CLASSIFICATION OF TIME GRANULAR TRAFFIC IN SDWMN BASED IOT NETWORKS TO AVOID CONGESTION

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

SUBRAMANIAM S V
(Reg. No.: 122003258, B.Tech. Computer Science and Engineering)

February 2021

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**Classification of Time Granular Traffic in SDWMN based IoT Network to Avoid Congestion** " submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri**. **SUBRAMANIAM S V (Reg. No.122003258, B.Tech. Computer Science and Engineering**) during the academic year 2020-21, in the School of Computing

Project Based Work *Viva voc*e held on 17/02/2021

Examiner 1 **Examiner 2**

List of Figures

List of Tables

Abbreviations

SDN	Software Defined Networking
WMN	Wireless Mesh Networks
SDWMN	Software Defined Wireless Mesh Network
TC	Traffic Classification
ML	Machine Learning
KNN	K Nearest Neighbor
SVM	Support Vector Machine
DT	Decision Tree
IoT	Internet of Things
TE	Traffic Engineering
MR	Mesh Router
BR	Border Router
NAT	Network Address Translation
LR-WPAN	Low-Rate Wireless Personal Area Network
FGT	Fine Granular Traffic
CGT	Coarse Granular Traffic
RFC	Random Forest Classifier
BGD	Boosted Gradient Descent
LR	Logistic Regression

Abstract

WMN(Wireless Mesh Network) is a wireless network for the communication in which nodes or systems are connected in a mesh topology. The inter-connection of all nodes or systems connected with all other systems is called mesh topology. In the modern Wireless IoT(Internet of Things) devices, WMN is mostly used because of its efficiency. SDWMN is the result of employing SDN(software Defined Networking) technology in the WMN . This kind of SDN and WMN combination provides the advantage of creating a Heterogeneous large wireless network. These large wireless networks increase the efficiency of the IoT networks.

There are also disadvantages in the SDWMN network because of heterogeneous(different types of nodes) networks. These heterogeneous large wireless networks will create a problem of varying granular network traffic. These things cause congestion in the network and packet transfer speed will become slow. TE(Traffic Engineering) is a method of optimizing the performance of telecommunication networks by dynamically, predicting, analyzing, regulating the behavior of the data transmitted over the network.

TC(Traffic Classification), it is used to classify the network traffic based on different factors and TC is also used in various applications like intrusion detection,firewall building,status report generation ,etc. This paper uses different kinds of ML(Machine Learning) techniques to classify the network traffic to avoid congestion. All ML Algorithms reached the accuracy above 95%.Boosted Gradient Descent classifier provides the best accuracy(above 98.8%) of all other ML techniques. 10-fold cross-validation technique is used to validate the model and to avoid overfitting.

KEY WORDS: ML classifier, Traffic Classification, WMN, SDN, Traffic Engineering, network congestion control

Table of Contents

1. INTRODUCTION

1.1 Introduction:

Nowadays,wireless devices are increasing and which lead to the growth of wireless networks. WMN(Wireless Mesh Network) is the best solution for long distance communication in IoT(Internet of Things) networks. Each node in the WMN is called either a MC(Mesh Client) or a MR(Mesh Router). SDN(Software Defined Networking) is a future generation network's technology and it offers unprecedented benefits to the network by simplifying the WMN traffic management. "The merge of two promising technologies, WMN and SDN, give rise to a new type of network called Software Defined WMN (SDWMN) [1]".

"Although WMNs seem to be a viable solution to long range wireless communication problems in IoT networks, many other issues need to be taken care of [2]".

WMN contains millions of heterogeneous nodes connected across the global because of that it will cause large amounts of congestion in the network. SDN is used to control the traffic by enabling the TE(Traffic Engineering) from the central point or server. The Heterogeneous IoT network contains different kinds of nodes,different subsystems,different packet transfer speed. These heterogeneous IoT nodes will produce varying granular traffic in the network. This kind of network will produce huge traffic and it needs to be processed carefully.

"TC(Traffic Classification) is one of the TE techniques that helps to categorize the varying granular network traffic and extracts some important heuristics, which assist to improve the network performance [3]".

We are going to classify the network traffic state as congested and non congested based on the different features in the dataset. We are going to use different ML classification algorithms and its variants to classify the network traffic. These classification algorithms give accuracy more than 95%.

This paper's contributions are summarized below

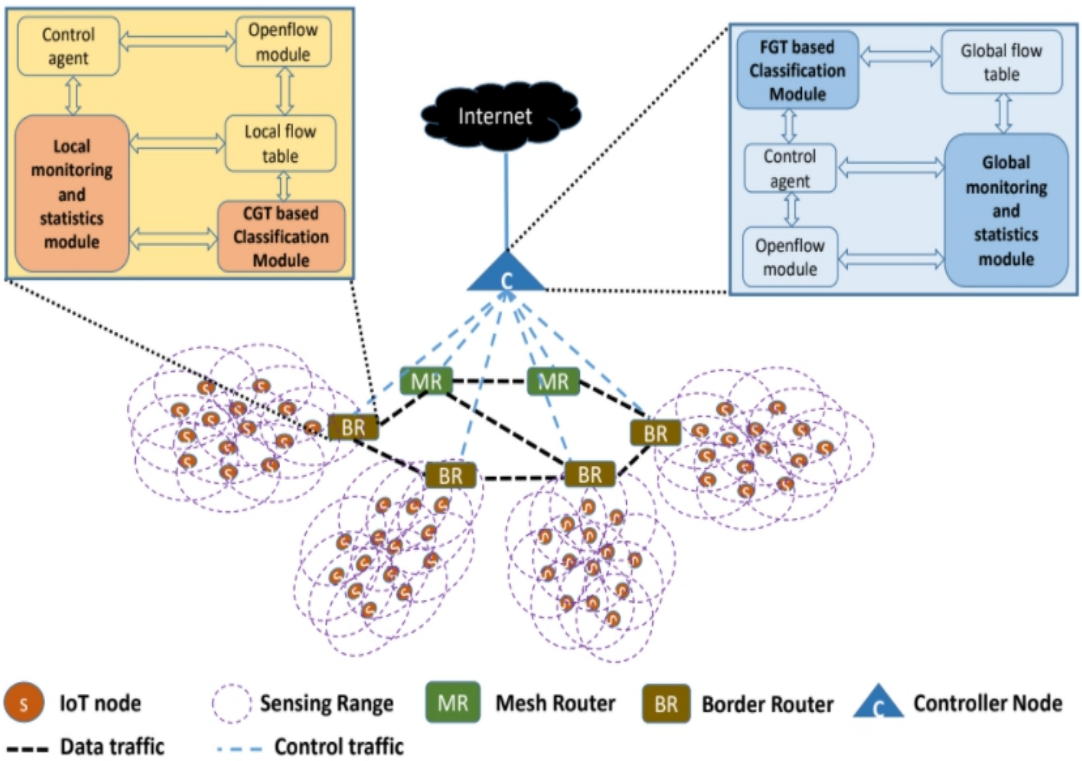
- SDWMN architecture based on IoT network are proposed
- Multi-granular network traffic is categorized into congested and non congested by different type of ML algorithms
- Results of the various ML algorithms are recorded and updated in the table.
- Observation from table are analyzed and concluded

