B.Sc. in Computer Science and Engineering Thesis

A Machine Learning Approach for Protecting Wireless Networks Against Virtual Jamming Based Denial of Service (DoS) Attacks

Submitted by

Yeaseen Arafat
201405091

Kazi Samin Yeaser
201405103

Arnab Dasgupta
201405063

Supervised by

Dr. A.K.M. Ashikur Rahman

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

April 2019

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "A Machine Learning Approach for Protecting Wireless Networks Against Virtual Jamming Based Denial of Service (DoS) Attacks", is the outcome of the investigation and research carried out by us under the supervision of Dr. A.K.M. Ashikur Rahman.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Yeaseen Arafat
201405091

Kazi Samin Yeaser
201405103

Arnab Dasgupta
201405063

CERTIFICATION

This thesis titled, **"A Machine Learning Approach for Protecting Wireless Networks Against Virtual Jamming Based Denial of Service (DoS) Attacks"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in April 2019.

Group Members:

Yeaseen Arafat

Kazi Samin Yeaser

Arnab Dasgupta

Supervisor:

Dr. A.K.M. Ashikur Rahman
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

Dhaka
April 2019

Yeaseen Arafat

Kazi Samin Yeaser

Arnab Dasgupta

Contents

CANDIDATES' DECLARATION i

CERTIFICATION ii

ACKNOWLEDGEMENT iii

List of Figures vii

List of Tables viii

ABSTRACT ix

1 Introduction 1

1.1 DoS attack in wireless network . . . 1
1.2 Our contribution . . . 2
1.3 Organization of the Thesis . . . 2

2 Preliminaries 3

2.1 Hidden Node Problem . . . 3
2.2 IEEE 802.11 Protocol : RTS-CTS Handshake . . . 4
2.3 Problems with RTS-CTS Handshake . . . 5
2.3.1 Preventing Non Interfering Parallel Transmission . . . 5
2.3.2 False Blocking . . . 6
2.3.3 Virtual Jamming . . . 7
2.4 Performance Matrix . . . 8
2.4.1 Throughput . . . 8
2.4.2 Delay . . . 8
2.4.3 Confusion Matrix . . . 9
2.4.4 Support Vector Machine . . . 9

3 Previous Works 11

3.1 RTS Validation . . . 11
3.1.1 Problems with RTS Validation . . . 12
3.2 Random RTS Validation . . . 13

4 Methodology **14**
4.1 Problem Definition . . . 14
4.1.1 Topology . . . 14
4.1.2 Scenario with no jamming . . . 15
4.1.3 Scenario With Jamming . . . 15
4.2 Random RTS Validation . . . 17
4.2.1 Scenario with random RTS validation . . . 17
4.2.2 Effect on aggregate throughput . . . 17
4.3 Machine Learning Approach . . . 18
4.3.1 Learning period . . . 18
4.3.2 Action period . . . 18
4.3.3 Analysis . . . 19
4.3.4 Selected Features . . . 19
4.3.4.1 Moving Average of IRR . . . 20
4.3.4.2 Deviation of Moving Average of IRR . . . 22
4.3.4.3 Moving Average of Inter Arrival Time of RTS packets . . . 23

5 Dataset Preparation **24**
5.1 Dataset Properties . . . 24
5.2 Model selection: Support Vector Machine . . . 25
5.3 Training Results . . . 26
5.3.1 Confusion Matrix Property . . . 26
5.3.2 False positive ratio and Accuracy . . . 26

6 Experimental Results **27**
6.1 Simulation Environment . . . 27
6.2 Performance comparison of different approaches . . . 27
6.2.1 Throughput comparison . . . 27
6.2.2 Delay comparison . . . 29

7 Conclusions and Future works **31**
7.1 Conclusion . . . 31
7.2 Future works . . . 31
7.2.1 Tuning Learn to Action Ratio . . . 32
7.2.2 Feature Selection hyperparameters tuning . . . 32
7.2.3 CTS only attack . . . 32
7.2.4 CTS plus ACK attack . . . 32

References **33**

A Codes **34**

A.1 Necessary code for Learning Part 34

List of Figures

2.1 A Hidden Node Scenario 4
2.2 Four Way Handshake 5
2.3 Different areas of perception for a transmission from A to B. 6
2.4 False Blocking. 7
2.5 Virtual Jamming 8
2.6 Hyperplanes in SVM 9
2.7 Hyperplanes in 2D and 3D 10

3.1 RTS Validation 11
3.2 False Negative with RTS Validation 12
3.3 Random RTS Validation 13

4.1 Static Scenario 14
4.2 Aggregate throughput without jamming. 15
4.3 Aggregate throughput with jamming. 16
4.4 Aggregate throughput with Random RTS Validation. 17
4.5 Effect in Cumulative Invalid RTS Count over Time 20
4.6 Effect in Moving average of IRR over Time 21
4.7 Effect in Deviation of Moving average of IRR over Time 22
4.8 Effect in Moving average of IRR over Time 23

6.1 Throughput comparison between RandomRTS and SVM 28
6.2 Throughput comparison between Normal Scenario and SVM 29
6.3 Throughput comparison under different approaches 30
6.4 Delay comparison under different approaches 30

7.1 CTS only attack carrier sensing 32

List of Tables

5.1 Dataset properties 24
5.2 Dataset 25
5.3 SVM coefficients 25
5.4 Confusion Matrix 26
5.5 False Positive ratio and Accuracy 26

ABSTRACT

In this project environment monitoring systems is implemented by using sensors and then sensors data are sent to a My SQL server using Wi-Fi. For enchanting data, DHT-11 (Temperature and Humidity sensor) and MQ-6 (LPG gas sensor) are being used. The basic objective of this research is to monitor and to develop a real-time monitoring of humidity and temperature, as well as the availability of gas using the very available DHT-11 sensor, MQ-6 sensor, and ESP-8266 NodeMCU module and then observe the data from a database. We can control the actuator from the server depending on the sensor value. Although great leap has been made in the control area, the precision motion control is challenging the control engineering to a greater extent. The control engineer needs to design a suitable controller which will effectively achieve the desired system characteristics, such as high precision, high speed requirements in precision motion control. here are two control schemes which have been proposed. In this research, both feedback and feedforward methods of control have been applied. This paper also makes a compact distinction between conventional and the local IP based observing system of an environment. The sensor's data are saved in a database by which we can monitor the sensors data without any access to the internet. In this research, the Arduino based ESP-8266 based NodeMCU was used. The various data attainment system of Arduino or Raspberry-pi is mother controller but using NodeMCU gives the benefit of using an Arduino along with a 2.4 GHz Wi-Fi module. As this was a demo project and needed far more inquiry in the real practice so, breadboards and jumper wires were used to test the project.

Chapter 1

Introduction

A wireless network is a computer network that uses wireless data connections between network nodes. As there is no wired connection between sender and receiver, so jamming can easily be occurred. The consequence is known as Denial of Service(DoS). DoS attack is cyber attack where an intruder seeks to make the network resources unavailable to its intended users temporarily.

1.1 DoS attack in wireless network

In wireless network scenario, when a sender wants to transmit data, he firstly creates a ready to signal to allocate the resource. Other nodes who received the signal do not try to allocate the resources in order to avoid the collision. Then the receiver replies with a ready to receive signal if it is not busy. After that, the sender can start sending data thoughout the transmission time. An intruder behaves like after allocating the resource, it will not send the data at the sending time. It will affect the aggregate throughput. As a result aggregate throughput on the attacking time will decreased. To prevent DoS attack channel sensing is one of the most significant way.

At the starting time of data sending of a sender, a sense on the channel can imply that the sender actually sends the data or not. If a sender is inactive at the data sending time, other sources who are waiting to allocate the channel, can easily access the channel. Again, if the sender actually active at the data sending time, there is nothing to do. By doing this, declining of aggregate throughput can be avoided.

1.2 Our contribution

There are some approaches to avoid DoS attack on wireless network. Randomly sensing the channel on the data transmission time can not recover the loss fully. We now try to increase the throughput more than random sensing. We propose a classification based solution to avoid DoS attack. The major contribution of this thesis are as follow.

- Firstly we try to find a pattern by which we can classify an intruder or a sender.
- In order to find any pattern, we have to find significant features. We create on some random scenarios and run the simulations in NS-2 model simulator. By observing these, we find some significant features.
- We propose two periods, learning period, action period.
- We feed the collection of features to a machine learning model to learn about them. Then classify them at the action period.

1.3 Organization of the Thesis

The rest of the book is organized as following.

- **Chapter 2** discusses the related problems, notations, and protocols of DoS attack in wireless network.
- **Chapter 3** discusses the previously done works to solve DoS attack in Wireless Network and shortcomings of those works.
- **Chapter 4** defines the problem and analyses the effects of DoS and current prevention on throughput. It also describes the Machine Learning Approach to solve this problem and discusses about the used features in the Machine Learning approach.
- **Chapter 5** describes the dataset used in the Machine Learning Approach.
- **Chapter 6** presents the simulation results and analyse the results.
- **Chapter 7** concludes the thesis and shows the scope of future works.

Chapter 2

Preliminaries

In this section at first we describe the notorious hidden node problem. Then we discuss the prescribed solution to hidden node problem in family of IEEE 802.11 protocols. Next we show how this solution can lead to false blocking and virtual jamming problem.

2.1 Hidden Node Problem

In wireless networks each node has a limited transmission range. Usually the *transmission range* of a node A is the area inside which other nodes can receive A's packet. Due to the limited transmission range the hidden node problem occurs because a node can communicate with a certain node within its transmission range but unable to communicate with the other nodes in the transmission range of that particular node.

Let us explain the problem and its main reason with a simple scenario. In Figure 2.1, there is a configuration of three nodes A, B and C. Here B is in the transmission range of A and C but C is out of the transmission range of A. In this scenario, C can communicate with B but cannot sense whether A is communicating with B and can interfere in B's reception of A's packet. As node C is out of the transmission range (the solid circle) of A, it can appear as a hidden node to A.

To some extent the problem can be mitigated by setting larger **carrier sense range.** *Carrier sense range* of node A is the area encompassing the node whose transmission A can sense while not necessarily being able to receive the transmitted package. The carrier sense range can be twice the transmission range [1]. In Figure 2.1 A is in the carrier sense range (the dashed circle) of C. So C is hidden no more as it can sense the transmission of A and can avoid interfering with it. The mechanism for eliminating the hidden node problem has been described in [1].

Carrier detection is usually controlled by thresholds applied to the level of perceived signal.

