                            else:
                                wr.writerow(log)
                                i = i+1
                        except:
                            break

                    f.close()
                    self.info_signal.emit(dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S') + "\n - Saving Log File\n(" + file_name + ")\n - Mileage : " + str(mileage_log))
                    time.sleep(HEADER_LOG_CYCLE)
                except:
                    print(traceback.format_exc())

    def stop(self):
        self.trig = False
        self.quit()
        self.wait(10)
```

- 수신한 패킷 헤더 내용과 성능 지표들을 csv 파일 형태로 기록하는 QThread
- "ETRI\_OBU\_장치ID\_시작시간\_종료시간\_저장시간.확장자.csv"의 양식으로 로그파일을 기록
- header\_q 명칭의 큐 안에 들어있는 패킷 헤더 정보와 성능 지표를 차례로 꺼내서 시간 순으로 기록
- 기록되는 헤더 : DB\_V2X

```

185 # DB_V2X (length = 54)
186 eDeviceType = struct.unpack(">i", header_log[0][38:42])[0]
187 eTeleCommType = struct.unpack(">i", header_log[0][42:46])[0]
188 unDeviceId = struct.unpack(">i", header_log[0][46:50])[0]
189 eServiceId = struct.unpack(">i", header_log[0][58:62])[0]
190 eActionType = struct.unpack(">i", header_log[0][62:66])[0]
191 eRegionId = struct.unpack(">i", header_log[0][66:70])[0]
192 ePayloadType = struct.unpack(">i", header_log[0][70:74])[0]
193 eCommId = struct.unpack(">i", header_log[0][74:78])[0]
194 usDbVer = struct.unpack(">H", header_log[0][78:80])[0]
195 usHwVer = struct.unpack(">H", header_log[0][80:82])[0]
196 usSwVer = struct.unpack(">H", header_log[0][82:84])[0]
197 ulPayloadLength = struct.unpack(">i", header_log[0][84:88])[0]
198 ulPayloadCrc32 = struct.unpack(">i", header_log[0][88:92])[0]

```

- 기록되는 헤더 이외 값 : Latency, PDR, Distance, Throughput, 패킷 수신 시간, 도로 상황, 날씨 상황
- class ViewWorker(QThread):

```

264 class ViewWorker(QThread):
265     def __init__(self, frame, label):
266         super().__init__()
267         self.frame = frame
268         self.video_label = label
269         self.trig = True
270
271     def run(self):
272         while self.trig:
273             try:
274                 show_frame = cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB)
275                 image = QImage(show_frame, show_frame.shape[1], show_frame.shape[0], QImage.Format_RGB888)
276                 pixmap = QPixmap.fromImage(image)
277                 self.video_label.setPixmap(pixmap)
278             except:
279                 print(traceback.format_exc())
280                 time.sleep(0.02)
281                 self.video_label.setPixmap(QPixmap(resource_path('./resource/stop_icons.png')))
282
283     def stop(self):
284         self.trig = False
285         self.quit()
286         self.wait(10)

```

- 수신한 패킷 내 영상 데이터가 조립되어 저장되는 frame 변수를 0.02 초마다 불러와 화면에 현재 영상 데이터 프레임을 보여주는 QThread
- 데이터 수신을 하고 있지 않을 때 stop\_icons.png를 화면에 표시

- class ReceiveWorker(QThread):

```
class ReceiveWorker(QThread):
    def __init__(self, sock, frame, pkt_num_q, header_q):
        super().__init__()
        global DEVICE_ADDR
        global DEVICE_PORT
        global RECV_FRAME_WIDTH
        global RECV_FRAME_HEIGHT
        global WS_RESP_MAGIC_NUM
        global RX_MAGIC_NUM
        global VIDEO_DATA_INDICATOR
        global ws_tag

        self.show_frame = frame
        self.pkt_num_q = pkt_num_q
        self.header_q = header_q
        self.sock = sock
        self.trig = True
        while ws_tag:
            try:
                self.sock.send(WS_REQ)
                ws_resp = self.sock.recv(1024)
                if ws_resp[0:2] != WS_RESP_MAGIC_NUM:
                    continue
                ws_tag = False
                break
            except:
                print(traceback.format_exc())

    def run(self):
        global sender_latitude
        global sender_longitude
        global latency_result

        break_pre_pkt_temp = ""
        while self.trig:
            # Receive Packet
            try:
                packet = self.sock.recv(1024 * 12)
            except:
                print(traceback.format_exc())
                continue

            packet_ptr = 0
            receive_time = datetime.now().strftime("%S%f")

            while True:
                try:
                    if packet[packet_ptr:packet_ptr + 2] == RX_MAGIC_NUM:
                        if packet[packet_ptr+38:packet_ptr+40] == PING_INDICATOR:
                            packet_header = packet[packet_ptr:packet_ptr + 38]
                            payload_length = int.from_bytes(packet_header[36:38], "big")
                            payload = packet[packet_ptr + 38:packet_ptr + 38 + payload_length]

                            sender_recv_time = int.from_bytes(payload[6:10], "big", signed=False)
                            sender_send_time = int.from_bytes(payload[10:14], "big", signed=False)
                            receiver_send_time = int.from_bytes(payload[2:6], "big", signed=False)
                            receiver_delay = int(receive_time) - receiver_send_time

                            if receiver_delay < 0:
                                receiver_delay += 60000000
                            sender_delay = sender_send_time - sender_recv_time
                            if sender_delay < 0:
                                sender_delay += 60000000
                            RTT = receiver_delay - sender_delay
                            latency_result = RTT / 2000
                            packet_ptr = packet_ptr + 38 + payload_length
                        else:
                            packet_header = packet[packet_ptr:packet_ptr + 38]
                            db_c2x_header = packet[packet_ptr + 38:packet_ptr + 38 + 54]
                            payload_length = int.from_bytes(packet_header[36:38], "big") - 54
                            payload = packet[packet_ptr + 38 + 54:packet_ptr + 38 + 54 + payload_length]

                            if len(packet[packet_ptr:]) < 38 + 54 + payload_length:
                                break_pre_pkt_temp = packet[packet_ptr:]
                                break

                            # Get and Save data
                            self.header_q.append([packet_header + db_c2x_header, road_condition, weather_condition,
                                                  pdr_result, throughput_result, latency_result, distance_result,
                                                  latitude, longitude, dt.datetime.now()])

                            sender_latitude = float(int.from_bytes(db_c2x_header[46:50], "big")) / 1000000
                            sender_longitude = float(int.from_bytes(db_c2x_header[50:54], "big")) / 1000000

                            if payload[0:2] != VIDEO_DATA_INDICATOR:
                                print("Receive RTT")
                            elif payload[0:2] == VIDEO_DATA_INDICATOR:
                                self.pkt_num_q.append(int.from_bytes(payload[2:6], "big"))
                                try:
                                    frame_line_num = struct.unpack(">h", payload[6:8])[0]
                                    frame_line_data = numpy.frombuffer(payload[8:], dtype=numpy.uint8)
                                    frame_line_data = numpy.reshape(frame_line_data, (RECV_FRAME_WIDTH, -1))
                                    self.show_frame[frame_line_num] = frame_line_data
                                except:
                                    print(traceback.format_exc())
                            else:
                                print("Wrong Indicator")
                                packet_ptr = packet_ptr + 38 + 54 + payload_length
                                if packet_ptr >= len(packet):
                                    break
                        else:
                            packet_ptr = packet_ptr + 1
                            if packet_ptr >= len(packet):
                                break
                except:
                    print(traceback.format_exc())
                    continue