1.2 Project related to work:

"A malware based TC approach is proposed using the convolution neural network for representation learning" in [4]. Similarly," an internet TC based approach" is proposed in [5]. purpose. In [6] discuss "user traffic accountability based methods under the congestion scenario in flow-based multi layer switches networks". In [7], "an internet TC based approach" is proposed.Authors discuss" a multi-granular aggregation approach "in [8]. In [9], "a hierarchical classification based approach is used for threats identification".

1.3 Proposed architecture:

Fig 1.1. Shows SDWMN flow classification architecture for varying granular Dataset



This architecture consists of MR(Mesh Router), IoT nodes,and C(a controller). IoT nodes represented in the diagram are operated by a low power battery and different IoT nodes forms LR-WPAN(Low-Rate Wireless Personal Area Network) which follows IEEE 802.15.4 standards. All IoT nodes in the network are connected to the BR(Border Router). BR provides services like IP forwarding, NAT(Network Address Translation), etc. BR is connected directly or indirectly connected to the MR(Mesh Router). WMN is formed by combining multiple MR and it follows IEEE 802.11 standard. Mesh Router is connected to all different nodes in the network directly or indirectly via single or multi-hop

communication.The controller node is node which contains SDN technology and all information about the network .This controller node is also called controller plane of SDWMN and its controls the whole WMN.

Every IoT node in the architecture is connected to the Border Router.Each Border Router contains its own monitoring and statistics modules. Border Router stores all traffic flows and traffic statistics.

All network traffic statistics are sent to the storage module for classification. This classification helps in reducing the network congestion. There are two types of OTP(Observation Time Period) one is fine granular OTP and another is coarse granular OTP.

The fine granular OTP creates more frequent network traffic data flow in SDWMN and these kinds of flow are sent to the control plane(C). Control Plane records its network traffic flow by its own module. Control Plane stores its own global traffic flow statistics and network monitoring data. These global statistical and monitoring data are sent to the FGT based classification module. These data are global data because it is connected to the internet.

The coarse granular OTP creates less frequent network traffic data flow inside SDWMN and these kinds of flow are sent to Border Router (BR). As mentioned above, Each Border router contains its own statistics and monitoring modules.Border Router stores all local network traffic flows and traffic statistics. These local statistics and monitored traffic data is sent to the CGT based classification module. All the traffic statistics and monitored data are local data because it is connected to the Mesh Router (not internet).

Mesh Router (MR) periodically updates its local routing tables and sends the local routing data to the control plane. The FGT classification module in the controller plane periodically updates its global routing table. Controller dynamically re-configures the network traffic behavior and manages all the traffic.All global and local routing data are stored in the controller plane.

“As IoT networks have limited resources and are bounded by multiple constraints, TC helps to improve the performance of multi-granular traffic based networks by dividing the aggregate traffic Dataset into multiple granularity based application specific subsets [11]”.

1.4 Proposed TC Approach

The performance of the network is measured by performance metrics like packet loss,throughput,inter arrival time , etc.These metrics are to be considered while determining the current network traffic congestion in TC approach.In this paper,TC

approach is used to classify the varying granular network traffic flow using various Machine Learning classification algorithms.

Each node in the SDWMN network sends the network traffic to the nearest connected local BR,which in turns sends network traffic to the MR. Finally, network traffic from MR reaches the controller to connect the internet. As we already said that Coarse granular OTP has less number of network flow and it is controlled in BR level. All BR contain statistics and monitoring modules which record the local incoming and outgoing traffic from the IoT nodes. Each traffic flow is taken as a separate traffic unit and if any respective traffic unit or resources allocation causes congestion, then it is DROP at the BR level itself to prevent condensation in the network.

Fine Granular OTP produces more number network traffic and which is handled by controller level. As already mentioned, the local routing table is periodically updated in the controller which in turn gives the controller knowledge of the network traffic in the MR and BR level. By using this, controllers avoid the network congestion by limiting the resources allocation and network traffic to the each process in the nodes. This gives good performance to the SDWMN network. In this paper, We assume that the ML model is trained and stored at the controller level. ML algorithms train the model by statistics and monitored data given to the controller from MR. This trained ML model replaces , updates, and removes the record in the flow table. Modification of the data in the flow table is done periodically.

1.5 Granularity Classes:

1) FGT(Fine Granular Traffic) having short OTP of 20, 15, 10 ,5 seconds
2) CGT(Coarse Granular Traffic) having longer OTP of 60, 55, 50, 45 seconds

1.6 Environment:

Description of Dataset : For this classification, 802.15.4 MAC layer performance data-set is used. “The Dataset provides the measurements collected from the wilab2 testbed facility in Ghent university and is given in [10]”.

This Dataset consists of seven features. The collected Dataset is preprocessed and following seven different features are selected: NumOfReceived (Number of received frames), PRR(Packet Reception Rate), PacketLoss (Number of erroneous frames), P LR (Packet Loss Rate), Throughput (Aggregated throughput), IP I (Inter-Packet-Interval), and Density (Number of active nodes) [base paper].These features are used in ML algorithm to train the model.

1.7 Machine learning algorithms:

In this paper, DT(Decision Tree), SVM(Support vector machine), KNN(K Nearest Neighbor), LR(Logistic Regression), RFC(Random Forest

classification),BGD(Boosted Gradient Descent) Machine Learning classification algorithm are implemented.

