SDR기반 송수신기 테스트베드 구현

2019203004 소프트웨어학부 박영서

1. 프로젝트 개요

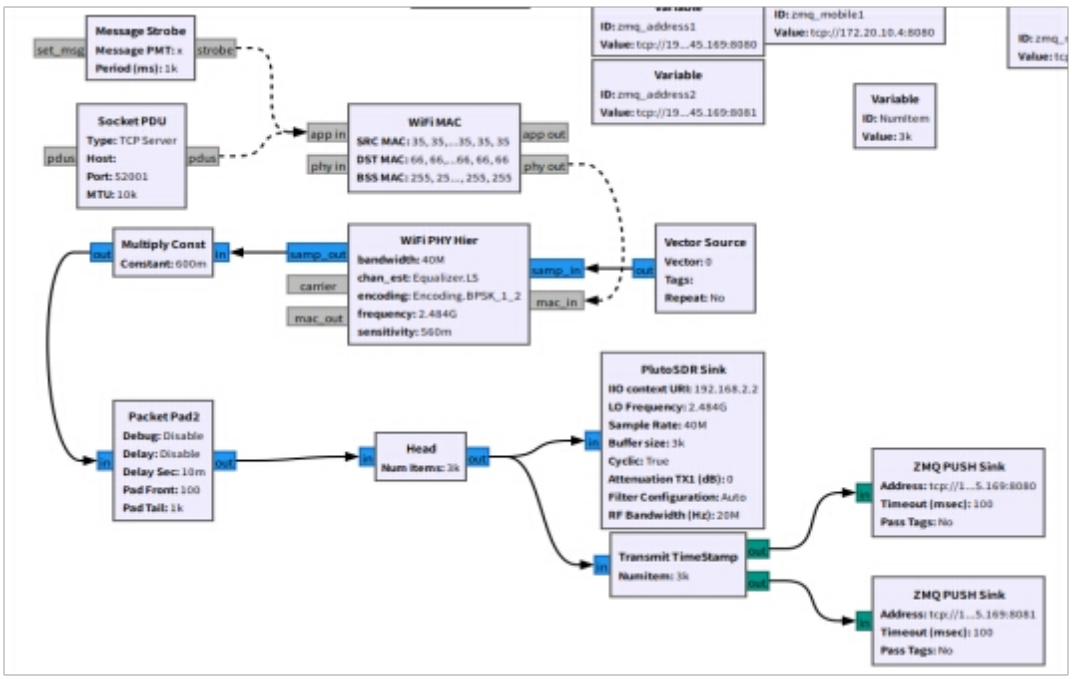
GNURadio를 통해 구현한 WiFi 송수신 소프트웨어를 Pluto SDR 2대를 사용하여 테스트베드를 만들고 수신된 신호를 분석하여 Throughput과 Packet Loss를 데이터셋으로 생성하는 프로젝트입니다.

FlowGraph구현, 테스트베드 설정, 데이터셋 분석, 그리고 느낀 점에 대해서 차례로 말씀드리겠습니다.

2. FlowGraph 구현

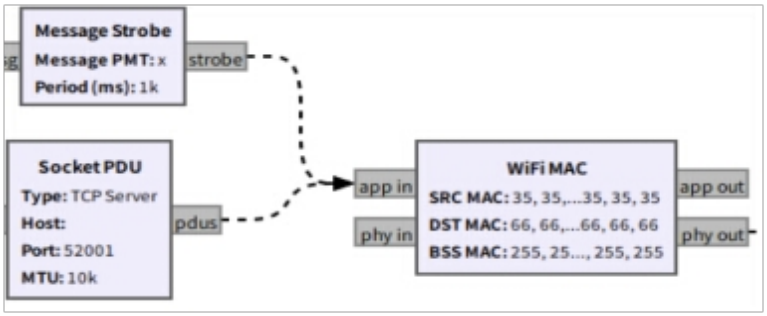
GNU Radio에서는 FlowGraph를 사용하여 SDR에 맞는 소프트웨어를 만들 수 있습니다. 송신단과 수신단을 각각 구현해야 되므로 두 개의 FlowGraph를 만들었습니다. FlowGraph의 경우 해당 오픈소스(https://github.com/bastibl/gr-ieee802-11)에서 example/wifi_tx.grc와 example/wifi_rx.grc를 사용하였습니다. 해당 FlowGraph에서는 IEEE802.11g 규격을 지원합니다.