    def stop(self):
        self.trig = False
        self.quit()
        self.wait(10)
```

- 지속적으로 패킷을 수신하는 역할의 QThread
- 패킷 수신 전 **WS\_REQ, WS\_RESP\_MAGIC\_NUM** 패킷을 통해 5G\_NR 장비와 socket 통신 연결을 형성
- 버퍼로부터 한 번에 1024\*12= 12288바이트 크기의 패킷을 수신
- 패킷 분류
  - 수신한 패킷이 **RX\_MAGIC\_NUM**로 시작
    1. 패킷 헤더를 제외한 패킷 내의 데이터가 **PING\_INDICATOR**로 시작하는 경우 RTT 기반 Latency 계산을 위한 패킷이기 때문에 돌아온 시간을 고려하여 **latency\_result** 변수에 Latency 값을 저장
    2. 패킷 헤더를 제외한 패킷 내의 데이터가 **PING\_INDICATOR**로 시작하지 않는 경우 영상 데이터와 관련된 패킷을 의미
    3. 영상 데이터 패킷인 경우 패킷의 헤더 부분과 데이터 부분을 **header\_q** 변수에 저장
    4. 데이터에 포함된 송신자 위도, 경도 정보를 **sender\_latitude, sender\_longitude**에 저장하고 영상 데이터 부분은 **show\_frame**에 저장
  - 수신한 패킷이 **RX\_MAGIC\_NUM**로 시작하지 않음

# 1. 패킷 포인터를 1 증가시키고 다시 패킷 분류 과정을 실시

- class PingWorker(QThread):

```
405 class PingWorker(QThread):
406     def __init__(self, sock):
407         super().__init__()
408         self.sock = sock
409
410         self.v2x_tx_pdu_p = packet_header_struct.V2X_TxPDU(
411             magic_num=htons(0xf2f2),
412             ver=0x0001,
413             psid=5271,
414             e_v2x_comm_type=0,
415             e_payload_type=4,
416             elements_indicator=0,
417             tx_power=20,
418             e_signer_id=0,
419             e_priority=0,
420             channel_load=0,
421             reserved1=0,
422             expiry_time=0,
423             transmitter_profile_id=100,
424             peer_l2id=0,
425             reserved2=0,
426             reserved3=0,
427             crc=0,
428             length=6
429         )
430         self.trig = True
431
432     def run(self):
433         while self.trig:
434             try:
435                 RST_T = datetime.now().strftime("%S%f") # receiver send time
436                 RST = int(RST_T).to_bytes(length=4, byteorder="big", signed=False)
437                 RTT_Packet = bytes(self.v2x_tx_pdu_p) + PING_INDICATOR + RST
438                 self.sock.send(RTT_Packet)
439                 time.sleep(RTT_TIMER)
440             except:
441                 print(traceback.format_exc())
442                 continue
443
444     def stop(self):
445         self.trig = False
446         self.quit()
447         self.wait(10)
```

- RTT 기반 Latency 측정을 위하여 **RTT\_TIMER** 주기로 RTT 관련 패킷을 전송
- 패킷 전송 시 패킷 전송 시간 **RST\_T**와 RTT 패킷임을 구별할 수 있는 **PING\_INDICATOR**를 패킷에 넣어서 전송

- class ReceiverVideoWindow(QWidget):

```
class ReceiverVideoWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.show_frame = numpy.zeros((RECV_FRAME_HEIGHT, RECV_FRAME_WIDTH, 3), numpy.uint8)
        self.pkt_num_q = deque()
        self.header_q = deque()

        while True:
            try:
                self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.sock.connect((DEVICE_ADDR, DEVICE_PORT))
                break
            except:
                print(traceback.format_exc())

        # UI declaration
        # Video Screen Area
        self.label = QLabel()
        self.label.setScaledContents(True)
        self.label.setPixmap(QPixmap(resource_path('./resource/stop_icons.png')))
        # Play video & receive video
        self.button_play = QPushButton("Receive")
        self.button_play.clicked.connect(self.play_receive_video)
        # Stop play & receive video
        self.button_pause = QPushButton("Pause")
        self.button_pause.clicked.connect(self.pause_video)
        self.button_pause.setDisabled(True)
        # Information Box
        self.info_box = QTextEdit()
        self.info_box.setReadOnly(True)

        # UI Arrangement
        self.layout = QGridLayout()
        self.left_layout = QVBoxLayout()
        self.right_layout = QVBoxLayout()
        self.left_layout.addWidget(self.button_play)
        self.left_layout.addWidget(self.button_pause)
        self.left_layout.addWidget(self.info_box)
        self.right_layout.addWidget(self.label)
        self.layout.setColumnStretch(0, 2)
        self.layout.setColumnStretch(1, 4)
        self.layout.addLayout(self.left_layout, 0, 0)
        self.layout.addLayout(self.right_layout, 0, 1)

        # Final UI Layout Arrangement
        self.setLayout(self.layout)
        self.setWindowFlag(Qt.FramelessWindowHint)
        self.setFixedSize(VIDEO_WIN_SIZE_W, VIDEO_WIN_SIZE_H)
        self.move(BLANK_SPACE, int(monitor_size_height/2) + BLANK_SPACE)

        # Receiver Graph Window Setting
        self.receiver_graph_window = ReceiverGraphWindow(self.pkt_num_q)
        self.receiver_graph_window.show()

        # Receiver Navigation Window Setting
        self.navigation_window = NavigationWindow()
        self.navigation_window.show()

        # GPS thread
        self.gps_worker_th = GPSWorker()
        self.gps_worker_th.start()

    def play_receive_video(self):
        self.show_frame = numpy.zeros((RECV_FRAME_HEIGHT, RECV_FRAME_WIDTH, 3), numpy.uint8)
        self.pkt_num_q.clear()
        self.header_q.clear()
        self.info_box.append(dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S') + " : Start Receiving")
        self.rec_th = ReceiveWorker(self.sock, self.show_frame, self.pkt_num_q, self.header_q)
        self.view_th = ViewWorker(self.show_frame, self.label)
        self.ping_th = PingWorker(self.sock)
        self.save_header_th = SaveHeaderWorker(self.info_box, self.header_q)
        self.save_header_th.info_signal.connect(self.update_infobox)
        self.rec_th.start()
        self.view_th.start()
        self.ping_th.start()
        self.save_header_th.start()
        self.button_play.setDisabled(True)
        self.button_pause.setDisabled(False)

    def pause_video(self):
        self.info_box.append(dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S') + " : Stop Receiving")
        self.rec_th.stop()
        self.ping_th.stop()
        self.view_th.stop()
        self.save_header_th.stop()
        self.button_play.setDisabled(False)
        self.button_pause.setDisabled(True)

    def update_infobox(self, log):
        self.info_box.setText(log)

    def closeEvent(self, event):
        event.accept()
```



- Receiver Window의 기본적인 UI와 UI 기능들을 선언하는 QWidget
- 265~498번째 줄은 Receiver Window UI 디자인 설정 및 각 부분에 대한 기능 설정 코드로 크게 6개로 이루어져 있다.
  - label : 현재 수신받는 영상 데이터가 재생되는 구역
  - button\_play : 영상 데이터 수신을 시작하는 버튼으로 **play\_receive\_video** 함수와 연동됨
  - button\_pause : 영상 데이터 전송을 중지하는 버튼으로 **pause\_video** 함수와 연동됨
  - info\_box : 현재 기록되는 로그 기록 상황을 글로 표현하는 구역
- receiver\_graph\_window와 navigation\_window는 각각 성능 그래프, 차량 네비게이션 및 악조건 상황 표출을 위한 다른 window를 의미하며 ReceiverVideoWindow가 열릴 때 같이 열림
- gps\_worker\_th는 지속적으로 자신의 GPS 정보를 GPS 센서로부터 받아오는 역할을 하는 QThread를 의미함

■ def play\_receive\_video(self):