Different variant of DT are Coarse DT(C_DT), Medium DT(M_DT), Fine DT(F_DT)

Different variant of SVM are Fine Gaussian SVM(FG_SVM), Medium Gaussian SVM(MG_SVM), Coarse Gaussian SVM(CG_SVM), Linear SVM(L_SVM), Quadratic SVM(Q_SVM), Cubic SVM(C_SVM)

Different variant of KNN are Cosine KNN(Cos_KNN), Cubic KNN(Cu_KNN), Weighted KNN(W_KNN), Fine KNN(F_KNN), Medium KNN(M_KNN), Coarse KNN(Coa_KNN)

DT variants are easy and wide used classification techniques that have simple interpretability, less memory usage and high prediction speed. SVM variants are common prediction models to resolve a range of problems like regression, classification, etc. SVM variants are hard to interpret except linear SVM but they have good prediction speed. KNN variants are most commonly used for many purposes such as classification and regression. KNN variants are slow in prediction, hard to interpret and medium memory usage.

RT is a combination of many DTs(Decision Tree), it is also simple as DT but it gives good accuracy compared to DT. LR is easy to interpret and it is the simple classifier, it is fast in prediction, medium memory usage, Flexibility of model is low because you cannot change parameters. BGD gives fine detail with either deeper trees or larger numbers of shallow trees, low memory usage, prediction speed is fast.

After getting the result of all the ML algorithms mentioned above, it is recorded in the table.Then table is analysed and observed, finally the best ML classification algorithm is selected for implementation in the controller unit in SDWMN network. Selection of the best ML classification algorithm is decided based on different criteria like prediction speed, prediction accuracy, interpretability, model flexibility, implementation difficulty, memory usage, etc.

By using selection criteria mentioned above,BGD(Boosted Gradient Descent) gives the best results as compared to the other ML classification algorithms.BGD gives prediction accuracy of more than 99%, fast prediction speed.

1.8 Merits:

SDWMN is most commonly used for long distance communication and it give short packet transportation time. Another advantage of SDWMN network is number of node connected in the network is large.Here network uses heterogeneous IoT node which gives various sub system and different type of node to connected in the single network. These huge number of IoT nodes , long distance communication and heterogeneous IoT

nodes create a problem of more frequent network congestion.There are various method to solve this problem, this paper uses traffic classification method to manage the network traffic with ML classification algorithm. This type of SDWMN network gives efficient communication of IoT node

1.9 Demerits:

Even through there are various ML classification algorithms to control the network congestion and all well optimized ML algorithm gives accuracy of 99.2% to 99.7% but 0.8% to 0.3% there are chances of network congestion which can be unavoidable. Now a days packet transfer speed of the network is rapidly increase by introducing the new generation network which make these 0.3% to 0.7% network congestion negligible.

2. SOURCE CODE

2.1 Importing all Header File:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from google.colab import files
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
```

2.2 Code for Uploading the Dataset :

```
uploaded = []
uploaded.append(files.upload())
```

2.3 Fine Decision Tree (F_DT) :

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
#IMPLEMENTING THE CLASSIFIER
  DT = DecisionTreeClassifier(criterion="entropy", max_depth = 100)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    DT.fit(X_train,Y_train)
    scores.append(DT.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.4 Medium Decision Tree (M_DT):

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
#IMPLEMENTING THE CLASSIFIER
  DT = DecisionTreeClassifier(criterion="entropy", max_depth = 20)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    DT.fit(X_train,Y_train)
    scores.append(DT.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.5 Coarse Decision Tree (C_DT) :

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
#IMPLEMENTING THE CLASSIFIER
  DT = DecisionTreeClassifier(criterion="entropy", max_depth = 4 )
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    DT.fit(X_train,Y_train)
    scores.append(DT.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.6 Linear SVM (L_SVM):

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
#IMPLEMENTING THE CLASSIFIER
  SVM = svm.SVC(kernel='linear')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    SVM.fit(X_train,Y_train)
    scores.append(SVM.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.7 Quadratic SVM (Q_SVM):

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  SVM = svm.SVC(C = 15,kernel='poly',degree = 2,gamma = 'auto')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    SVM.fit(X_train,Y_train)
    scores.append(SVM.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.8 Cubic SVM (C_SVM) :

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  SVM = svm.SVC(C = 20,kernel='poly',degree=3,gamma = 'auto')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    SVM.fit(X_train,Y_train)
    scores.append(SVM.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.9 Fine Gaussian SVM (FG_SVM):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  gamma = np.sqrt((1/7))*4
  SVM = svm.SVC(C=15,kernel='rbf',gamma = gamma)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    SVM.fit(X_train,Y_train)
    scores.append(SVM.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.10 Medium Gaussian SVM (MG_SVM):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  gamma = np.sqrt((1/7))
  SVM = svm.SVC(C=15,kernel='rbf',gamma = gamma)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    SVM.fit(X_train,Y_train)
    scores.append(SVM.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.11 Coarse Gaussian SVM (CG_SVM):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  gamma = np.sqrt((1/7))/4
  SVM = svm.SVC(C=15,kernel='rbf',gamma = gamma)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    SVM.fit(X_train,Y_train)
    scores.append(SVM.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.12 Fine K Nearest Neighbor (F_KNN):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  KNN = KNeighborsClassifier(n_neighbors=5,metric = 'euclidean')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    KNN.fit(X_train,Y_train)
    scores.append(KNN.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.13 Medium K Nearest Neighbors (M_KNN):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  KNN = KNeighborsClassifier(n_neighbors=10,metric='euclidean')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    KNN.fit(X_train,Y_train)
    scores.append(KNN.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.14 Coarse K Nearest Neighbors (Coa_KNN):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  KNN = KNeighborsClassifier(n_neighbors=100,metric = 'euclidean')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    KNN.fit(X_train,Y_train)
    scores.append(KNN.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.15 Cosine K Nearest Neighbors (Cos_KNN):

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  KNN = KNeighborsClassifier(n_neighbors=5,metric='cosine')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    KNN.fit(X_train,Y_train)
    scores.append(KNN.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.16 Cubic K Nearest Neighbors (Cu_KNN):

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  KNN = KNeighborsClassifier(n_neighbors=10,p=3)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    KNN.fit(X_train,Y_train)
    scores.append(KNN.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.17 Weighted K Nearest Neighbors (W_KNN):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  W = 1.65
  #IMPLEMENTING THE CLASSIFIER
  KNN = KNeighborsClassifier(n_neighbors=15,weights='distance')
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    KNN.fit(X_train,Y_train)
    scores.append(KNN.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.18 Random Forest Classifier (RFC)

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  RF = RandomForestClassifier(n_estimators = 100,max_depth=100,random_state=4,)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    RF.fit(X_train,Y_train)
    scores.append(RF.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.19 Logistic Regression (LR):

