B.E. PROJECT ON

RF Jamming Classification in Wireless Ad-Hoc Networks using Machine Learning

Submitted By

ANSH JAIN(712IT15)

G.S KASTURI(727IT15)

JAHNVI ARYA(732IT15)

Under The Guidance Of

Prof. SANJAY KUMAR DHURANDHER

A Project in partial fulfillment of requirement for the award of

B.E. in **Information Technology**

Department of Information technology , (NSIT)

NEW DELHI-110078

2019

SELF DECLARATION

The project entitled “RF jamming classification in wireless ad-hoc networks” is a bonafide work carried out by Ansh Jain, G S Kasturi, Jahnvi Arya in Department of Information Technology, Netaji Subhas University of Technology under the supervision and guidance of Prof. Sanjay Kumar Dhurandher in partial fulfilment of the requirement for the Degree of Bachelor of Engineering in Information Technology, University of Delhi for the year 2018-19.

The content of the report is to the best of our knowledge and hasn’t been used to the best of our Activity.

ANSH JAIN (712IT15)
G.S KASTURI(727IT15)
JAHNVI ARYA(732IT15)

नेताजी सुभाष प्रौद्योगिकी संस्थान
NETAJI SUBHAS INSTITUTE OF TECHNOLOGY
(An Institution of Govt. of NCT of Delhi-Formerly, Delhi Institute of Technology)
Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi - 110 078
Telephone : 25099050; Fax : 25099025, 25099022 Website : http://www.nsit.ac.in

CERTIFICATE

This is to certify that report entitled "**RF Jamming Classification in Wireless ad-hoc Networks using Machine Learning"** being submitted by Ansh Jain (712IT15), G.S.Kasturi (727IT15), and Jahnvi Arya(732IT15), to the Division of Information Technology, NSIT, for the award of bachelor's degree of engineering, is the record of the bonafide work carried out by them under my supervision and guidance. The results contained in this report have not been submitted by either in part or in full to any other university or institute for the award of any degree or diploma.

Supervisor

Prof Sanjay Kumar Dhurandher
Department of IT
NSIT, New Delhi, India
G.S KASTURI(727IT15)
JAHNVI ARYA(732IT15)

ACKNOWLEDGEMENT

B.E. project is a part of curriculum of final year and plays a pivotal role in the overall grooming by providing real time project experience. We have tried to inculcate all that we have learnt from various experienced professors. We wish to express our sincere regards to individuals who have been a great support and motivation throughout the project tenure and helped us perform better.

We would extend our sincere thanks to Prof. Sanjay Kumar Dhurandher for giving us an opportunity to work under her guidance and encouraging us at the same time to innovate and come up with fresh ideas. We would also like to thank each other for putting a collective effort to take project at its final stage.

We wish to place on record our gratitude towards NSIT for providing us such an opportunity to work on a project of such practical considerations. Working on this project was a great learning experience. This exposure has enriched us with knowledge and has also introduced us to the attributes of a successful IT engineer.

ANSH JAIN (712IT15)
G.S KASTURI(727IT15)
JAHNVI ARYA(732IT15)

INDEX

ABSTRACT

Wireless networking plays a vital role in achieving ubiquitous computing where network devices embedded in environments provide continuous connectivity and services, thus improving quality of life of humans. However, current wireless networks can be easily attacked by jamming technology since wireless links are opened and shared. An attacker exploits the limitations in wireless protocols at different layers and disrupt the existing wireless communication by decreasing the signal-to-noise ratio at receiver sides through the transmission of interfering wireless signals. Thus, the design of accurate jamming detection algorithms becomes important to react to ongoing jamming attacks.

Jamming causes Denial-of-Service (DoS) problem which may result in several other higher-layer security problems. Moreover, Jamming is a well-known threat to reliability for wireless networks. With the rise of safety-critical applications, jamming attack is likely to become a constraining issue in the future. Hence, detection of jamming attack and its classification is of great importance in order to take suitable countermeasures.

We therefore propose jamming detection and classification techniques which are based on Machine Learning . We analyse previously used jamming strategies and evaluate them with various performance metrics for jamming detection . In order to collect data , We simulate jamming in Wifi Ad-Hoc network using the Ns3 jamming model. The effectiveness of our model is then evaluated and compared.

LIST OF FIGURES

LIST OF TABLES

ORGANISATION OF THESIS

Chapter 1: Introduction

Presents an introduction to wireless networks, discusses the problem of RF jamming and provides the motivation for our work.

Chapter 2: Literature Survey

Presents a literature survey on wireless networks, RF jamming and machine learning techniques used in our project. It encompasses a comprehensive view of NS3 and the machine learning techniques.

Chapter 3: System Design

It presents our problem statement formally and provides the details of implementation of jamming in wireless ad-hoc networks in NS3 and how the data was collected.

Chapter 4: Experiments and Results

It presents the details of different experiments that were conducted and also the quantitative results obtained from each algorithm. This section also presents different graphs and their analysis.

Chapter 5: Conclusion and Future Work

It presents the conclusion to our project and the scope of further research that can be carried out in this topic.

List of References

Appendix

Chapter 1
INTRODUCTION

Wireless networks are the most popular communication technology in today's world, and with this popularity wireless network's credibility and security is becoming a serious issue. Attackers now-a-days utilise advanced jamming strategies by exploiting the shortcomings of PHY and MAC layer protocols. There can be multiple types of attacks, attacker can transmit a permanent signal so that the packets are always jammed (i.e constant jammer) or inject packets at random intervals of time called random jammer. Attacker can also exploit semantics of the protocol layers like ACK packets, DATA packets and launch intelligent jamming strategy. Therefore, malicious attackers can prevent users from performing operations on the MAC layer and also block transmissions.

1.1 CURRENT SCENARIO

Many countermeasures have been proposed in previous works to resolve the problem of RF jamming using different techniques. Example, Frequency-Hopping Spread Spectrum[1] (FHSS) , Direct Sequence Spread Spectrum[2] (DSSS) , and Hybrid FHSS/DSSS techniques applied at the PHY layer. These techniques are widely adopted to avoid jamming interference. Also spread spectrum technique is used to expand narrow band signal into a wider band for resisting interference which may be intentional or unintentional. The drawback of these techniques is that bandwidth is wasted. Other techniques used include Ultra Wideband (UWB) techniques, multi-antenna techniques etc. Techniques like CSMA/CA , TDMA are also used to address multi-channel jamming issue.

[1] "Compressive Sampling for Detection of Frequency-Hopping Spread ...." 2 Aug. 2016, https://ieeexplore.ieee.org/document/7529084. Accessed 27 May. 2019.
[2] "Design and construction of Direct Sequence Spread Spectrum CDMA ...." http://ieeexplore.ieee.org/abstract/document/4803054/. Accessed 27 May. 2019.

Therefore we propose machine learning based techniques which can easily detect the jammer type with minimal overhead.

1.2 MOTIVATION

The motivation for this project is to protect the privacy of users and prevent them from being prone to jamming attacks. We aim to classify different jammer types so that specific and appropriate countermeasures can be adopted. Furthermore it is important to understand the jamming techniques in depth to prevent them so we also provide graphs for data analysis. For this purpose we use RSS and PDR values as it is important to extract metrics from different layers (PDR is from application layer and RSS from PHY layer). We finally design a detection method by incorporating the information from different layers. The method is based on machine learning algorithms which we analyse in detail. Our work lays the foundation for further research in devising techniques to prevent jamming.

Chapter 2

LITERATURE SURVEY

2.1 AN INTRODUCTION TO WIRELESS NETWORKING

Wireless networking is used by telecommunications, homes and businesses to avoid the process of introducing cables in a building, or to connect various equipments. This is required as wires are expensive, hard to manage and take up a lot of space. Radio communication are used to generally implement and administer Wireless telecommunications network. This implementation takes place at the physical layer of the OSI network architecture.

Examples of wireless networks -

1) cell phones networks,
2) wireless local area networks (WLANs),
3) wireless sensor net[3]works,
4) satellite communication networks.

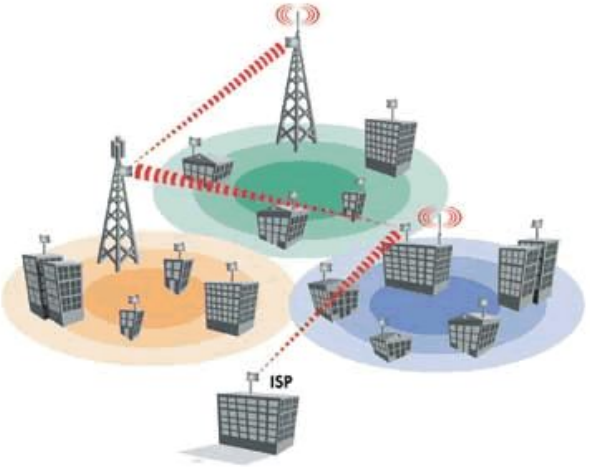

Fig 1:Wireless Networks [3]

[3] http://www.tech-faq.com/wireless-networks-and-their-types.html

2.1.1 WIRELESS LAN

A Wireless Local Area Network (WLAN) uses high frequency radio waves for communication between different devices and is a type of Local Area Network.

2.1.2 ACCESS POINT

Wireless communication devices, like PDAs, mobile computers etc use hardware devices called access points (AP) to connect to a wireless network. An AP usually provides a bridge for data communication between wired and wireless devices by connecting to a wired network.

2.1.3 AD-HOC MODE

Ad-hoc mode is a networking topologies which is provided in the 802.11 standard. It involves communication where no access point is involved and at least 2 wireless stations are used. Ad-hoc mode WLANs do not need APs for communication and so they are usually less expensive to run. This topology however, lacks security features like access control and MAC filtering and cannot scale for large networks.

2.1.4 INFRASTRUCTURE MODE

Infrastructure mode is a network topology which is also provided in 802.11 standard. It has a large number of access points and wireless stations. Usually the access points connect to large wired networks. This network topology overcomes one of the disadvantages of Ad-hoc as it can scale to large-scale highly complex networks with arbitrary coverage.

2.2 SECURITY IN WIRELESS NETWORKS

2.2.1 JAMMING ATTACK

Jamming attack is one of the most common type of Denial of Service attack . This attack occupies the channel on which nodes are communicating and does not allow nodes from using the channel for communication.. A jammer in wireless network is defined as an entity which purposely tries to interfere with the transmission and reception of messages via wireless communication.

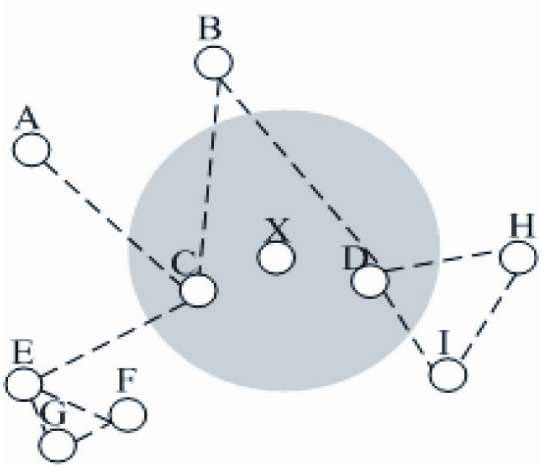

Fig 2 Example of jamming attack [4]

(the nodes C and D have been jammed by node X which is malicious, so communication between(C, D) which are jammed nodes and (A, B, E, H, I) which are normal nodes is disrupted.)

