International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 12, Issue 6(June 2016), PP.07-19

Mitigation And Control Of Defeating Jammers Using P-1 Factorization

G Krithika, K Kamala
Student, ME-Communication Systems, Valliammai Engineering College, Potheri, Chennai 603203. India
Assistant Professor (SG), Dept of ECE, Valliammai Engineering College, Potheri, Chennai 603203. India

Abstract:-The Jamming resistant broadcast communication is crucial for safety-critical applications such as emergency alert broadcasts or the dissemination of navigation signals in adversarial settings. These applications share the need for guaranteed authenticity and availability of messages which are broadcasted by base stations to a large or unknown number of (potentially untrusted) receivers. Common techniques to counter jamming attacks such as Direct-Sequence Spread Spectrum(DSSS) and Frequency Hopping are based on secrets that need to be shared between the sender and the receiver before the start of the communication. However broadcast anti jamming communication that relies on Pollard's Rho method. In this work we therefore propose a solution called P-Rho method to enable spread-spectrum anti-jamming broadcast communication without the requirement of shared secrets. The location of a jammer in a network is identified by a technique called JADE. The type of jammer present in the network is detected by a method called MIR. By identifying the type and location of jammer present in the network transmission between the sender and the receiver can be achieved safely and successfully.

Index Terms:- Jammers, JADE, MIR, Wireless Networks.

I. INTRODUCTION

The Wireless networks makes use of shared transmission medium therefore, they are vulnerable to several malicious attacks. These attacks are typically referred to as jamming. An attacker with a radio transceiver intercepts a transmission, injects spurious packets, blocks or jams the legitimate transmission is called as jammer.

Jammers disrupt wireless communication by generating high-power noise across the entire bandwidth near the transmitting and receiving nodes. Since jamming attacks drastically degrade the performance of wireless networks, some effective mechanisms are required to detect their presence and to avoid them.

Some techniques conventionally used for the detection and location of jammers in a wireless networks are as follows. Conventional anti-jamming techniques rely on spread spectrum (SS) communications, such as direct sequence spread spectrum (DSSS) and frequency hopping spread spectrum (FHSS). DSSS provides bit-level protection by spreading bits according to a secret pseudorandom noise (PN) code, known only to the communicating parties. In FHSS, the sender and the receiver hop synchronously using a secret random frequency hopping (FH) sequence.

For jamming-resistant broadcast communications, a common secret to be shared between n the sender and all (potentially non-trustworthy) receiver. The disclosure of this common secret due to the compromise of any receiver nullifies the SS gains. After the studies, Researchers assumed that the jammer cannot flip a bit '1' to a bit '0'. It was shown that a jammer cannot erase packets from the wireless channel. They proposed a method called Uncoordinated DSSS (UDSSS), and RD-DSS.

Accurate detection of radio jamming attacks is challenging in mission critical scenarios. Many detection techniques have been proposed in the literature, but the precision components is always an issue. Some of them either produce high false alarm rates or do partial detection of jamming attacks.

II. ARCHITECTURE OF THE SYSTEM

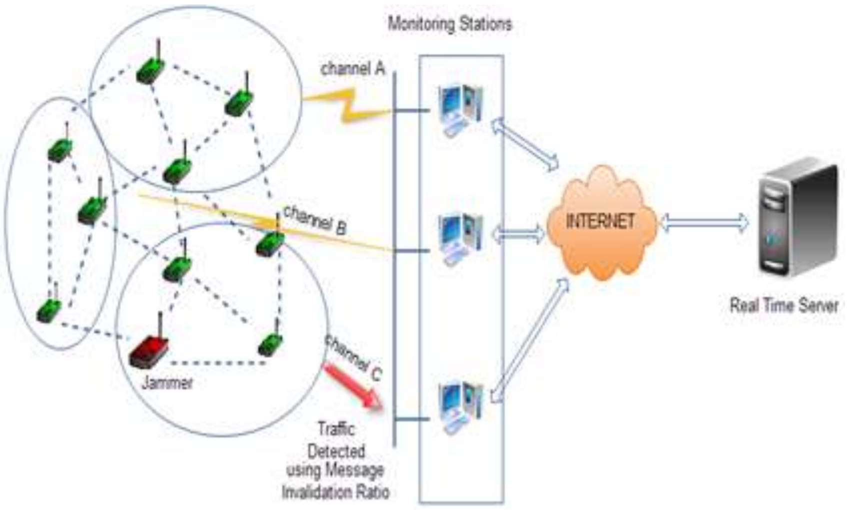

Fig.1. Transmission of packets in the presence of Jammers

Set of nodes forms a cluster based on the location of the nodes. It registers the node details such as its IP address etc.. with the server. Server checks the path for transmission and identifies the trusted nodes for secure transmission. The node checks for the trusted value given by the server for each node and the value should be the same for all the nodes in the path. If the values are found to be not equal in a path then that node is identified to be the attacker node and no transmission is continued further in that path. Therefore the server chooses the alternate path by again identifying the trusted node values.