```
accuracies = []
acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
  #IMPLEMENTING THE CLASSIFIER
  LR = LogisticRegression()
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    LR.fit(X_train,Y_train)
    scores.append(LR.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

2.20 Boosted Gradient Descent (BGD):

```
accuracies = [];acc = []
for j in range(5,65,5):
  if(j >= 25 and j<= 40):
    continue
  #READING THE  DATASET
  X = pd.read_csv('802_15_4_MACperf_'+str(j)+'s.csv')
  #DATA PREPROCESSING
  X = X.sample(frac = 1)
  Y = X['COR'].values
  kf = KFold(n_splits=10)
  X_ = X.drop("COR", axis=1)
  X_ = preprocessing.StandardScaler().fit(X_).transform(X_)
  label = preprocessing.LabelEncoder()
  label.fit([0,20])
  Y_ = label.transform(Y)
  scores = []
#IMPLEMENTING THE CLASSIFIER
  GBC = GradientBoostingClassifier(loss = 'exponential',learning_rate=0.5,criterion =
'mse',max_depth = 6)
  for train, test in kf.split(X):
    X_train, X_test, Y_train, Y_test = X_[train],X_[test],Y_[train],Y_[test]
    GBC.fit(X_train,Y_train)
    scores.append(GBC.score(X_test,Y_test))
  accuracy = np.mean(scores)
  accuracies.append(accuracy)
  print(str(j)+"sec OTP Dataset :","   Accuracy is ",accuracy)
  if(j == 20):
    print("Average accuracy of fine Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
  if(j == 60):
    print("Average accuracy of coarse Granular OTP Dataset ",np.mean(accuracies))
    acc.append(np.mean(accuracies))
    accuracies = []
print("Average Accuracy ",np.mean(acc))
```

3. SNAPSHOT

3.1 Fine Decision Tree (F_DT) : Fig. 3.1.Output of F_DT

```
5sec OTP dataset:    Accuracy is  0.9946932989058939
10sec OTP dataset:    Accuracy is  0.9934503807474553
15sec OTP dataset:    Accuracy is  0.9914615389920755
20sec OTP dataset:    Accuracy is  0.990216453329498
Average accuracy of fine Granular OTP dataset 0.9924554179937307
45sec OTP dataset:    Accuracy is  0.9802046020278065
50sec OTP dataset:    Accuracy is  0.978587460721233
55sec OTP dataset:    Accuracy is  0.9764012334482132
60sec OTP dataset:    Accuracy is  0.9802221983683985
Average accuracy of coarse Granular OTP dataset 0.9788538736414129
Average Accuracy  0.9856546458175718
```

3.2 Medium Decision Tree (M_DT) : Fig. 3.2.Output of M_DT

```
5sec OTP dataset:    Accuracy is  0.9944379121283184
10sec OTP dataset:    Accuracy is  0.992832486380076
15sec OTP dataset:    Accuracy is  0.9897914353304686
20sec OTP dataset:    Accuracy is  0.9880129680233951
Average accuracy of fine Granular OTP dataset 0.9912687004655645
45sec OTP dataset:    Accuracy is  0.9802076376662013
50sec OTP dataset:    Accuracy is  0.984707466706569
55sec OTP dataset:    Accuracy is  0.9831353165245783
60sec OTP dataset:    Accuracy is  0.9765081580077286
Average accuracy of coarse Granular OTP dataset 0.9811396447262694
Average Accuracy  0.986204172595917
```

3.3 Coarse Decision Tree (C_DT) : Fig. 3.3.Output of C_DT

```
5sec OTP dataset:    Accuracy is  0.9626623795788631
10sec OTP dataset:    Accuracy is  0.9724412092356056
15sec OTP dataset:    Accuracy is  0.9704895476270942
20sec OTP dataset:    Accuracy is  0.9703988685938922
Average accuracy of fine Granular OTP dataset 0.9689980012588638
45sec OTP dataset:    Accuracy is  0.9620636269807539
50sec OTP dataset:    Accuracy is  0.969426155918001
55sec OTP dataset:    Accuracy is  0.9702884092145837
60sec OTP dataset:    Accuracy is  0.9692035208243881
Average accuracy of coarse Granular OTP dataset 0.9677454282344315
Average Accuracy  0.9683717147466477
```

3.4 Linear SVM (L_SVM) : Fig. 3.4.Output of L_SVM

```
5sec OTP dataset:    Accuracy is  0.9655393314430926
10sec OTP dataset:    Accuracy is  0.9619368523859666
15sec OTP dataset:    Accuracy is  0.9617686615031278
20sec OTP dataset:    Accuracy is  0.962820605014622
Average accuracy of fine Granular OTP dataset 0.9630163625867023
45sec OTP dataset:    Accuracy is  0.9636998360755266
50sec OTP dataset:    Accuracy is  0.9614544366302559
55sec OTP dataset:    Accuracy is  0.9635633956103755
60sec OTP dataset:    Accuracy is  0.9626556462000858
Average accuracy of coarse Granular OTP dataset 0.9628433286290609
Average Accuracy  0.9629298456078816
```

3.5 Quadratic SVM (Q_SVM) : Fig. 3.5.Output of Q_SVM

```
5sec OTP dataset:    Accuracy is  0.9654115767712836
10sec OTP dataset:    Accuracy is  0.9639161287368949
15sec OTP dataset:    Accuracy is  0.9654750984543868
20sec OTP dataset:    Accuracy is  0.9637980008629368
Average accuracy of fine Granular OTP dataset 0.9646502012063756
45sec OTP dataset:    Accuracy is  0.9582296156881792
50sec OTP dataset:    Accuracy is  0.954103695944935
55sec OTP dataset:    Accuracy is  0.9514239071286053
60sec OTP dataset:    Accuracy is  0.9523829969944181
Average accuracy of coarse Granular OTP dataset 0.9540350539390344
Average Accuracy  0.959342627572705
```

3.6 Cubic SVM (C_SVM) : Fig. 3.6.Output of C_SVM

```
5sec OTP dataset:    Accuracy is  0.9712935620143321
10sec OTP dataset:    Accuracy is  0.9715765538921699
15sec OTP dataset:    Accuracy is  0.9673362484568008
20sec OTP dataset:    Accuracy is  0.9633078047845054
Average accuracy of fine Granular OTP dataset 0.968378542286952
45sec OTP dataset:    Accuracy is  0.9532663469127558
50sec OTP dataset:    Accuracy is  0.9632762232530301
55sec OTP dataset:    Accuracy is  0.9534191910030836
60sec OTP dataset:    Accuracy is  0.9552866036925721
Average accuracy of coarse Granular OTP dataset 0.9563120912153603
Average Accuracy  0.9623453167511562
```

