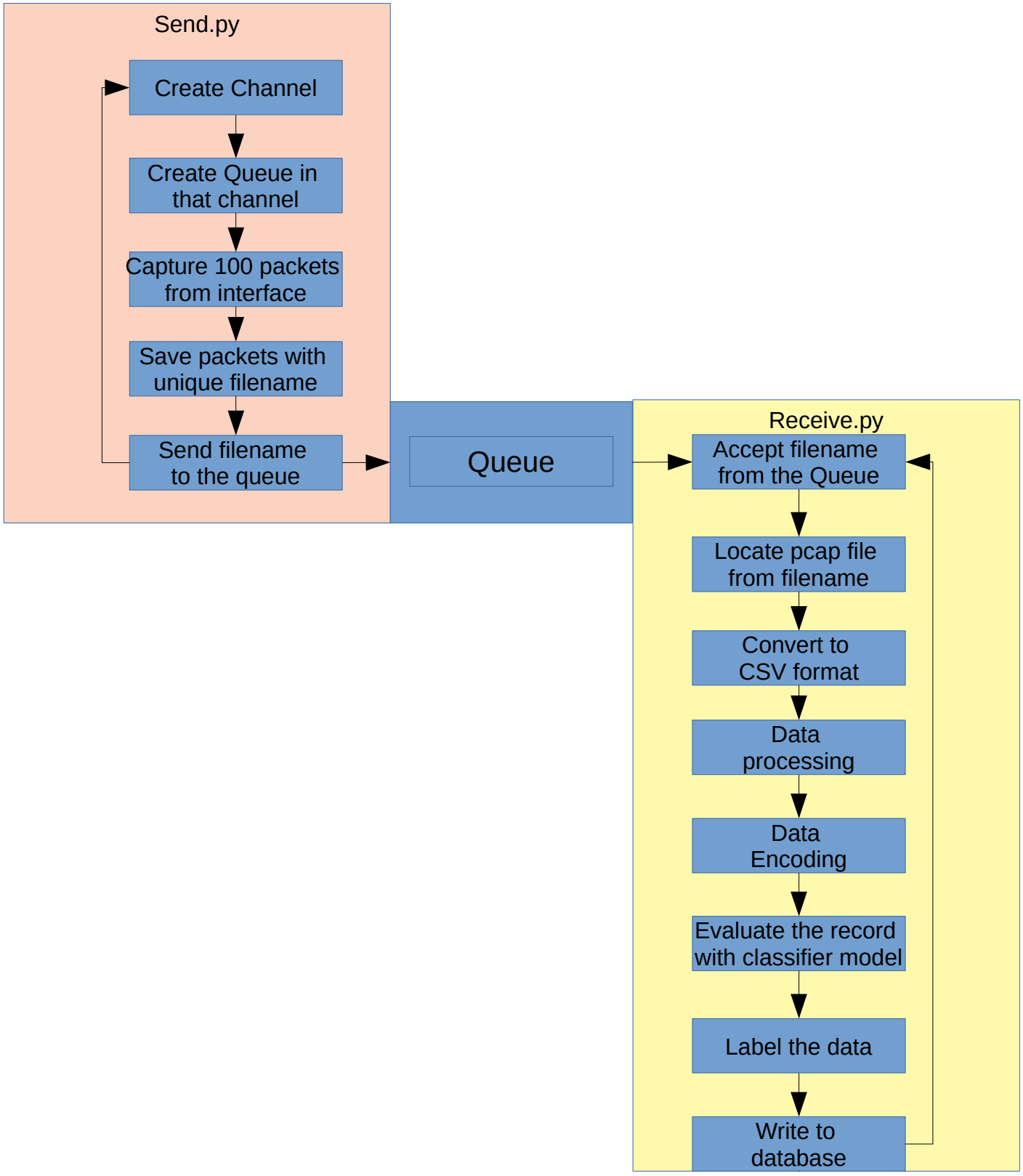
System Block Diagram of Near Real Time Intrusion Detection System



code for send.py

Importing relevant package inside send.py script

```
import pika
import sys
import os
import time
import pyshark