2.2.2 SOLUTIONS FOR JAMMING

The most effective method that an attacker may use to bre[4]ach the reliability of our communication network is flooding all wireless stations by random packets which are unauthenticated. Jamming attack is very easy to perform. It can easily be performed by using

[4] https://krazytech.com/technical-papers/jamming-and-anti-jamming-techniques

A hardware that is easily available at an electronics store and a software which can readily be downloaded from the internet. Attackers who are experienced may able to jam every possible network frequency which may be impossible to defend .

The simplicity of the jamming attacks and its possible repercussions makes the detection and prevention of jamming attacks very important . The wireless communication systems used should not only be able to detect the authorized users but also prevent unauthorized users from accessing the the system or making any modification in the system. The wireless communication system should also be able to pinpoint the physical location of the malicious devices .
Therefore, a wireless network should be able to identify RF jamming and take necessary counter measures accordingly in order to maintain the network performance.

2.3 JAMMING ATTACKS MODELS

2.3.1 CONSTANT JAMMERS

A constant jammer is a jammer that produces high-power noise which represents random bits continuously. The bit generator in a constant jammer works independent of the channel sensing or the traffic on the channel and does not follow any MAC protocol.

2.3.2 RANDOM JAMMERS

A random jammer is a type of jammer that operates in sleep and jam intervals randomly and does not follow any MAC protocol.It can either act as a constant jammer or reactive jammer during the jam interval and sleeps during the sleep interval irrespective of any traffic on the network.

2.3.3 REACTIVE JAMMERS

A reactive jammer becomes active when it senses transmission on the channel. It transmit noise when there is transmission on the channel which corrupts some number of bits in a legitimate packet .Due to this, a receiver cannot recover the checksum and the packet is discarded causing drop in the PDR .

2.4 JAMMING DETECTION METRICS

2.4.1 PACKET DELIVERY RATIO

It is the ratio of the total number of packets correctly received that is that pas cyclic redundancy check to the total number of packets received. The Packet Delivery Ratio is measured at the receiver side for an environment having noise and interference.

2.4.2 CARRIER SENSING TIME

It is the time a station has to wait for the channel to get idle in order to start its transmission.

2.4.3 RECEIVED SIGNAL STRENGTH (RSS)

The signal power that is observed on the receiver end is RSS. Two approaches that are used to characterize variation in signal strength are: (1) average signal strength value in time window and (2) spectral discrimination technique.

2.4.4 NOISE

Noise is measured as the signal strength on an idle channel i.e when there is no transmission. This measured strength is recorded as noise of a jamming signal if a jammer is jamming the channel.

2.5 NS3 NETWORK SIMULATOR

Network simulator is a tool that is used for simulating real world networks on a computer by writing scripts in C++ or Python. Normally if we want to perform experiments ,we don't have required number of computers and routers for making different topologies , to see how our network works using various parameters. It is very expensive to build such a network for experiment purposes even if we have these resources.

So we used NS3 to overcome these drawbacks. NS3 helps to create various virtual nodes (i.e., computers in real life) and it allows us to install devices, internet stacks, application, etc to our nodes with the help of various Helper classes.

We can create PointToPoint, Wireless, CSMA, etc connections between nodes using NS3. A PointToPoint connection is the same as a LAN connection between two computers.

A Wireless connection similar to a WiFi connection between different computers and routers.

CSMA connection is similar to a bus topology between computers. We try to install NIC to every node to enable network connectivity after building connections.

We add different parameters in the channels (i.e., real world path used to send data) when network cards are enabled in the devices, which are data-rate, packet size, etc. Then we use Application to generate traffic and send packets using the same applications.

Ns3 gives us special features which can be used for real life integrations. Some of these features are listed as follows:

- **Tracing of the nodes:**

  NS3 allows the tracing of the routes of the nodes which helps in knowing how much data is send or received. Trace files are used to monitor these activities.
- **NetAnim:**

  It stands for Network Animator. It is an animated version of the real network and shows how data is transferred from one node to other.
- **Pcap file:**

  NS3 is used to generate the pcap file which can be used to get all information of the packets (e.g., Source IP, Sequence number, destination IP, etc). These pcaps are seen using a software tool known as wireshark.
- **gnuPlot:**

  GnuPlot plots graphs from the data obtained from trace file of NS3. Gnuplot is less complex than other tools and gives more accurate graph compare to other graph making tools and also it

2.6 WIRELESS JAMMING MODEL - NS3

- It provides a simulation environment for the researchers for implementing and simulating different jamming detection/mitigation strategy.
- It also enables researchers, to evaluate the performances of jamming detection strategies as well as jamming itself.

This model focuses on modeling Intelligence layer like decisions regarding which packets to jam and when. This model uses existing infrastructures and build upon them example uses existing PHY protocol model to send packets. The goal of this design is to abstract underlying protocols from the higher level layers like the intelligence layer. This is done so that same strategies can be applied to different PHY layers without the need for re writing same code for different types of PHY layers.

2.6.1 WIRELESS JAMMING MODEL HIERARCHY

The wireless jamming model of the following components:

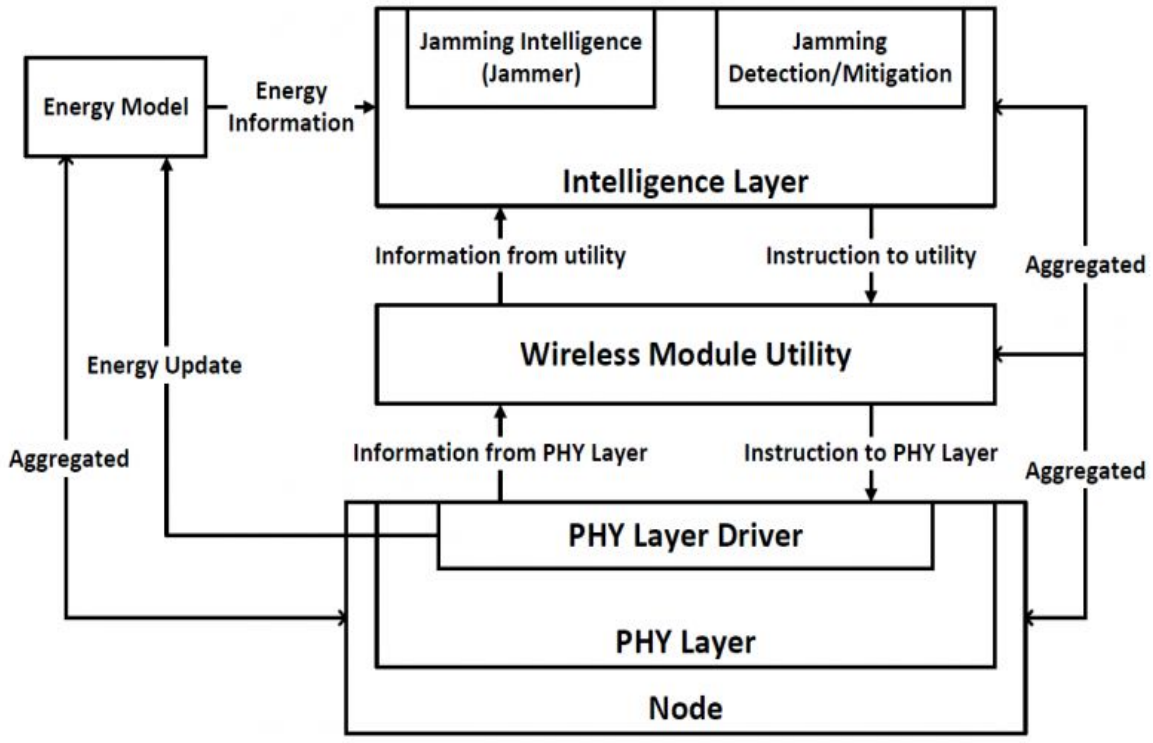

Fig 3: Wireless jamming model heirarchy [5]

- **Mitigation**.
  - wireless module utility derives its interface from this class. It is the base class for wireless module utility
  - To implement detailed jamming mitigation strategies this class can be derived and different child classes can be implemented

[5] https://www.nsnam.org/wiki/File:Ns-3_jamming_model.PNG

- **Jammer (Intelligence Layer)**
    - wireless module utility derives its interface from this class as well. It is the base class for wireless module utility
    - To implement detailed jamming strategies this class can be derived and different child classes for constant, reactive etc jammers can be implemented

- **Utility**.
    - This class provides many functions implementing jamming and jamming mitigation strategies.
    - This class provides abstraction of PHY to intelligence layer and also acts as a bridge between them.

2.6.2 JAMMING INTELLIGENCE (JAMMER)

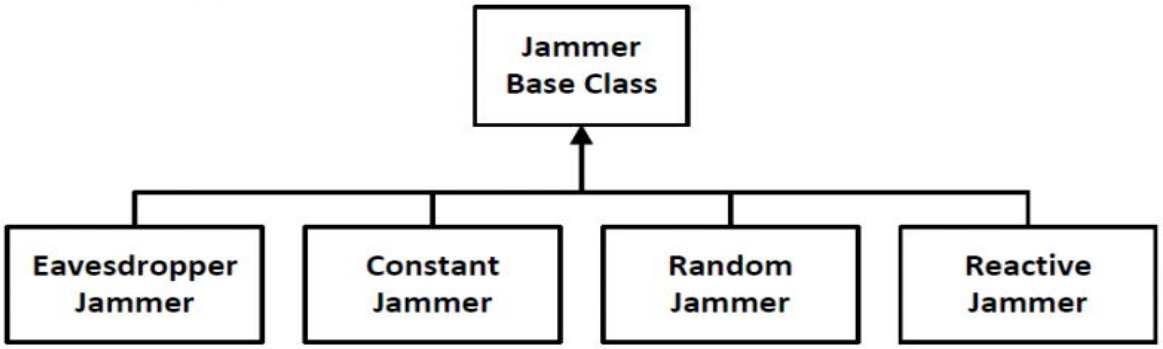

Fig 4: Jammer class structure[6]

The types of jammers provided by the jamming model are as follows:

- **Constant jammer**:

  Constantly sends jamming signal

- **Random jammer**:

  Sends jamming signal at random intervals

[6] https://www.nsnam.org/wiki/File:Ns-3_jammers.PNG

- **Reactive jammer**:

Sends jamming signal whenever it detects a message being sent on the channel We can define our own jamming strategies by following format provided in the classes. The abstraction which is provided by intelligence class hides details of sending jamming signals and information extraction from different channels