3.7 Fine Gaussian SVM (FG_SVM) : Fig. 3.7.Output of FG_SVM

```
5sec OTP dataset:    Accuracy is  0.9963556212872703
10sec OTP dataset:    Accuracy is  0.9944387980893954
15sec OTP dataset:    Accuracy is  0.9933185508066019
20sec OTP dataset:    Accuracy is  0.9941308308164342
Average accuracy of fine Granular OTP dataset 0.9945609502499254
45sec OTP dataset:    Accuracy is  0.9851587638880457
50sec OTP dataset:    Accuracy is  0.9877599880293282
55sec OTP dataset:    Accuracy is  0.9865091601668784
60sec OTP dataset:    Accuracy is  0.986813009875483
Average accuracy of coarse Granular OTP dataset 0.9865602304899339
Average Accuracy  0.9905605903699297
```

3.8 Medium Gaussian SVM (MG_SVM) : Fig. 3.8.Output of MG_SVM

```
5sec OTP dataset:    Accuracy is  0.9942458919948033
10sec OTP dataset:    Accuracy is  0.9934511437684078
15sec OTP dataset:    Accuracy is  0.9927619645357298
20sec OTP dataset:    Accuracy is  0.9924145452802147
Average accuracy of fine Granular OTP dataset 0.993218386394789
45sec OTP dataset:    Accuracy is  0.988455467184749
50sec OTP dataset:    Accuracy is  0.9834804728415382
55sec OTP dataset:    Accuracy is  0.9865136949029567
60sec OTP dataset:    Accuracy is  0.9780270502361528
Average accuracy of coarse Granular OTP dataset 0.9841191712913493
Average Accuracy  0.9886687788430691
```

3.9 Coarse Gaussian SVM (CG_SVM) : Fig. 3.9.Output of CG_SVM

```
5sec OTP dataset:    Accuracy is  0.9887476610313524
10sec OTP dataset:    Accuracy is  0.9883833112057256
15sec OTP dataset:    Accuracy is  0.9870095385230806
20sec OTP dataset:    Accuracy is  0.9867910733975741
Average accuracy of fine Granular OTP dataset 0.9877328960394332
45sec OTP dataset:    Accuracy is  0.9785592860178497
50sec OTP dataset:    Accuracy is  0.978598683226096
55sec OTP dataset:    Accuracy is  0.9804598222383458
60sec OTP dataset:    Accuracy is  0.9772810218978101
Average accuracy of coarse Granular OTP dataset 0.9787247033450255
Average Accuracy  0.9832287996922293
```

3.10 Fine K Nearest Neighbor (F_KNN) : Fig. 3.10.Output of F_KNN

```
5sec OTP dataset:    Accuracy is  0.9963556212872703
10sec OTP dataset:    Accuracy is  0.9944387980893954
15sec OTP dataset:    Accuracy is  0.9933185508066019
20sec OTP dataset:    Accuracy is  0.9941308308164342
Average accuracy of fine Granular OTP dataset 0.9945609502499254
45sec OTP dataset:    Accuracy is  0.9851587638880457
50sec OTP dataset:    Accuracy is  0.9877599880293282
55sec OTP dataset:    Accuracy is  0.9865091601668784
60sec OTP dataset:    Accuracy is  0.986813009875483
Average accuracy of coarse Granular OTP dataset 0.9865602304899339
Average Accuracy  0.9905605903699297
```

3.11 Medium K Nearest Neighbors (M_KNN) : Fig. 3.11.Output of M_KNN

```
5sec OTP dataset:    Accuracy is  0.9906654110456519
10sec OTP dataset:    Accuracy is  0.9866535427062827
15sec OTP dataset:    Accuracy is  0.98162472153444
20sec OTP dataset:    Accuracy is  0.9809213049522988
Average accuracy of fine Granular OTP dataset 0.9849662450596685
45sec OTP dataset:    Accuracy is  0.9736051241576102
50sec OTP dataset:    Accuracy is  0.9749289241358671
55sec OTP dataset:    Accuracy is  0.9743606022129512
60sec OTP dataset:    Accuracy is  0.9743398454272219
Average accuracy of coarse Granular OTP dataset 0.974308623983412
Average Accuracy  0.9796374345215406
```

3.12 Coarse K Nearest Neighbors (Coa_KNN) : Fig. 3.12.Output of Coa_KNN

```
5sec OTP dataset:    Accuracy is  0.9756409795478129
10sec OTP dataset:    Accuracy is  0.9726896488577577
15sec OTP dataset:    Accuracy is  0.9662223862170756
20sec OTP dataset:    Accuracy is  0.9667355817632675
Average accuracy of fine Granular OTP dataset 0.9703221490964784
45sec OTP dataset:    Accuracy is  0.9626191488069941
50sec OTP dataset:    Accuracy is  0.9596176866676641
55sec OTP dataset:    Accuracy is  0.9608697623798295
60sec OTP dataset:    Accuracy is  0.9633694718763419
Average accuracy of coarse Granular OTP dataset 0.9616190174327073
Average Accuracy  0.9659705832645928
```

3.13 Cosine K Nearest Neighbors (Cos_KNN) : Fig. 3.13.Output of Cos_KNN

```
5sec OTP dataset:    Accuracy is  0.9937344647540917
10sec OTP dataset:    Accuracy is  0.9901139953303117
15sec OTP dataset:    Accuracy is  0.9877485499099945
20sec OTP dataset:    Accuracy is  0.986789874874155
Average accuracy of fine Granular OTP dataset 0.9895967212171384
45sec OTP dataset:    Accuracy is  0.97966425839354
50sec OTP dataset:    Accuracy is  0.9786061648960048
55sec OTP dataset:    Accuracy is  0.9750272084164701
60sec OTP dataset:    Accuracy is  0.9758265349935595
Average accuracy of coarse Granular OTP dataset 0.9772810416748936
Average Accuracy  0.983438881446016
```