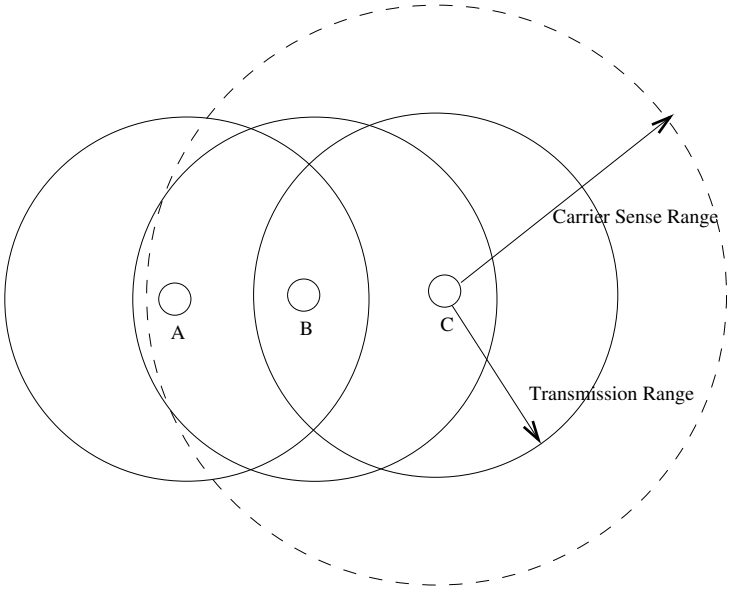


Figure 2.1: A Hidden Node Scenario

Low thresholds tend to be sensitive to many factors involving more than the distance between the nodes, e.g., the natural noise level in the neighborhood. Increasing the carrier sense range is possible but low thresholds may trigger many false indications. It will result in unnecessary back-offs and reduced throughput, possibly below one that could be achieved by simply ignoring the hidden node problem altogether.

2.2 IEEE 802.11 Protocol : RTS-CTS Handshake

To reduce the hidden node problem, Karn [2] proposed a two way handshake involving exchange of short packets between sender and receiver that would proceed the actual transmission. The complete exchange involves 4 packets. RTS/CTS/DATA/ACK.

The protocol is illustrated in Figure 2.2. The sender starts by transmitting a Request-To-Send(RTS) packet. The packet is broadcast through the network. The indented recipient then sends a Clear-To-Send(CTS) packet to the sender. Both these packets contain the length of time needed to transmit the actual data packet. Any third party receiving these packets becomes aware of the transmission and does not interfere with it. After receiving the CTS packet, the sender transmit the actual data packet and upon receiving it correctly the receiver sends a ACK to the sender. Here in the protocol the first two packet handles the hidden node problem while the ACK packet ensures reliable delivery. Failure to receive ACK packet triggers re transmissions.This protocol has been standardized into the popular IEEE 802.11 family of access schemes.

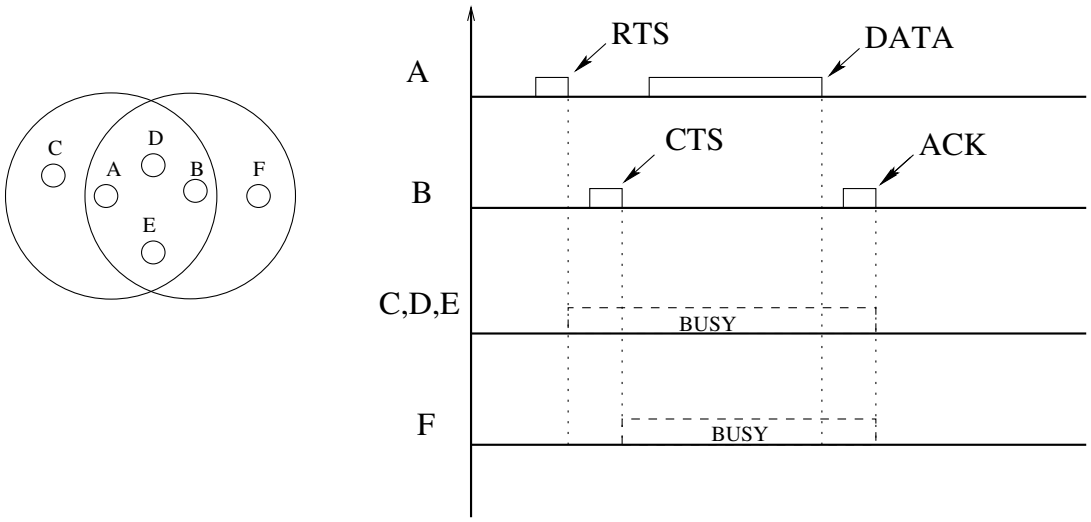


Figure 2.2: Four Way Handshake

2.3 Problems with RTS-CTS Handshake

To discuss the problems with RTS-CTS handshake first we consider a scenario.In Figure 2.3 suppose Node *A* is the sender and Node *B* is the receiver. Let R_A and R_B denote the respective transmission ranges of *A* and *B*, and P_A and P_B denote the corresponding transmission areas, i.e., the sets of points of the rectangle *U* covered by the circles with the radii R_A and R_B drawn around the nodes. We can classify the other nodes into 4 sections with respect to the sender and receiver and their transmission range. Any node $v \in P_A - P_B$ is painted green, any node $v \in P_B - P_A$ is painted red, any node $v \in (P_A \cap P_B)$ is painted yellow and any node $v \in (P_A \cup P_B)$ is painted blue.

2.3.1 Preventing Non Interfering Parallel Transmission

RTS-CTS handshake prevents some parallel transmission that can happen without interfering the actual transmission thus impacting the throughput. In Figure 2.3, the green nodes(inside the transmission range of *A* but outside the transmission range of *B*) won't be able receive anything while the transmission going on but can transmit to blue nodes. Similarly a red node (inside the transmission range of *B* but outside the transmission range of *A*) cannot be able to transmit anything but can receive from the blue node while the transmission of *A* and *B* going on. The truly restricted nodes are the yellow one, they can not receive or transmit anything during the transmission.

A node can classify itself with respect to ongoing transmission by going through the RTS-CTS handshake mechanism. When *A* sends an RTS packet all the nodes in it's transmission range(green and yellow) will be able to receive it. Similarly when *B* sends an CTS packet all the nodes in it's transmission range(red and yellow) will be able to receive it. So the nodes can

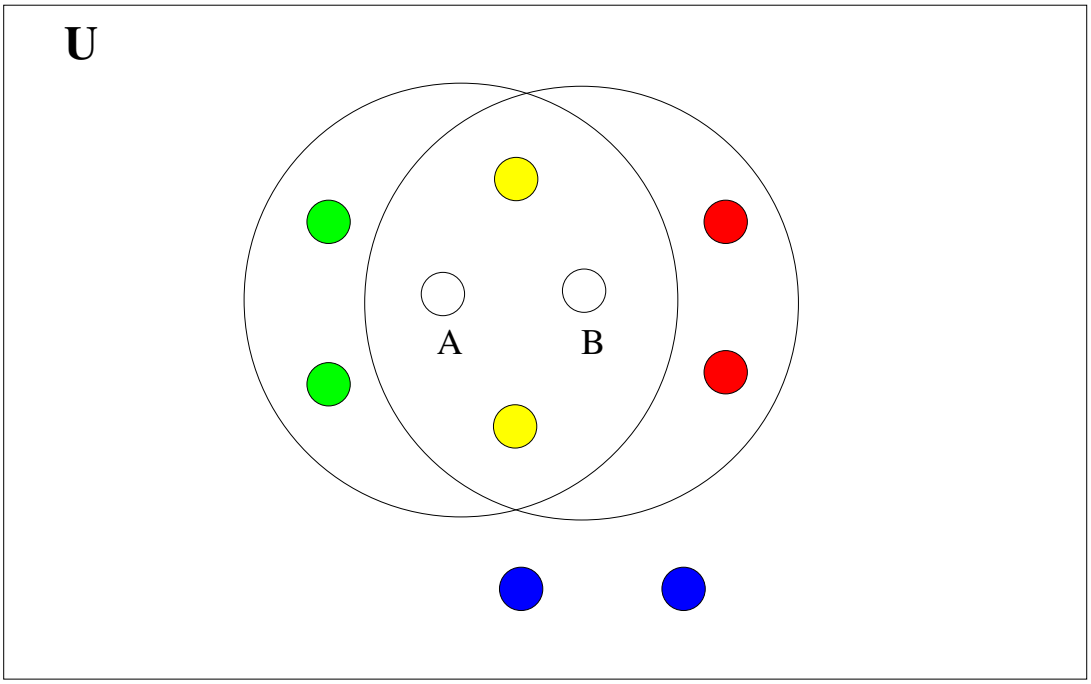


Figure 2.3: Different areas of perception for a transmission from A to B.

be classified according to the packet they receive; green, if only RTS packet is received; red, if only CTS packet is received; yellow, if both RTS and CTS packets are received; blue, if no packet is received.

The RTS/CTS mechanism hinders some non-interfering transmission that could be carried out in parallel. While eliminating hidden nodes will reduce collisions, thereby positively impacting the throughput, the elimination of some legitimate transmission will have the opposite effect. Although a scheme has been proposed in [3] to admit some parallel transmissions while avoiding the hidden node problem, it requires a significant modification of the IEEE 802.11s RTS/CTS mechanism.

2.3.2 False Blocking

False Blocking happens when some node becomes blocked for a non-existent transmission. To understand false blocking we consider the scenario shown in Figure 2.4(a). While *A* is transmitting to *B*, all green, yellow and red nodes are temporarily blocked. However, white nodes, being outside the range of both *A* and *B*, are free to transmit and receive.Now if a blue node *D* tries to transmit to a yellow node *C*. *D* will start the handshake with an RTS packet addressed to *C* [Figure 2.4(b)], but *C* will fail to respond with a CTS(blocked due to the transmission between *A* and *B*). *D* will assume that *C* is busy and will try to transmit later. But due to the ineffective RTS packet all blue nodes within *D*s transmission range will become green, and they will remain blocked for the entire time of the non-existent transmission, as announced by *D*. This

false blocking [4] will further propagate if some other blue node tries to send something to any of the newly-painted green nodes[Figure 2.4(c)], thus hindering possible transmission.