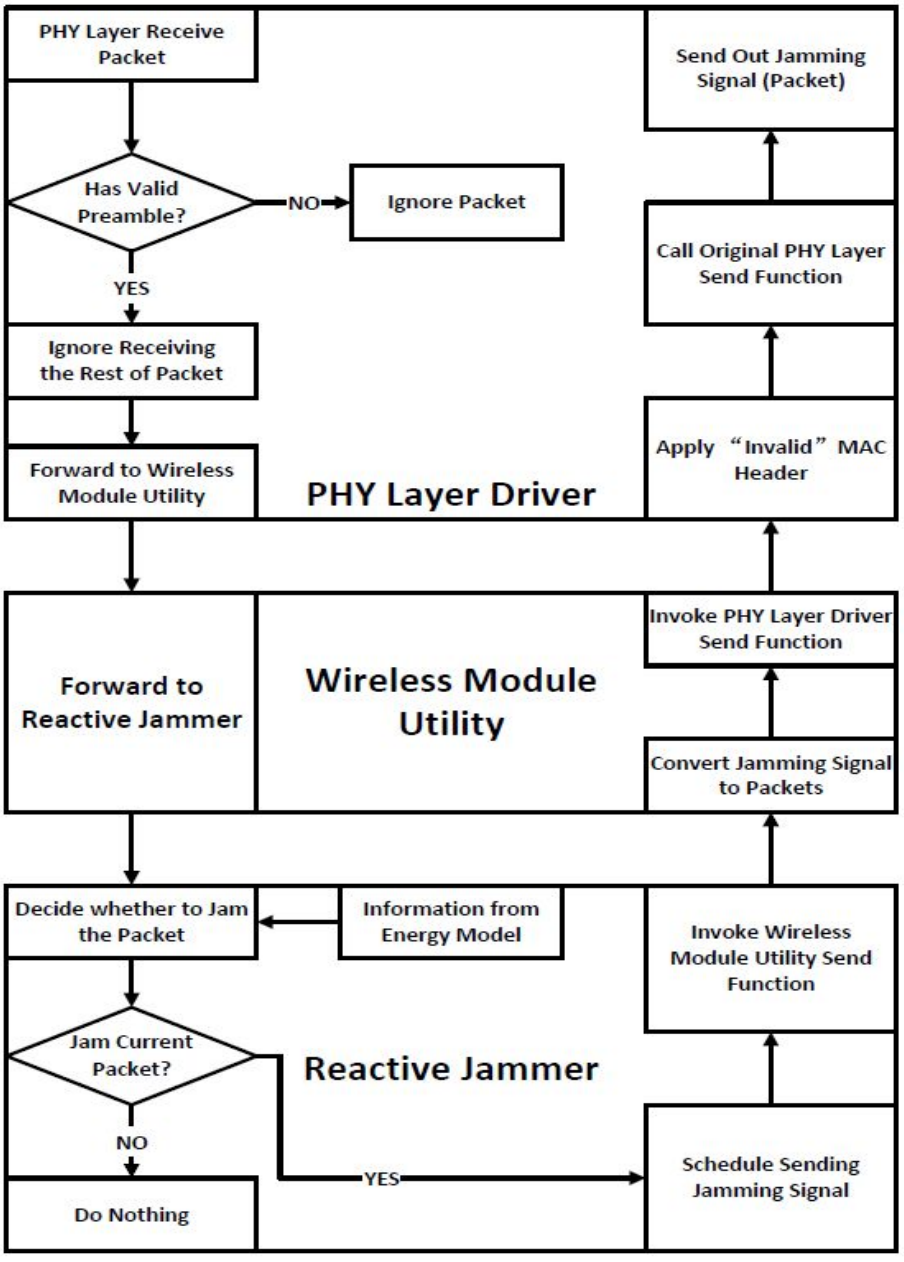

Fig 5: Work Flow of Reactive jammer[7]

[7] https://www.nsnam.org/wiki/File:Ns-3_reactive_jammer_flow.PNG

2.7 MACHINE LEARNING

Machine learning is nothing but an application of artificial intelligence (AI). It provides a system the ability to automatically learn and improve from past experience without the need of being programmed explicitly.

The learning process begins with observations of data. The main aim is to enable the computers learn automatically without any assistance from humans,

2.7.1 SOME MACHINE LEARNING METHODS

Machine learning algorithms can be categorized as supervised or unsupervised.

- **Supervised machine learning algorithms** are used when the data is labeled. These algorithms can apply what has been learned in the past to new data using labeled examples to predict future occurrences. Beginning from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the outputs. After sufficient training, the system is able to provide targets for any new input. The supervised learning algorithm also compares its output with the ground truth value and find errors in order to modify the model in such a way that the error is minimised.
- **Unsupervised machine learning algorithms** are used when the data used to train is neither labeled nor classified. Unsupervised learning algorithm tries to find a hidden structure from unlabeled data. It studies how systems can infer a function to describe a hidden structure from unlabeled data. Correct output cannot be determined using unsupervised learning algorithm. It mainly explores the data and draws inferences from datasets to describe hidden structures from unlabeled data.
- **Semi-supervised machine learning algorithms** use both labeled and unlabeled data for training. They fall somewhere in between supervised and unsupervised learning. They typically use a small amount of labeled data and a large amount of unlabeled data. The systems that use this type of algorithms are able to considerably improve

learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to learn from it. Otherwise, obtaining unlabeled data generally doesn't require additional resources.

- **Reinforcement machine learning algorithm** is a learning algorithm that interacts with its environment by producing actions and discovers errors which are rewards. The most relevant characteristics of reinforcement learning are trial and error search and delayed reward. This type of learning algorithm allows software agents and machines to automatically determine the ideal behavior within a specific context so as to maximize its performance. Simple reward feedback is required for the agent to learn which action is best. This feedback signal is known as the reinforcement signal.

2.8 SUPERVISED LEARNING

Supervised Learning algorithms use labeled data to learn. After understanding the data, the supervised learning algorithm determines which label should be given to the new data based on a pattern and associating the pattern to the unlabeled new data.

Categories of supervised learning ::Classification & Regression

2.8.1 CLASSIFICATION

Classification is a method of determining the class that a dependent belongs to based on the one or more independent variables.

2.8.1.1 K-NEAREST NEIGHBOURS (K-NN)

K-NN algorithm is one of the simplest algorithm used for classification .It predicts the class of a new data point by identifying the classes of nearest neighbour and considering the majority vote as the result .

Advantages: KNN algorithm is very simple to implement .Its is robust against noisy data . It is also very effective when the dataset is large.

Disadvantages: It is difficult to determine a suitable value of K. Moreover, KNN incurs high computation cost .

How does the KNN algorithm work ?

Let us take an example to understand the algorithm .The following represents the spread of green(GS) and red data points(RC).

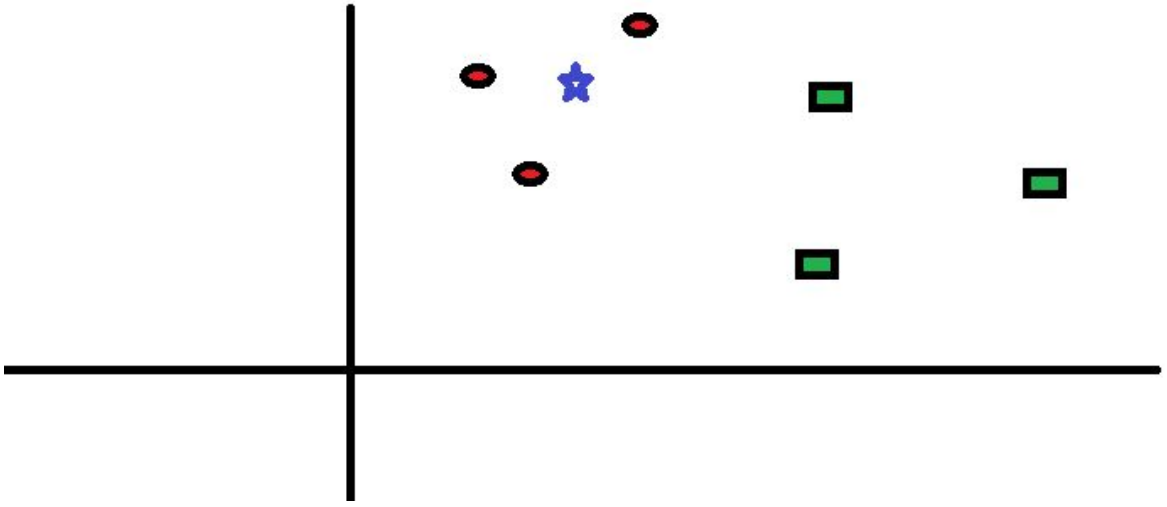

Fig 6 : Working of KNN [8]

We are supposed to find out the class of the blue star(BS). BS can either be GS or RC and nothing else. The "K" in the KNN algorithm is the number of the nearest neighbors we wish to take vote from. Let us assume that K = 3. Now, we now will make a circle with blue star as it's center. The circle should be just big enough to enclose just the three nearest neighbours.
The three closest points to BS are RC. Since the three closest neighbours are RC, so by majority vote we can say BS in RC . The choice of K plays a very important role .

2.8.1.2 RANDOM FOREST

Definition: Random forest is a meta-estimator that fits many decision trees on various sub-samples of a datasets and uses the average to improve accuracy of the model and

[8] https://www.analyticsvidhya.com/wp-content/uploads/2014/10/scenario1.png

controlling over-fitting. The sub-sample size should always be the same as the original input sample size, but samples can be drawn with replacement.

Advantages: Reduction in over-fitting is more accurate than decision trees for most cases.

Disadvantages: Slow real time prediction, is difficult to implement, and a comparitively complex algorithm.