3.14 Cubic K Nearest Neighbors (Cu_KNN) : Fig. 3.14.Output of Cu_KNN

```
5sec OTP dataset:    Accuracy is  0.9898345358423963
10sec OTP dataset:    Accuracy is  0.9859117337362082
15sec OTP dataset:    Accuracy is  0.9834813884999759
20sec OTP dataset:    Accuracy is  0.9796952154945107
Average accuracy of fine Granular OTP dataset 0.9847307183932728
45sec OTP dataset:    Accuracy is  0.97416671726064
50sec OTP dataset:    Accuracy is  0.9755237168936107
55sec OTP dataset:    Accuracy is  0.9770587701795757
60sec OTP dataset:    Accuracy is  0.9736099184199227
Average accuracy of coarse Granular OTP dataset 0.9750897806884373
Average Accuracy  0.979910249540855
```

3.15 Weighted K Nearest Neighbors (W_KNN) : Fig. 3.15.Output of W_KNN

```
5sec OTP dataset:    Accuracy is  0.992967160471634
10sec OTP dataset:    Accuracy is  0.9906081276991866
15sec OTP dataset:    Accuracy is  0.9875644005489995
20sec OTP dataset:    Accuracy is  0.9867904741358646
Average accuracy of fine Granular OTP dataset 0.9894825407139212
45sec OTP dataset:    Accuracy is  0.9802076376662013
50sec OTP dataset:    Accuracy is  0.9798107137513092
55sec OTP dataset:    Accuracy is  0.9824641755849809
60sec OTP dataset:    Accuracy is  0.979443967367969
Average accuracy of coarse Granular OTP dataset 0.9804816235926151
Average Accuracy  0.9849820821532682
```

3.16 Random Forest Classifier (RFC) : Fig. 3.16.Output of RFC

```
5sec OTP dataset:    Accuracy is  0.9949492985136825
10sec OTP dataset:    Accuracy is  0.9930790947519419
15sec OTP dataset:    Accuracy is  0.9914642977840004
20sec OTP dataset:    Accuracy is  0.9904603528452945
Average accuracy of fine Granular OTP dataset 0.9924882609737298
45sec OTP dataset:    Accuracy is  0.9873535304474531
50sec OTP dataset:    Accuracy is  0.98469998503666
55sec OTP dataset:    Accuracy is  0.9865091601668784
60sec OTP dataset:    Accuracy is  0.9868022756547873
Average accuracy of coarse Granular OTP dataset 0.9863412378264447
Average Accuracy  0.9894147494000872
```

3.17 Logistic Regression (LR): Fig. 3.17.Output of LR

```
5sec OTP dataset:    Accuracy is  0.9654113724945457
10sec OTP dataset:    Accuracy is  0.9608246730455218
15sec OTP dataset:    Accuracy is  0.9597247415356815
20sec OTP dataset:    Accuracy is  0.9601281221535067
Average accuracy of fine Granular OTP dataset 0.9615222273073141
45sec OTP dataset:    Accuracy is  0.9626191488069941
50sec OTP dataset:    Accuracy is  0.9620866377375432
55sec OTP dataset:    Accuracy is  0.9635452566660619
60sec OTP dataset:    Accuracy is  0.9625912408759124
Average accuracy of coarse Granular OTP dataset 0.9627105710216279
Average Accuracy  0.962116399164471
```

3.18 Boosted Gradient Descent (BGD): Fig. 3.18.Output of BGD

```
5sec OTP dataset:    Accuracy is  0.9975703733361658
10sec OTP dataset:    Accuracy is  0.9954278258480977
15sec OTP dataset:    Accuracy is  0.9955462752860521
20sec OTP dataset:    Accuracy is  0.992416942327053
Average accuracy of fine Granular OTP dataset 0.9952403541993422
45sec OTP dataset:    Accuracy is  0.9873504948090585
50sec OTP dataset:    Accuracy is  0.9865217716594344
55sec OTP dataset:    Accuracy is  0.9878514420460729
60sec OTP dataset:    Accuracy is  0.9860562473164448
Average accuracy of coarse Granular OTP dataset 0.9869449889577526
Average Accuracy  0.9910926715785474
```

4. CONCLUSION AND FUTURE PLAN

4.1 Conclusion:

Difference ML classification algorithms and its different variants are used to get good classification accuracy for fine granular OTP network traffic data. Results of the ML classifier are listed in table 1,2,3,4.

In this paper, validation is done by K-fold technique. K-fold validation is a validation technique in which Total Dataset is divided into a K number of pieces or fold and one of the pieces is selected as a test Dataset . This is used to avoid overfitting of models. Here we used 10 fold validation in the Dataset. BGD ML algorithm works well for fine granular OTP traffic.RFC, FG_SVM, BGD ML algorithms works well for Coarse granular OTP traffic. Likewise, F_KNN reaches the highest precision for 5, 10 and 15 Sec OTP-based Datasets, then FG_SVM produces the better Dataset s accuracy of 99.4 percent for 20 sec OTP-based Datasets. From all observations of the average accuracy plots of the various classifiers, it's also evident that the BGD displays the best average accuracy for both FGT and CGT data subsets which is shown in fig. 4.1.

4.2 Fine Granular OTP:

Table 4.1 ML classifier accuracy for FGT in DT

Dataset	F_DT	M_DT	C_DT
5 sec OTP Dataset	99.4	99.4	96.2
10 sec OTP Dataset	99.3	99.2	97.2
15 sec OTP Dataset	99.1	98.9	97.0
20 sec OTP Dataset	99.0	98.8	97.0
Average	99.2	99.1	96.8

Table 4.2 ML classifier accuracy for FGT in SVM

Dataset	L_SVM	Q_SVM	C_SVM	FG_SVM	MG_SVM	CG_SVM
5 sec OTP Dataset	96.5	96.5	97.1	99.6	99.4	98.8
10 sec OTP Dataset	96.1	96.3	97.1	99.4	99.3	98.8
15 sec OTP Dataset	96.1	96.5	96.7	99.3	99.2	98.7
20 sec OTP Dataset	96.2	96.3	96.3	99.4	99.2	98.7
Average	96.3	96.4	96.8	99.4	99.3	98.7

Table 4.3 ML classifier accuracy for FGT in KNN

Dataset	F_KNN	M_KNN	Coa_KNN	Cos_KNN	Cu_KNN	W_KNN
5 sec OTP Dataset	99.6	99.0	97.5	97.3	98.9	99.2
10 sec OTP Dataset	99.4	98.6	97.2	99.0	98.5	99.0
15 sec OTP Dataset	99.3	98.1	96.6	98.7	98.3	98.7
20 sec OTP Dataset	99.4	98.0	96.6	98.6	97.9	98.6
Average	99.4	98.4	97.0	98.9	98.4	98.9