III. SYSTEM DESCRIPTION

A. CLUSTER CONFIGURATION

In this module, each node registers the details such as Node IP address, Port number and distance. Node details are stored and maintained in a server database. After that nodes enter the IP and port number to activate themselves in the network.

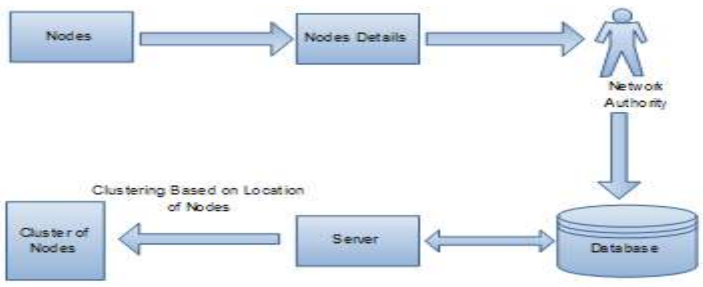

Fig.2. Cluster Configuration

B. CONSTRUCTION OF GOOSE

Generally time-critical traffic is used for monitoring, control and protection of electronic devices on Physical infrastructures. So in this module GOOSE message format is used. GOOSE message consists of a GOOSE header and the message information to the receiver. It can deliver the message to time-critical application with the delay of 3 ms to 10 ms

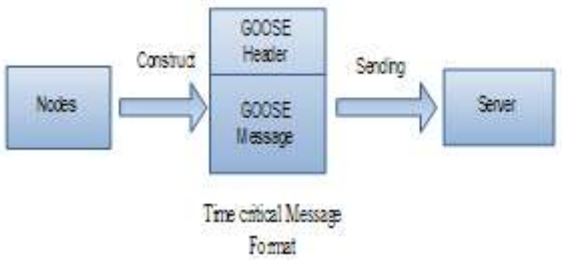

Fig.3. Construction of GOOSE

C. ESTIMATE MESSAGE INVALIDATION RATIO

Message Invalidation Ratio is mainly used to quantify the performance of the time critical applications in the presence of jammers.

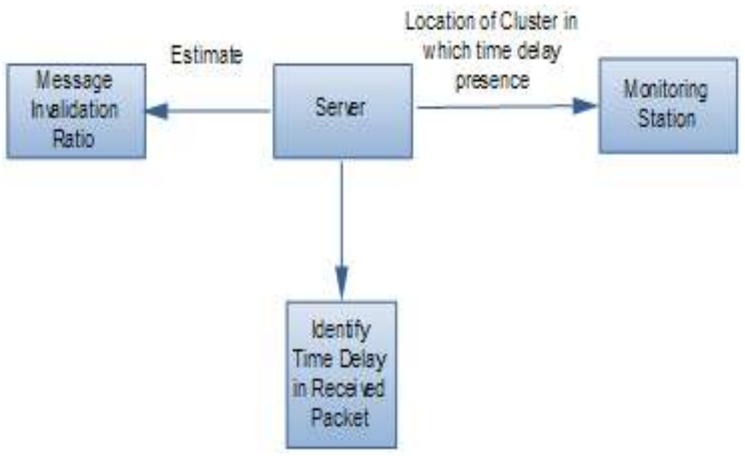
Estimate
Location of Cluster in which time delay presence
Message Invalidation Ratio
Server
Monitoring Station
Identify Time Delay in Received Packet

Fig.4. Estimate Message Invalidation Ratio

D. DETECTION OF JAMMER

With the help of Message Invalidation Ratio probability and the number of needed samples the malicious jammer in the group can be detected. JADE technique is used detecting the location of the jammer in the network.