```

512 def play_receive_video(self):
513     self.show_frame = numpy.zeros((RECV_FRAME_HEIGHT, RECV_FRAME_WIDTH, 3), numpy.uint8)
514     self.pkt_num_q.clear()
515     self.header_q.clear()
516     self.info_box.append(dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S') + " : Start Receiving")
517     self.rec_th = ReceiveWorker(self.sock, self.show_frame, self.pkt_num_q, self.header_q)
518     self.view_th = ViewWorker(self.show_frame, self.label)
519     self.ping_th = PingWorker(self.sock)
520     self.save_header_th = SaveHeaderWorker(self.info_box, self.header_q)
521     self.save_header_th.info_signal.connect(self.update_infobox)
522     self.rec_th.start()
523     self.view_th.start()
524     self.ping_th.start()
525     self.save_header_th.start()
526     self.button_play.setDisabled(True)
527     self.button_pause.setDisabled(False)

```

- 영상 데이터 수신을 시작하는 함수로 실행 시 로그를 기록하는 **SaveHeaderWorker**, RTT 기반 Latency를 측정하는 **PingWorker**, label에 현재까지 수신한 영상 데이터의 프레임을 출력하는 **ViewWorker**를 실행
- button\_play를 누르면 button\_play는 비활성화되고 button\_pause는 활성화 상태로 변환

■ def pause\_video(self):

```

529 def pause_video(self):
530     self.info_box.append(dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S') + " : Stop Receiving")
531     self.rec_th.stop()
532     self.ping_th.stop()
533     self.view_th.stop()
534     self.save_header_th.stop()
535     self.button_play.setDisabled(False)
536     self.button_pause.setDisabled(True)

```

- 영상 데이터 수신을 중지하며 button\_play를 눌러서 실행된 QThread들 모두를 정지함

- button\_pause를 누르면 button\_pause는 비활성화되고 button\_play는 활성화 상태로 변환

■ def update\_infobox(self, log):

- info\_box에 현재 로그 파일 기록 상황을 출력할 때 사용

- class PDRWorker(QThread):

```

545 class PDRWorker(QThread):
546     def __init__(self, pkt_num_q, pdr_subplot, pdr_graph_canvas):
547         super().__init__()
548         self.pkt_num_q = pkt_num_q
549         self.pdr_subplot = pdr_subplot
550         self.pdr_graph_canvas = pdr_graph_canvas
551         self.pdr_data = []
552         self.current_time = []
553         self.trig = True
554
555     def run(self):
556         global pdr_result
557
558         while self.trig:
559             try:
560                 pkt_count = 0
561                 start_num = 0
562                 end_num = 0
563                 if len(self.pkt_num_q) != 0:
564                     pkt_count = 1
565                     start_num = self.pkt_num_q.popleft()
566                     end_num = start_num
567                 while True:
568                     if len(self.pkt_num_q) == 0:
569                         break
570                     end_num = self.pkt_num_q.popleft()
571                     pkt_count = pkt_count + 1
572                 if end_num < start_num:
573                     end_num = end_num + 1000000
574                 pdr_result = (pkt_count * 100) / (end_num - start_num + 1)
575
576                 if len(self.pdr_data) == 60:
577                     del self.pdr_data[0]
578                     del self.current_time[0]
579                 self.pdr_data.append(pdr_result)
580                 self.current_time.append(dt.datetime.now())
581
582                 self.pdr_subplot.clear()
583                 self.pdr_subplot.set_ylim(0, 105)
584                 self.pdr_subplot.plot(self.current_time, self.pdr_data)
585                 self.pdr_subplot.text(dt.datetime.now() - dt.timedelta(seconds=0.01), pdr_result + 2,
586                                     "{:.3f}%".format(pdr_result))
587
588                 self.pdr_subplot.set_ylabel("Packet Delivery Ratio(%)")
589                 self.pdr_subplot.fill_between(self.current_time, self.pdr_data, alpha=0.5)
590
591                 self.pdr_graph_canvas.draw()
592             except:
593                 print(traceback.format_exc())
594                 time.sleep(1)
595
596     def stop(self):
597         self.trig = False
598         self.quit()
599         self.wait(10)

```

- 성능 지표 값 PDR(Packet Delivery Ratio)를 계산 및 그래프에 추가할 때 사용되는 QThread
- 1초 주기로 pkt\_num\_q에 저장되어 있는 패킷의 순서 번호를 차례로 세어 현재 PDR을 계산
- $$PDR(\%) = \frac{\text{큐 내 모든 패킷 수량}}{\text{제일 큰 패킷 순번} - \text{제일 작은 패킷 순번}} * 100$$

```
- class ThroughputWorker(QThread):
```

```

602 class ThroughputWorker(QThread):
603     def __init__(self, throughput_subplot, throughput_graph_canvas):
604         super().__init__()
605         self.throughput_subplot = throughput_subplot
606         self.throughput_graph_canvas = throughput_graph_canvas
607         self.throughput_data = []
608         self.current_time = []
609         self.trig = True
610
611     def run(self):
612         global throughput_result
613
614         while self.trig:
615             try:
616                 initial_stats = psutil.net_io_counters(pernic=True)
617
618                 # Wait for the specified interval
619                 time.sleep(1)
620
621                 # Get the updated network statistics
622                 updated_stats = psutil.net_io_counters(pernic=True)
623
624                 for interface, initial in initial_stats.items():
625                     if interface == NET_IF:
626                         updated = updated_stats[interface]
627                         throughput_result = updated.bytes_recv - initial.bytes_recv
628                         break
629                 throughput_result = 0.0
630                 throughput_result = float(throughput_result / 125000)
631
632                 if len(self.throughput_data) == 60:
633                     del self.throughput_data[0]
634                     del self.current_time[0]
635                 self.throughput_data.append(throughput_result)
636                 self.current_time.append(dt.datetime.now())
637
638                 self.throughput_subplot.clear()
639                 self.throughput_subplot.set_ylim(0, 50)
640                 self.throughput_subplot.plot(self.current_time, self.throughput_data)
641                 self.throughput_subplot.text(dt.datetime.now() - dt.timedelta(seconds=0.01), throughput_result + 2,
642                                             "{:.3f}Mbps".format(throughput_result))
643
644                 self.throughput_subplot.set_ylabel("Throughput(Mbps)")
645                 self.throughput_subplot.fill_between(self.current_time, self.throughput_data, alpha=0.5)
646
647                 self.throughput_graph_canvas.draw()
648             except:
649                 print(traceback.format_exc())
650
651     def stop(self):
652         self.trig = False
653         self.quit()
654         self.wait(10)

```

- 성능 지표 값 Throughput(Mbps)을 계산 및 그래프에 추가할 때 사용되는 QThread
- 현재 5G\_NR 장비와 연결된 이더넷 포트(**NET\_IF**)의 1초동안 수신받는 데이터량을 측정하여 Throughput(Mbps)를 계산

- class DistanceWorker (QThread):