Table 4.4 ML classifier accuracy for FGT

Dataset	RFC	LR	BGD
5 sec OTP Dataset	99.4	96.5	99.7
10 sec OTP Dataset	99.3	96.0	99.5
15 sec OTP Dataset	99.1	95.9	99.5
20 sec OTP Dataset	99.0	96.0	99.2
Accuracy	99.2	96.1	99.5

4.3 Coarse Granular OTP:

Table 4.5 ML classifier accuracy for CGT in DT

Dataset	F_DT	M_DT	C_DT
45 sec OTP Dataset	98.0	98.0	96.2
50 sec OTP Dataset	97.8	98.4	96.9
55 sec OTP Dataset	97.6	97.3	97.0
60 sec OTP Dataset	98.0	97.6	96.9
Average	97.8	98.1	96.7

Table 4.6 ML classifier accuracy for CGT in SVM

Dataset	L_SVM	Q_SVM	C_SVM	FG_SVM	MG_SVM	CG_SVM
45 sec OTP Dataset	96.3	95.8	95.3	98.5	98.8	97.8
50 sec OTP Dataset	96.1	95.4	96.3	98.7	98.3	97.8
55 sec OTP Dataset	96.3	95.1	95.3	98.6	98.6	98.0
60 sec OTP Dataset	96.2	95.2	95.5	98.6	97.8	97.7
Average	96.2	95.4	95.6	98.6	98.4	97.8

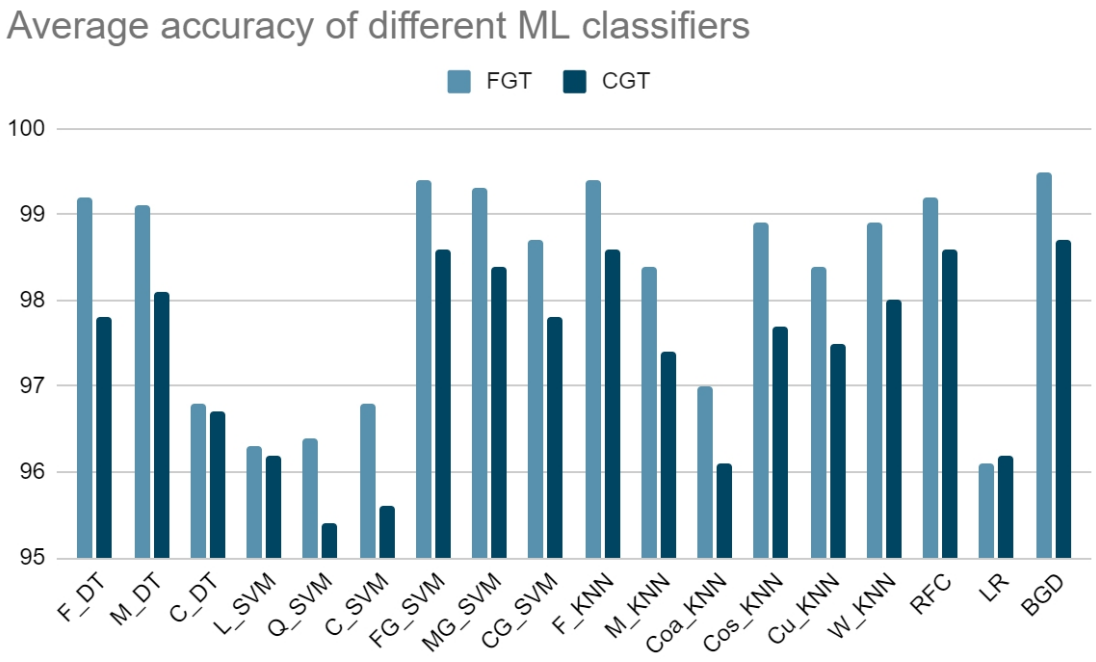
Table 4.7 ML classifier accuracy for CGT in KNN

Dataset	F_KNN	M_KNN	Coa_KNN	Cos_KNN	Cu_KNN	W_KNN
45 sec OTP Dataset	98.5	97.3	96.2	97.9	97.4	98.0
50 sec OTP Dataset	98.7	97.4	95.9	97.8	97.5	97.9
55 sec OTP Dataset	98.6	97.4	96.0	97.5	97.7	98.2
60 sec OTP Dataset	98.6	97.4	96.3	97.5	97.3	97.9
Average	98.6	97.4	96.1	97.7	97.5	98.0

Table 4.8 ML classifier accuracy for CGT

Dataset	RFC	LR	BGD
45 sec OTP Dataset	98.7	96.2	98.7
50 sec OTP Dataset	98.4	96.2	98.6
55 sec OTP Dataset	98.6	96.3	98.7
60 sec OTP Dataset	98.6	96.2	98.6
Accuracy	98.6	96.2	98.7

Fig. 4.1.Average accuracy of different ML classifiers



4.4 Future Plan

Long distance network traffic is routed by help of WMN.WMN is monitored at the central point by SDN and this SDN provides the suitable traffic engineering method to control the congestion in the network.This paper suggests the architecture of the SDWMN to classify multi-granular IoT network traffic to be efficient Management of the network. All ML trained model are stored in the controller node. These trained models change or update the network routing table or flow tables periodically.

Accuracy is taken as a main criteria for the selection of ML classification algorithms.Trained ML model must be trained with a new Dataset to ensure that models are updated and evolved. These training with new Dataset should be done periodically.

Fig. 4.2.Maximum and minimum accuracies of different ML classifiers for FGT Dataset