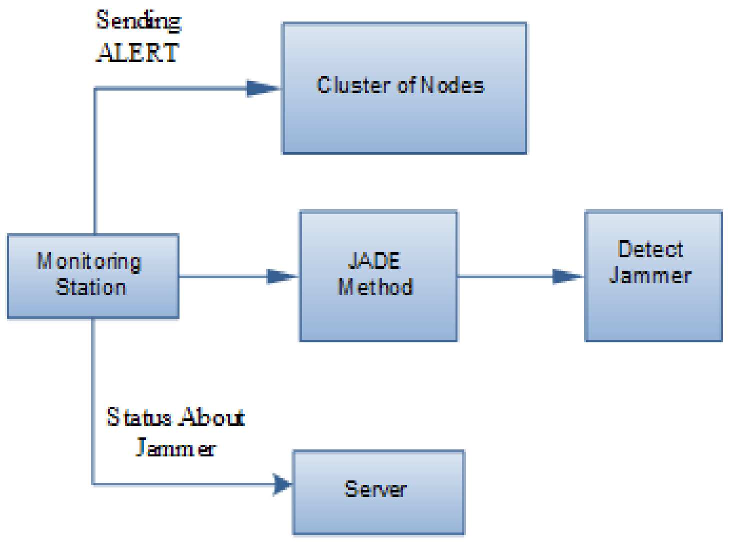

Fig.5. Detection of Jammer

E. SYSTEM DESIGN

The system design is given mainly for the identification of the system architecture accurately. In this the set of nodes sends information regarding the nodes from the monitoring stations to the server. The request for the clustering of nodes based on the location is given and the response is obtained. The GOOSE message is constructed which consist of a header and the message which is used to deliver the message with the delay of 3 ms to 10 ms. Nodes construct GOOSE message and send it to the server. The estimation of Message Invalidation Ratio is used to quantify the performance of networks and identify the type of jammers present in the network. It mainly identifies the location of the cluster in which the time delay is present and gives it to the server. With the help of jamming probability and number of needed samples the malicious jammer in the group can be detected using JADE technique.

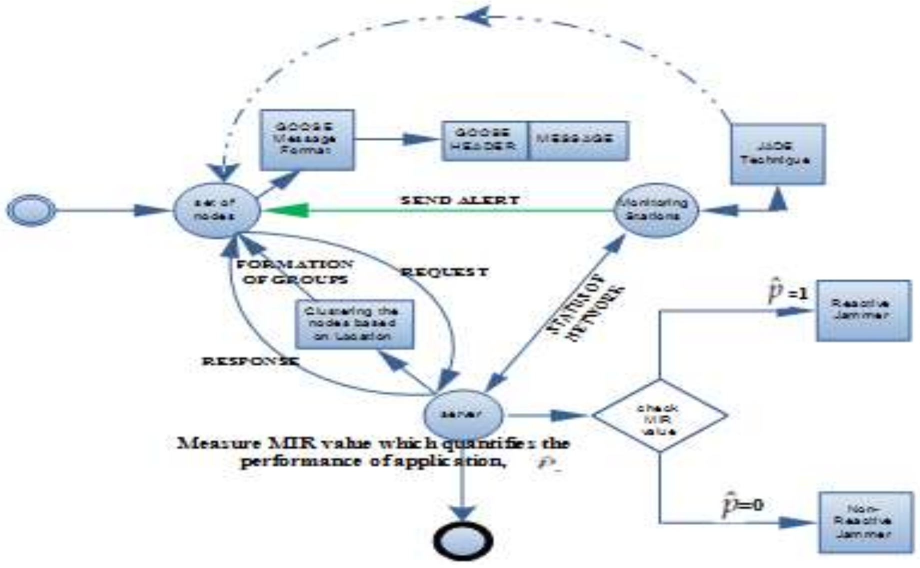

Fig.6. System Design

F. OVERALL ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. This is the simple method for the explanation of the overall proposed system. In this method the node registers its details such as the login details with the server so that untrusted nodes cannot enter the cluster and disrupt the communication. After registration the valid user can access the data and the MIR value is calculated. The jammer can be easily detected and the untrusted nodes can be avoided for further transmission.