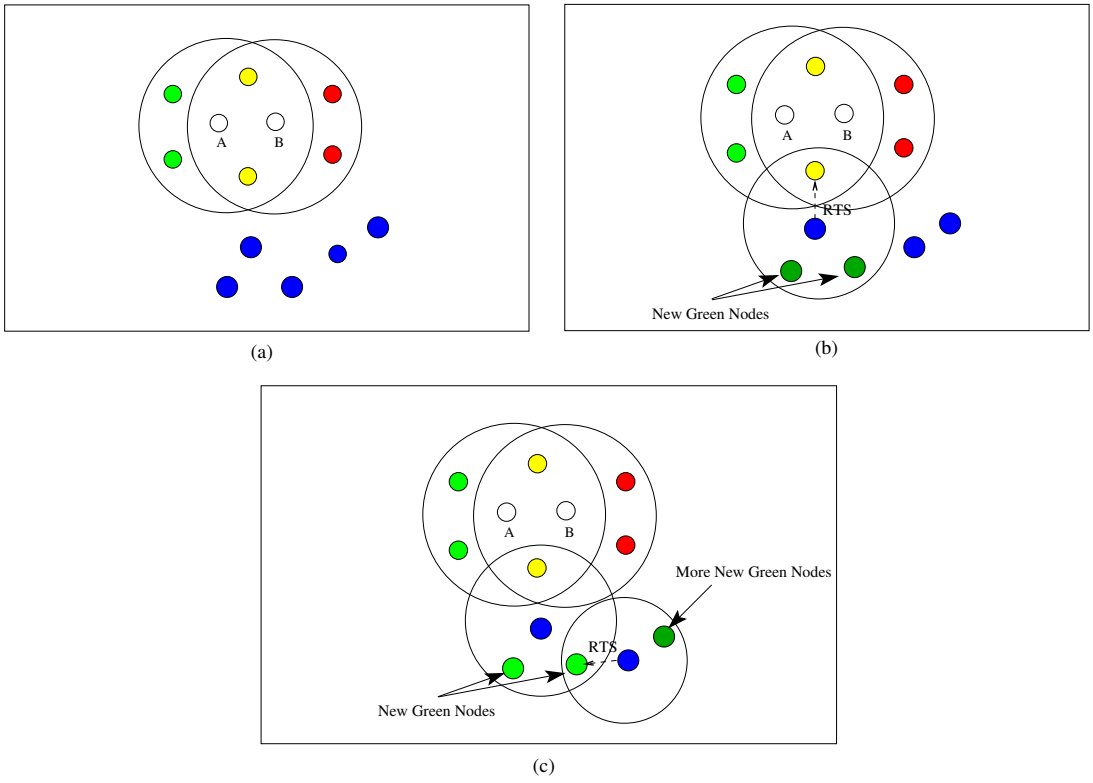


Figure 2.4: False Blocking.

2.3.3 Virtual Jamming

False blocking opens up an opportunity for malicious node to a **Denial of Service Attacks** against network using the four-way handshake which is meant for collision avoidance. A malicious node can deliberately send (short) RTS packets at some intervals announcing long transmissions never to occur. Hearing the RTS packets, some part of the network will be blocked for a never occurring transmission and the block can be propagated. This way the node will be able to effectively jam a possibly large segment of the network with a trivially small expenditure of power(RTS is a small data packet). The significance of these attacks is not in the fact that a node can jam a wireless network (which can hardly be doubted), but that the amount of power needed to carry out this kind of attack can be trivially small [5], [6].

This virtual jamming is illustrated in Figure 2.5, where node *A* sends false RTS packets to node *B* with a large legitimate value in the duration field. When nodes *C,D,E* and *F* receive such a packet, they will both become blocked for the amount of time requested by *A*. *A* will try to send the false RTS packet again after the waiting time of *C,D,E* and *F* will be over.

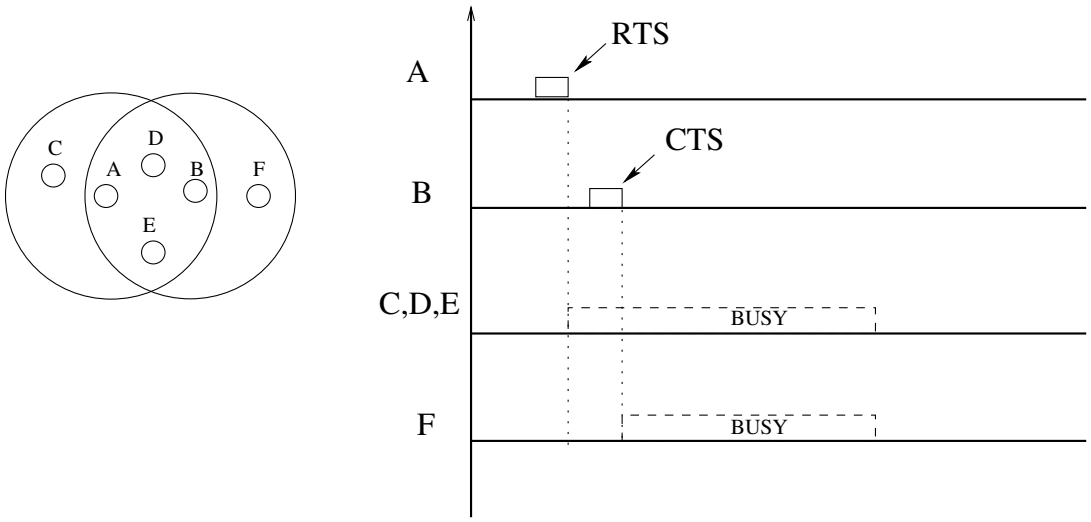


Figure 2.5: Virtual Jamming

2.4 Performance Matrix

2.4.1 Throughput

Throughput is a measure of how many units of information a system can process in a given amount of time. In networking, it is the rate of successful message delivery over a communication channel. Throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second (p/s or pps) or data packets per time slot. The goal of every communication channel is to maximize throughput. In describing throughput two term is used. **Instantaneous Throughput** is the throughput of a network measured over a short time of period. Mathematically, this is the limit taken with respect to throughput as time approaches zero. If the throughput of a network is measured over a long time of period it is called **Average Throughput**.

2.4.2 Delay

Delay is a important performance characteristic of a computer network system. The delay of a network means how long it takes for a bit of data to travel across the network from one node or endpoint to another. It is typically measured in multiples or fractions of seconds. Delay may differ slightly, depending on the location of the specific pair of communicating nodes. Here we calculated **Instantaneous Delay** and **Average Delay** to calculate the performance of network. **Instantaneous Delay** is the delay of a network measured over a short amount of time. **Average Delay** is measured the full transmission time.

2.4.3 Confusion Matrix

In the field of **machine learning** and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes.

2.4.4 Support Vector Machine

A support vector machine (SVM) is an important machine learning algorithm that analyzes data for classification and regression analysis. The objective of the support vector machine algorithm is to find a hyper-plane in an N-dimensional space(NŁŁthe number of features) that distinctly classifies the data points. It represents the examples as points in space so that the examples of the separate categories are divided by a clear gap . It is trained with a series of data already classified into two categories, building the model as it is initially trained. The task of an SVM algorithm is to determine which category a new data point belongs in. This makes SVM a kind of non-binary linear classifier.

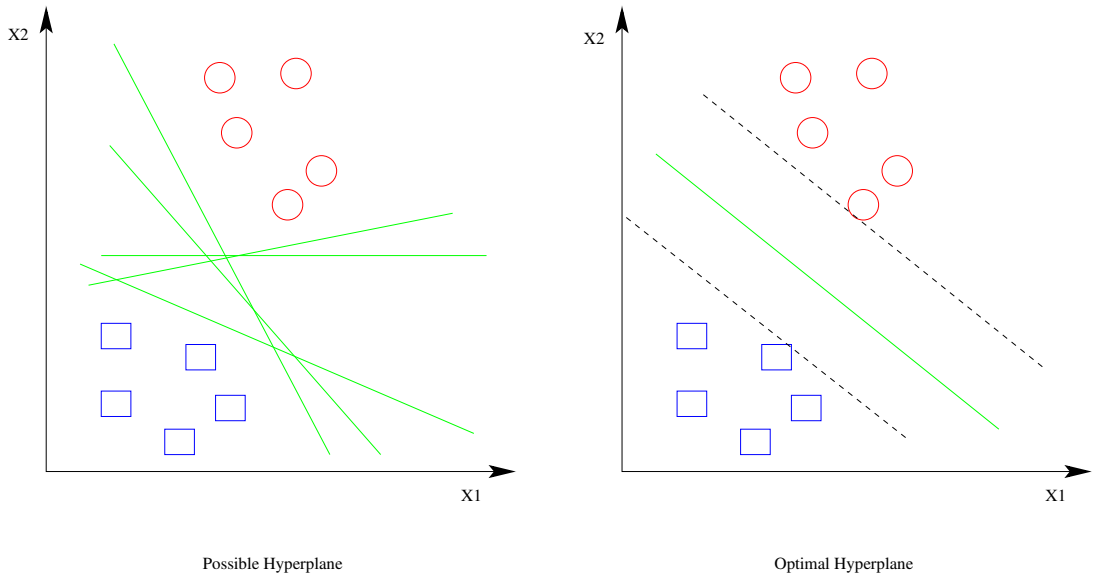


Figure 2.6: Hyperplanes in SVM

An SVM algorithm should not only place objects into categories, but have the margins between them on a graph as wide as possible 2.6. A support vector machine is also known as a support vector network (SVN).

Two or more featured can be used in a SVM. If two features are used the hyper-plane will be a

line. If three features are used the hyper-plane will be a plane 2.7.