```

657 ~ class DistanceWorker(QThread):
658 ~     def __init__(self, distance_subplot, distance_graph_canvas):
659 ~         super().__init__()
660
661         self.distance_subplot = distance_subplot
662         self.distance_graph_canvas = distance_graph_canvas
663         self.distance_data = []
664         self.current_time = []
665         self.trig = True
666
667 ~     def run(self):
668 ~         global sender_latitude
669 ~         global sender_longitude
670 ~         global latitude
671 ~         global longitude
672 ~         global distance_result
673
674 ~         while self.trig:
675 ~             try:
676 ~                 distance_result = haversine.haversine((sender_latitude, sender_longitude),
677 ~                                                         (latitude, longitude), unit='m')
678
679 ~                 if len(self.distance_data) == 60:
680 ~                     del self.distance_data[0]
681 ~                     del self.current_time[0]
682 ~                 self.distance_data.append(distance_result)
683 ~                 self.current_time.append(dt.datetime.now())
684
685 ~                 self.distance_subplot.clear()
686 ~                 self.distance_subplot.set_ylim(0, 105)
687 ~                 self.distance_subplot.plot(self.current_time, self.distance_data)
688 ~                 self.distance_subplot.text(dt.datetime.now() - dt.timedelta(seconds=0.01), distance_result
689 ~                                             "{:.3f}m".format(distance_result))
690
691 ~                 self.distance_subplot.set_ylabel("Distance(Meters)")
692 ~                 self.distance_subplot.fill_between(self.current_time, self.distance_data, alpha=0.5)
693
694 ~                 self.distance_graph_canvas.draw()
695 ~             except:
696 ~                 print(traceback.format_exc())
697 ~                 time.sleep(1)
698
699 ~     def stop(self):
700 ~         self.trig = False
701 ~         self.quit()
702 ~         self.wait(10)

```

- 성능 지표 값 Distance(m)를 계산 및 그래프에 추가할 때 사용되는 QThread
- Sender 컴퓨터와 Receiver 컴퓨터 사이 거리를 1초마다 계산
- Sender 컴퓨터의 GPS 좌표의 경우 패킷 수신을 하며 패킷 안에 포함된 값을 사용

```
- class LatencyWorker(QThread):
```

```

705 class LatencyWorker(QThread):
706     def __init__(self, latency_subplot, latency_graph_canvas):
707         super().__init__()
708         self.latency_subplot = latency_subplot
709         self.latency_graph_canvas = latency_graph_canvas
710         self.latency_data = []
711         self.current_time = []
712         self.trig = True
713
714     def run(self):
715         global latency_result
716
717         while self.trig:
718             try:
719                 self.latency_data.append(latency_result)
720                 self.current_time.append(dt.datetime.now())
721
722                 self.latency_subplot.clear()
723                 self.latency_subplot.set_ylim(0, 40)
724                 self.latency_subplot.plot(self.current_time, self.latency_data)
725                 self.latency_subplot.text(dt.datetime.now() - dt.timedelta(seconds=0.01), latency_result +
726                                           | | | | | | | | "{:.3f}ms".format(latency_result))
727
728                 self.latency_subplot.set_ylabel("Latency(ms)")
729                 self.latency_subplot.fill_between(self.current_time, self.latency_data, alpha=0.5)
730
731                 self.latency_graph_canvas.draw()
732             except:
733                 print(traceback.format_exc())
734             time.sleep(1)
735
736     def stop(self):
737         self.trig = False
738         self.quit()
739         self.wait(10)

```

- 성능 지표 값 Latency(ms)를 그래프에 추가할 때 사용되는 QThread
- PingWorker를 통해 계산된 Latency 값을 그래프에 지속적으로 추가함

- class ReceiverGraphWindow(QWidget):

```

742 class ReceiverGraphWindow(QWidget):
743     def __init__(self, pkt_num_q):
744         super().__init__()
745         self.pkt_num_q = pkt_num_q
746         style = dict(ha='center', va='center', fontsize=28, color='Gray')
747
748         # UI declaration
749         self.pdr_graph_figure = Figure()
750         self.pdr_graph_figure.text(0.5, 0.5, 'PDR', style)
751         self.pdr_graph_canvas = FigureCanvas(self.pdr_graph_figure)
752         self.pdr_subplot = self.pdr_graph_figure.add_subplot()
753
754         self.throughput_graph_figure = Figure()
755         self.throughput_graph_figure.text(0.5, 0.5, 'Throughput', style)
756         self.throughput_graph_canvas = FigureCanvas(self.throughput_graph_figure)
757         self.throughput_subplot = self.throughput_graph_figure.add_subplot()
758
759         self.latency_graph_figure = Figure()
760         self.latency_graph_figure.text(0.5, 0.5, 'Latency', style)
761         self.latency_graph_canvas = FigureCanvas(self.latency_graph_figure)
762         self.latency_subplot = self.latency_graph_figure.add_subplot()
763
764         self.distance_graph_figure = Figure()
765         self.distance_graph_figure.text(0.5, 0.5, 'Distance', style)
766         self.distance_graph_canvas = FigureCanvas(self.distance_graph_figure)
767         self.distance_subplot = self.distance_graph_figure.add_subplot()
768
769         # UI Arrangement
770         self.layout = QGridLayout()
771         self.layout.addWidget(self.pdr_graph_canvas, 0, 0)
772         self.layout.addWidget(self.throughput_graph_canvas, 0, 1)
773         self.layout.addWidget(self.latency_graph_canvas, 1, 0)
774         self.layout.addWidget(self.distance_graph_canvas, 1, 1)
775
776         # Final UI Layout Arrangement
777         self.setLayout(self.layout)
778         self.setWindowTitle("Receiver Graph Window")
779         self.setFixedSize(GRAPH_WIN_SIZE_W, GRAPH_WIN_SIZE_H)
780         self.move(BLANK_SPACE, BLANK_SPACE)
781
782         # Init Graph Window
783         self.init_graph()
784
785     def init_graph(self):
786         self.pdr_subplot.set_ylim(0, 105)
787         self.pdr_subplot.set_ylabel("Packet Delivery Ratio(%)")
788
789         self.throughput_subplot.set_ylim(0, 50)
790         self.throughput_subplot.set_ylabel("Throughput(Mbps)")
791
792         self.latency_subplot.set_ylim(0, 100)
793         self.latency_subplot.set_ylabel("Latency(ms)")
794
795         self.distance_subplot.set_ylim(0, 100)
796         self.distance_subplot.set_ylabel("Distance(Meters)")
797
798         self.pdr_worker_th = PDRWorker(self.pkt_num_q, self.pdr_subplot, self.pdr_graph_canvas)
799         self.pdr_worker_th.start()
800
801         self.distance_worker_th = DistanceWorker(self.distance_subplot, self.distance_graph_canvas)
802         self.distance_worker_th.start()
803
804         self.latency_worker_th = LatencyWorker(self.latency_subplot, self.latency_graph_canvas)
805         self.latency_worker_th.start()
806
807         self.throughput_worker_th = ThroughputWorker(self.throughput_subplot, self.throughput_graph_canvas)
808         self.throughput_worker_th.start()
809

```



- Receiver Window에 의해 열리는 Receiver Graph Window의 기본적인 UI와 UI 기능들을 선언하는 QWidget
- 748~781번째 줄은 Receiver Graph Window UI 디자인 설정 및 그래프 선언 코드로 크게 4개의 그래프로 나뉘어 있다.
- pdr\_graph\_figure : 실시간 PDR 데이터가 그래프로 표현되는 구역
- throughput\_graph\_figure : 실시간 Throughput 데이터가 그래프로 표현되는 구역
- latency\_graph\_figure : 실시간 RTT 기반 Latency 데이터가 그래프로 표현되는 구역
- distance\_graph\_figure : 실시간 Distance 데이터가 그래프로 표현되는 구역
- def init\_graph(self):
  - 4개의 그래프의 축 이름과 축 범위를 선언
  - 각 그래프를 갱신하는 QThread들(PDRWorker, DistanceWorker, DistanceWorker, ThroughputWorker)을 시작하는 함수로 그래프 영역이 모두 선언된 뒤 실행

- class WeatherWorker(QThread):