2-1. 송신 FlowGraph



위 FlowGraph가 전체 송신 FlowGraph입니다. 와이파이에 담을 정보를 설정하고 와이파이 신호를 만들어서 PlutoSDR이 송신하도록 합니다. Transmit TimeStamp라는 커스텀 파이썬 블록을 사용하여 실제 보낸 데이터의 개수와 신호를 송신한 시간 측정하여 ZMQ(Zero Messag Queue)라는 비동기 메시징 라이브러리를 사용하여 수신단에 보내도록 하였습니다. 원래는 PlutoSDR의 와이파이 패킷에 함께 담아서 보내려고 하였으나 지원하지 않는 관계로 ZMQ 라이브러리를 사용하였습니다.

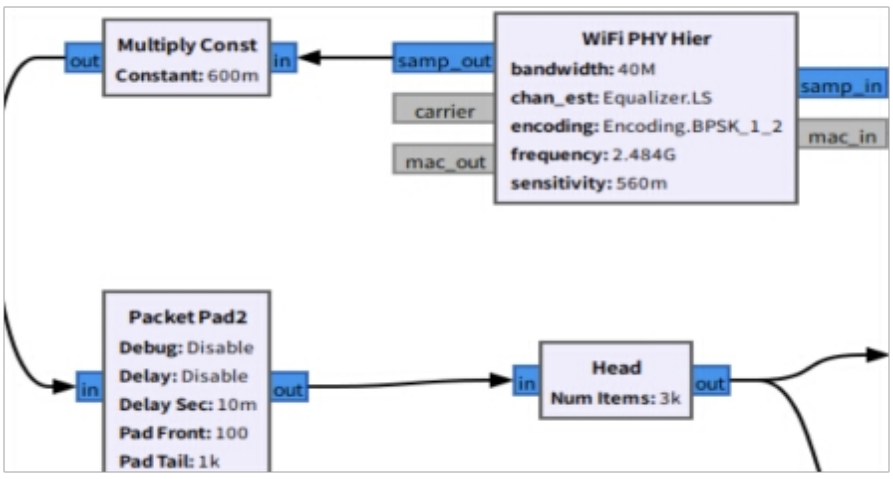
각 블록별로 나누어서 설명하도록 하겠습니다.



Message Strobe : 와이파이에 어떤 메시지를 몇 밀리초마다 메시지를 생성할지 정하는 블록입니다. 저는 여기서 메시지를 x로 설정하고 초당 1초(1000밀리초)마다 하나씩 메시지를 생성하도록 설정하였습니다.

Socket PDU : 메시지가 담긴 와이파이를 실제 전송할 때 사용할 Port 번호와 IP주소를 할당하는 블록입니다. TCP방식으로 Port 번호 52001로 설정하였습니다. IP는 연결된 PlutoSDR이 하나밖에 없으므로 Host란은 비워두었습니다.

WiFi MAC : 해당 블록은 메시지를 WiFi 표준에 맞게 MAC Frame을 생성하는 역할을 합니다.

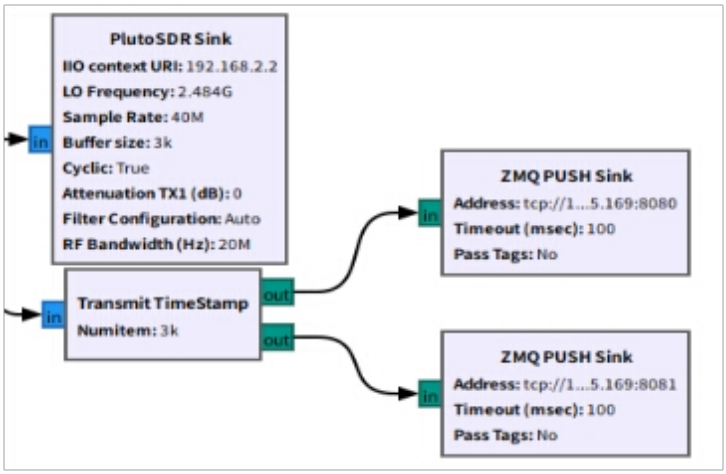


WiFi PHY Hier : 물리계층을 처리하는 블록입니다. MAC Frame을 기반으로 실제 물리적 신호를 생성합니다. 데이터의 균일성, 인코딩 등의 수학적, 물리적 작업을 진행합니다. 속성값들은 기본적으로 제공되는 값들로 진행하였습니다.