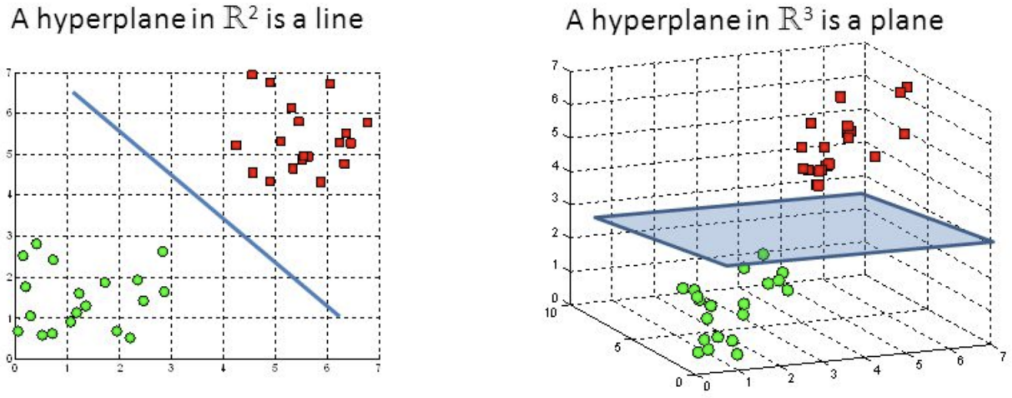


Figure 2.7: Hyperplanes in 2D and 3D

Chapter 3

Previous Works

3.1 RTS Validation

One solution to the **Denial of Service attack** is suggested in [6] and it is called RTS validation. Here a node receiving an RTS packet views it with a limited trust. The node only waits for and remains blocked until the time when CTS packet expected to arrive from the intended recipient. After that the sender should start the transmission of data packet. If the transmission does not happen, the the node unblocks. Otherwise, the node waits for the reminder of the transmission time announced in the RTS packet by the sender. The whole scenario is illustrated in Figure 3.1.

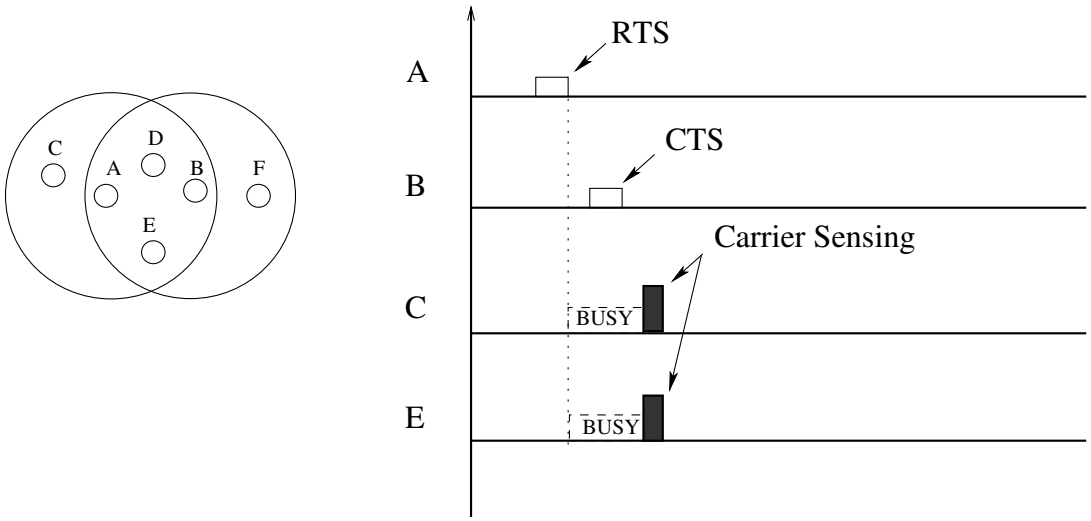


Figure 3.1: RTS Validation

3.1.1 Problems with RTS Validation

RTS validation does not work in the case of **advance attacker**. A scenario is illustrated in the Figure 3.2. Sometimes attacker sends a small data packet after receiving the CTS packet but declare a much longer time for transmission in the RTS packet. In RTS validation process, the neighboring node will check for transmission after waiting for the time needed to send the CTS packet.They will find transmission is going on(due to small data packet) and will be blocked for the entire time. It is a case of **False Negative**. An advance attacker can overcome RTS validation process and still jam a wireless network with small amount of power.

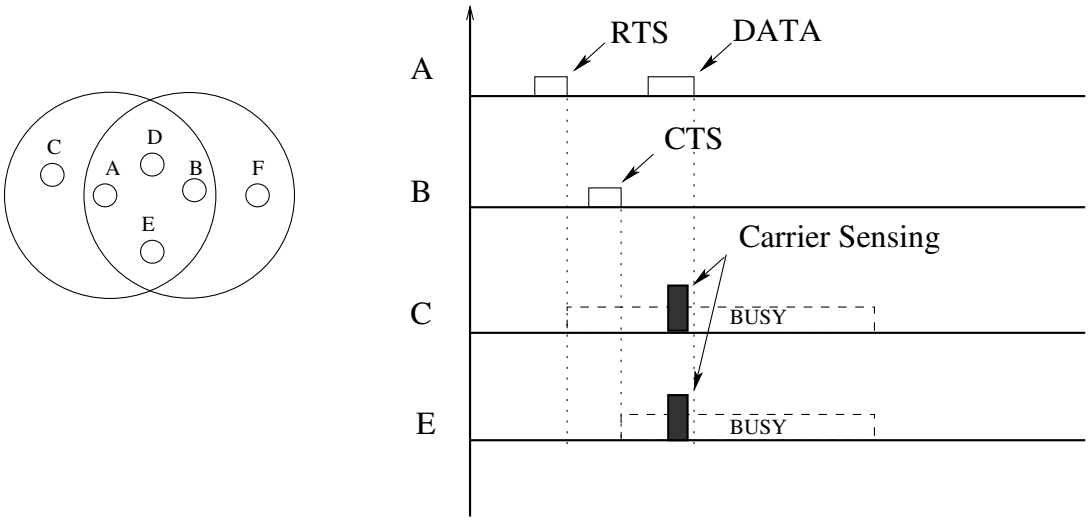


Figure 3.2: False Negative with RTS Validation

3.2 Random RTS Validation

To prevent the advance attacker A system called Random RTS Validation is proposed in [7]. It is a improved version of RTS Validation. In RTS Validation the neighboring nodes checks for transmission at the start of the time period of data transmission announced in the RTS packet. In Random RTS Validation the neighboring nodes check for transmission randomly at any part of the data transmission period. In this process the advance attacker can be detected and almost 50% of the data transmission period can be recovered. THe process is illustrated in Figure 3.3.

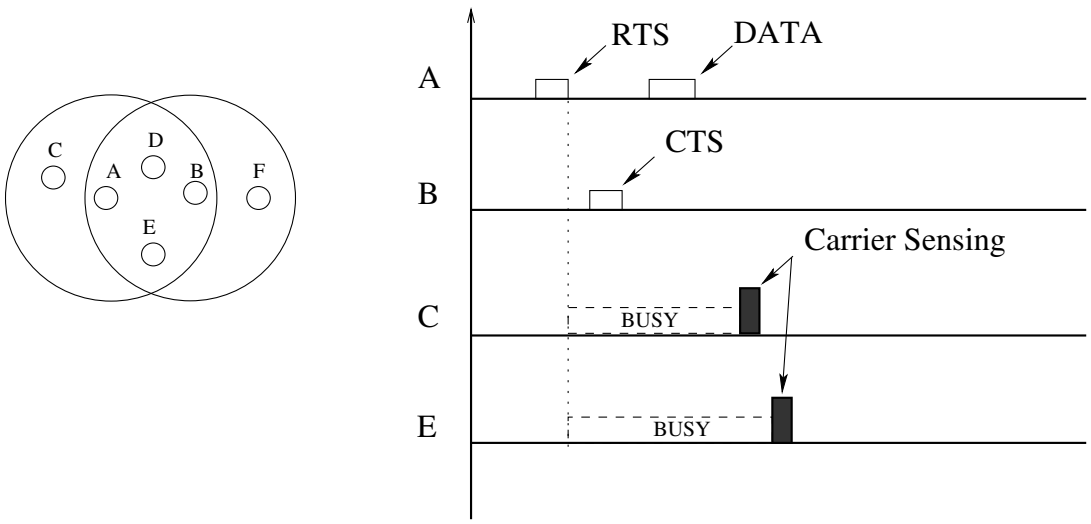


Figure 3.3: Random RTS Validation

Chapter 4

Methodology

4.1 Problem Definition

We discuss on the ”Advance” variant of virtual jamming problem, which allows a malicious node to effectively jam a large fragment of a wireless network at a minimum expense of power. We propose a solution to this problem and provide experimental data illustrating the impact of virtual jamming and the effectiveness of our proposed solution

4.1.1 Topology

To study the effect of ”Advanced” virtual jamming, we implemented an NS-2 model [8] of the two-dimensional static scenario shown in Figure Figure 4.1.

In this scenario, nodes 1 - 4 form a clique. Each of the outer nodes 5 - 8 is only reachable from exactly one of the inner nodes.

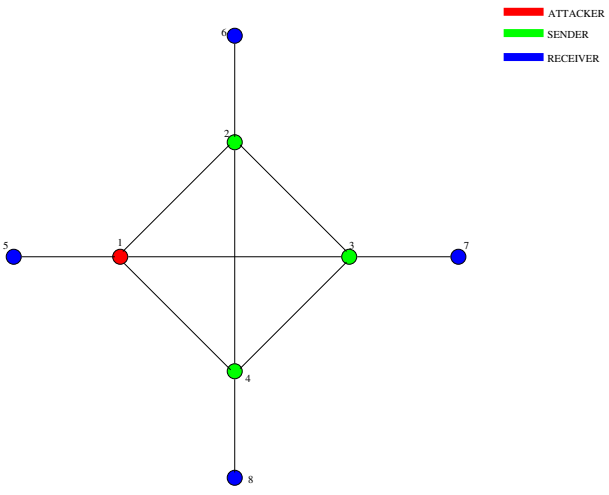


Figure 4.1: Static Scenario

4.1.2 Scenario with no jamming

In this scenario the traffic consists of three CBR flows (1024-Byte packets): $2 \rightarrow 6$,$3 \rightarrow 7$ and $4 \rightarrow 8$. The transmission rate (sufficient to keep the network saturated) is 1Mbps for all three sources.

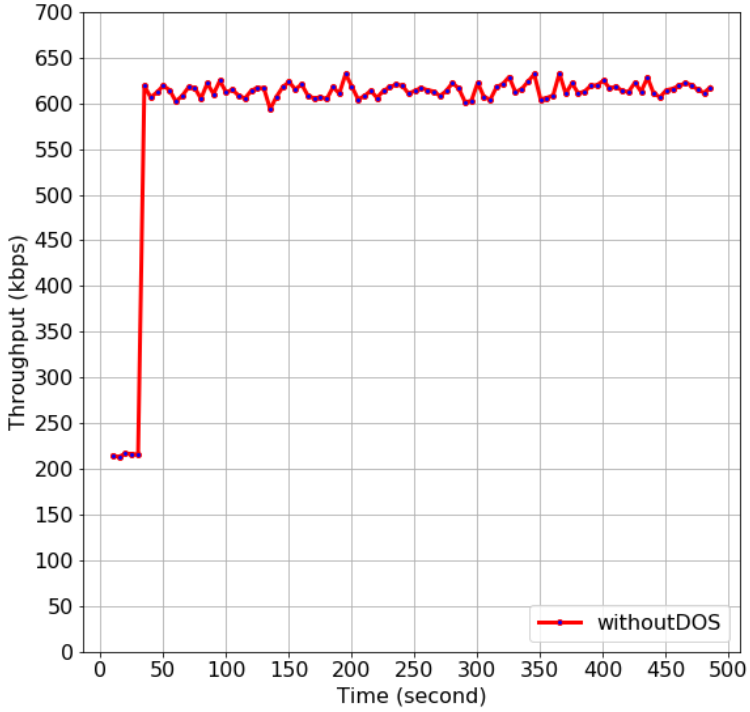


Figure 4.2: Aggregate throughput without jamming.