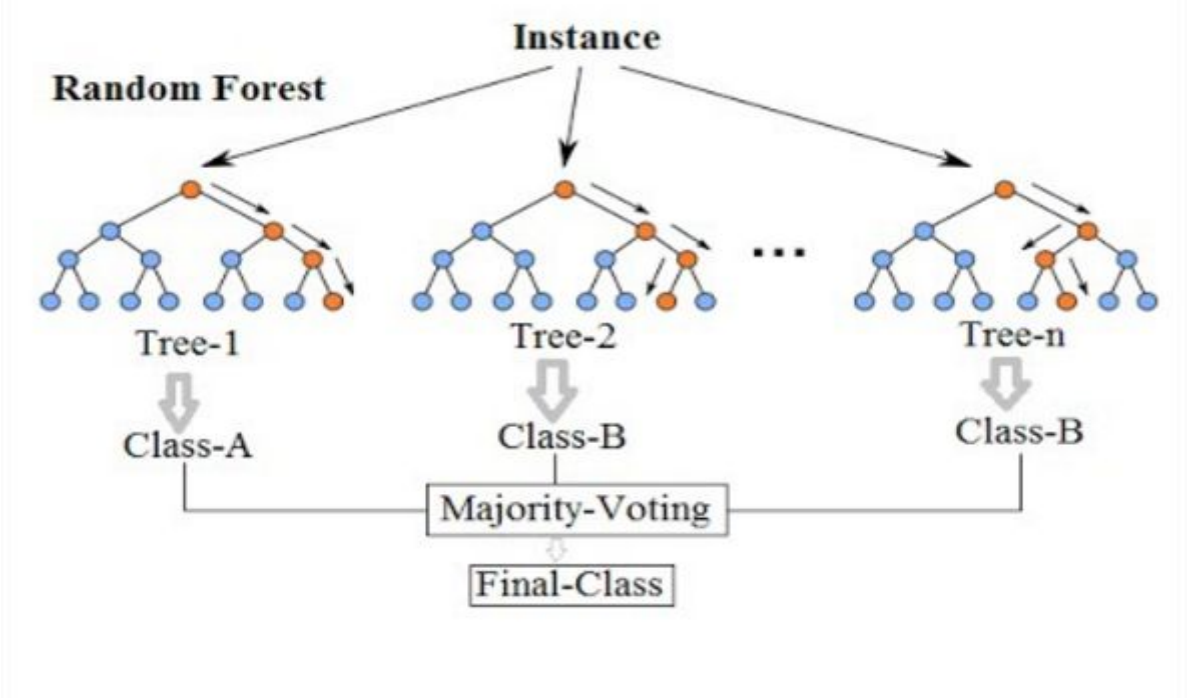

Fig 7 : Random forest[9]

Features of random forest:

- It can run efficiently on large databases.
- It can handle a large number of input variables without deleting variables.
- It gives accurate estimates of the variables which are important in the classification.
- It generates internal unbiased estimates of the generalization error with the progression of forest building.
- It has an effective method for the estimation of missing data and maintaining accuracy when large proportions of the data are missing.
- It has different methods for balancing error in class populated unbalanced data sets.

[9] https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

- Generated forests can be used for other data.

2.8.1.3 DECISION TREE

Decision tree is supervised learning algorithm and is mostly used for classification tasks. This algorithm is designed for both continuous input/output and categorical input/output. Decision tree involves splitting up of data set on the basis on the most desirable splitter in the input features.

Each internal node is used to represent a "test" on an attribute, each branch represents test's outcome, and each leaf node represents a class label. The paths from root node to the leaf node represent the classification rules as shown in figure where a decision is taken at each node until leaf node is reached .

Advantages: Decision trees algorithm is very easy to implement to understand. It can handle both continuous and categorical data. It is very easy to visualise. Moreover, very less data processing is required for implementing decision tree algorithm.

Disadvantages: Decision trees are very unstable in nature since a small alteration in the dataset can entirely change the structure of the tree and create a much more complex structure which might not generalise well.

Common terms used with Decision trees:

1. **Root Node:** Here data gets divided into two or more sets on the basis of most abstract splitting feature.
2. **Splitting:** It is a process in which a node is divided into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.

4. **Leaf/ Terminal Node:** Nodes that do not split is called Terminal or leafnode.
5. **Pruning:** It is a process of of removing sub-nodes of a decision node. It is opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.

2.8.1.4 GRADIENT BOOSTING

Gradient boosting is a machine learning technique for classification and regression problems It produces a prediction model in the form of an ensemble of weak prediction models. These weak prediction models are typically decision trees. The model is built in a stage-wise fashion like various other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

ELEMENTS OF GRADIENT BOOSTING

Gradient boosting algorithm mainly involves three elements:

1. Loss function which needs to be optimized.
2. Weak learner which is used to make predictions.
3. Additive model to add weak learners with the aim of minimizing the loss function.

1. **Loss Function**

Selection of the loss function is dependent on the type of problem which is to be solved. The major criterion for a loss function is that it should be differentiable. There are many Loss functions which are supported like MSE Loss , Cross-Entropy Loss. Moreover, We can define our own loss function. For example, Regression may use MSE loss whereas a classification problem may use logarithmic loss.

The major advantage of the gradient boosting algorithm is that a new boosting algorithm does not have to be derived for each loss function. Instead, it provides a generic framework that any loss function which is differentiable can use.

2. **Weak Learner**

Gradient Boosting trees use decision trees as weak learners. We use Regression trees which output real values for splits and moreover, their outputs can be added together. It allows the outputs of the subsequent models to be added in such a way that the "residuals" in the predictions are corrected. Construction of trees takes place in a greedy manner which involves finding out the best spilt points on the basis of the purity score in order to minimise the loss . Adaboost only uses small decision trees , usually single levels which are called decision stumps . Gradient boosting uses larger tree levels and we can use trees having levels from 3-9 . Weak learners can be constrained by limiting the number of maximum nodes , number of leaf nodes or constraining the number of layers.

3. **Additive Model**

We add the decision trees i.e weak learners one at time sequentially and the already existing trees are not changed. We use Gradient descent algorithm which is used for minimisation of set of parameters lie the trainable weights in a artificial neural network of the coeffients of a given regression equation. The weights are updated in such a way that the loss between the prediction made by the model and the ground truth is minimised.

Chapter 3

SYSTEM DESIGN :

Wireless networks are vulnerable to Radio Frequency (RF) jamming attacks since an attacker can easily emit an interference signal to prevent legitimate access to the medium or disrupt the reception of signal by using various jamming strategies.

Therefore, Our project aims at detecting and classifying various jamming attacks in order to take necessary countermeasures. Our project involves mainly these tasks:

1. Simulation of jamming in Wifi Ad-Hoc Network.
2. Collection of data which is used for training and validating the machine learning algorithms
3. Use Gradient Boosting machine learning algorithm to detect and classify jamming attacks.

3.1 SIMULATION OF JAMMING IN WIRELESS AD-HOC NETWORK :

We use the ns3 jamming model which allows to simulate a wireless jamming strategy and jamming detection/mitigation strategy.

We simulate the jammed wifi ad-hoc network using EnergyModel, Jammer, JammingMitigation, WirelessModuleUtility and NslWifiPhy modules.

Channel configuration:

Channel takes care of getting signal from one device to other devices on the same Wi-Fi channel. The main configurations of WifiChannel are propagation loss model and

propagation delay model .We use Constant Speed Propagation Delay Model and Friis Propagation Loss Model.

Physical Layer Configuration:

The ns3::WifiPhy is an abstract base class representing the 802.11 physical layer functions. WifiPhy takes care of actually sending and receiving wireless signal from Channel. We implement WifiPhy using the NslWifiPhy class. We set Tx gain of the transmitter as -10 dB and Rx gain of the receiver as 1 dB in the initial configuration . We set the energy detection threshold as 0.0 dBm above which the transmitter transmits.

Mac layer Configuration:

We add a non-Qos upper MAC and disable rate control . We disable fragmentation and turn off RTS/CTS for frames below 2200 bytes . We use DSSS and a constant data rate of 1Mbps. We Fix non-unicast data rate to be the same as that of unicast. We configure the wifi architecture as Ad-hoc.

We use the 802.11 b Wifi Standard.

Network Layer Configuration:

We use the InternetStackHelper Class in order to provide implementation of TCP/IPv4 and IPv6-related components. We use IPv4 protocol assign IP addresses to all the nodes .

Transport Layer Configuration:

Each transport protocol implementation is a socket factory. An application that needs a new socket. We use UDP socket factory for the implementation of UDP protocol .

Mobility Configuration:

We use Constant Position Mobility Model in order to set the position of the nodes. The network topology is shown below :

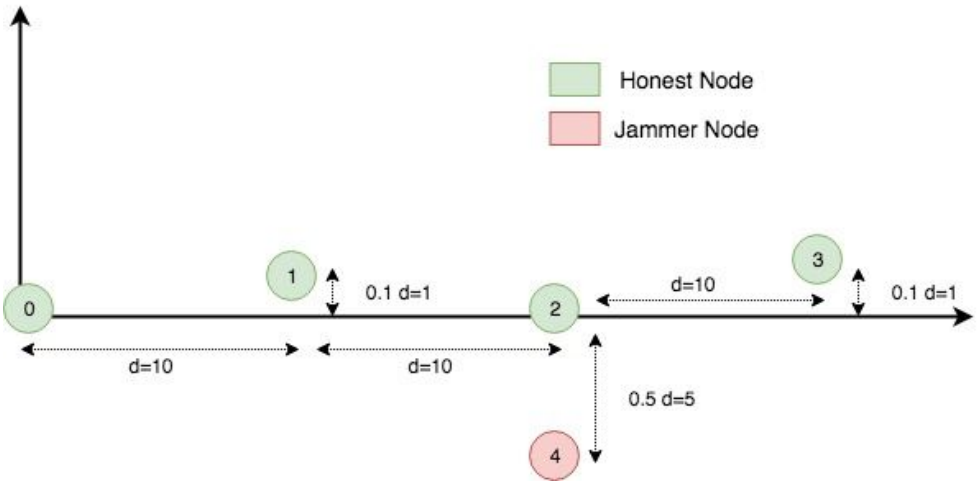

Fig 8: Model Configuration

Our network topology comprises of 5 nodes . Among 5 nodes 1 node is the jammer node and other 4 nodes are honest nodes. Node 0 broadcasts UDP packets to all other nodes. The jammer, that is node 4, tries to jam node 3.

Energy Configuration:

We install an energy source (BasicEnergySource) and energy model (WifiRadioEnergyModel) onto each node. WifiRadioEnergyModel represents energy model for Wifi radio devices . We set the inital energy of basic energy source installed on each device as 0.1 J. We set the current draw of the Wifi radio devices as 0.0174 A.

Installation of Wireless module Utility:

Wireless Module Utility class provides essential functions for jamming intelligence and jamming detection/mitigation intelligence to operate. It can also be installed separately for monitoring network performance such as throughput.

We install Wireless Module Utility onto every node for monitoring packet delivery ratio and received signal strength for the receiver.

Installation of Jammer :

Jammer class is used for sending jamming signal of a certain duration. It can be used to implement Eavesdropper jammer, Constant jammer, Random jammer and Reactive jammer

We use Jammer class to install jammer onto node 4. We change the jammer type by using the command :

jammerHelper.SetJammerType ("ns3::ConstantJammer")

We also set other attributes like jamming duration, Constant jammer constant interval etc.

3.2 CREATION OF DATASET:

We use packet delivery rate and Received signal strength at the receiver end for jamming detection scheme. These metrics can easily be obtained from existing hardware and can be measured at the receiver alone whereas metrics like noise need information about the sender as well. The RSS can be measured easily at the receiver's end and PDR can be calculated as the ratio of correctly received packets to the total number of preambles received.

Unlike the signal strength and carrier sense time, the PDR is calculated in a sliding window, that is , the packet delivery rate is updated once a packet is successfully received. If no packet is received within the sliding window, the PDR within the window is zero.

Some researchers have pointed out that using only PDR as a metric is sufficient in order to detect a jamming attack. This might be because in case of a jamming attack PDR reduces significantly. But, they show that the PDR can still be as high as 78% in congested networks, whereas it is greatly reduced if receiver has poor link quality.

Therefore both PDR and RSS need to be considered for classification of jammers, using only one of them would not yield good results.