```

This class helps us to select an interface(wifi or ethernet) from which the packets are to be captured

```
class PcapCapture:
    def __init__(self,interface_name,output_file):
        self.capture = None
        self: PcapCapture terface_name
        self.output_file = output_file

    def start(self,packet_count=100):
        print("*** starting packet capture *** ")
        self.capture = pyshark.LiveCapture(interface=self.interface,output_file=self.output_file)
        self.capture.sniff(packet_count=packet_count)

    def stop(self):
        print("*** stopping packet capture *** ")
        self.capture.close()

```

pika is used to create connection and provide channel for creating exchange queues. In this code we created an connection ato create channel inside the local host. Then we declared an exchange for sending captured packet’s filename.Inside the infinte loop the program captures 100 packets from wlp2s0 interface and sends it to exchange using the channel. The process works on loop continuously sending the packets filename

```

if __name__ == "__main__":

    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()
    channel.exchange_declare(exchange='logs',exchange_type='fanout')

    while True:
        # check starting time
        start = time.process_time()
        timestr = time.strftime("%Y%m%d-%H%M%S")

        # sniff live packet
        filename = f"dump-{timestr}.pcap"
        cap = PcapCapture(interface_name='wlp2s0',output_file=filename)
        cap.start()
        cap.stop()

        filename = str(''.join(filename))
        channel.basic_publish(exchange='logs',routing_key='',body=filename)
        print(f"[x] sent {filename}")

        # your code here
        print(time.process_time() - start)
    connection.close()
```

code for receive.py

Importing relevant package for receive.py

```
import os
import time
import base64
import pika
import numpy as np
import pandas as pd
import itertools
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
import psycopg2
import psycopg2.extras

```

PcapToNetFlow class is used to get the pcap filename and convert it into netflow csv format for data processing.

```
class PcapToNetFlow():
    def __init__(self,pcap_file_name):
        self.pcap_file_name = pcap_file_name

    def convert(self):
        cmd = f"./bin/cfm ./{self.pcap_file_name} ./"
        result = os.system(cmd)
        print(result)
        return f"{self.pcap_file_name}_Flow.csv"
```

Class PreProcessNetFlowCsv is used to preprocess the raw data so that we can encode the data for classifier model to use it. It reads the CSV filename and converts it into dataframe. Unwanted columns are droped. One hot encoding is done to categorical data and data types are converted into float for machine for normalizing. Then data is splitted into X and y for data encoding and target labelling. In this process we have used MinMaxScaler which normalizes the data using min and max values of the data.

```python
class PreProcessNetFlowCsv():
    def __init__(self,csv_file_name):
        self.csv_file_name = csv_file_name
        self.df = pd.read_csv(self.csv_file_name)
        self.drop_columns = None
        self.x = None
        self.y = None

    def set_drop_columns(self,drop_columns):
        self.drop_columns = drop_columns

    def drop_unused_column(self):
        self.df = self.df.drop(self.drop_columns,axis=1)

    # Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,green,blue)
    def encode_text_dummy(self, name):
        dummies = pd.get_dummies(self.df[name])
        for x in dummies.columns:
            dummy_name = f"{name}-{x}"
            self.df[dummy_name] = dummies[x]
        self.df.drop(name, axis=1, inplace=True)

    def pre_process(self):
        self.df['Flow Byts/s'] = self.df['Flow Byts/s'].astype('float32')
        self.df['Flow Pkts/s'] = self.df['Flow Pkts/s'].astype('float32')
        self.df = self.df.replace([np.inf, -np.inf], np.nan)
        self.df = self.df.dropna()
        self.df.loc[self.df['Label']=='No Label', 'Label'] = 0

    def split_x_y(self,label_name):
        # Convert a Pandas dataframe to the x,y inputs that TensorFlow needs
        def to_xy(df, target):
            result = []
            for x in df.columns:
                if x != target:
                    result.append(x)
            # find out the type of the target column.  Is it really this hard? :(
            target_type = df[target].dtypes
            target_type = target_type[0] if hasattr(
                target_type, '__iter__') else target_type
            # Encode to int for classification, float otherwise. TensorFlow likes 32 bits.
            if target_type in (np.int64, np.int32):
                # Classification
                dummies = pd.get_dummies(df[target])
                return df[result].values.astype(np.float32), dummies.values.astype(np.float32)
            # Regression
            return df[result].values.astype(np.float32), df[[target]].values.astype(np.float32)
        self.x, self.y = to_xy(self.df,label_name)
        return self.x

    def normalize(self):
        scaler = MinMaxScaler()
        self.x = scaler.fit_transform(self.x)
        print(self.x)
```