Figure 4.2. shows the aggregate throughput achieved by all three streams without jamming. The aggregate throughput is almost steady and constant for the total duration of the experiment, except at the beginning, before the network has reached the steady state.

4.1.3 Scenario With Jamming

In ihis scenario the traffic consists of three CBR flows (1024-Byte packets): $2 \rightarrow 6$,$3 \rightarrow 7$ and $4 \rightarrow 8$.But in addition to this node 1 (the attacker) sends dummy RTS packets to node 5 at the frequency of a regular traffic source. Those RTS packets are received by nodes 2, 3 and 4 and cause them to become blocked.The attack begins at second 200 and continues until second 300.

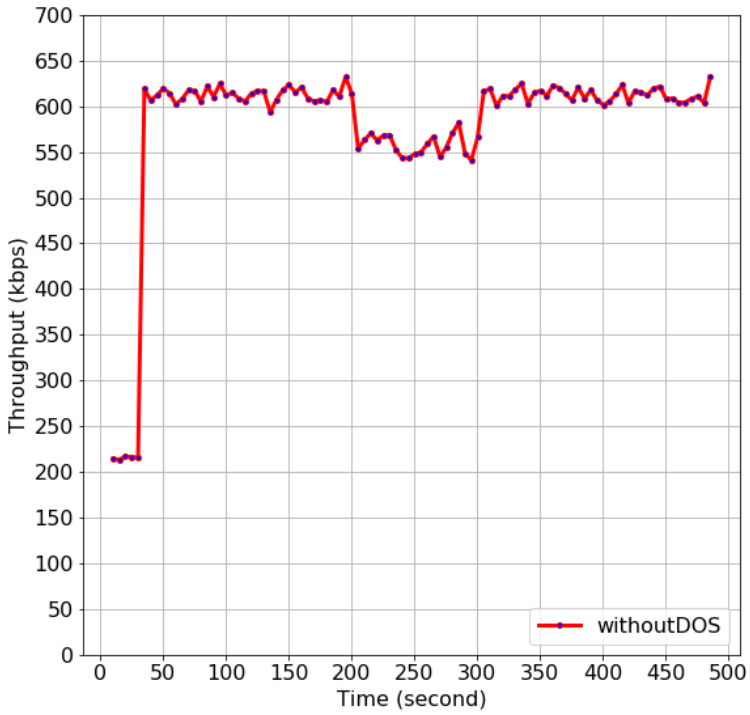


Figure 4.3: Aggregate throughput with jamming.

Figure 4.3 shows the aggregate throughput achieved by all three streams with node 1 jamming. The aggregate throughput is almost steady and constant for the total duration of the experiment, except at the beginning, before the network has reached the steady state and between 200 to 300 second when the node 1 was jamming the network. In the jamming period the aggregate throughput drops by almost 16%.

4.2 Random RTS Validation

With one solution to this problem, suggested in [7] and dubbed *Random RTS validation*, a node receiving an RTS packet views it with a limited trust. The node only remains blocked until a random time. In this time the node checks if the sender is transmitting the data packet or not. If the transmission does not happen, then the node unblocks itself.Otherwise, it will continue waiting for the remainder of the transmission time announced in the RTS packet by the sender.

4.2.1 Scenario with random RTS validation

In this case scenario is same as 4.1.3 referred earlier. Only addition that this time every node is applying *Random RTS validation*.

4.2.2 Effect on aggregate throughput

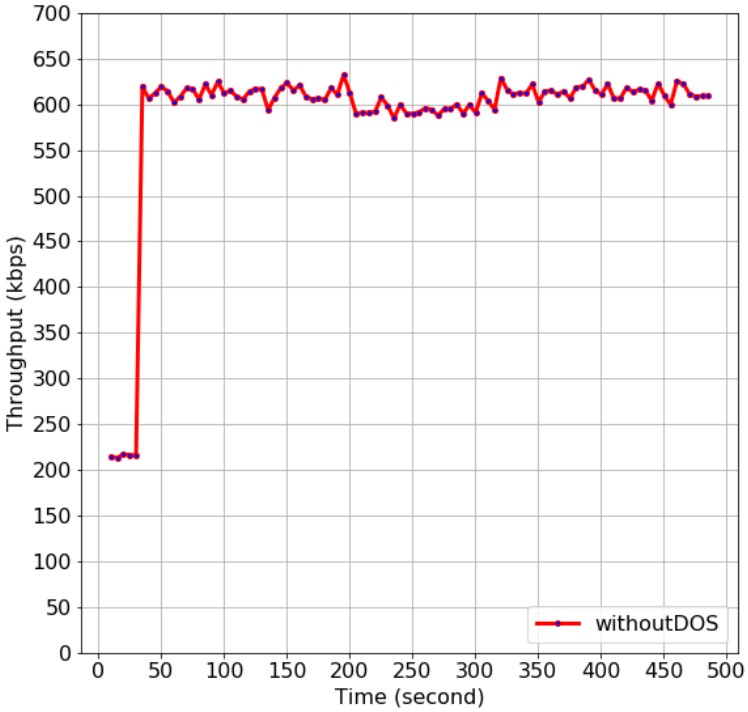


Figure 4.4: Aggregate throughput with Random RTS Validation.

fig. 4.4 shows the aggregate throughput achieved by *Random RTS Validation*. We can see from the figure that almost 50% throughput loss is recovered.

4.3 Machine Learning Approach

In the previous method for every packet we validate the RTS and classified the nodes as malicious and trivial nodes. When an observer node classified a node as malicious node it unblocked itself. But it didn't remember the malicious node. So when in the next term malicious node again did virtual jamming it have to reclassify the malicious node. In this process we lose 50% dropped throughput.

But from the behaviour of a malicious node it is expected that it will keep jamming the network repeatedly. So if we can classify a node confidently we can ignore it for a certain period of time. This is the basis of Machine Learning Approach. In congested traffic there are many False Blocking scenarios which results in similar malicious behaviour. So there stands a possibility that we misclassify a good node. If it is ignored permanently that will be unwanted.So its behaviour have to be reevaluated after a certain period.

So this machine learning algorithm runs periodically. Every period has 2 parts.

- Learning period.
- Action period.

4.3.1 Learning period

In this time interval the algorithm observe the behaviour of the neighbouring nodes and collects features about them. In this period *Random RTS Validation* runs for every nodes. So in this period in average 50% throughput loss is regained.

4.3.2 Action period

In this period each node classifies its neighbouring nodes as good node or bad node by the features gathered in the last *Learning Period*, previous *Action period* and previously measured features. Each node ignores RTS packets from those nodes which are classified as bad node for rest of the period. And for other nodes it keeps validating them with *Random RTS Validation* and also keep observing their behaviour and collecting features.

So in the action period 100% throughput is recovered.

4.3.3 Analysis

In order to analyse the algorithm we introduce the term **LAR** or *Learning to Action Ratio*. **LAR** is the ratio of Learning period and Action period.

In the Learning phase we get 50% recovery and in the action phase we get 100% recovery. By the definition of **LAR**, we can say that if the Learning phase runs for LAR s than the action phase will run for 1 s. SO the recovery in Learning period,R_L will be

$$R_L = \frac{LAR}{1 + LAR} * 50\%$$

And the recovery in Action period, R_A will be

$$R_A = \frac{1}{1 + LAR} * 100\%$$

So the total average recovery, R will be $R_L + R_A$

$$R = \frac{1}{2} + \frac{1}{2 * (1 + LAR)}$$

We know the *Random RTS Validation* approach has 50% recovery. So ML approach will have an improvement of $\frac{1}{2*(1+LAR)} * 100\%$

4.3.4 Selected Features

In order to find the suitable features to classify the nodes as bad and good different random scenario was studied with topology containing 20 to 100 nodes. Where about 25% was sender. Both with attacker and without attacker scenarios was studied.The following 3 features were selected.

- Moving Average of IRR.
- Deviation of Moving Average of IRR.
- Moving Average of Inter Arrival Time of RTS packets

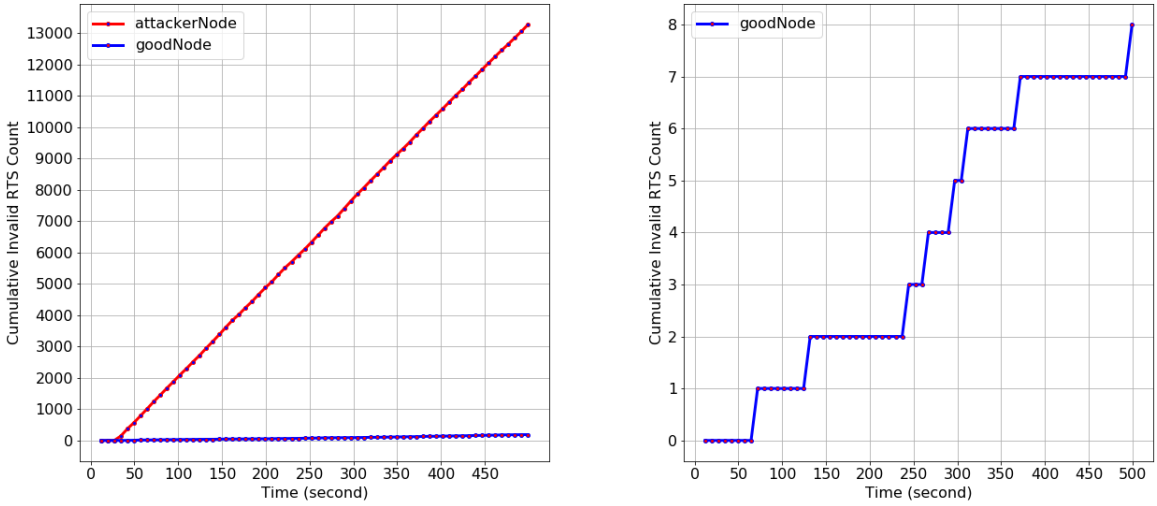
To understand these features we will consider 2 scenarios both having 25 nodes and 6 sources. But in one scenario there are 2 attackers and in the other there is none.