```
class WeatherWorker(QThread):
    def __init__(self, label):
        super().__init__()
        global weather_condition
        self.condition_label = label

        self.trig = True
        while True:
            try:
                weather_condition = 0
                self.weather_img = QPixmap(resource_path('./resource/weather_0.png'))
                self.condition_label.setPixmap(self.weather_img)
                break
            except:
                print(traceback.format_exc())

    def run(self):
        global latitude
        global longitude
        global weather_condition

        while self.trig:
            try:
                base_date = time.strftime('%Y%m%d')
                base_time = time.strftime('%H%M')
                params = {
                    'serviceKey': WEATHER_API_SERVICE_KEY,
                    'pageNo': '1',
                    'numOfRows': '1000',
                    'dataType': 'JSON',
                    'base_date': base_date,
                    'base_time': base_time,
                    'nx': int(latitude),
                    'ny': int(longitude)
                }
            except:
                continue







            try:
                response = requests.get(WEATHER_API_URL, params=params, timeout=WEATHER_CONDITION_WAIT_TIMER)
            except:
                time.sleep(WEATHER_CONDITION_ERROR_RESEND_TIMER)
                continue

            try:
                result = str(response.content, 'utf-8')
                result = json.loads(result)

                if (result['response']['header']['resultCode'] == '00'):
                    for i in result['response']['body']['items']['item']:
                        if i.get('category') == 'PTY':
                            weather_condition = int(i.get('obsrValue'))
                            if weather_condition == 1:
                                self.weather_img = QPixmap(resource_path('./resource/weather_1.png'))
                            elif weather_condition == 2:
                                self.weather_img = QPixmap(resource_path('./resource/weather_2.png'))
                            elif weather_condition == 3:
                                self.weather_img = QPixmap(resource_path('./resource/weather_3.png'))
                            elif weather_condition == 5:
                                self.weather_img = QPixmap(resource_path('./resource/weather_5.png'))
                            elif weather_condition == 6:
                                self.weather_img = QPixmap(resource_path('./resource/weather_6.png'))
                            elif weather_condition == 7:
                                self.weather_img = QPixmap(resource_path('./resource/weather_7.png'))
                            else:
                                self.weather_img = QPixmap(resource_path('./resource/weather_0.png'))
                            while True:
                                try:
                                    self.condition_label.setPixmap(self.weather_img)
                                    break
                                except:
                                    print("Retry to change image")
                                    break
                        elif (result['response']['header']['resultCode'] == '03'):
                            print("No API data")
                        else:
                            print(result['response']['header']['resultCode'] + ' Content Error')
            except:
                continue
            time.sleep(WEATHER_CONDITION_RESEND_TIMER)

    def stop(self):
        self.trig = False
        self.quit()
        self.wait(100)
```

- 현재 GPS 좌표 지역의 날씨 상황을 알기 위해 공공 데이터 포털을 통한 초단기 날씨 상황 정보를 얻어오는데 사용되는 QThread
- 날씨는 크게 7가지 상태로 나뉘며 날씨 정보를 성공적으로 얻어오면 **WEATHER\_CONDITION\_RESEND\_TIMER** 이후 다시 날씨 정보를 요청
- 날씨 종류 사진

<div>날씨 정보</div> <div>N/A 정보 없음</div>	<div>날씨 정보</div> <div> 비</div>	<div>날씨 정보</div> <div> 비 / 눈</div>
<div>날씨 정보</div> <div> 눈</div>	<div>날씨 정보</div> <div> 빗방울</div>	<div>날씨 정보</div> <div> 빗방울눈날림</div>
	<div>날씨 정보</div> <div> 눈날림</div>	

- 공공 데이터 포털 기상청\_단기예보 링크

➤ <https://www.data.go.kr/data/15084084/openapi.do>

- class RoadWorker(QThread):

```
class RoadWorker(QThread):
    def __init__(self, label):
        super().__init__()
        global road_condition
        self.road_label = label

        self.trig = True
        while True:
            try:
                road_condition = 0
                self.road_img = QPixmap(resource_path('./resource/road_0.png'))
                self.road_label.setPixmap(self.road_img)
                break
            except:
                print(traceback.format_exc())

    def run(self):
        global latitude
        global longitude
        global road_condition

        congestion_degree = 0
        congestion_counter = 0
        headers = {
            "appKey": ROAD_API_SERVICE_KEY
        }

        while self.trig:
            try:
                congestion_degree = 0
                congestion_counter = 0
                params = {
                    "version": "1",
                    "format": "json",
                    "reqCoordType": "WGS84GEO",
                    "resCoordType": "WGS84GEO",
                    "zoomLevel": 17,
                    "trafficType": "AUTO",
                    "centerLon": longitude,
                    "centerLat": latitude
                }

                try:
                    response = requests.get(ROAD_API_URL, headers=headers, params=params, timeout=ROAD_CONDITION_WAIT_TIMER)
                except:
                    continue
                if response.status_code == 200:
                    data = response.json()

                    for feature in data["features"]:
                        properties = feature["properties"]
                        if feature["geometry"]["type"] == "LineString":
                            for Coordinates in feature["geometry"]["coordinates"]:
                                if haversine.haversine((latitude, longitude), (Coordinates[1], Coordinates[0]), unit='m') <= 50:
                                    congestion_counter = congestion_counter + 1
                                    congestion_degree = congestion_degree + properties["congestion"]

                    if congestion_counter != 0:
                        congestion_degree = math.ceil(congestion_degree / congestion_counter)

                    if congestion_degree == 0:
                        self.road_img = QPixmap(resource_path('./resource/road_0.png'))
                    elif congestion_degree == 1:
                        self.road_img = QPixmap(resource_path('./resource/road_1.png'))
                    elif congestion_degree == 2:
                        self.road_img = QPixmap(resource_path('./resource/road_2.png'))
                    elif congestion_degree == 3:
                        self.road_img = QPixmap(resource_path('./resource/road_3.png'))
                    elif congestion_degree == 4:
                        self.road_img = QPixmap(resource_path('./resource/road_4.png'))
                    self.road_label.setPixmap(self.road_img)
                    time.sleep(ROAD_CONDITION_RESEND_TIMER)
            except:
                continue

    def stop(self):
        self.trig = False
        self.quit()
        self.wait(100)
```

- 현재 GPS 좌표 지역의 도로 상황을 알기 위해 T MAP API를 통한 도로 상황 정보를 얻어오는데 사용되는 QThread
- 자신의 GPS 좌표 일정 범위 내의 모든 도로 정보를 json 형태로 받아온 뒤 도로 혼잡도에 대한 평균을 계산
- 도로 상태는 크게 5가지 상태로 나뉘며 도로 정보를 성공적으로 얻어오면 **ROAD\_CONDITION\_RESEND\_TIMER** 이후 다시 도로 정보를 요청
- 도로 정보 사진



- T MAP API 가이드 링크

➤ <https://tmapapi.sktelecom.com/main.html#websevice/docs/tmapTrafficDoc>

- Class NavigationWindow(QWidget)