We vary various simulation parameters in order to collect Packet Delivery Rate (PDR) and Received Signal Strength (RSS) for different jamming situations :

Distance between the nodes :

We vary the distance between the nodes ‘d’ (as shown in figure 13) in order to vary the received signal strength which affects the packet delivery ratio in return . We see that increasing the distance between the nodes reduces the received signal strength and packet delivery ratio.

Transmission Power :

The TX power setting specifies the strength of the signal that the sender (node 0) produces during the times it is transmitting. A stronger signal will generally provide a more reliable wireless connection. A lower power setting means the signal will not go as far and will imply reduced received signal strength.

We vary the sender transmission power of the sender. Decreasing the transmission reduces the link quality thereby affecting the packet delivery ratio.

Distance between the sender and receiver :

The distance between sender (Node 0) and receiver (Node 3) varied . As the distance between the sender and receiver increases, the received signal strength decreases and therefore, the packet delivery ratio decreases.

Number of Packets:

As we increase the number of packets, congestion in the network increases therefore, packet delivery ratio decreases.

We calculate the PDR and RSS value for the Receiver (Node 3) using the wireless module utility. We use the function TraceConnectwithoutContext() which invokes a callback

function whenever the trace attribute " RSS " changes. The callback function is used for storing RSS and PDR of Node 3.

Data Annotation :

After collecting packet delivery ratio (PDR) and received signal strength (RSS) for different jamming scenarios, we label the data. These labels serve as ground truth for our machine learning algorithms. We use

- 0 for No Jamming
- 1 for Constant Jamming
- 2 for Reactive Jamming
- 3 for Random Jamming

3.3 PROPOSED MODEL:

Previous works have analysed various machine learning algorithms to detect and classify Jamming attacks. Various machine algorithms include KNN, Decision Trees, Random Forest etc. Previous works have achieved best performance with Random Forest Classifier. We use a new machine learning algorithm that is Gradient boosting and compare its results with various machine algorithms used before . Gradient Boosting outperforms Random forest in detection and classification of the jamming attack.

Gradient boosting is a machine learning technique used for both regression and classification problems. It forms an ensemble of weak prediction models like decision trees to produce a prediction model.

Gradient Boosting is an example of the boosting algorithm. Boosting is an ensemble technique in which predictors are made sequentially, not independently. GBT build trees one at a time, and each new tree helps in correcting errors made by previously trained tree.

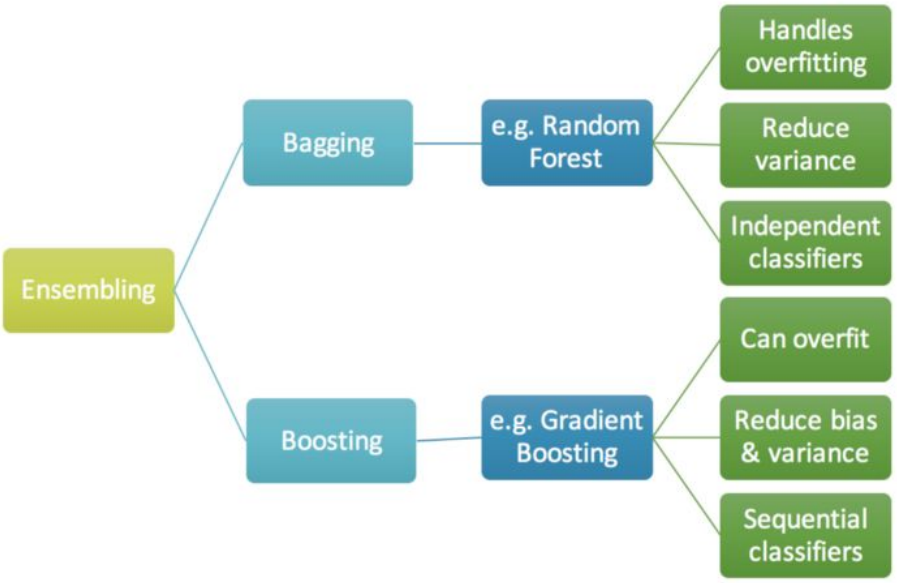

Fig 9: Gradient Boosting and Random Forest Difference [10]

Ensembling - This refers to a collection of different predictors that come together to predict the final outcome. Eg In random forest a number of decision trees use mean or majority voting to predict final output.

Bagging - Bagging is an ensembling technique which constructs independent predictors or learners and combine them using model averaging techniques like mean.

Boosting - This is similar to bagging as it is also an ensemble technique but different as it build the various predictors sequentially and not independently.

[10] https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

Hence, Gradient Boosting is an Ensembling technique and builds the predictors sequentially. This makes it different from random forest which uses bagging technique. Hence our proposed model uses this technique to improve over previous work.

3.4 ALGORITHM:

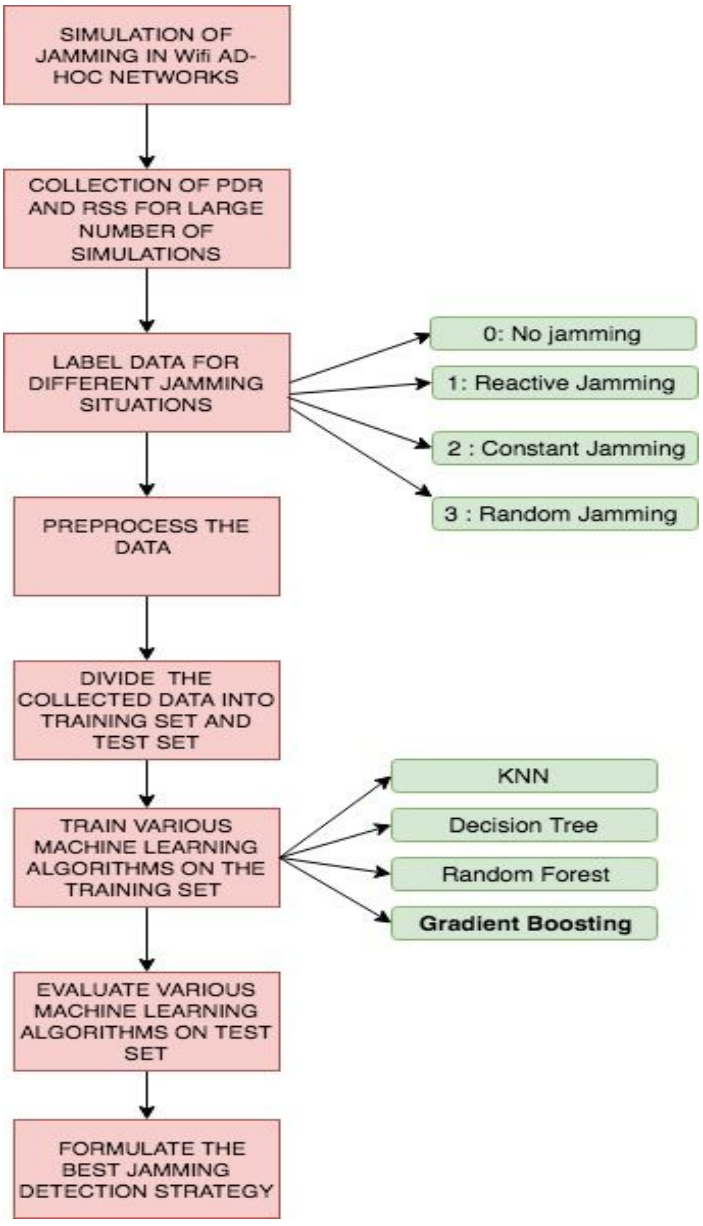

Fig 10:Algorithm

Chapter 4

EXPERIMENTS AND RESULTS

4.1 EXPERIMENTS

For the purpose of our model we conducted various simulations in NS-3 and collected a lot of data. We varied parameters like distance between nodes and power of transmission of nodes. We then used the data to train our model using machine learning algorithms and analyse the relation between different types of jamming attacks. The data collected was preprocessed to remove duplicate and missing values. Also the values were normalised to remove bias towards any single feature.

4.1.1 NO JAMMER

To collect the data for no jammer case we varied the distance of sender to the receiver. The simulation ran for 60 s and we collected values for RSS and PDR.

4.1.2 DISTANCE

The first experiment conducted was by changing the distance between the nodes. The position of each node was defined as a function of "d" variable as following (as shown in figure :13):-

1) Node0 (source) = (0.0, 0.0, 0.0)
2) Node1 = (d, 0.1 * d, 0.0));
3) Node2 = (2 *d, 0.0, 0.0);
4) Node3 (receiver) = (3 * d, 0.1 * d, 0.0);

5) Node4 (Jammer) = (2 * d, -0.5 * d , 0.0);

The variable distanceToRx was varied from 7m to 13m at an interval of 0.2s. The simulation ran for 60 seconds in each case and the jammer started running at 7s. This was repeated for all 3 types of jammers.

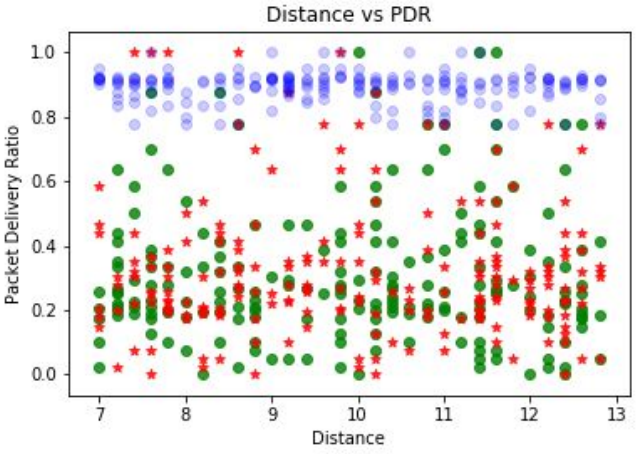

Fig 11: Distance vs PDR

This figure shows how PDR of the receiver varies as the distance between the nodes is increased in the three jamming situations.

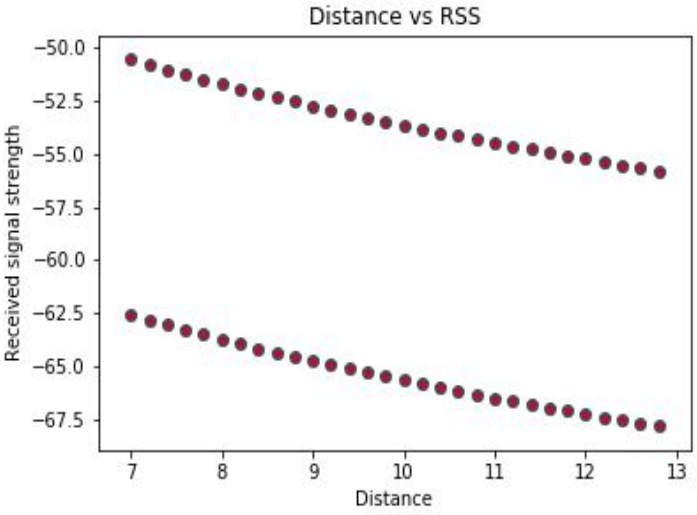

Fig 12: Distance vs RSS

This graph shows how the Received signal strength varies as the distance between the node 'd' is increased. We see the same trend for all three cases .