4.3.4.1 Moving Average of IRR

IRR *IRR* means Invalid RTS ratio. We call a RTS invalid if there is no Data packet following it.So the IRR refers to the ratio of Invalid RTS count and Total RTS count.

$$IRR = \frac{InvalidRTS}{TotalRTS}$$

In every Learning period and Action period when an RTS packet is received Total RTS count is incremented and in the time of *Random RTS Validation* if no data packet is sensed then the Invalid RTS count is incremented. It is very obvious that for an attacker the Invalid RTS count will be very high then the normal node. The effect in cumulative Invalid RTS count is shown in fig. 4.5



(a) In Scenario with attacker.

(b) In scenario with no attacker.

Figure 4.5: Effect in Cumulative Invalid RTS Count over Time

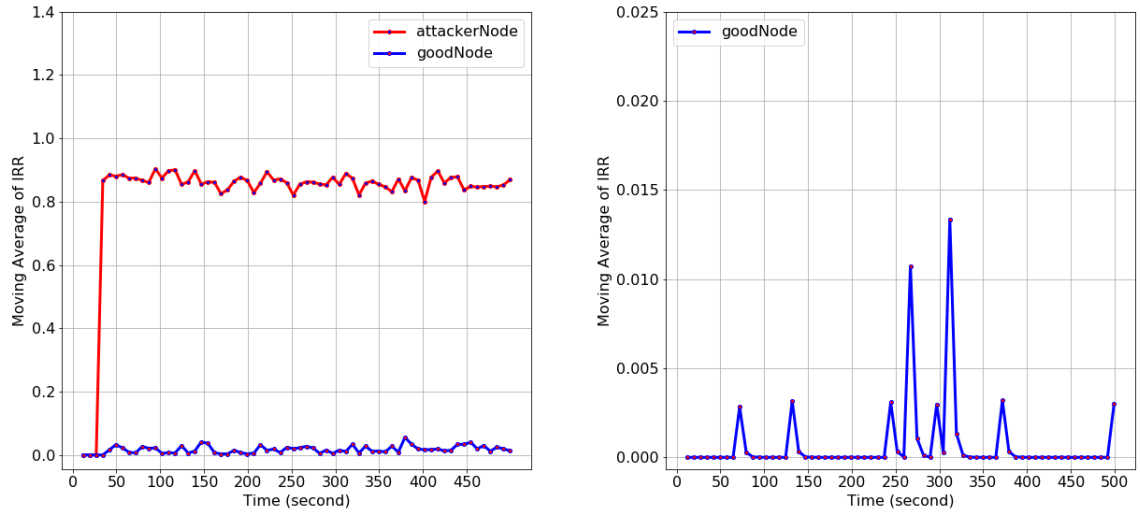
In fig. 4.5a the cumulative Invalid RTS count is shown for 2 source in respect to one of their neighbour. It is very clear that for the attacker it is very high compared to the other source.In fig. 4.5b the cumulative Invalid RTS count is shown for 1 innocent source with respect to its neighbour.There was no attacker in this scenario.

So it seems that we can classify attackers and innocent sender in respect to Cumulative RTS count. But there are some problems regarding this parameter. It will vary with the Senders data transmission rate and data transmission start time. In order to skip these problem we are considering the moving average of IRR.

So as we taking IRR it is certain that for an attacker its value will be high and for non attacker it will be low. In order to find the moving average at 1st IRR_L in last Learning period and IRR_A in last Action period will be calculated. Now we will calculate the IRR_{new} with respect to IRR_{old} calculated in last period from the following equation.

$$IRR_{new} = (1 - coffIRR_L - coffIRR_A) * IRR_{old} + coffIRR_L * IRR_L + coffIRR_A * IRR_A$$

Here $coffIRR_L$ and $coffIRR_A$ are coefficient of Learning and Action period use to prioritize them and act as hyperparameter.



(a) In Scenario with attacker.

(b) In scenario with no attacker.

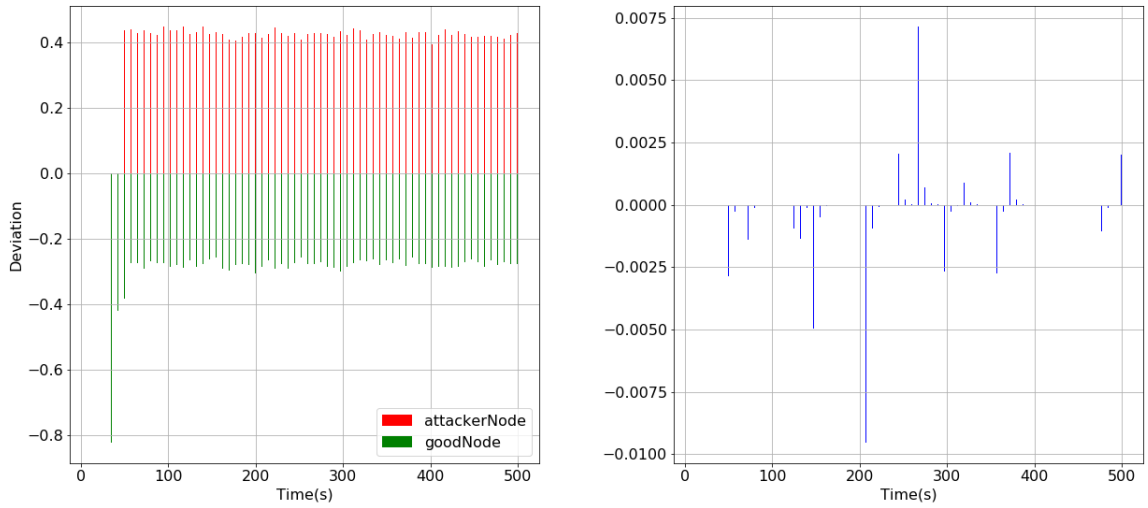
Figure 4.6: Effect in Moving average of IRR over Time

fig. 4.6 shows the effect in moving Average of IRR of senders with respect to its neighbour.fig. 4.6a shows the moving average of IRR of 1 attacker and 1 innocent sender with respect to their neighbour for a scenario containing malicious node. fig. 4.6b shows the moving average of IRR of 1 innocent sender with respect to its neighbour for a scenario containing no malicious node

Decision Point From the fig. 4.6 it is certain that we can classify bad and good nodes with respect to Moving Average of IRR.

4.3.4.2 Deviation of Moving Average of IRR

The deviation of moving average is calculated for each neighbouring sender respect to each receiver. It is obvious that for a attacker the deviation is almost always positive. And its value should be relatively high. On the other hand for innocent senders deviation will be almost always negative. And if its positive its value will be very low.



(a) In Scenario with attacker.

(b) In scenario with no attacker.

Figure 4.7: Effect in Deviation of Moving average of IRR over Time

fig. 4.7 shows the effect in Deviation of moving Average of IRR of senders with respect to its neighbour.fig. 4.7a shows the deviation of moving average of IRR of 1 attacker and 1 innocent sender with respect to their neighbour for a scenario containing malicious node. fig. 4.7b shows the deviation of moving average of IRR of 1 innocent sender with respect to its neighbour for a scenario containing no malicious node.

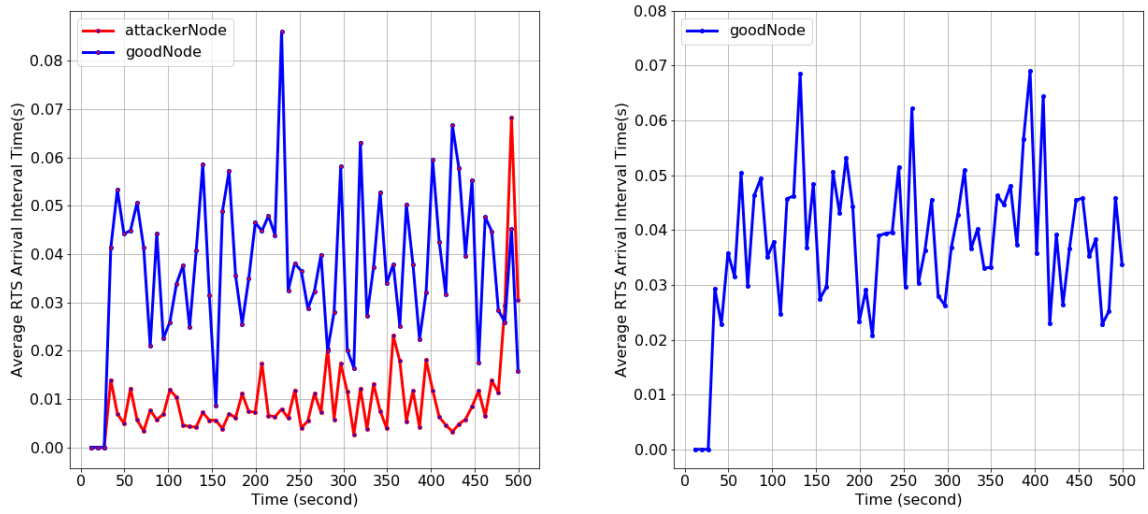
Decision Point From the fig. 4.7 it is certain that we can classify bad and good nodes with respect to Deviation Moving Average of IRR.

4.3.4.3 Moving Average of Inter Arrival Time of RTS packets

When a good sender sends data it have to face many delay caused by traffic and retransmitting data. It also have to wait for receivers ACK packets and cope with its delay. Even it has to back off to give other opportunity to transmit packets. So when judging from a neighbour for normal sender average inter arrival time of RTS will be high. But an attacker is not bounded by these delays. So the average inter arrival time of RTS will be low. For every RTS received we calculate the moving average of Inter Arrival TIME or RTS by the following equation.

$$IAT_{new} = (1 - coffIAT) * IAT_{old} + coffIAT * (CurrentTime - lastRTSTime)$$

Here $CurrentTime$ refers to the current time when we got the new RTS , $lastRTSTime$ refers to the time lasr RTS packet was received and $coffIAT$ is the coefficient of the RTS arrival interval act as hyperparameter.



(a) In Scenario with attacker.