Multiply Const : 상수값을 곱합니다. 신호의 세기를 조정합니다. 기본값(0.6)을 사용하였습니다.

Packet Pad2 : 패킷의 크기를 일정하게 유지하도록 조정하여 송신의 안정성을 높입니다.

Head : 송신할 데이터의 개수를 설정합니다. 원래는 1000개로 설정하여 테스트할 예정이었지만, 송신이 잘 진행되지 않아서 반복적인 테스트를 통해 3000개로 데이터의 개수를 설정하였습니다.



PlutoSDR Sink : 실제 PlutoSDR을 통하여 와이파이를 송신합니다. Sample Rate의 경우 샤논-나이퀴스트 정리에 따라 주파수 대역폭의 두 배로 설정하였습니다. Buffer Size는 바로 전의 Head블록에서 설정한 데이터의 개수와 똑같은 3000개로 설정하였습니다. Atteunuation TX1은 신호 전송시 신호의 세기를 몇 데시벨만큼 줄여서 전송할지를 정하는 매개변수인데, 제가 진행할 프로젝트에서는 굳이 신호의 세기를 줄일 필요가 없어서 0 dB로 설정하였습니다. 또한 IEEE802.11g 규격을 사용하므로 RF Bandwidth를 20Mhz로 설정하였습니다.

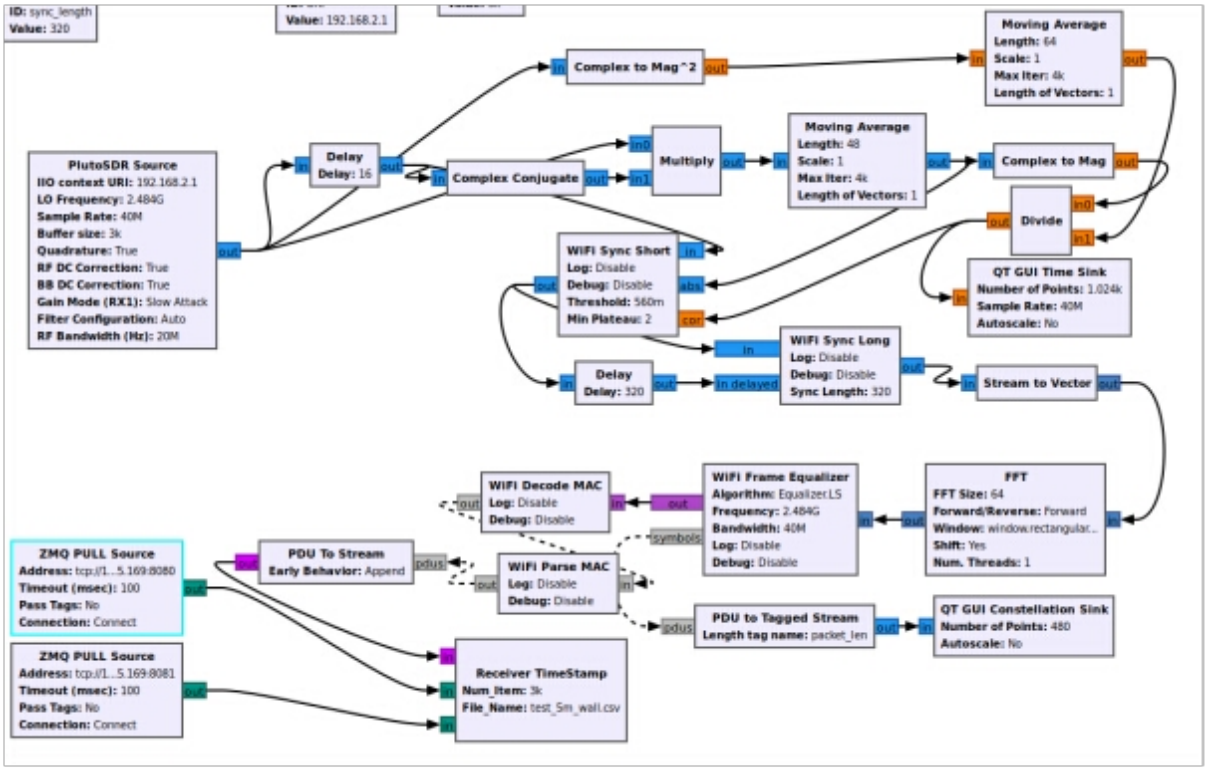
Transmit TimeStamp : 파이썬 커스텀블록을 사용하여 직접 만든 블록으로 ZMQ에 전달할 데이터 송신 시간과 데이터 송신 개수를 구하는 역할을 담당합니다. 아래는 해당 블록에서 구현한 코드입니다.

```
def work(self, input_items, output_items):
    if self.start_time is None:
        self.start_time = time.time()

    if self.count < self.num_item:
        self.count += 1
        #output = np.array(output, dtype=np.float64) float64 is not supported so only 64 allowed, also at receiver part

        fractional, integer = np.modf(self.start_time)

        output_items[0][0] = int(integer)
        output_items[1][0] = self.count
        # output[0] += np.round(self.start_time%100.0, 7)
        # output[0] = output[0] + self.start_time/100 + (self.start_time - int(self.start_time))

        # print(f"{self.count}\n")

        if self.count == self.num_item:
            print(f"\n current time : {output_items[0][0]}, count : {output_items[1][0]} Transmition Completed\n")


    return 2
```