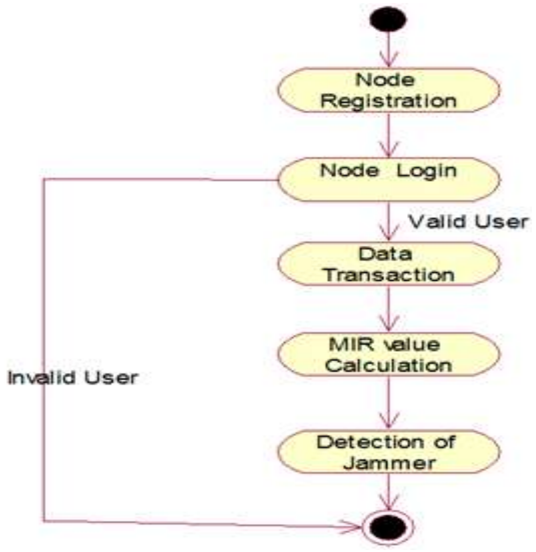

Fig.7. Overall Activity Diagram

IV. SIMULATION TOOL

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkely written in C++ and OTCL. NS is primarily useful for simulating local and wide area networks. Although NS is fairly easy to use once you get to know the simulator, it is quite difficult for a first time user, because there are few user-friendly manuals. Even though there is a lot of documentation written by the developers which has in depth explanation of the simulator, it is written with the depth of a skilled NS user.The purpose of this project is to give a new user some basic idea of how the simultor works, how to setup simulation networks, where to look for further information about network components in simulator codes, how to create new network components, etc., mainly by giving simple examples and brief explanations based on our experiences. Although all the usage of the simulator or possible network simulation setups may not be covered in this project, the project should help a new user to get started quickly.

NS is an event driven network simulator developed at UC Berkeley that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations. The NS project is now a part of the VINT project that develops tools for simulation results display, analysis and converters that convert network topologies generated by well-known generators to NS formats. Currently, NS (version 2) written in C++ and OTCL (TCL script language with Object-oriented extensions developed at MIT) is available.

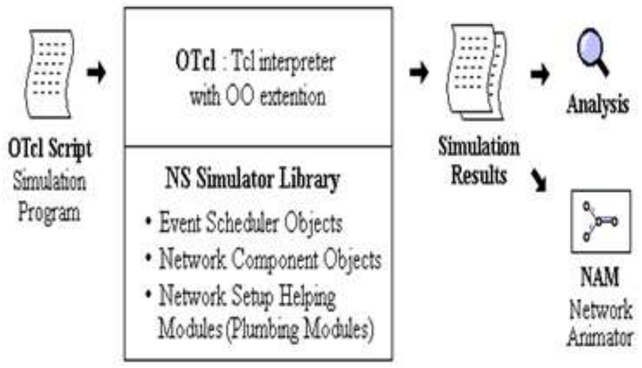
OTcl Script
Simulation Program
OTcl : Tcl interpreter with OO extention
NS Simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

Fig.8. Simplified View of NS

As shown in Figure in a simplified view, NS is Object-oriented TCL (OTCL) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object). In other words, to use NS, you program in OTCL script language. To setup and run a simulation network, a user should write an OTCL script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler. The term "plumbing" is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the "neighbor" pointer of an object to the address of an appropriate object. When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object. This may sound like complicated job, but the plumbing OTCL modules actually make the job very easy. The power of NS comes from this plumbing.

Another major component of NS beside network objects is the event scheduler. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event. Network components communicate with one another passing packets, however this does not consume actual simulation time. All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet. For example, a network switch component that simulates a switch with 20 microseconds of switching delay issues an event for a packet to be switched to the scheduler as an event 20 microsecond later. The scheduler after 20 microsecond dequeues the event and fires it to the switch component, which then passes the packet to an appropriate output link component. Another use of an event scheduler is timer. For example, TCP needs a timer to keep track of a packet transmission time out for retransmission (transmission of a packet with the same TCP packet number but different NS packet ID). Timers use event schedulers in a similar manner that delay does. The only difference is that timer measures a time value associated with a packet and does an appropriate action related to that packet after a certain time goes by, and does not simulate a delay.

NS is written not only in OTCL but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTCL interpreter through an OTCL linkage that creates a matching OTCL object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTCL object. In this way, the controls of the C++ objects are given to OTCL. It is also possible to add member functions and variables to a C++ linked OTCL object. The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTCL. Likewise, an object (not in the data path) can be entirely implemented in OTCL. Figure 6.2 shows an object hierarchy example in C++ and OTCL. One thing to note in the figure is that for C++ objects that have an OTCL linkage forming a hierarchy, there is a matching OTCL object hierarchy very similar to that of C++.