NeuralnetworkClassifier class loads the model and predicts the label of the record

```
81  class NeuralNetworkClassifier():
82      def __init__(self,model_name):
83          self.model_name = model_name
84          self.classifier = None
85          self.x = None
86          self.y = None
87
88      def load_model(self):
89          self.classifier = load_model(self.model_name)
90
91      def predict(self,x):
92          y_pred = self.classifier.predict(x)
93          self.y = np.argmax(y_pred,axis=1)
94          print(self.y)
95          return self.y
96
```

this function is used to hide tensor-flow warnings that are unnecessary

```
98  def tensorflow_shutup():
99      """
100     Make Tensorflow less verbose
101     """
102     try:
103         # noinspection PyPackageRequirements
104         import os
105         from tensorflow import logging
106         logging.set_verbosity(logging.ERROR)
107         os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
108
109         # Monkey patching deprecation utils to shut it up! Maybe good idea to disable this once after upgrade
110         # noinspection PyUnusedLocal
111         def deprecated(date, instructions, warn_once=True):
112             def deprecated_wrapper(func):
113                 return func
114             return deprecated_wrapper
115
116         from tensorflow.python.util import deprecation
117         deprecation.deprecated = deprecated
118
119     except ImportError:
120         pass
```

```python
if __name__ == "__main__":


    tensorflow_shutup()
    nnc = NeuralNetworkClassifier('../model/dnn-model.hdf5')
    nnc.load_model()

    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.exchange_declare(exchange='logs',exchange_type='fanout')

    result = channel.queue_declare(queue='',exclusive=True)
    queue_name = result.method.queue

    channel.queue_bind(exchange='logs',queue = queue_name)

    conn = psycopg2.connect(user = "netvizuser",password = "netviz123",host = "127.0.0.1",port = "5432",database = "netviz")

    print("[*] Waiting for logs. To Exit press ctrl + C")

    def callback(ch,method,properties,body):

        try:
            filename = body.decode("utf-8")
            print(f"working on filename: {filename}")

            # check starting time
            start = time.process_time()

            # convert to netflow csv
            nc = PcapToNetFlow(pcap_file_name=filename)
            csv = nc.convert()

            df = pd.read_csv(csv)
            df = df.reset_index(drop=True)

            print(df)
              # preprocess netflow csv
              drop_columns = ['Flow ID','Timestamp','Src IP','Dst IP','Src Port','Dst Port','Protocol']
              preprocessor = PreProcessNetFlowCsv(csv_file_name=csv)
              preprocessor.set_drop_columns(drop_columns)
              preprocessor.drop_unused_column()
              preprocessor.pre_process()
              x = preprocessor.split_x_y('Label')
              y_pred = nnc.predict(x)
              df["Label"] = y_pred
              equiv = {0:"Normal",1:"Threat"}
              df["Label"] = df["Label"].map(equiv)
              df.reset_index(drop=True,inplace=True)

              ## time to save the df
              cur = conn.cursor()
              columns = [col.replace(" ","_").replace("/","_per_").lower() for col in df.columns.values]
              print(columns)
              table = "dash_trafficlog"
              # df is the dataframe
              if len(df) > 0:
                  df_columns = columns
                  # create (col1,col2,...)
                  columns = ",".join(df_columns)

                  # create VALUES('%s', '%s",...) one '%s' per column
                  values = "VALUES({})".format(",".join(["%s" for _ in df_columns]))

                  #create INSERT INTO table (columns) VALUES('%s',...)
                  insert_stmt = "INSERT INTO {} ({}) {}".format(table,columns,values)

                  print(values)
                  print(insert_stmt)
                  psycopg2.extras.execute_batch(cur, insert_stmt, df.values)
                  conn.commit()
                  cur.close()
                  print("dataframe inserted")


                  os.remove("data.csv")
                  df.to_csv("data.csv")
```

```
202                 # also dump this data to realtimedata.csv for django to make it realtime
203                 # df.reset_index(drop=True,inplace=True)
204                 # df["new_old"] = "new"
205                 # df_old = pd.read_csv("data.csv")
206                 # df_old["new_old"] = "old"
207                 # with open('data.csv', 'a') as f:
208                 #     df.to_csv(f, header=False)
209 
210             # let's drop the unwanted pcaps
211             if os.path.exists(filename):
212                 os.remove(filename)
213                 os.remove(csv)
214             else:
215                 print("Can not delete the file as it doesn't exists")
216 
217             # your code here
218             print(time.process_time() - start)
219         except:
220             pass
221 
222     channel.basic_consume(queue=queue_name,on_message_callback=callback,auto_ack=True)
223     channel.start_consuming()
```