전송할 데이터가 들어오면 들어온 시간을 측정하고 개수를 세어서 각각 ZMQ블록으로 전달합니다.

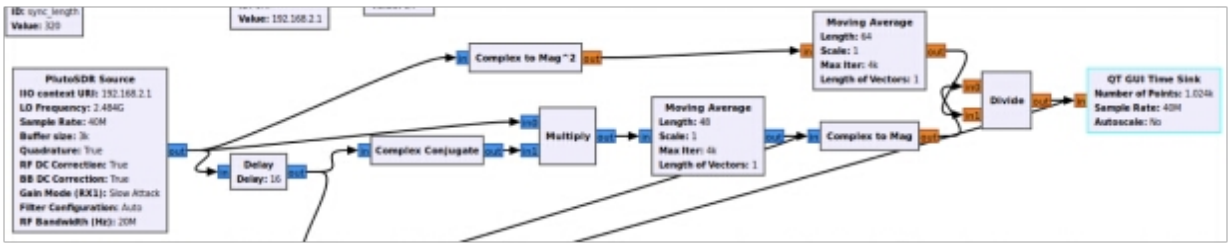
ZMQ PUSH SINK : Zero Message Queue라는 TCP방식의 비동기 메시징 블록입니다. 해당 블록을 두 개 사용하여 각각 데이터 송신 시간과 보낸 데이터의 개수를 전송합니다. 해당 데이터들을 수신단에서 받으면 Throughput과 Packet Loss을 구할 때 활용될 것입니다. 원래는 Message Strobe에서 한꺼번에 메시지와 묶어서 보낼 계획이었으나, Message Strobe블록 내에서 직접적으로 파이썬 코드의 접근을 제한하여서 ZMQ 메시징 블록을 사용하였습니다. 위 두 블록은 수신단에서 ZMQ PULL SOURCE와 연동됩니다.

2-2. 수신 FlowGraph



전체 수신 FlowGraph입니다. 송신단에서 보낸 와이파이 신호를 받아서 받은 신호의 내용을 해석합니다. 또한 ZMQ PULL Source에서도 송신단에서 보낸 데이터들을 받아서 이를 받은 신호와 함께 Receiver TimeStamp라는 커스텀 파이썬 블록에서 Throughput과 Packet Loss를 구합니다. 구한 Throughput과 Packet Loss는 CSV파일에 저장하여 데이터셋으로 만듭니다.

각 블록별로 나누어서 설명드리겠습니다.



PlutoSDR Source : 컴퓨터에 연결된 PlutoSDR이 송신단에서 보낸 와이파이 신호를 수신하여 전기적 신호로 바꾸도록 합니다. 해당 블록 또한 기본 설정을 유지하되, Buffer Size와 RF Bandwidth는 송신단의 PlutoSDR Sink와 같도록 하였습니다.

