**코드 정리**

**Load\_data**

pcap파일 디렉토리와 파일 명을 받아와서 데이터 프레임 형태로 반환

file\_dir=**'./all\_packet**\\**'**file\_list= os.listdir(file\_dir)

def load\_data(file\_dir, filename):  
  
 filename = file\_dir + filename  
  
 time = []  
 div\_time = [0]  
 size = []  
  
 idx = 0  
 for ts, pkt in dpkt.pcap.Reader(open(filename, **'rb'**)):  
 if idx == 0: tmp\_time = ts  
  
 eth = dpkt.ethernet.Ethernet(pkt)  
 if eth.type != dpkt.ethernet.ETH\_TYPE\_IP:  
 continue  
 idx += 1  
 ip = eth.data  
 time.append(round(ts - tmp\_time, 4))  
 size.append(len(pkt))  
  
 for i in range(1, len(time)):  
 div\_time.append(abs(round(time[i] - time[i - 1], 4)))  
  
 df = pd.DataFrame({**'time'**: div\_time, **'size'**: size})  
  
 return df

**MA\_filter**

Moving Average filter를 적용하여 데이터 프레임 반환

def MA\_filter(df):  
 df[**'time'**]= df[**'time'**].rolling(100).mean()  
 df[**'size'**]= df[**'size'**].rolling(100).mean()  
  
 return df

**Std\_scaler**

스케일링을 한 데이터 프레임 반환

def std\_scaler(data):  
 std\_data= (data- min(data))/(max(data)-min(data))  
 return std\_data

**Plot\_data**

데이터 프레임을 subplot으로 time, size 순으로 plot / 주석 친 부분은 Thresholding 값을 확인할 때 사용이 됨

def plot\_data(df):  
 global filename  
 global tmp  
 x = np.linspace(0, df[**'time'**].shape[0], df[**'time'**].shape[0])  
 plt.subplot(2, 1, 1)  
 plt.plot(x, df[**'time'**])  
  
 *# plt.scatter(tmp, df['time'][tmp], s= 100, c='blue')  
 # plt.scatter(int(df['time'].idxmax()\*0.8), df['time'][int(df['time'].idxmax()\*0.8)], s= 50, c='blue')  
 # plt.scatter(df['time'].idxmax(), df['time'][df['time'].idxmax()], s= 50, c='red')* plt.subplot(2, 1, 2)  
 x = np.linspace(0, df[**'size'**].shape[0], df[**'size'**].shape[0])  
 plt.xlabel(filename)  
 plt.plot(x, df[**'size'**])  
 plt.show()

**Calculate\_normal**

각 파일에서의 최대치를 갖는 값의 index를 가져와 그 전 80%의 데이터의 평균과 표준 편차의 값을 구해서 반환

def calculate\_normal():  
  
 idx= 0  
 normal\_mean= []  
 normal\_std= []  
 for filename in file\_list:  
 df = load\_data\_threshold\_cnt(file\_dir, filename, idx)  
 df[**'time'**].fillna(method=**'bfill'**)avg= df[**'time'**].iloc[:int(df[**'time'**].idxmax()\*0.8)].mean()  
 std= df[**'time'**].iloc[:int(df[**'time'**].idxmax()\*0.8)].std()  
 normal\_mean.append(avg)  
 normal\_std.append(std)  
 max\_time.append(df[**'time'**].max())plot\_data(df)  
 mean\_val= round((sum(normal\_mean)/len(normal\_mean)), 4)  
 std\_val= round((sum(normal\_std)/len(normal\_std)), 4)  
 print(**"mean val:"**, mean\_val, **"std\_val"**, std\_val)  
 return mean\_val, std\_val

**Load\_data\_threshold\_cnt**

+-100개의 값을 기준으로 정확도를 측정하여 반환

def load\_data\_threshold\_cnt(file\_dir, filename, x\_idx):  
 global acc  
 global x\_threshold  
 global tmp  
 filename = file\_dir + filename  
  
 time = []  
 div\_time = [0]  
 size = []  
  
 idx = 0  
 for ts, pkt in dpkt.pcap.Reader(open(filename, **'rb'**)):  
 if idx == 0: tmp\_time = ts  
  
 eth = dpkt.ethernet.Ethernet(pkt)  
 if eth.type != dpkt.ethernet.ETH\_TYPE\_IP:  
 continue  
 idx += 1  
 ip = eth.data  
 time.append(round(ts - tmp\_time, 4))  
 size.append(len(pkt))  
  
 for i in range(1, len(time)):  
 div\_time.append(abs(round(time[i] - time[i - 1], 4)))  
  
 df= pd.DataFrame({**'time'**: div\_time, **'size'**: size})  
  
 tmp = 0  
 for i, item in enumerate(df[**'time'**]):  
 if item > x\_threshold[x\_idx]:  
 tmp= i; break;  
 *# print(tmp)* if tmp-100 <=df[**'time'**].idxmax()<= tmp+100:  
 acc[x\_idx]+=1;  
  
 return df

**Load\_data\_time**

Time\_interval 기준으로 정확도 측정하여 반환

def load\_data\_time(file\_dir, filename, x\_idx):  
 global acc  
 global x\_threshold  
 global tmp  
 global tmp\_min\_idx  
 global tmp\_max\_idx  
 filename = file\_dir + filename  
  
 time = []  
 div\_time = [0]  
 size = []  
  
 idx = 0  
 for ts, pkt in dpkt.pcap.Reader(open(filename, **'rb'**)):  
 if idx == 0: tmp\_time = ts  
  
 eth = dpkt.ethernet.Ethernet(pkt)  
 if eth.type != dpkt.ethernet.ETH\_TYPE\_IP:  
 continue  
 idx += 1  
 ip = eth.data  
 time.append(round(ts - tmp\_time, 4))  
 size.append(len(pkt))  
  
 for i in range(1, len(time)):  
 div\_time.append(abs(round(time[i] - time[i - 1], 4)))  
  
 df= pd.DataFrame({**'time'**: div\_time, **'size'**: size})  
 time\_max\_idx= df[**'time'**].idxmax()  
  
 *# tmp에서 threshold를 넘는 최초의 값의 index를 저장* tmp = 0  
 for i, item in enumerate(df[**'time'**]):  
 if item > x\_threshold[x\_idx]:  
 tmp= i; break;  
  
 tmp\_max\_idx= len(df[**'time'**])  
 tmp\_min\_idx= 0  
  
 sum\_time = 0  
 for i in range(tmp + 1, len(df[**'time'**])):  
 sum\_time += df[**'time'**][i]  
if sum\_time > time\_interval:  
 tmp\_max\_idx= i  
break;  
  
 sum\_time = 0  
 for i in range(tmp-1, 0, -1):  
 sum\_time+= df[**'time'**][i]  
if sum\_time > time\_interval:  
 tmp\_min\_idx = i  
break;  
if tmp\_min\_idx<= time\_max\_idx<= tmp\_max\_idx:  
 acc[x\_idx]+=1;  
 avg\_cnt.append(tmp\_max\_idx-tmp\_min\_idx)  
return df