4.1.3 TRANSMISSION POWER

The transmission power of the nodes were varied from -40 to 40 dBm at an interval of 1 dBm and the simulation ran for 60s with a message transmitting at each second and the jammer starting at 7s. The data was also collected for each jammer

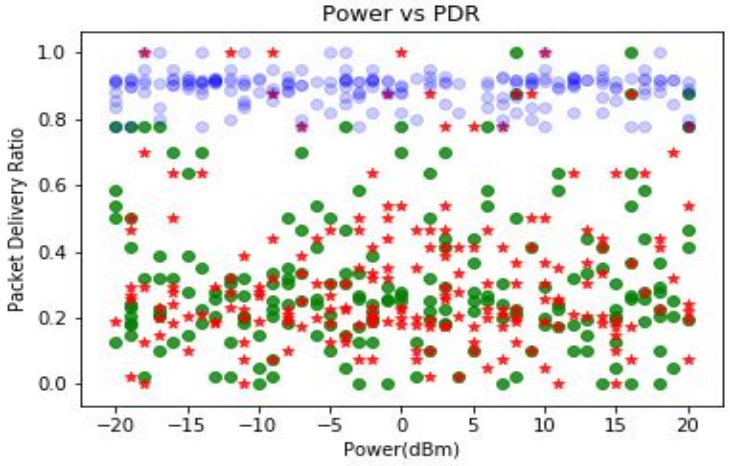

Fig 13: Power vs PDR

This graph shows the variation of packet delivery ratio with transmission power of the sender

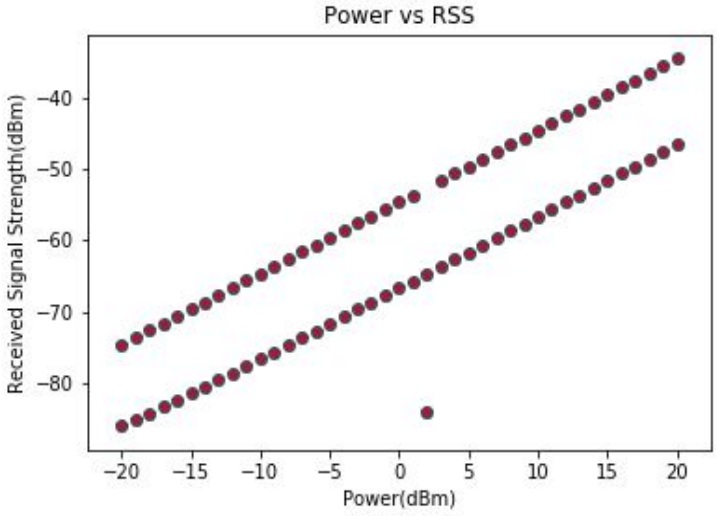

Fig 14: Power vs RSS

This graph shows the variation of Received signal Strength with the Transmission power of the sender. In all scenarios, RSS increases with the transmission power

4.1.4 DISTANCE AND TRANSMISSION POWER

In this case both distance and transmission power were varied with the same specifications as above. For each particular distance the power was varied from -40 to 40 dBm and the distance itself was varied from 7m to 13 m. In this case also the simulations were run for 60s for each value of distance and power with the jammer starting at 7s.

4.1.5 PREDICTION

The data was collected in above 3 cases was fed to different machine learning algorithms with changing parameters. The data collected was labelled as following :-

1) 0 - No Jammer
2) 1 - Constant Jammer
3) 2 - Reactive Jammer
4) 3 - Random Jammer

After labelling the data was randomly shuffled and split into training and test sets. We experimented with 2 different splits and recorded the best results:-

1) 60% train and 40% test
2) 70% train and 30% test

The parameter changes in the machine learning algorithms were as follows:-

1) *Random Forest* - The number of estimators were taken to be 2, 5, 10, 100, 1000 and the best result was recorded.
2) *KNN*- The number of neighbours were taken to be between 2 and 20 with an interval of 2 and the best result was recorded.
3) *Decision Tree* - No parameter changes were tested.

4) ***Gradient Boosting -*** The number of estimators were taken to be 1, 10, 100. The max depth of each tree was varied as 1, 10, 100 as well and the learning rate was taken from 0.1 to 1 at an interval of 0.1

4.2 RESULTS

4.2.1 DISTANCE BETWEEN NODES

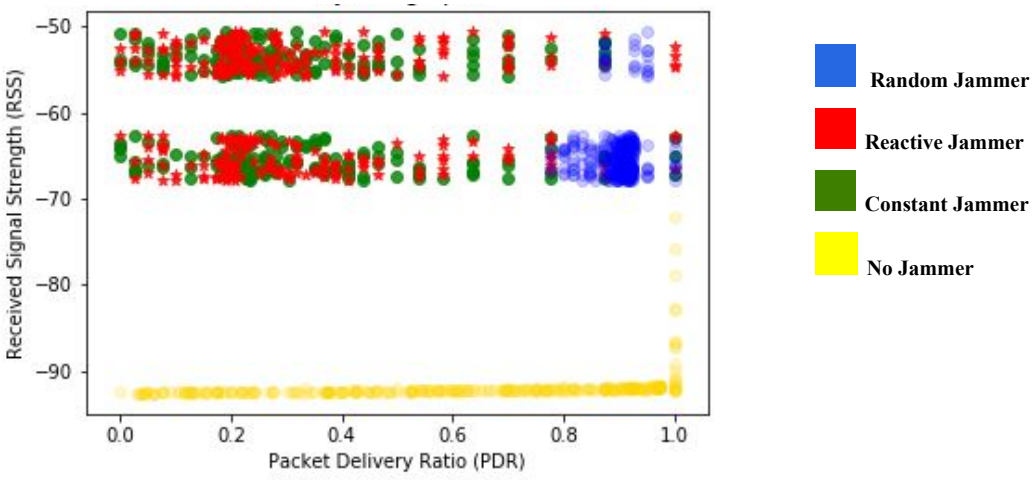

Fig 15: PDR vs RSS (for changing distance)

Distance ranges from 7 - 13m and for each distance simulation runs for 60s collecting RSS and PDR values

The above graph shows many results.

1) It is clearly easy to differentiate between jammed and no jammed situations. This is because the value of RSS is more in case of jammed signal as the strength for both the original message and jamming signal is added.
2) The graph for reactive and constant jammer is overlapping more and so it is difficult to differentiate between them.

3) The graph for reactive and constant jammer is congested for lower values of PDR and becomes more spaced for higher values. This shows that it is difficult to achieve higher values of PDR in these cases whereas for random jammer the PDR is still quite high so this jamming technique is not as effective as the other two.
4) In case of no jamming situations PDR remains high even if RSS is low, it is only when RSS gets too low that PDR starts to drop which is not the case in jamming situations.

The results are recorded for the parameters that gave the best accuracy.

MACHINE-LEARNING ALGO	**ACCURACY**
KNN	73.91%
Decision Tree	71.37%
Random Forest	74.26 %
Gradient Boosting (Our Model)	**75.7%**

Table 1 : Accuracies (for changing distance)

4.2.2 TRANSMISSION POWER

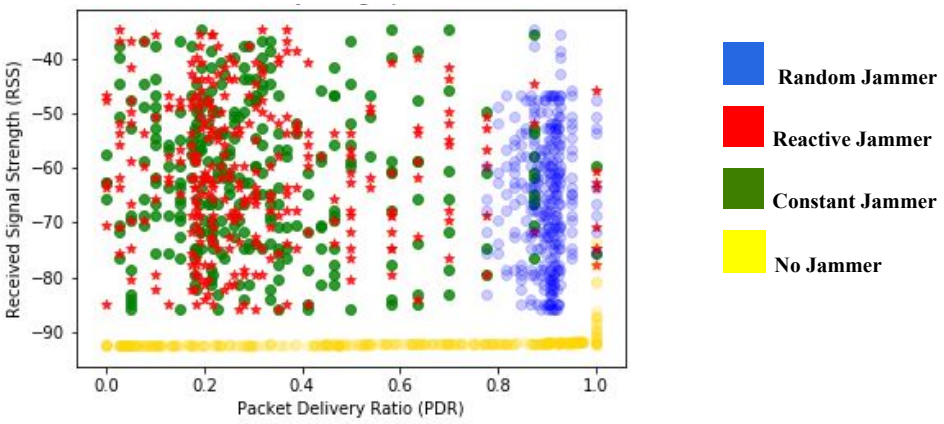

Fig 16: PDR vs RSS (for changing power)

The above graph provides similar analysis to the graph in case of varying distance the difference is that in this case the range of RSS is much more for a particular PDR.

MACHINE-LEARNING ALGO	ACCURACY
KNN	70.02%
Decision Tree	65.1%
Random Forest	67.9%
Gradient Boosting (Our Model)	**71.95%**

Table 2 : Accuracies (for changing Transmission power)

4.2.2 DISTANCE AND TRANSMISSION POWER

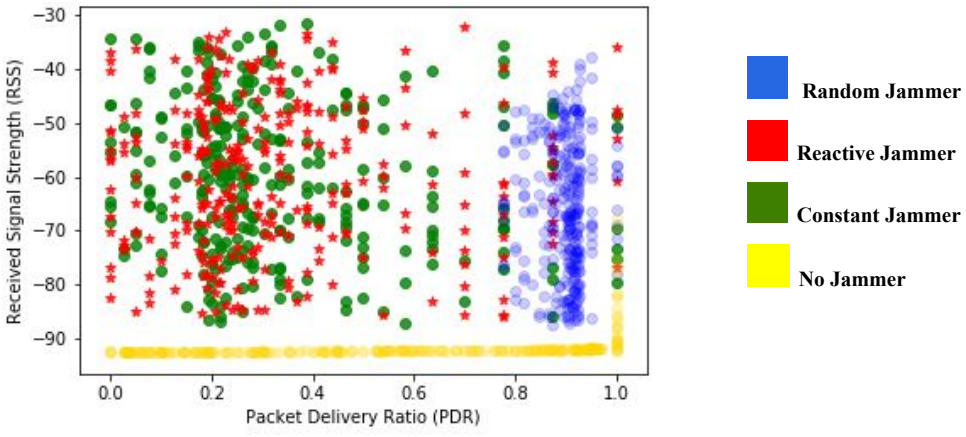

Fig 17: PDR vs RSS (for changing distance and power)

distance ranges from 7-13m and power from -20 to 20dBm and for each value of distance and power simulation runs for 60s collecting RSS and PDR values

The above graph provides similar analysis to the graph in case of varying distance the difference is that in this case the range of RSS is much more for a particular PDR.