Complex Conjugate : PlutoSDR Source에서 받은 전기적 신호를 시간적 변화로 표현하게 되면 복소수로 표현할 수 있는데, 해당 복소수의 켤레를 구하는데 사용합니다.

Multiply : 말그대로 곱셈을 하는 블록입니다. 복소수와 복소수 켤레를 곱하여 허수 부분을 제거하고 실수 부분만을 남깁니다.

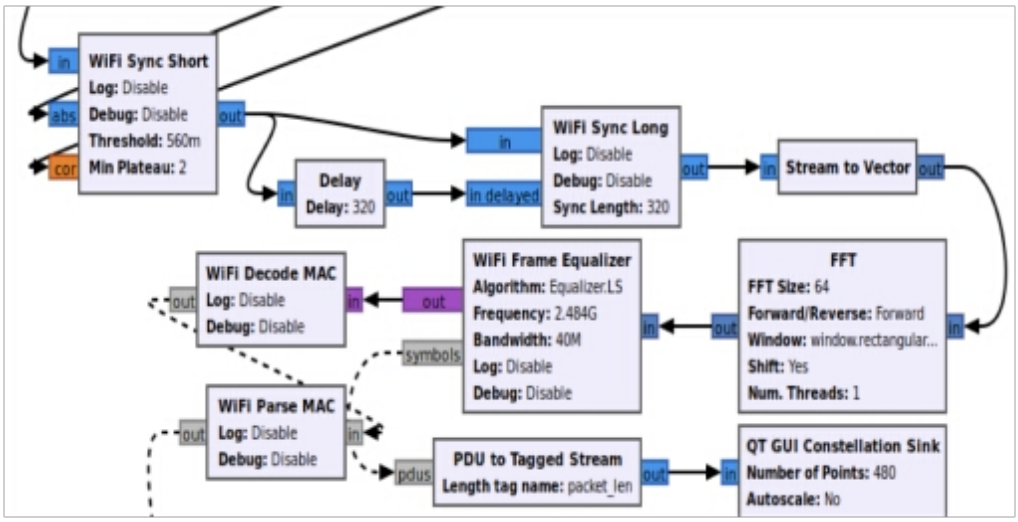
Moving Average : Multiply블록에서 구한 실수값들을 평균을 냅니다.

Complex to Mag : 복소수의 크기를 구합니다.

Complex to Mag^2 : 실수와 허수 부분을 각각 제곱하여 더한 값으로 신호의 세기를 나타냅니다.

Divide : 신호의 상대적인 세기를 구합니다.

QT GUI Time Sink : 이렇게 구해낸 신호의 상대적인 세기를 시간의 변화에 따라 GUI에 띄웁니다.



WiFi Sync Short / WiFi Sync Long : 둘 다 수신한 와이파이 신호를 동기화하는데 사용합니다. 두 번의 동기화를 거쳐 신호를 온전하게 복원합니다.

Stream to Vector : 신호를 벡터화합니다.

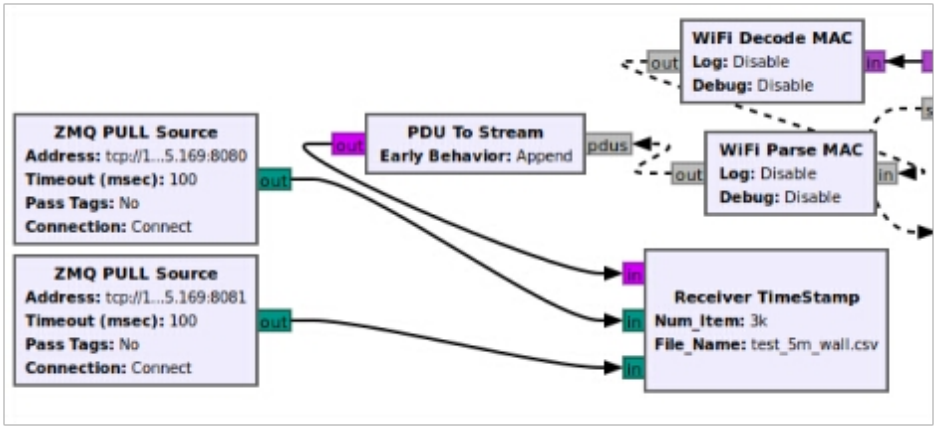
FFT : 고속 푸리에 변환을 사용합니다. 고속 푸리에 변환은 푸리에 변환을 쿨리-튜키 알고리즘 기법을 사용하여 보다 빠른 성능의 푸리에 변환을 처리합니다. 푸리에 변환을 통해 신호를 분리 및 추출합니다.