(b) In scenario with no attacker.

Figure 4.8: Effect in Moving average of IRR over Time

fig. 4.8 shows the effect in moving Average of Inter Arrival Time of RTS of senders with respect to its neighbour. fig. 4.8a shows the moving average of inter arrival time of RTS of 1 attacker and 1 innocent sender with respect to there neighbour for a scenario containing malicious node. fig. 4.8b shows the moving average of inter arrival time of RTS of 1 innocent sender with respect to its neighbour for a scenario containing no malicious node.

Decision Point From the fig. 4.8 it is certain that we can classify bad and good nodes with respect to the Moving Average of inter arrival time of RTS packets.

Chapter 5

Dataset Preparation

So far there is no dataset to feed our machine learning algorithm. To activate our machine, we need to create a dataset. In this topic we cover how we collected our data. We also cover which machine learning model is used to learn. Finally, learner's properties are given as well as learner's result on test dataset is given at the end of this topic.

5.1 Dataset Properties

To create data set we randomly choose some scenarios by varying number of nodes in NS-2 [8]. We change number of nodes from 30 to 100, in each step we increase 10 nodes.For each scenario, there are 10 variations of the node positions. Packet sending interval is 0.08s. Each packet size is 1024 bit. We run the simulation 500s for each scenario.

From Table 5.1 we find the properties of dataset.

Number of Attributes	4
Number of rows	2320449
Size	75.4MB
Missing Data	No
Outliers	Present

Table 5.1: Dataset properties

In Table 5.2, we can see random 5 rows of the dataset.

movingAverage	Deviation	averageRTSInterval	isAttacker
0.0411155	-0.0462200	0.0550714	0
0.3691936	0.2818581	0.0039915	1
0.0458663	-0.0414691	0.0892042	0
0.0250579	-0.0622775	0.0899507	0
0.3288425	0.2734656	0.0086242	1

Table 5.2: Dataset

5.2 Model selection: Support Vector Machine

As we know that our dataset is linearly seperable, we can feed our dataset to SVM model to learn. After learning, it will return a vector of coefficients and a bias. Actually these coefficients ar the coefficients of a hyperplane. SVM tries to separate the dataset with respect to a hyperplane.

At first we have to do train test splitting where SVM model learn from training set and validating at the test set. For our data set we split 80:20 ratio with respect to train:test.

Here we use scikit-learn API [9] to do this.

Then SVM model training started.

After finishing the training we get the output.

In A.1, necessary code is given.

From table 5.3 we can find the coefficients along the feature axes of dataset.

Axis name	Coefficients
movingAverage	2.59804693
deviation	2.5296639
averageRTSInterval	-0.36001139
bias	-1.09308601

Table 5.3: SVM coefficients

5.3 Training Results

5.3.1 Confusion Matrix Property

From table 5.4 we can see the confusion matrix properties value.

Property	value
True Positive	85083
False Positive	10081
False Negative	17436
True Negative	351490

Table 5.4: Confusion Matrix

5.3.2 False positive ratio and Accuracy

From table 5.5 we can see the ratio and accuracy.

Property	value
False Positive Rate	0.02788
Accuracy	0.94070

Table 5.5: False Positive ratio and Accuracy

Now we set the trained model in NS-2 [8] simulator and run the simulation to capture the effectiveness of this model.

Chapter 6

Experimental Results

For the verification and the comparison of the effectiveness of Random RTS validation approach and Our proposed Machine learning approach on virtual jamming, we execute some simulation experiments on NS-2 [8] simulator. Instantaneous Throughput, defined in Section 2.4, is used as the performance metric to compare. Higher curve of throughput across the simulation time implies the acceptance of that approach.We also use Instantaneous Delay, defined in Section 2.4, as the performance metric where a lower curve implies the acceptance of corresponding approach.

6.1 Simulation Environment

To create topological scenarios, we take a 2D area of 625 $\times$ 625 sq unit where eight nodes are stayed. The transmission range is 250 unit. Packet size is 1024 bit. A sender can send maximum 10000 packets. Interval time between consecutive packets is 0.04 s that means a sender sends 25 packets in a second.Code We run the simulation for 500 seconds. Later, by varying the number of nodes, we create some random scenario. We vary the number of nodes from 10 to 30. Then we run these with two mentioned approaches.

6.2 Performance comparison of different approaches

6.2.1 Throughput comparison

Firstly, we run the simulation in NS-2 [8] simulator while there is no virtual jamming. Then we collect instantaneous throughput across the time. By plotting them, we get Throughput vs Time graph. We calculate the average throughput for normal scenario.

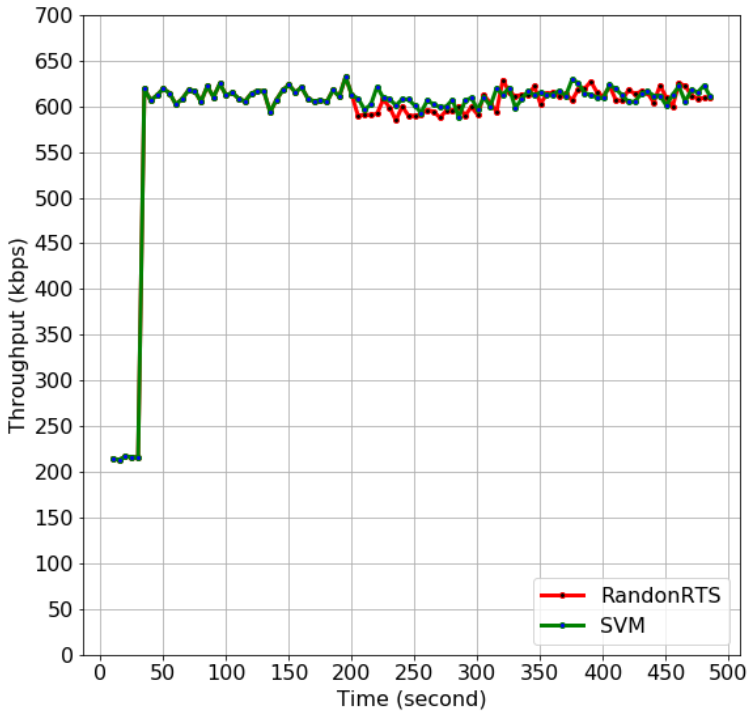


Figure 6.1: Throughput comparison between RandomRTS and SVM

Normal scenario consists of 3 senders with packet sending interval of 0.04 second. The calculated average throughput here is 593.643 kbps.

Advanced variant of Virtual jamming occurs at 200^{th} second and continues for 100 seconds by a node.For this, RTS receivers of that node become blocked. Hence the throughput in the jamming interval decreased. So do average throughput. Average throughput while jamming is 580.998 kbps.

To overcome virtual jamming we go through the existing Random RTS validation process. By doing this, average throughput becomes 588.418 kbps.

We want to increase the average throughput to attain the normal average throughput.

Then finally, we execute our machine learning based classification approach to identify the bad nodes.After that, the average throughput gets higher. Average throughput becomes 590.534 kbps which is greater than Random RTS validation's throughput.

From Figure 6.1, we can see that, the throughput in the attacking time, gets higher.

In compare to normal scenario, how good our model can be gound in figure Figure 6.2

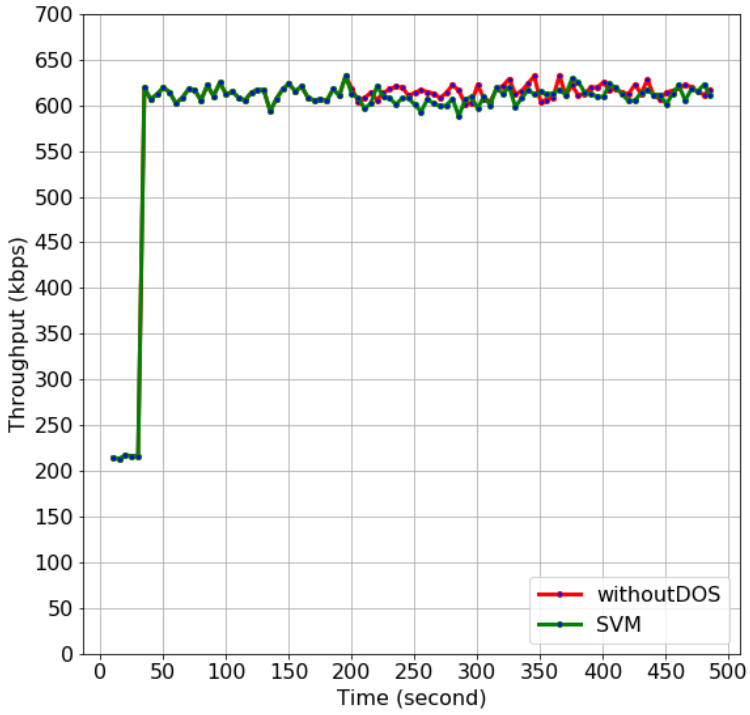


Figure 6.2: Throughput comparison between Normal Scenario and SVM

From Figure 6.3 we can capture all comparisons under different scenarios.

6.2.2 Delay comparison

Now we draw the instantaneous delay curve for normal scenario, virtual jamming scenario, random RTS validation and our machine learning approach.

From figure Figure 6.4 we can see that while jamming is active, delay gets higher.

Then to decrease the delay in the jamming time, we run Random RTS validation approach.

Finally, on this situation we approach with classification based model in order to decrease the delay through the jamming time with respect to Random RTS validation.