```

1005 class NavigationWindow(QWidget):
1006     def __init__(self):
1007         super().__init__()
1008         global webView
1009         global latitude
1010         global longitude
1011
1012         # UI declaration
1013         self.weather_label = QLabel()
1014         self.weather_label.setAlignment(Qt.AlignCenter)
1015         self.road_label = QLabel()
1016         self.road_label.setAlignment(Qt.AlignCenter)
1017         webView = QWebEngineView()
1018         webView.load(QUrl.fromLocalFile(resource_path(HTML_FILE_PATH)))
1019
1020         # UI Arrangement
1021         self.layout = QGridLayout()
1022         self.layout.addWidget(webView, 0, 0, 2, 1)
1023         self.layout.addWidget(self.weather_label, 0, 1, 1, 1)
1024         self.layout.addWidget(self.road_label, 1, 1, 1, 1)
1025
1026         # Final UI Layout Arrangement
1027         self.setLayout(self.layout)
1028         self.setWindowFlag(Qt.FramelessWindowHint)
1029         self.setFixedSize(NAVIGATION_WIN_SIZE_W, NAVIGATION_WIN_SIZE_H)
1030         self.move(int(monitor_size_width/2), int(monitor_size_height/2) + BLANK_SPACE)
1031
1032         self.timer = QTimer(self)
1033         self.timer.timeout.connect(self.receiving)
1034         self.timer.start(1000) # Cycle : 1 second
1035
1036         # Weather Condition Thread
1037         self.weather_worker_th = WeatherWorker(self.weather_label)
1038         self.weather_worker_th.start()
1039
1040         # Road Condition Thread
1041         self.road_worker_th = RoadWorker(self.road_label)
1042         self.road_worker_th.start()
1043
1044     def receiving(self):
1045         global webView
1046         global latitude
1047         global longitude
1048         global sender_latitude
1049         global sender_longitude
1050
1051         script = f"receiving({latitude},{longitude},{sender_latitude},{sender_longitude})"
1052         try:
1053             webView.page().runJavaScript(script)
1054         except:
1055             print(traceback.format_exc())

```

## ■ 1005~1056 번째 줄

```
# UI declaration
self.weather_label = QLabel()
self.weather_label.setAlignment(Qt.AlignCenter)
self.road_label = QLabel()
self.road_label.setAlignment(Qt.AlignCenter)
webView = QWebEngineView()
webView.load(QUrl.fromLocalFile(resource_path(HTML_FILE_PATH)))
```

## ■ UI 선언 코드

- WebView 변수를 사용하여 Tmap.html을 Receiver Window에 출력한다.  
HTML\_FILE\_PATH 경로 파일을 실행하며 './resource/Tmap.html'으로 설정되어있다.

```
# UI Arrangement
self.layout = QGridLayout()
self.layout.addWidget(webView, 0, 0, 2, 1)
self.layout.addWidget(self.weather_label, 0, 1, 1, 1)
self.layout.addWidget(self.road_label, 1, 1, 1, 1)

# Final UI Layout Arrangement
self.setLayout(self.layout)
self.setWindowFlag(Qt.FramelessWindowHint)
self.setFixedSize(NAVIGATION_WIN_SIZE_W, NAVIGATION_WIN_SIZE_H)
self.move(int(monitor_size_width/2), int(monitor_size_height/2) + BLANK_SPACE)
```

## ■ UI 레이아웃 배치 코드

```
self.timer = QTimer(self)
self.timer.timeout.connect(self.receiving)
self.timer.start(1000) # Cycle : 1 second
```

- 설정 주기마다 receiving함수를 실행하여 지도 화면 갱신
- Self.timer.start를 활용하여 ms 단위로 갱신주기 설정

```
# Weather Condition Thread
self.weather_worker_th = WeatherWorker(self.weather_label)
self.weather_worker_th.start()

# Road Condition Thread
self.road_worker_th = RoadWorker(self.road_label)
self.road_worker_th.start()
```

- 날씨정보 및 도로정보 획득을 위한 스레드 생성

```
def receiving(self):
    global webView
    global latitude
    global longitude
    global sender_latitude
    global sender_longitude

    script = f"receiving({latitude},{longitude},{sender_latitude},{sender_longitude})"
    try:
        webView.page().runJavaScript(script)
    except:
        print(traceback.format_exc())
```

- Tmap.html (지도)의 자바스크립트 함수 receiving을 실행하여 지도를 갱신
- receiving 함수는 지도 위치 변경 및 차량 위치 갱신 기능을 수행하며 Tmap.html의 receiving 함수 참조
- 파라미터: 송수신측 위도, 경도



## E. Tmap.html

### i. 역할

- Receive Window의 네비게이션을 웹으로 출력

### ii. 코드 설명

- <https://tmapapi.sktelecom.com/index.html> 에서 API 사용법 참조 가능.
- Tmapv2.LatLng: 위도, 경도를 담은 변수타입
- setCenter(LatLng): 위도, 경도 위치로 지도이동
- setPosition(LatLng): 위도, 경도 위치로 마커이동
- API KEY:

#### ■ 6~7번 줄

#### ■ Tmap 을 활용하기 위한 API KEY.

#### ■ 무료이용 횟수에 제한이 있으며 소진 시 자동 차단(해당 날짜)

#### ■ 교통정보: 1000건/일

#### ■ 지도보기: 100,000건/일

```
<script  
src="https://apis.openapi.sk.com/tmap/jsv2?version=1&appKey=f0sIyENUEf8Arejv1qGDU4p66e0sMRjB5kII22do"></script>
```

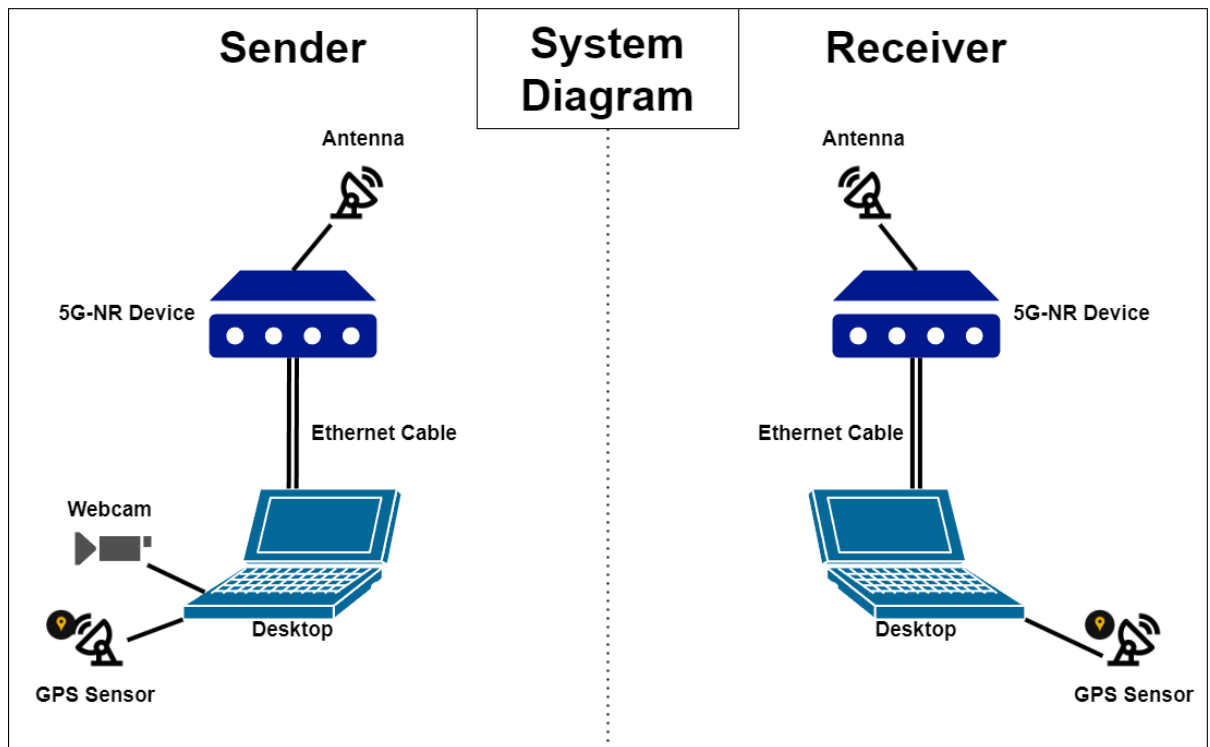
- initTmap
- 13~31번 줄
- Tmap 지도 생성
- map: 지도 instance
  - center: 초기 지도 위치
  - width: 출력 넓이
  - height: 출력 높이
  - zoom: 지도 확대 수준
- marker: 파란 차 instance
  - position: 차량 위치
- marker1: 빨간 차 instance
  - position: 차량 위치