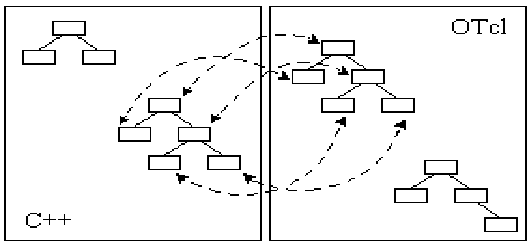

Fig.9. C++ and OTCL: The Duality

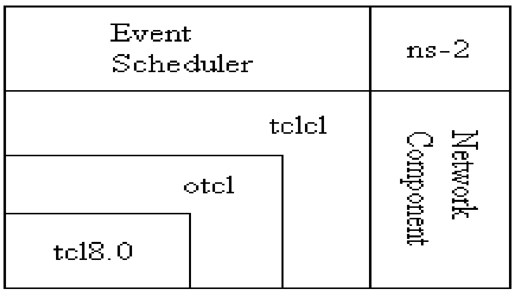

Fig.10. Architectural View of NS

Figure 10 shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in TCL using the simulator objects in the OTCL library. The event schedulers and most of the network components are implemented in C++ and available to OTCL through an OTCL linkage that is implemented using TCLcl. The whole thing together makes NS, which is a OO extended TCL interpreter with network simulator libraries.This section briefly examined the general structure and architecture of NS. At this point, one might be wondering about how to obtain NS simulation results. As shown in Figure, when a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input TCL (or more specifically, OTCL) script. The data can be used for simulation analysis (two simulation result analysis examples are presented in later sections) or as an input to a graphical simulation display tool called Network Animator (NAM). NAM has a nice graphical user interface similar to that of a CD player (play, fast forward, rewind, pause and so on), and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis.

Network simulator (NS) is an object–oriented, discrete event simulator for networking research. NS provides substantial support for simulation of TCP, routing and multicast protocols over wired and wireless networks. The simulator is a result of an ongoing effort of research and developed. Even though there is a considerable confidence in NS, it is not a polished product yet and bugs are being discovered and corrected continuously.

NS is written in C++, with an OTCL1 interpreter as a command and configuration interface. The C++ part, which is fast to run but slower to change, is used for detailed protocol implementation. The OTCL part, on the other hand, which runs much slower but can be changed very fast quickly, is used for simulation configuration. One of the advantages of this split-language program approach is that it allows for fast generation of large scenarios. To simply use the simulator, it is sufficient to know OTCL. On the other hand, one disadvantage is that modifying and extending the simulator requires programming and debugging in both languages.

A. NETWORK COMPONENTS

This section talks about the NS components, mostly compound network components. Figure shows a partial OTCL class hierarchy of NS, which will help understanding the basic network components. The root of the hierarchy is the TCL Object class that is the super class of all OTCL library objects (scheduler, network components, timers and the other objects including NAM related ones). As an ancestor class of TCL Object, Ns

Object class is the super class of all basic network component objects that handle packets, which may compose compound network objects such as nodes and links. The basic network components are further divided into two subclasses, Connector and Classifier, based on the number of the possible output DATA paths. The basic network and objects that have only one output DATA path are under the Connector class, and switching objects that have possible multiple output DATA paths are under the Classifier class.