MACHINE-LEARNING ALGO	**ACCURACY**
KNN	64.22%
Decision Tree	69.57%
Random Forest	69.85%
Gradient Boosting (Our Model)	**72.05%**

Table 3 : Accuracies (for changing distance and power simultaneously)

4.2.3 CONFUSION MATRIX

A confusion matrix is used to check the confusion of an algorithm between different classes. The entry a_{ij} refers to the number of examples that were labelled as 'i' class but predicted as class 'j'. We present confusion matrix for the Gradient Boosting algorithm in the case of varying distance and power both. This matrix further solidifies our result that the algorithms finds it difficult to classify between Constant and Reactive Jammers. Example, a_{12} shows the samples which were actually for Constant Jammers but were predicted to Reactive Jammers.

	0	1	2	3
0	[[3956,	7,	18,	12],
1	[14,	294,	636,	21],
2	[26,	714,	257,	24],
3	[17,	13,	23,	726]]

Chapter 5

Conclusion and Future work

In our report we used NS-3 to simulate different jamming techniques on wireless networks. We collected data and used machine learning to classify different types of attacks so that appropriate countermeasures can be taken. Our simulation results show that it **Gradient Boosting gives the best performance**. We also conclude that it is **difficult to classify between Reactive and Constant Jammers** whereas detecting whether jamming is occurring or not is relatively simple task.

So we can further optimise our algorithm to distinguish between reactive and constant jammers. Other techniques and algorithms can be tried to improve the accuracy even further. Also further research on this topic can include extending these algorithms for networks with mobile nodes like VANETS by including features such as relative speed of nodes, location etc.

The most natural step although, after classification of different types of jammers is to form techniques to prevent each type of attack. This is also a challenging and interesting research topic.

References

[1] https://www.analyticsindiamag.com/7-types-classification-algorithms/

[2]https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb

[3]https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

[4]https://www.nsnam.org/wiki/Wireless_jamming_mode

[5] https://www.geeksforgeeks.org/computer-network-network-simulator-3/

[6]https://jupyter.org/

[7]https://www.cisco.com/c/en_in/solutions/small-business/resource-center/networking/wireless-network.html

[8]https://www.igi-global.com/chapter/jamming-attacks-countermeasures-wireless-sensor/41122

[9]Feng, Zhutian, and Cunqing Hua. "Machine Learning-based RF Jamming Detection in Wireless Networks." *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*. IEEE, 2018.

[10]Puñal, Oscar, et al. "Machine learning-based jamming detection for IEEE 802.11: Design and experimental evaluation." *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014.

[11]Xu, Wenyuan, et al. "Jamming sensor networks: attack and defense strategies." *IEEE network* 20.3 (2006): 41-47.

[12]Xu, Wenyuan, et al. "The feasibility of launching and detecting jamming attacks in wireless networks." *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2005.

[13]Nguyen, Danh, et al. "A real-time and protocol-aware reactive jamming framework built on software-defined radios." *Proceedings of the 2014 ACM workshop on Software radio implementation forum*. ACM, 2014.

[14]Misra, Sudip, et al. "Using honeynodes for defense against jamming attacks in wireless infrastructure-based networks." *Computers & electrical engineering* 36.2 (2010): 367-382.

Appendix

A.1 NS3 code for simulation of jamming in Wifi Ad-Hoc Networks

This is an example code simulating reactive jamming in Wifi Ad-Hoc Network.

```
#include"ns3/all_headers.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

NS_LOG_COMPONENT_DEFINE ("ReactiveJammer");

using namespace ns3;

/**
 *brief: Packet receiving sink.
 *parameter:socket Pointer to socket.
 **/
void
ReceivePacket (Ptr<Socket> socket)
{
  Ptr<Packet> pkt;
  Address _from;
  while (pkt = socket->RecvFrom (_from))
    {
```

```
    if (pkt->GetSize () > 0)
      {
        InetSocketAddress _addr = InetSocketAddress::ConvertFrom (_from);
        NS_LOG_UNCOND ("--\nReceived one packet! Socket: "<< _addr.GetIpv4 ()
                    << " port: " << _addr.GetPort () << " at time = " <<
                    Simulator::Now ().GetSeconds () << "\n");
      }
   }
}

/**
 *brief: Traffic generator.
 * param :socket Pointer to socket.
 * param :pkt_size, Packet size.
 * param :node_ Pointer, to node.
 * param :pkt_count, Number of packets to generate.
 * param :pkt_interval, Packet sending interval.
 **/
static void
GenerateTraffic (Ptr<Socket> socket, uint32_t pkt_size, Ptr<Node> node_,
                 uint32_t pkt_count, Time pkt_interval)
{
  if (pktCount > 0)
    {
      socket->Send (Create<Packet> (pkt_size));
      Simulator::Schedule (pkt_interval, &GenerateTraffic, socket, pkt_size, node_,
          pkt_count - 1, pkt_interval);
    }
  else
    {
```

```
      socket->Close ();
    }
}

/**
 * brief : Trace function for remaining energy at node.
 * param : oldValue,  Old remaining energy value.
 * param :remaining_energy, New remaining energy value.
 */
void
RemainingEnergy (double oldValue, double remaining_energy)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << " Current Remaining Energy = "
<< remaining_energy << "J");
}

/**
 * brief: Trace function for total energy consumption at node.
 * param: oldValue, Old total energy consumption.
 * param: total_energy. New total energy consumption.
 */
void
TotalEnergy (double oldValue, double total_energy)
{
   NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "Total energy consumed by
radio = "<<total_energy << "J");
}

/**
 * brief :Trace function for node RSS.
```

```
 * param :oldValue, Old RSS value.
 * param: RSS, New RSS value.
 */
void
NodeRss (double oldValue, double RSS)
{
   NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << " Node RSS = " << RSS
<<"W");
}

/**
 * brief: Trace, function for node PDR.
 * param: oldValue, Old PDR value.
 * param: pdr, New PDR value.
 */
void
NodePdr (double oldValue, double PDR)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "Node PDR = " << PDR);
}

/**
 * brief: Trace function for node RX throughput.
 * param: oldValue, Old RX throughput value.
 * param: rx_throughput, New RX throughput value.
 */
void
NodeThroughputRx (double oldValue, double rx_throughput)
{
```

```
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "Node RX throughput = " <<
rx_throughput);
}

int
main (int argc, char *argv[])
{
 std::string phy_mode ("DsssRate1Mbps");
 double Prim_rss = -80;            // dBm
 uint32_t packet_size = 200;   // bytes
 bool verbose = false;

 // simulation parameters
 uint32_t num_packets = 10000;  // number of packets to send
 double interval = 1;          // seconds
 double start_time = 0.0;       // seconds
 double distance_to_Rx = 10.0;   // meters
 /*
  * This is number used to set the transmit power
  */
 double offset = 81;

 CommandLine cmd;
 cmd.AddValue ("phyMode", "Wifi Phy mode", phy_mode);
 cmd.AddValue ("Prim_rss", "Intended primary RSS (dBm)", Prim_rss);
 cmd.AddValue ("packet_size", "size of application packets sent", packet_size);
 cmd.AddValue ("num_packets", "Total no. of packets to send", num_packets);
 cmd.AddValue ("star_time", "Start time of Simulation", start_time);
 cmd.AddValue ("distance_to_Rx", "X-Axis distance between nodes", distance_to_Rx);
 cmd.AddValue ("verbose", "Turn on all log components", verbose);
```

```
  cmd.Parse (argc, argv);

  // Convert to time object
  Time interPacketInterval = Seconds (interval);

                                                                Config::SetDefault
("ns3::WifiRemoteStationManager::FragmentationThreshold",StringValue ("2200"));
       Config::SetDefault    ("ns3::WifiRemoteStationManager::RtsCtsThreshold",StringValue
("2200"));
      Config::SetDefault    ("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue
(phyMode));

  NodeContainer n;
  n.Create (5);     // create 4 nodes and 1 jammer
  NodeContainer network_nodes;
  network_nodes.Add (n.Get (0));
  network_nodes.Add (n.Get (1));
  network_nodes.Add (n.Get (2));
  network_nodes.Add (n.Get (3));
/**The below set of helpers will help us to put together the wifi NICs we want**/
  WifiHelper wifi;
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

  /** WifiPHY**/

/**************************************************************************
*/
  NslWifiPhyHelper wifi_Phy = NslWifiPhyHelper::Default ();
  wifi_Phy.Set ("NslRxGain", DoubleValue (-10));
  wifi_Phy.Set ("NslTxGain", DoubleValue (offset + Prss));
```

```
  wifi_Phy.Set ("NslCcaMode1Threshold", DoubleValue (0.0));

/**************************************************************************
*/

  /** wifi channel **/
  NslWifiChannelHelper wifi_channel;
  wifi_channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
  wifi_channel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
  // create wifi channel
  Ptr<NslWifiChannel> wifi_channel_ptr = wifi_channel.Create ();
  wifi_Phy.SetChannel (wifi_channel_ptr);

  /** MAC layer **/
  // Add a non-QoS upper MAC , disable rate control
  NqosWifiMacHelper wifi_mac = NqosWifiMacHelper::Default ();
                        wifi.SetRemoteStationManager            ("ns3::ConstantRateWifiManager",
"DataMode",StringValue (phy_mode),"ControlMode", StringValue (phy_mode));
  // Configure it to ad-hoc mode
  wifi_mac.SetType ("ns3::AdhocWifiMac");

  /** install MAC + PHY **/
  NetDeviceContainer devices = wifi.Install (wifi_Phy, wifi_mac, network_nodes);
  // install PHY & MAC onto jammer
  NetDeviceContainer jammer_net_device = wifi.Install (wifi_Phy, wifi_mac, n.Get (4));

  /** mobility **/
  MobilityHelper mobility;
  Ptr<ListPositionAllocator> position_Alloc =
     CreateObject<ListPositionAllocator> ();
```

```
  position_Alloc->Add (Vector (0.0, 0.0, 0.0));
  position_Alloc->Add (Vector (distanceToRx, 0.1 * distanceToRx, 0.0));
  position_Alloc->Add (Vector (2 * distanceToRx, 0.0, 0.0));
  position_Alloc->Add (Vector (3 * distanceToRx, 0.1 * distanceToRx, 0.0));
   position_Alloc->Add (Vector (2 * distanceToRx, -0.5 * distanceToRx, 0.0)); // location of
jammer
  mobility.SetPositionAllocator (position_Alloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (n);

  /** Energy Model **/

/**********************************************************************
*/
  /* energy source */
  BasicEnergySourceHelper basic_Source_Helper;
  // configure energy source
  basic_Source_Helper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue (0.1));
  // install on node
  EnergySourceContainer energy_Sources = basicSourceHelper.Install (n);
  /* device energy model */
  WifiRadioEnergyModelHelper radio_Energy_Helper;
  // configure radio energy model
  radio_Energy_Helper.Set ("TxCurrentA", DoubleValue (0.0174));
  // install on devices
  DeviceEnergyModelContainer device_Models =
     radio_Energy_Helper.Install (devices, energy_Sources);
  DeviceEnergyModelContainer jammer_Device_Models =
     radio_Energy_Helper.Install (jammer_net_device.Get (0), energy_Sources.Get (4));
```

```
/**************************************************************************
*/

  /** Wireless Module Utility **/

/**************************************************************************
*/
  WirelessModuleUtilityHelper utility_Helper;
  // set inclusion/exclusion list for all nodes
  std::vector<std::string> AllIncList;
  AllIncList.push_back ("ns3::UdpHeader");          // record only Udp Header
  std::vector<std::string> AllExcList;
  AllExcList.push_back ("ns3::olsr::PacketHeader"); // ignore all olsr headers/trailers
  // assign lists to helper
  utility_Helper.SetInclusionList (AllIncList);
  utility_Helper.SetExclusionList (AllExcList);
  // install on all nodes
  WirelessModuleUtilityContainer utilities = utility_Helper.InstallAll ();

/**************************************************************************
*/

  /** Jammer configuration **/

/**************************************************************************
*/
  JammerHelper jammer_Helper;
  // configure jammer type
  jammer_Helper.SetJammerType ("ns3::ReactiveJammer");
```

```
  // set jammer params
  jammer_Helper.Set ("ReactiveJammerRxTimeout", TimeValue (Seconds (2.0)));
  jammer_Helper.Set ("ReactiveJammerReactionStrategy",
                UintegerValue(ReactiveJammer::FIXED_PROBABILITY));
  // install jammer
  JammerContainer jammers = jammerHelper.Install (n.Get (4));
  // Get pointer to Jammer
  Ptr<Jammer> jammer_ptr = jammers.Get (0);
  // enable all jammer debug statements


/*************************************************************************
*/

  /** Internet stack **/
  InternetStackHelper internet;
  internet.Install (network_nodes);

  Ipv4AddressHelper ip_v4;
  NS_LOG_INFO ("Assign IP Addresses.");
  ip_v4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ip_v4.Assign (devices);

  TypeId t_id = TypeId::LookupByName ("ns3::UdpSocketFactory");
   Ptr<Socket> recv_Sink = Socket::CreateSocket (network_nodes.Get (3), t_id);   // node 3,
receiver
  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
  recv_Sink->Bind (local);
  recvS_ink->SetRecvCallback (MakeCallback (&ReceivePacket));
```

```
  Ptr<Socket> source = Socket::CreateSocket (network_nodes.Get (0), t_id);      // node 0,
sender
  InetSocketAddress rem = InetSocketAddress (Ipv4Address::GetBroadcast (), 80);
  source->SetAllowBroadcast (true);
  source->Connect (rem);

  /** connect trace sources **/

/**************************************************************************
*/
  // energy source
  Ptr<EnergySource> basic_Source_Ptr = energy_Sources.Get (2);
       basic_Source_Ptr->TraceConnectWithoutContext    ("RemainingEnergy",MakeCallback
(&RemainingEnergy));
  // using honest node device energy model
  Ptr<DeviceEnergyModel> basic_Radio_Model_Ptr = device_models.Get (2);
  basic_Radio_Model_Ptr->TraceConnectWithoutContext ("TotalEnergyConsumption",
                                    MakeCallback (&TotalEnergy));
  // wirelessModuleUtility
  Ptr<WirelessModuleUtility> utility_Ptr = utilities.Get (2);
  utility_Ptr->TraceConnectWithoutContext ("Rss", MakeCallback (&NodeRss));
  utility_Ptr->TraceConnectWithoutContext ("Pdr", MakeCallback (&NodePdr));

/**************************************************************************
*/
  /** simulation setup **/
  // start traffic
      Simulator::Schedule (Seconds (startTime), &GenerateTraffic, source,PpacketSize,
network_nodes.Get (0), numPackets,
                  interPacketInterval);
```

```
  //start the jammer (node 4) at 7s
  Simulator::Schedule (Seconds (startTime + 7.0), &ns3::Jammer::StartJammer,
                       jammerPtr);
  Simulator::Stop (Seconds (60.0));
  Simulator::Run ();
  Simulator::Destroy ();

  return 0; }
```