```
function initTmap(){
    screen_height = (window.screen.availHeight/2.5) + "px"
    map = new Tmapv2.Map("map_div", {
        center : new Tmapv2.LatLng(37.56520450, 126.98702028),
        width : "100%",
        height : screen_height,
        zoom : 17
    });
    marker = new Tmapv2.Marker({
        position: new Tmapv2.LatLng(37.56520450, 126.98602028),
        icon: './blue_car.png',
        map: map
    });
    marker1 = new Tmapv2.Marker({
        position: new Tmapv2.LatLng(37.56520450, 126.98702028),
        icon: './red_car.png',
        map: map
    });
}
```

- receiving
- 34~47번줄
- 기능
  - 위도 경도 정보를 받아 지도 갱신
- Input:
  - Lat\_r: 수신측 위도
  - Lng\_r: 수신측 경도
  - Lat\_s: 송신측 위도
  - Lng\_s: 송신측 경도
- Output: 지도 정보 갱신

## 4. 성능 검증 툴 사용 설명서

### A. 성능 검증 툴 실행 전 기기 간 연결

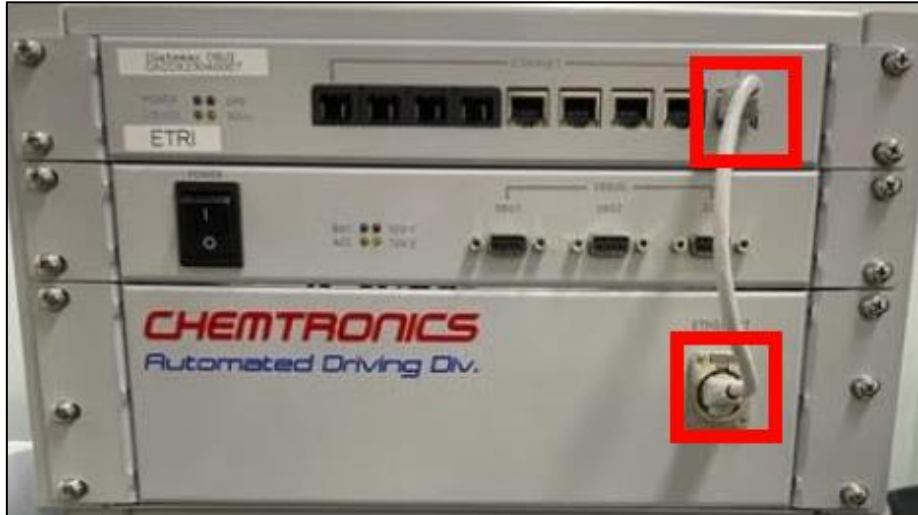


#### i. 5G-NR 장비 통신을 위한 세팅

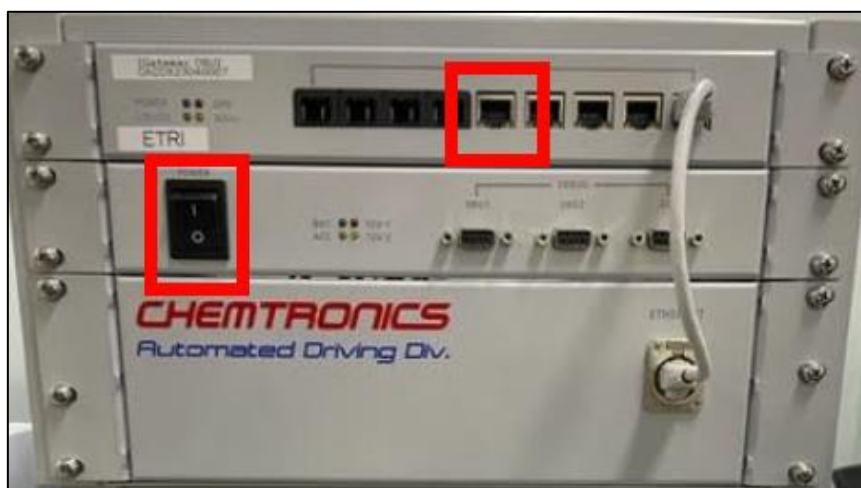
- 통신용 안테나와 5G-NR 장비를 3개의 포트에 연결



- 5G-NR 장비 앞면 지정된 2개의 이더넷 포트를 이더넷 케이블로 연결



- ii. 성능 검증 툴 실행 PC와 5G-NR 장비를 이더넷 케이블로 연결 및 전원 상태 확인
  - 'O': 전원 꺼짐 상태
  - '|': 전원 켜짐 상태



iii. 성능 검증 툴 실행 PC에 센서 연결

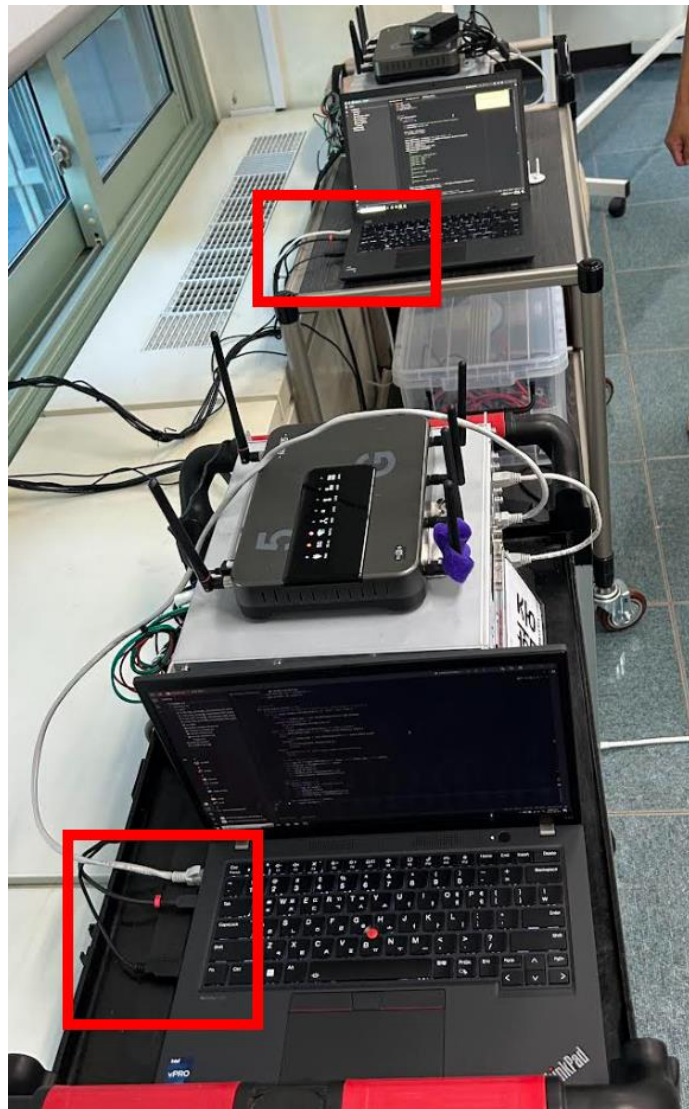
- Sender PC

■ 실시간 동영상 촬영이 가능한 Webcam

■ GPS 센서

- Receiver PC

■ GPS 센서



## B. Select Window

- i. '5GNR\_TOOL' 프로그램을 실행



- ii. 나온 화면에서 각 5G-NR 장비의 역할 버튼을 클릭

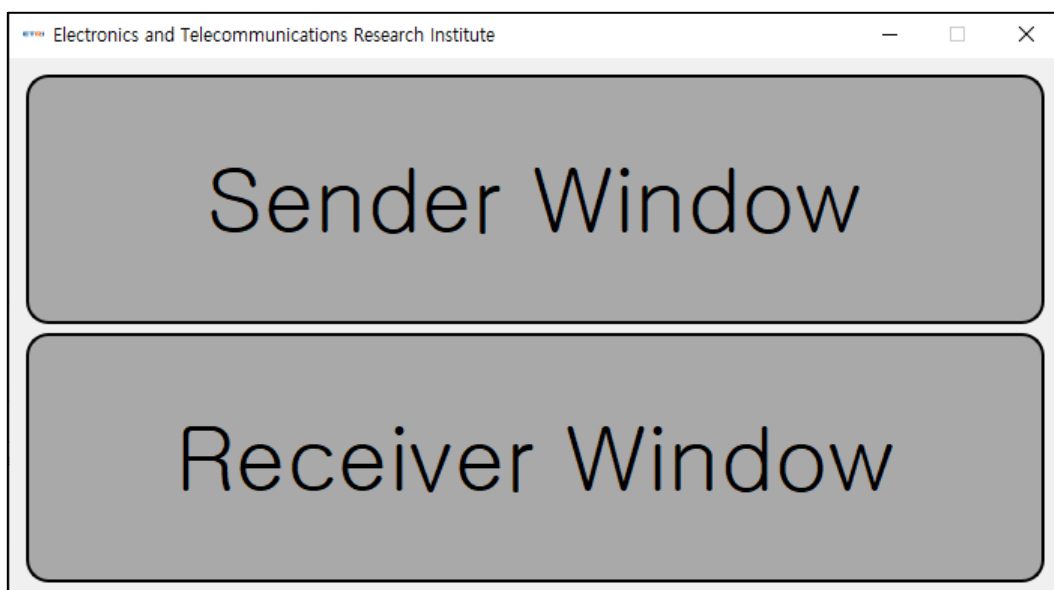
- Sender Window

- 성능 분석 결과 분석을 위한 데이터를 송신