WiFi Frame Equalizer : 고속 푸리에 변환으로 추출된 신호들을 복원하고 왜곡된 신호를 보정하는 처리 과정을 거칩니다. 그리고 기존에 수신하기로 했던 신호만을 추출하여 다음 블록에 전달합니다. 디지털 신호 처리에서는 샘플링율이 실제 대역폭을 결정하므로 Bandwidth는 PlutoSDR Source에서 Sample Rat와 같은 값인 40M로 설정했습니다.

PDU to Tagged Stream / QT GUI Constellation Sink : 알맞게 추출한 와이파이 신호를 실수형으로 변환하여 GUI에 띄웁니다.

WiFi Decode MAC : 추출한 전기적 신호를 MAC Frame으로 변환합니다. 이 때부터 와이파이의 내용을 읽어들일 수 있습니다.

WiFi Parse MAC : MAC Frame에서 실질적인 정보가 들어있는 payload를 추출합니다.



ZMQ PULL Source : 송신단의 ZMQ PUSH Sink에서 보낸 두 데이터(송신 시각/보낸 데이터의 개수)를수신합니다.

Receiver TimeStame : 직접 구현한 파이썬 커스텀 블록입니다. 해당 블록에서는 최종적으로 받아낸 payload와 송신 시각, 보낸 데이터 개수를 활용하여 Throughput과 Packet Loss를 계산하여 최종적으로 CSV파일에 데이터셋 형태로 저장하게 됩니다. 아래는 해당 블록에 구현된 코드입니다.(코드수정)

```
        self.start_time = None
        self.end_time = None
        self.data_size = 0.0
        self.count = 0
        self.last_index = 0
        self.num_item = num_item
        self.throughput = 0.0
        self.ber = 0.0
        self.packet_loss = 0.0
        self.path = file_name
        self.header = ["Throughput (bit per sec)", "Bit Error Rate (%)", "Packet Loss (%)"]

    def work(self, input_items, output_items):
        if len(input_items[0]) != 0 & input_items[2][0] <= self.num_item:
            self.last_index = input_items[2][0]

            if self.last_index != 0:
                millsec, sec = np.modf(time.time())
                self.start_time = input_items[1][0]
                self.end_time = int(sec)
                self.data_size += len(input_items[0])
                self.count += 1
                self.packet_loss = (1 - (self.count / self.num_item)) * 100
                self.ber = (10 ** input_items[3][0]) * 100
                self.throughput = self.data_size/(self.end_time - self.start_time)

                # [Throughput(bps), Bit Error Rate(%), Packet Loss(%)]
                data = [self.throughput, self.ber, self.packet_loss]
                # text = f"Throughput : {self.throughput} bps, Bit Error Rate : {self.ber} %
                # print(f"\n {text} \n")

                file_exists = os.path.isfile(self.path)
                lines = []
                if file_exists:
                    with open(self.path, mode='r', newline='', encoding='utf-8') as file:
                        reader = list(csv.reader(file))
                        rows = sum(1 for row in reader)

                        #if rows == 2:
                        #    lines.append(reader[1])
                        if rows >= 2 and self.count == 1:
                            for r in reader[1:]:
                                lines.append(r)
                        elif rows > 2:
                            for r in reader[1:-1]:
                                lines.append(r)

                lines.append(data)

                with open(self.path, mode="w", newline='',encoding="utf-8") as file:
                    writer = csv.writer(file)
                    writer.writerow(self.header)
                    writer.writerows(lines)
```

조건문을 통해 받아온 데이터들에 이상이 없는지 검사한 뒤에 받은 시간을 측정합니다. 빈 객체가 아닌지, 정해진 데이터의 개수를 초과해서 받지는 않았는지를 검사합니다. 그 후, Throughput과 Packet Loss에 대해서 각각 계산합니다.