A.2 Script for running Machine Learning algorithms on data collected

```
import numpy as np
import matplotlib.pyplot as plt
import math

path = "./BTPData/DistanceToRx"

def unique_rows(a):
    a = np.ascontiguousarray(a)
    unique_a = np.unique(a.view([('', a.dtype)]*a.shape[1]))
    return unique_a.view(a.dtype).reshape((unique_a.shape[0], a.shape[1]))

def delNaN(a,b):
    count = 0
    for i in range(len(a)):
        if(math.isnan(a[i])):
            count = count + 1

    for i in range(len(a) - count):
```

```
        if(math.isnan(a[i])):
            a = np.delete(a, (i), axis = 0)
            b = np.delete(b, (i), axis = 0)
            i = i - 1
    return a,b

# NO JAMMER
a = np.loadtxt("./BTPData/rss_nojammer_node2.txt")
a = 10*np.log10(1000*a)
b = np.loadtxt("./BTPData/pdr_nojammer_node2.txt") + 10

a,b = delNaN(a,b)

d = a.reshape(len(a),1)
e = b.reshape(len(b),1)
data = np.concatenate((d,e),axis = 1)
data = unique_rows(data)
for i in range(len(data)):
    if(data[i][0] < -93.5):
        data[i][0] = -84
        data[i][1] = 2
data = unique_rows(data)
idx = 0
for i in range(len(data)):
    if(data[i][1] == 2):
        idx = i
data = np.delete(data, (idx), axis = 0)
np.random.shuffle(data)
data1 = data
print(len(data1))
```

```
# CONSTANT JAMMER
a = np.loadtxt(path + "/rss_constantjammer_node2.txt")
a = 10*np.log10(1000*a)
b = np.loadtxt(path + "/pdr_constantjammer_node2.txt") - 80

a,b = delNaN(a,b)



d = a.reshape(len(a),1)
e = b.reshape(len(b),1)
data = np.concatenate((d,e),axis = 1)
data = unique_rows(data)
for i in range(len(data)):
    if(data[i][0] < -93.5):
        data[i][0] = -84
        data[i][1] = 2
data = unique_rows(data)
idx = 0
for i in range(len(data)):
    if(data[i][1] == 2):
        idx = i
data = np.delete(data, (idx), axis = 0)
np.random.shuffle(data)
data2 = data
print(len(data2))



#REACTIVE JAMMER
```

```
a = np.loadtxt(path + "/rss_reactivejammer_node2.txt")
a = 10*np.log10(1000*a)
b = np.loadtxt(path + "/pdr_reactivejammer_node2.txt") - 80

a,b = delNaN(a,b)



d = a.reshape(len(a),1)
e = b.reshape(len(b),1)
data = np.concatenate((d,e),axis = 1)
data = unique_rows(data)
for i in range(len(data)):
    if(data[i][0] < -93.5):
        data[i][0] = -84
        data[i][1] = 2
data = unique_rows(data)
idx = 0
for i in range(len(data)):
    if(data[i][1] == 2):
        idx = i
data = np.delete(data, (idx), axis = 0)
np.random.shuffle(data)
data3 = data
print(len(data3))



# RANDOM JAMMER

a = np.loadtxt(path + "/rss_randomjammer_node2.txt")
b = np.loadtxt(path + "/pdr_randomjammer_node2.txt") - 80
```

```
a = 10*np.log10(1000*a)

a,b = delNaN(a,b)



d = a.reshape(len(a),1)
e = b.reshape(len(b),1)
data = np.concatenate((d,e),axis = 1)
data = unique_rows(data)
for i in range(len(data)):
    if(data[i][0] < -93.5):
        data[i][0] = -84
        data[i][1] = 2
data = unique_rows(data)
idx = 0
for i in range(len(data)):
    if(data[i][1] == 2):
        idx = i
data = np.delete(data, (idx), axis = 0)
np.random.shuffle(data)
data4 = data
print(len(data4))

# PLOTTING GRAPH
noOfData = 200
c = np.random.randint(0,1,(noOfData))
#  plt.scatter(data1[:noOfData,1],  data1[:noOfData,0],  marker  =  'o',  alpha  =  0.2,  color  =
['yellow'])
plt.scatter(data2[:noOfData,1],  data2[:noOfData,0],  marker  =  'o',  alpha  =  0.8,  color  =
['green'])
```

```
plt.scatter(data3[:noOfData,1], data3[:noOfData,0], marker = '*', alpha = 0.8, color = ['red'])
plt.scatter(data4[:noOfData,1], data4[:noOfData,0], marker = 'o', alpha = 0.2, color = ['blue'])

plt.title('Power vs RSS')
# naming the x axis
plt.xlabel('Power(dBm)')
# naming the y axis d
plt.ylabel('Received Signal Strength(dBm)')

# function to show the plot
# plt.show('a.png')


# PREPARATION OF DATASET

from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
min_max_scaler = preprocessing.MinMaxScaler()

#No Jammer
c = np.zeros((len(data1),1))
data = min_max_scaler.fit_transform(data1)
train_features1 = np.concatenate((data,c),axis = 1)
# train_features1 = train_features1[0:9000]


#Constant Jammer
c = np.ones((len(data2),1))
```

```
data = min_max_scaler.fit_transform(data2)
train_features2 = np.concatenate((data,c),axis = 1)
# train_features2 = train_features2[0:1000]

#Reactive Jammer
c =  np.full((len(data3),1),2)
data = min_max_scaler.fit_transform(data3)
train_features3 = np.concatenate((data,c),axis = 1)
# train_features3 = train_features3[0:1000]

#Random Jammer
c =  np.full((len(data4),1),3)
data = min_max_scaler.fit_transform(data4)
train_features4 = np.concatenate((data,c),axis = 1)
# train_features4 = train_features4[0:1000]

#Combine all
train_features =
np.concatenate((train_features1,train_features2,train_features3,train_features4),axis = 0)
np.random.shuffle(train_features)

train_data, test_data, train_label, test_label = train_test_split(train_features[:,0:2],
train_features[:,2],
                                                  test_size=0.3, random_state=1)


#K NEAREST NEIGHBOURS

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=2)
```

```
classifier.fit(train_data, train_label)
predictions = classifier.predict(test_data)
print("Accuracy:",metrics.accuracy_score(test_label, predictions))


#DECISION TREE

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf = clf.fit(train_data,train_label)
predictions = clf.predict(test_data)
print("Accuracy:",metrics.accuracy_score(test_label, predictions))



#RANDOM FOREST

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 4, random_state = 1)
rf.fit(train_data, train_label)
predictions = rf.predict(test_data)
print("Accuracy:",metrics.accuracy_score(test_label, predictions))



#GRADIENT BOOSTING

from sklearn import ensemble
params = {
```

```
    'n_estimators': 10,
    'max_depth': 10,
    'learning_rate': 0.1,
    'criterion': 'mse'
}
gradient_boosting_classifier = ensemble.GradientBoostingClassifier(**params)
gradient_boosting_classifier.fit(train_data, train_label)
predictions = gradient_boosting_classifier.predict(test_data)
print("Accuracy:",metrics.accuracy_score(test_label, predictions))
```