In this code we are utilizing all the class we created to receive the filename of the packet captured and converting to csv format. After that the data is preprocessed and normalized and evaluated using neural network classifier. After evaluation the data is labelled if the data is normal or attack and the final output is stored in the database for web viewing.

Output from send.py

```
(venv) keshavchaurasia@keshavchaurasia-XPS:~/Desktop/codes/mp/campus/src$ python send.py
*** starting packet capture ***
*** stopping packet capture ***
[x] sent dump-20190717-102046.pcap
0.138205525
*** starting packet capture ***
*** stopping packet capture ***
[x] sent dump-20190717-102046.pcap
0.23236186200000003
*** starting packet capture ***
*** stopping packet capture ***
[x] sent dump-20190717-102047.pcap
0.19926562199999998
*** starting packet capture ***
*** stopping packet capture ***
[x] sent dump-20190717-102048.pcap
0.21480670000000002
*** starting packet capture ***
*** stopping packet capture ***
```

output from receive.py

```
(venv) keshavchaurasia@keshavchaurasia-XPS:~/Desktop/codes/mp/campus/src$ python receive.py
Using TensorFlow backend.
WARNING: Logging before flag parsing goes to stderr.
W0717 10:20:41.734602 139648471050048 deprecation_wrapper.py:119] From receive.py:106: The name tf.log
ging.set_verbosity is deprecated. Please use tf.compat.v1.logging.set_verbosity instead.

W0717 10:20:41.734893 139648471050048 deprecation_wrapper.py:119] From receive.py:106: The name tf.log
ging.ERROR is deprecated. Please use tf.compat.v1.logging.ERROR instead.

[*] Waiting for logs. To Exit press ctrl + C
working on filename: dump-20190717-102046.pcap
cic.cs.unb.ca.ifm.Cmd You select: ./dump-20190717-102046.pcap
cic.cs.unb.ca.ifm.Cmd Out folder: ./
cic.cs.unb.ca.ifm.Cmd CICFlowMeter received 1 pcap file
Working on... dump-20190717-102046.pcap
dump-20190717-102046.pcap is done. total 3 flows
Packet stats: Total=101,Valid=101,Discarded=0
--------------------------------------------------------------------------------
0
                                    Flow ID           Src IP  ...  Idle Min     Label
0  192.168.10.12-117.205.249.183-35821-15754-6  117.205.249.183  ...         0  No Label
1   192.168.10.12-24.168.68.230-51413-58052-17    24.168.68.230  ...         0  No Label

[2 rows x 84 columns]
[1 0]
```