Throughput = (통신 시간 동안 받은 데이터의 크기) / (통신 시간) 이므로

Throughput = self.data_size / (self.end_time - self.start_time) 으로 설정하였습니다.

Packet Loss = (전체 패킷의 수 - 성공적으로 받은 패킷의 수) / (전체 패킷의 수) 이므로

Packet Loss = (1 - (self.count / self.num_item)) * 100 으로 백분율로 설정하였습니다.

이렇게 만들어진 데이터들을 CSV파일로 만들어서 한 실험 당 한 줄이 써지도록 설정하였습니다.

3. 테스트베드

테스트베드는 우분투 리눅스 환경의 노트북 2대와 각각에 연결할 PlutoSDR 2대, 그리고 송수신용 안테나 2대를 사용하였습니다.

테스트는 실내환경에서 이루어졌으며, 장애물이 없는 거리 내에서의 1m, 5m, 10m와 송수신기 사이에 60cm의 벽을 두고 1m, 5m 거리에서 10번씩 3000개의 데이터를 송수신하는 테스트를 진행하여 측정한 데이터셋을 확보하였습니다.

테스트를 진행할 때 통신 범위를 설정하기 위해 **ITU Indoor Path Loss Model**을 참조하였습니다.

해당 공식은 실내 환경에서 신호의 손실을 측정하기 위해 사용됩니다. 아래 공식을 사용하여 몇m까지 통신 가능한지 계산하였습니다.

$$L = 20 \log_{10} f + N \log_{10} d + P_f(n) - 28$$

일반적인 실내환경에서 경로 손실 지수 N은 30으로 가정합니다. 주파수 f의 경우 MHz 단위를 사용하므로 2.4GHz = 2.4 x 1024 MHz로 치환하여 계산하였고, Pf(n)은 다른 건물 층 사이에서 통신할 때 사용하므로 해당 테스트베드에서는 0으로 됩니다. Pluto SDR의 최대 74.5 db까지 신호의 세기를 조절할 수 있으므로 정리하게 되면

$$74.5 = 20\log(2.4 \times 1024) + 30\log(d) - 28$$

$$d = 14.3326292792$$

즉, 이론상 최대 약 14미터까지만 통신이 가능하므로, 테스트베드를 5m단위로 해서 10m까지만 측정하는 것으로 설정하였습니다.

벽을 두고 하는 테스트의 경우 Multi-Wall Path Loss Model을 사용하였습니다.

$$L = PL0 + n\log(10 \times d) + Lwall$$

PL0는 기존 실내에서의 손실이므로 위의 ITU모델을 사용하고, 경로 손실 지수 n의 경우 마찬가지로 벽을 제외하고 위의 경우와 똑같은 환경이므로 30으로 가정합니다. 또한 Lwall의 경우 벽에 의한 직접적인 손실인데 일반적으로 벽의 길이가 20cm이상이면 20db로 설정하므로

$$74.5 = 20\log(2.4 \times 1024) + 30\log(10 \times d^2) - 8$$

$$d = 0.47$$

즉, 약 0.5m까지 통신이 가능할 것으로 예상되나 테스트를 5m까지는 진행할 예정입니다.

4. 데이터셋과 실험 결과

Throughput (bit per sec)	Packet Loss (%)
300.1	0
300.1	0
333.333333333333	0
300	0
300	0
300	0
299.9	0.0333333333333297
300	0
333.222222222222	0.0333333333333297
333.444444444444	0

1m 거리에서 장애물없이 진행한 데이터셋 : Throughput은 대부분 300 bit/sec 이상의 값이 나왔으며 Packet Loss 또한 대부분 0%에 가까웠습니다. Throughput이 일정한 값으로 나왔고, Packet Loss도 1% 미만으로 나와서 안정적인 네트워크 상태를 보여주었습니다.

Throughput (bit per sec)	Packet Loss (%)
300.1	0
300.1	0
333.333333333333	0
300	0
300	0
300	0
299.9	0.0333333333333297
300	0
333.222222222222	0.0333333333333297
333.444444444444	0

5m 거리에서 장애물없이 진행한 데이터셋 : 이 데이터셋 또한 마찬가지로 1m 거리에서의 데이터셋과 비슷한 네트워크 성능 지표를 보여주었습니다. Throughput이 꾸준하게 나왔고, Packet Loss 또한 1% 미만으로 안정적인 네트워크 상태를 보여주었습니다.