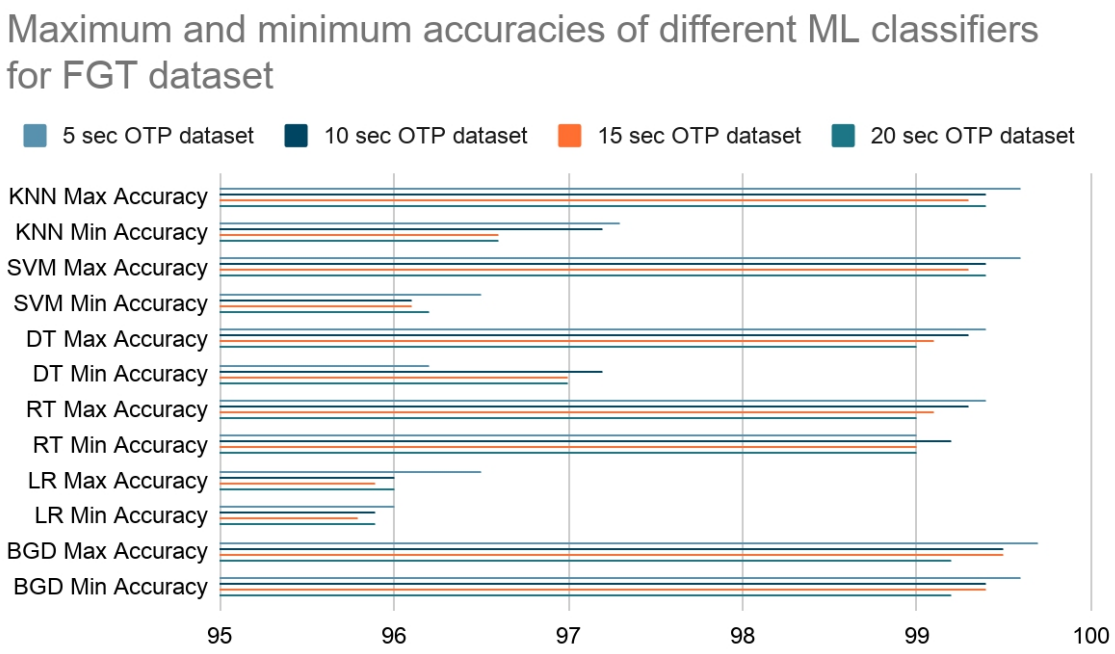
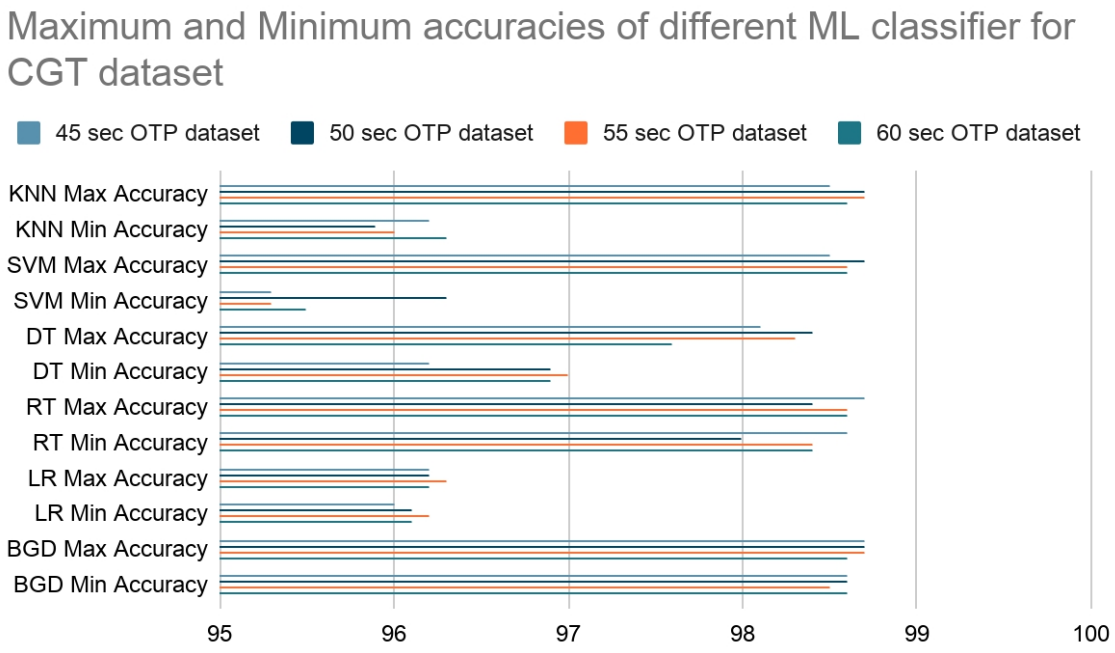


Fig. 4.3.Maximum and minimum accuracies of different ML classifiers for CGT Dataset



5. REFERENCE

[1] Rademacher, Michael, Karl Jonas, Florian Siebertz, Adam Rzyska, Moritz Schlebusch, and Markus Kessel. ”Software-defined wireless mesh networking: Current status and challenges.” The Computer Journal,Vol. 60, No. 10, pp. 1520-1535, Jul. 2017.

[2] Conti, Marco, Chiara Boldrini, Salil S. Kanhere, Enzo Mingozzi, Elena Pagani, Pedro M. Ruiz, and Mohamed Younis. ”From MANET to people-centric networking: Milestones and open research challenges.” Computer Communications, Vol. 71, No. 1, pp. 1-21, Nov. 2015.

[3] Thupae, Ratanang, Bassey Isong, Naison Gasela, and Adnan M. Abu Mahfouz. ”Machine Learning Techniques for Traffic Identification and Classification in SDWMN: A Survey.” In proceedings of the IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, pp. 4645-4650, Oct. 2018.

[4] Wang, Wei, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. ”Malware traffic classification using convolutional neural networks for representation learning.” In proceedings of the International Conference on Information Networking (ICOIN), pp. 712-717, Jan. 2017.

[5] Ducange, Pietro, Giuseppe Mannara, Francesco Marcelloni, Riccardo ` Pecori, and Massimo Vecchio. ”A novel approach for internet traffic classification based on multi-objective evolutionary fuzzy classifiers.” In proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1-6, Jul. 2017.

[6] Mir, Faisal Ghias, Marcus Brunner, Rolf Winter, and Dirk Kutscher. ”User traffic accountability under congestion in flow-based multilaye switches.” U.S. Patent 9,231,876,

[7] Vladut u, Alina, Dragos Com aneci, and Ciprian Dobre. ”Internet traffic classification based on flows’ statistical properties with machine learning.” International Journal of Network Management, Vol. 27, No.3, pp-19-27, May 2017.

[8] Ding, Tao, Ahmed AlEroud, and George Karabatis. ”Multi-granular aggregation of network flows for security analysis.” In proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 173-175, May 2015.

[9] Bartos, Karel, and Michal Sofka. ”Identifying threats based on hierarchical classification.” U.S. Patent 9,800,597, issued October 24, 2017.

[10] https://github.com/merimak/802_15_4MACperf_Datasets“ 802_15_4_MAC_perf Dataset s”

[11] https://ieeexplore.ieee.org/document/9027336 “A Binary Classification Approach for Time Granular Traffic in SDWMN based IoT Networks”