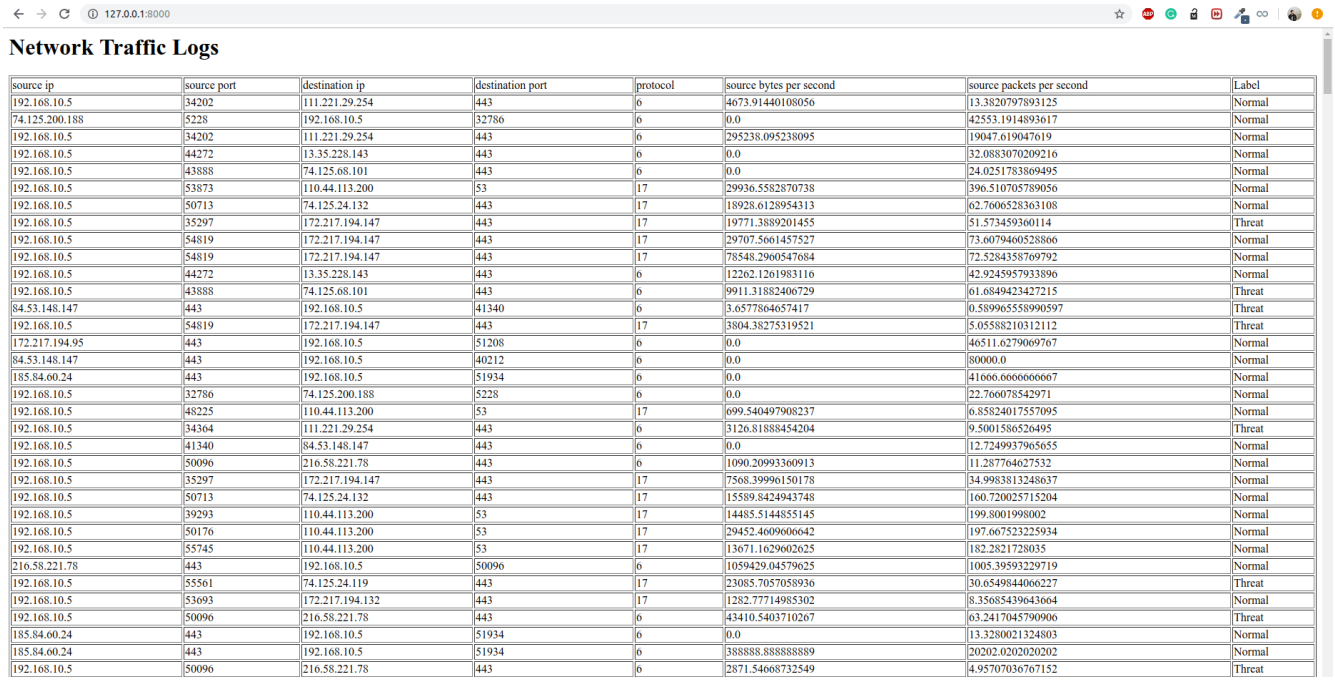
```
working on filename: dump-20190717-102047.pcap
cic.cs.unb.ca.ifm.Cmd You select: ./dump-20190717-102047.pcap
cic.cs.unb.ca.ifm.Cmd Out folder: ./
cic.cs.unb.ca.ifm.Cmd CICFlowMeter received 1 pcap file
Working on... dump-20190717-102047.pcap
dump-20190717-102047.pcap is done. total 3 flows
Packet stats: Total=102,Valid=102,Discarded=0
--------------------------------------------------------------------------------
0
                                    Flow ID           Src IP  ...  Idle Min     Label
0   192.168.10.12-24.168.68.230-51413-58052-17    24.168.68.230  ...         0  No Label
1  192.168.10.12-117.205.249.183-35821-15754-6  117.205.249.183  ...         0  No Label

[2 rows x 84 columns]
[0 1]
```

```
working on filename: dump-20190717-102048.pcap
cic.cs.unb.ca.ifm.Cmd You select: ./dump-20190717-102048.pcap
cic.cs.unb.ca.ifm.Cmd Out folder: ./
cic.cs.unb.ca.ifm.Cmd CICFlowMeter received 1 pcap file
Working on... dump-20190717-102048.pcap
dump-20190717-102048.pcap is done. total 4 flows
Packet stats: Total=112,Valid=112,Discarded=0
--------------------------------------------------------------------------------
0
                                    Flow ID           Src IP  ...  Idle Min     Label
0   192.168.10.12-87.202.162.246-51413-8999-17  87.202.162.246  ...         0  No Label
1   192.168.10.12-24.168.68.230-51413-58052-17   24.168.68.230  ...         0  No Label
2  192.168.10.12-117.205.249.183-35821-15754-6   192.168.10.12  ...         0  No Label

[3 rows x 84 columns]
[0 0 1]
```