Throughput (bit per sec)	Packet Loss (%)
23.8035714285714	55.5666666666667
60	0
111.111111111111	0
115.384615384615	0
69.7674418604651	0
103.448275862069	0
111.111111111111	0
88.2352941176471	0
71.3809523809524	0.33333333333333
68	0.33333333333333

10m 거리에서 장애물없이 진행한 데이터셋 : 기존의 1m, 5m내에서 진행하였던 테스트와 달리 Throughput만을 보면 많이 성능이 낮아진 것을 알 수 있습니다. Packet Loss의 경우 첫 번째 테스트만을 제외하면 나머지 테스트에서는 안정적인 네트워크 상태를 보여줬다고 할 수 있으나, 성능적인 측면에서는 크게 감소한 것을 알 수 있었습니다.

Throughput (bit per sec)	Packet Loss (%)
146.47619047619	2.76666666666666
60.3333333333333	16
157.842105263158	1.16666666666667
13.0434782608696	0
300	0
300	0
694.2	43.7
937.75	43.6333333333333
300	0
37.9746835443038	0

1m 거리에서 벽을 사이에 두고 진행한 데이터셋 : 이번 데이터셋은 벽을 두고 송수신 테스트를 진행한 데이터셋입니다. 확실히 장애물없이 테스트한 1m와는 성능이 전체적으로 줄어들었습니다. Throughput의 경우 안정적이지 못하고 Packet Loss 또한 벽없이 실험을 진행한 것과 많이 대조됩니다.

Throughput (bit per sec)	Packet Loss (%)
2.86666666666667	91.5
5.82954545454545	82.9
0.0972222222222222	99.7666666666667
0.808510638297872	97.4666666666667
0.273684210526316	99.1333333333333
0.160714285714286	99.7
1.11111111111111	98
0.28	99.5333333333333
1.38709677419355	98.5666666666667
0.4	99.8
1.45714285714286	98.3

5m 거리에서 벽을 사이에 두고 진행한 데이터셋 : 벽없이 5m거리에서 송수신한 데이터셋과 비교했을 때 많이 성능 차이가 나는 것으로 나타납니다. Muti Wall Path Loss Model을 사용하였을 때 1m의 거리에서도 잡히지 않았을 것이라는 예상과 달리 5m내에서도 통신이 되긴 했습니다. 그러나 전체적으로 매우 불안정한 모습을 보였습니다. Throughput의 경우 0~2 bit/sec의 매우 낮은 수치를 보여주였고, Packet Loss 또한 90%의 수치를 보이면서 안정적으로 통신하는 것이 어려운 것을 보여준 데이터셋이었습니다.

종합하자면, PlutoSDR을 사용하여 송수신을 테스트할 경우에, 장애물이 없는 경우에는 10m까지는 원활한 통신이 가능했으나, Throughput이 다소 떨어지는 모습을 보였습니다. 벽을 두고 통신한 경우에는 이론을 통해 예측했을 때보다는 통신 가능범위가 넓었으나, 5m에서는 통신 상태가 급격하게 불량해지는 것으로 볼 때, 신호가 왜곡되는 현상이 벽과 같은 장애물에 의해 많은 영향을 받는다는 것을 보여주었습니다.

5. 아쉬웠던 점 및 개선해야할 점

FlowGraph를 구현할 때 어려운 점이 많았지만 기존의 Bit Error Rate을 구해내지 못했던 것이 굉장히 아쉬웠습니다. 디버깅이나 로그에서 나온 payload자체에는 문제가 없었지만 Bit끼리 비교를 하면 문제가 발생하는 것이 큰 어려움이었습니다. Bit Error Rate을 제대로 구해내기 위해서 여러 가지 많은 블록을 사용해보고, WireShark와 같은 외부 프로그램과 연결하여 패킷을 분석하는 등 많은 시간과 노력을 하였음에도 아쉬웠습니다. 개선을 하고자 한다면, 더 좋은 장비들을 사용하여 Bit Error Rate을 구해서 테스트베드의 신뢰성을 높이고 싶습니다.