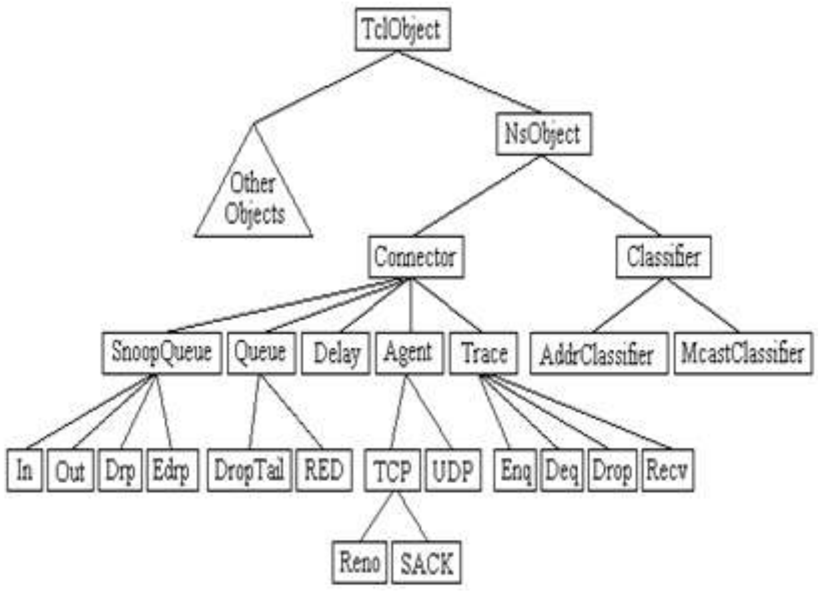

Fig.11. OTCL Class Hierarchy

B. CLASS TCL

The class TCL encapsulates the actual instance of the OTCL interpreter and provides the methods to access and communicate with that interpreter, code. The class provides methods for the following operations

1. obtain a reference to the Tel instance
2. invoke OTCL procedures through the interpreter
3. retrieve, or pass back results to the interpreter
4. report error situations and exit in an uniform manner
5. store and lookup "TCLObjects"
6. acquire direct access to the interpreter.

C. MOBILE NETWORKING

The wireless model essentially consists of the Mobile Node at the core with additional supportingfeatures that allows simulations of multi-hop ad-hoc networks, wireless LANs etc. The Mobile Node object is a split object. The C++ class Mobile Node is derived from parent class Node. A Mobile Node thus is the basic Node object with added functionalities of a wireless and mobile node like ability to move within a given topology, ability to receive and transmit signals to and from a wireless channel etc. A major difference between them is that a mobile Node is not connected by means of Links to other nodes or mobile nodes.Mobile Node is the basic nsNode object with added functionalities like movement, ability to transmit and receive on a channel that allows it to be used to create mobile, wireless simulation environments. The class Mobile Node is derived from the base class Node. The four ad-hoc routing protocols that are currently supported are, Dynamic Source Routing (DSR), Temporally ordered Routing Algorithm (TORA) and Adhoc On-demand Distance Vector (AODV).

D. TRACE ANALYSIS

This section shows a trace analysis. Running the TCL script generates a NAM trace file that is going to be used as an input to NAM and a trace file called "out.tr" that will be used for our simulation analysis. Figure shows the trace format and example trace DATA from "out.tr". Where each line in trace file represents an event associated to a packet.

E. NETWORK ANIMATOR (NAM)

NAM-Tool for viewing network simulation traces and real world packet teaches. It supports topology layout, packet level animation and various DATA inspection tools. Before starting to use NAM, trace file need to be created. This trace file is usually generated by NS. It contains topology information, e.g. nodes and links, as well as packet traces .during a simulation, the user can produce topology configurations, layout information and packet traces using tracing events in NS.Once the trace file is generated, NAM can be used to animate it. Upon startup, NAM will read the trace file, create topology, pop up a window, do layout if necessary and then pause at time 0. Through its user interface, NAM provides control over many aspects of animation. In Figure a screenshot of a NAM window is shown, where the most important function are explained. Although the NAM software contains bugs, as do the NS software, it works fine most of the times and times and causes only little trouble. NAM is an excellent first step to check.

V. RESULTS AND DISCUSSIONS

NS2 is one of the most popular open source network simulators. We conduct the following experiments with NS2.34 simulator

A network is taken into consideration in which 24 nodes are randomly deployed. The time taken for simulation is 30 ms.