Web View of traffic logs with their output Label as Normal or threat



127.0.0.1:8000

Network Traffic Logs

source ip	source port	destination ip	destination port	protocol	source bytes per second	source packets per second	Label
192.168.10.5	34202	111.221.29.254	443	6	4673.91440108056	13.3820797893125	Normal
74.125.200.188	5228	192.168.10.5	32786	6	0.0	42553.1914893617	Normal
192.168.10.5	34202	111.221.29.254	443	6	295238.095238095	19047.619047619	Normal
192.168.10.5	44272	13.35.228.143	443	6	0.0	32.0883070209216	Normal
192.168.10.5	43888	74.125.68.101	443	6	0.0	24.0251783869495	Normal
192.168.10.5	53873	110.44.113.200	53	17	29936.5582870738	396.510705789056	Normal
192.168.10.5	50713	74.125.24.132	443	17	18928.6128954313	62.7606528363108	Normal
192.168.10.5	35297	172.217.194.147	443	17	19771.3889201455	51.573459360114	Threat
192.168.10.5	54819	172.217.194.147	443	17	29707.5661457527	73.6079460528866	Normal
192.168.10.5	54819	172.217.194.147	443	17	78548.2960547684	72.5284358769792	Normal
192.168.10.5	44272	13.35.228.143	443	6	12262.1261983116	42.9245957933896	Normal
192.168.10.5	43888	74.125.68.101	443	6	9911.31882406729	61.6849423427215	Threat
84.53.148.147	443	192.168.10.5	41340	6	3.6577864657417	0.589965558990597	Threat
192.168.10.5	54819	172.217.194.147	443	17	3804.38275319521	5.05588210312112	Threat
172.217.194.95	443	192.168.10.5	51208	6	0.0	46511.6279069767	Normal
84.53.148.147	443	192.168.10.5	40212	6	0.0	80000.0	Normal
185.84.60.24	443	192.168.10.5	51934	6	0.0	41666.6666666667	Normal
192.168.10.5	32786	74.125.200.188	5228	6	0.0	22.766078542971	Normal
192.168.10.5	48225	110.44.113.200	53	17	699.540497908237	6.85824017557095	Normal
192.168.10.5	34364	111.221.29.254	443	6	3126.81888454204	9.5001586526495	Threat
192.168.10.5	41340	84.53.148.147	443	6	0.0	12.7249937965655	Normal
192.168.10.5	50096	216.58.221.78	443	6	1090.20993360913	11.287764627532	Normal
192.168.10.5	35297	172.217.194.147	443	17	7568.39996150178	34.9983813248637	Normal
192.168.10.5	50713	74.125.24.132	443	17	15589.8424943748	160.720025715204	Normal
192.168.10.5	39293	110.44.113.200	53	17	14485.5144855145	199.8001998002	Normal
192.168.10.5	50176	110.44.113.200	53	17	29452.4609606642	197.667523225934	Normal
192.168.10.5	55745	110.44.113.200	53	17	13671.1629602625	182.2821728035	Normal
216.58.221.78	443	192.168.10.5	50096	6	1059429.04579625	1005.39593229719	Normal
192.168.10.5	55561	74.125.24.119	443	17	23085.7057058936	30.6549844066227	Threat
192.168.10.5	53693	172.217.194.132	443	17	1282.77714985302	8.35685439643664	Normal
192.168.10.5	50096	216.58.221.78	443	6	43410.5403710267	63.2417045790906	Threat
185.84.60.24	443	192.168.10.5	51934	6	0.0	13.3280021324803	Normal
185.84.60.24	443	192.168.10.5	51934	6	388888.888888889	20202.0202020202	Normal
192.168.10.5	50096	216.58.221.78	443	6	2871.54668732549	4.95707036767152	Threat