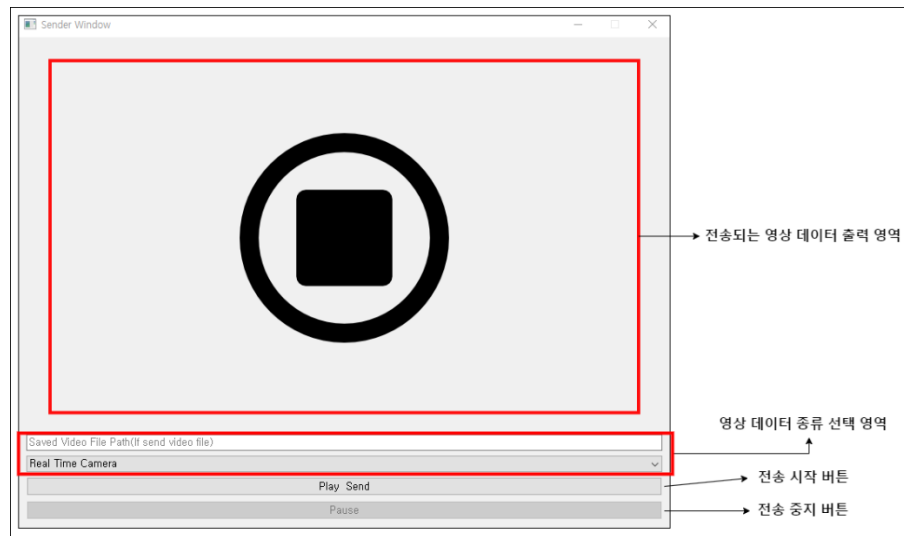
- Receiver Window

- 송신된 데이터를 수신

- 성능 분석 결과 출력



## C. Sender Window



### i. 역할

- 수신 역할 5G-NR 장비에 영상 데이터와 GPS 기반 현재 자신의 위치 정보를 전송
- 왕복 시간(RTT) 기반 지연시간(Latency) 측정을 위해 송신 역할 5G-NR 장비로부터 수신한 패킷에 답장

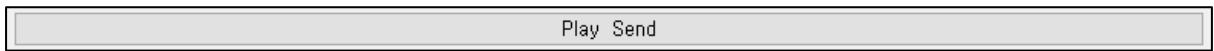
### ii. 실시간 영상 데이터(카메라) 송신 기능

- 외장 카메라 연결이 제대로 되어있는지 확인
- 선택 박스에서 'Real Time Camera'를 선택



- '**Play Send**' 버튼을 클릭하여 영상 데이터 송신 시작



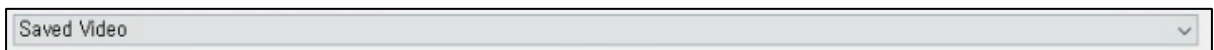


### iii. 저장된 영상 데이터 송신 기능

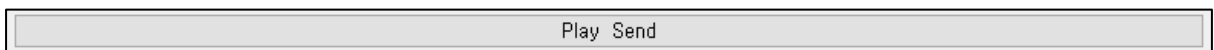
- 저장된 영상 데이터 경로를 입력 칸에 기입



- 선택 박스에서 '**Saved Video**'를 선택



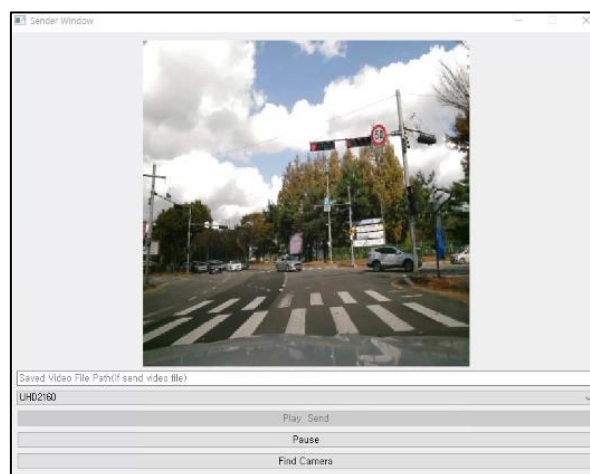
- '**Play Send**' 버튼을 클릭하여 영상 데이터 송신 시작



### iv. 영상 송신 중지

- '**Pause**' 버튼을 클릭하여 영상 데이터 송신 중지  
(재전송 시 저장된 영상 데이터 전송은 영상의 처음부터 송신)

### v. 실제 작동 모습 예시



## D. Receiver Window

### i. 역할

- 송신 역할 5G-NR 장비로부터 영상 데이터, GPS 기반 위치 정보, 왕복 시간(RTT) 기반 지연시간(Latency) 측정용 패킷 수신
- 수신 패킷들을 분석하여 네트워크 성능 지표들을 실시간 그래프로 화면에 표현  
(Packet Delivery Rate, Throughput, Latency, Distance)
- 실시간으로 수신하는 영상 데이터 화면에 재생
- TMAP API를 이용하여 송신 역할 5G-NR 장비 위치와 수신 역할 5G-NR 장비의 위치를 지도에 표시
- TMAP API를 이용하여 현재 자신의 위치 주변 도로 혼잡도 정보를 화면에 표시
- 공공 데이터 포털을 통한 현재 자신이 있는 지역의 날씨 정보(악조건)를 화면에 표시
- 수신 패킷 내용과 실시간 네트워크 성능 지표들 값 저장 및 저장 상황 로그 출력

### ii. 영상 데이터 수신 기능

- '**Receive**' 버튼을 클릭하여 영상 데이터 수신 시작

### iii. 영상 데이터 수신 정지

- 'Pause' 버튼을 클릭하여 영상 데이터 수신 중지  
(수신 정지 시 네트워크 성능 분석 기능도 정지)

### iv. 실제 작동 모습 예시