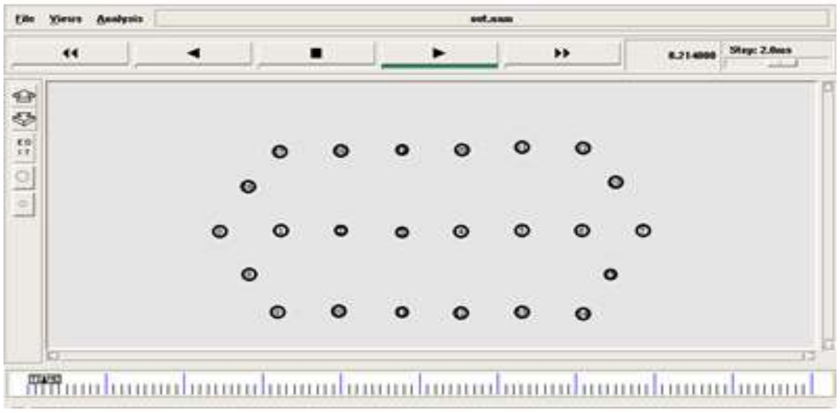

Fig.12. Nodes Deployed

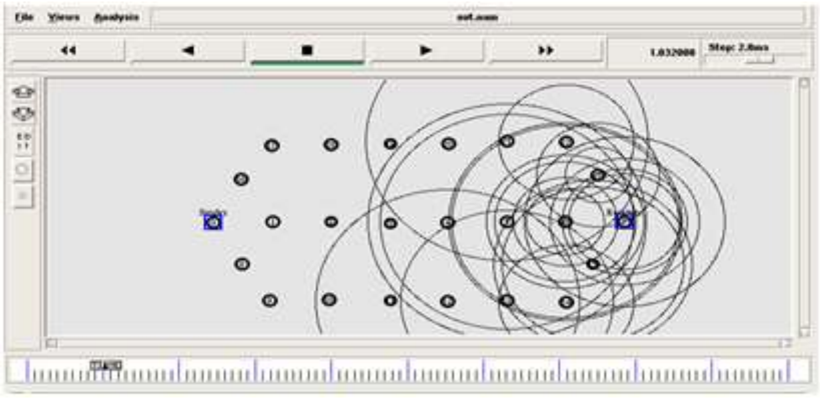

Fig.13. Coverage Area of Nodes

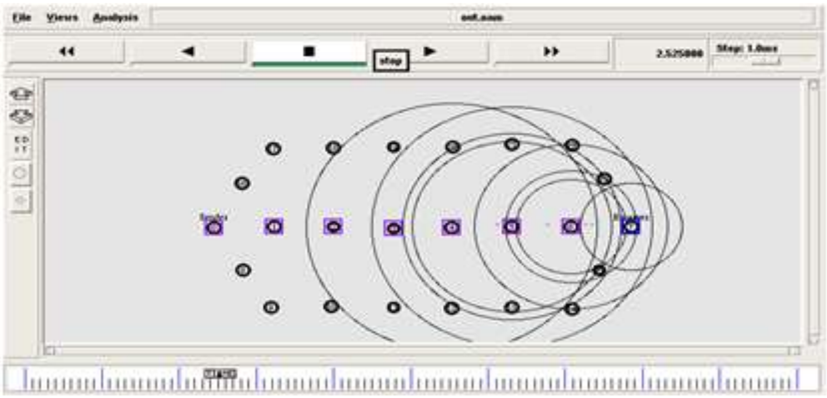

Fig.14. Source and Destination Nodes

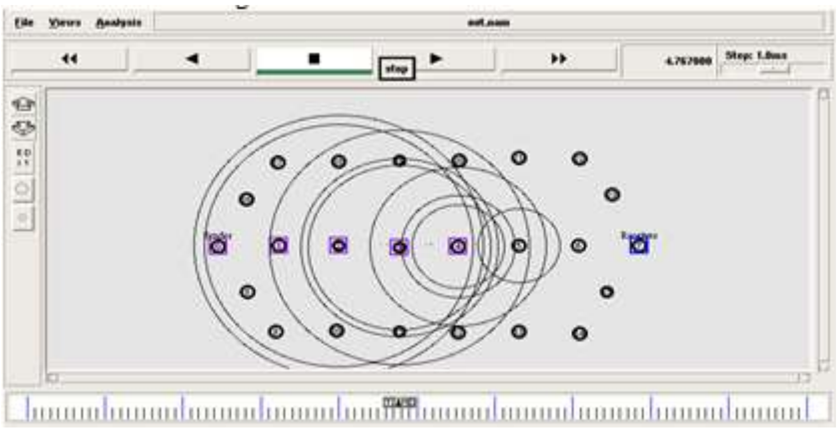
Fig.15. Identification of Route

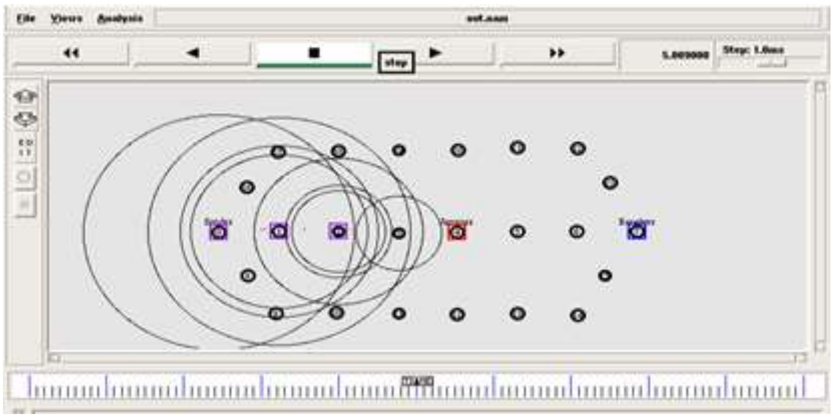