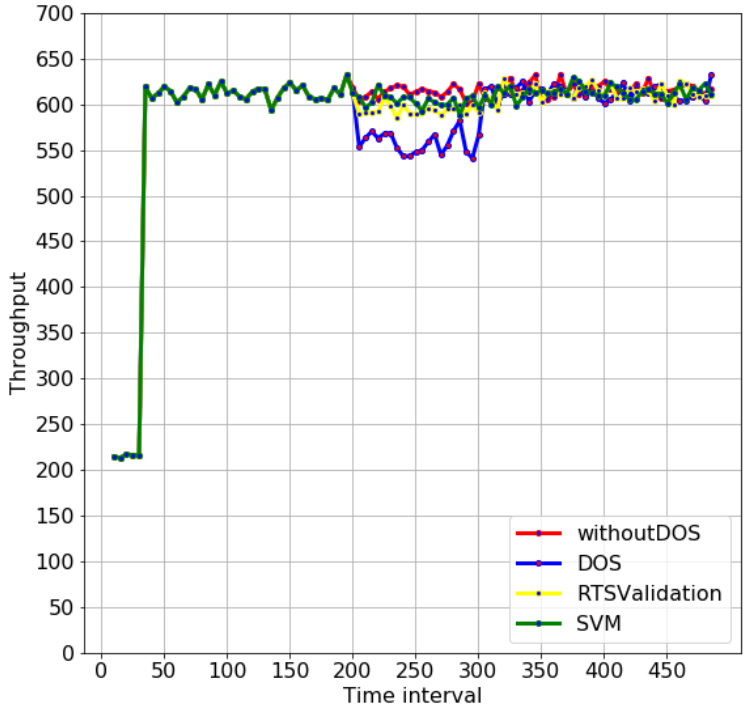


Figure 6.3: Throughput comparison under different approaches

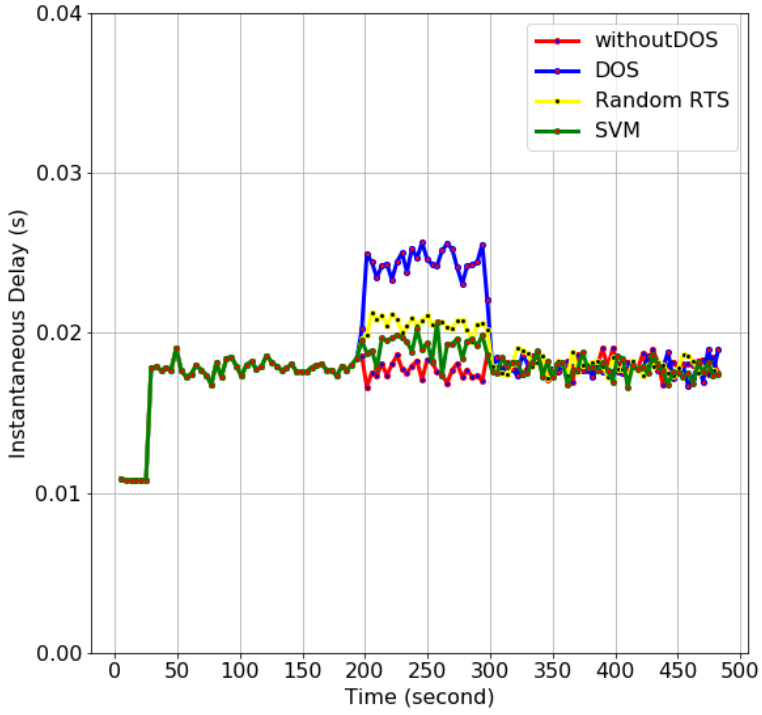


Figure 6.4: Delay comparison under different approaches

Chapter 7

Conclusions and Future works

In this section we discuss about the summary of our works in short. Future works are also included here.

7.1 Conclusion

For wireless routing channel system, virtual jamming is one of the reason of denial of service of channel's bandwidth. It decreases the average throughput. Our proposed machine learning approach take the data from learning period and use them in the starting of action period to classify a good or bad node. Our system will ignore the bad node. From the results, we can conclude that our system is capable of increasing the average throughput. We execute our proposed process in the network simulator, NS-2 [8]. Then we plot the instantaneous delay curve over the time under different approaches to capture the effectiveness respectively.

On the other hand, if there is any miss-classification in the current action period, our system has the flexibility so that it can learn in the next learning period and fix the problem in the next action period. For example, somehow outlier data from a node can arrive in the learning period, then our system will give the node another chance in the next learning period, then system will classify appropriately.

We finally run our proposed system in some random scenarios. It is capable in there too. It increases the average throughput for simple scenario.

7.2 Future works

In future, we plan to carry out more simulations under this classification based model.We now think only RTS with small packet attack. We can think of other problem scenarios if machine

learning can help to overcome these p[roblem.

7.2.1 Tuning Learn to Action Ratio

In this model, we take $LAR = 0.5$, defined in 4.3.3 . we can change it to get new data to feed the model wheather the model could perform better.

7.2.2 Feature Selection hyperparameters tuning

We use some hyperparameter, defined in 4.3.4, in the features selection step. We can change $coeffIRR_L$, $coffIRR_A$, $coeffIAT$ and get new dataset to feed the machine, in our case SVM.

7.2.3 CTS only attack

In future, we plan to solve the CTS only attack with machine learning. So far, we think of RTS plus small data packet attack. Now, for only CTS attack, we have to find some features to to feed the machine learning model so that he can learn and classify later. Here Acknowledgement validation can help us to detect the intruders.

The scenario from figure 7.1, A is launching the jamming by sending CTS packets to its neighbours despite he received any RTS packet from them. Then channel bandwidth usage decreases. One feature can be significant the valid ACK count at the ACK sending time by that noe.

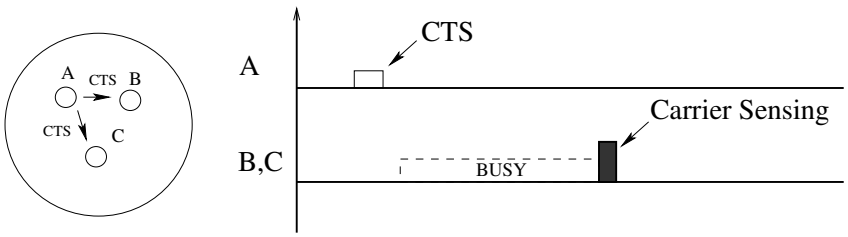


Figure 7.1: CTS only attack carrier sensing

7.2.4 CTS plus ACK attack

In this case, machine learning can not help us, beacuse there is no source id. In fact CTS packet can't hold any source id. Signal strength based classification can be used there.

References

[1] M. G. K. Xu and S. Bae, "How effective is the ieee 802.11 rts/cts handshake in ad hoc network?," *IEEE GLOVECOM*, vol. 1, pp. 17–21, November 2002.

[2] P. Karn, "Maca-a new channel access method for packet radio.," *9th Computer Networking Conference on ARRL/CRRL Amateur Radio*, pp. 134–140, September 1990.

[3] A. M. A. Acharya and S. Bansal, "Maca-p,a mac for concurrent transmission in multi-hop wireless newtorks," *First IEEE Internation COnference on Pervasive Computing and Communications PERCOM*, 2003.

[4] J. B. C. S. Ray and D. Starbinski, "Rts/cts-induced congestion in ad hocwireless lans," *WCNC*, 2003.

[5] M. Y. S. Uddin and R. Rafiq, "802.11 denial-of0service attacks: Real vulnerabilities and practical solutions," *Proceedings of the USENIX Security Symposium*, August 2003.

[6] J. D. D. Chen and P. K. Varshney, "Protecting wireless networks against a denial of service attack based on vietual jamming," *The Ninth ACM Annual International Conference on Mobile Computing and Networking(Mobicam) Poster*, September 2003.

[7] P. G. Ashikur Rahman, "Hidden problems with the hidden node problem," *International Journal of Environmental Science and Development*.

[8] "The network simulator: Ns-2: notes and documentation. http://www.isi.edu/nsnam/ns/,"

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Appendix A

Codes

A.1 Necessary code for Learning Part

Here is the code for training the machine part and results.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 21 19:47:51 2019

@author: yeaseen
"""

from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
import numpy as np

from sklearn.metrics import confusion_matrix, classification_report
def split(s, delim):
    words = []
    word = []
    for c in s:
        if c not in delim:
            word.append(c)
        else:
            if word:
                words.append(''.join(word))
                word = []
    if word:
```

```
25          words.append(''.join(word))
26      return words
27
28 def loadfile(filename):
29      file = open(filename, "r")
30      rows = list()
31      for line in file:
32          vals = split(line, [' ' ,'\t', '\n'])
33          rows.append(vals)
34      return rows
35
36
37 data=loadfile('/media/yeaseen/Y D U T S/ScenarioFile/svm.txt')
38 data=np.array(data)
39 data= data.astype(np.float)
40
41 data=np.delete(data,-1,axis=1)
42
43 dataY=data[:,-1].copy()
44 dataY=np.array(dataY)
45
46 dataX=np.delete(data,-1,axis=1)
47
48 X_train, X_test, Y_train, Y_test = train_test_split(dataX, dataY, test_size=0.2,
49
50
51 clf = LinearSVC(random_state=0, tol=1e-5)
52
53 print("started fitting")
54 clf.fit(X_train, Y_train)
55
56
57 w=clf.coef_
58
59 b=clf.intercept_
60
61 PY_train=[]
62 for r in X_train:
63      value=clf.predict([[r[0], r[1], r[2]]])
64      PY_train.append(value[0])
65
```

```
66 from sklearn.metrics import accuracy_score
67 print("Accuracy␣is␣in␣train␣set:␣␣␣"+str(accuracy_score(Y_train, PY_train)))
68
69
70 ww=np.array([w[0], w[1], w[1], b])
71
72
73 PYY_test=[]
74 for r in X_test:
75     rr=np.array([r[0], r[1], r[2], 1.0])
76     val=np.matmul(ww,rr.T)
77     if(val<0):
78         PYY_test.append(0.0)
79     elif(val>=0):
80         PYY_test.append(1.0)
81
82
83 from sklearn.metrics import accuracy_score
84 print("Accuracy␣is␣test␣set:␣␣␣"+str(accuracy_score(Y_test, PYY_test)))
85
86
87 print(confusion_matrix(Y_test, PYY_test))
88 print(classification_report(Y_test, PYY_test))
89
90 def perf_measure(y_actual, y_hat):
91     TP = 0
92     FP = 0
93     TN = 0
94     FN = 0
95     for i in range(len(y_hat)):
96         if y_actual[i]==y_hat[i]==1:
97            TP += 1
98         if y_hat[i]==1 and y_actual[i]!=y_hat[i]:
99            FP += 1
100        if y_actual[i]==y_hat[i]==0:
101           TN += 1
102        if y_hat[i]== 0 and y_actual[i]!=y_hat[i]:
103           FN += 1
104    return (TP, FP, TN, FN)
105
106 def printMeasure(TP,FP,TN,FN):
```

```
107     print(TP,FP,FN,TN)
108     print('false positive rate: ')
109     FDR = FP/(FP+TN)
110     print(FDR)
111
112 TP,FP,TN,FN=perf_measure(Y_test,PYY_test)
113 printMeasure(TP,FP,TN,FN)
```

Generated using Undegraduate Thesis LaTeX Template, Version 1.4. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Saturday 20th May, 2023 at 6:56pm.