Fig.16. Location of Jammer Node

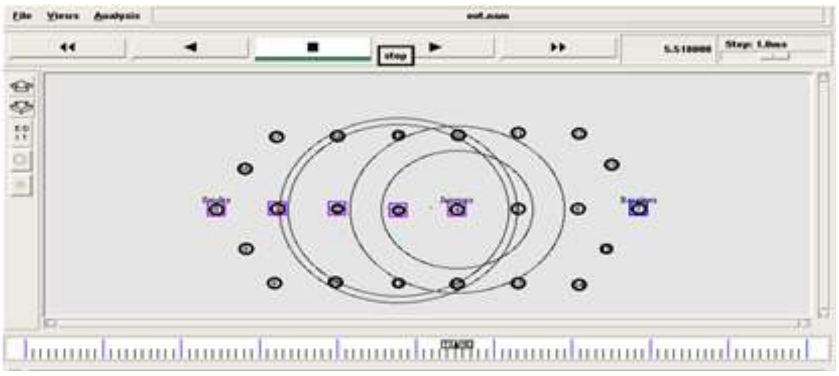
Fig.17. Transmission Blocked due to Jammer node

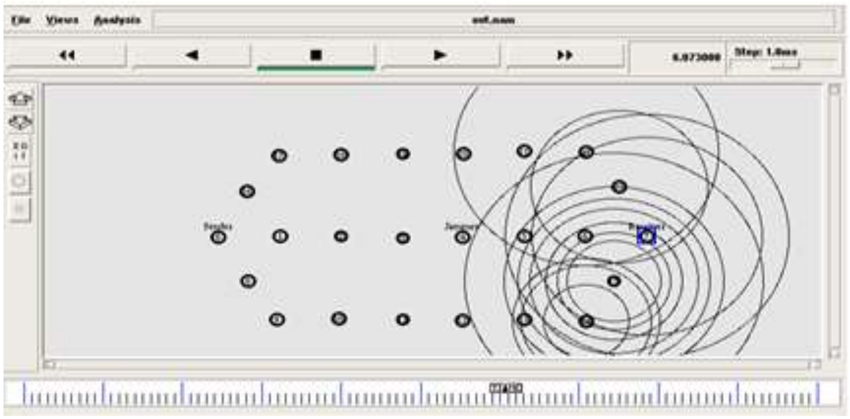
Fig.18. Identification of Alternate path

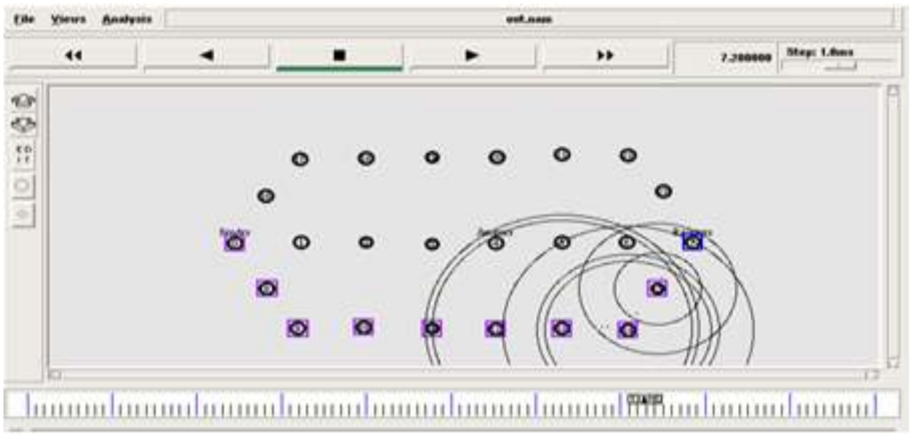

Fig.19. New Path found for Secure Transmission

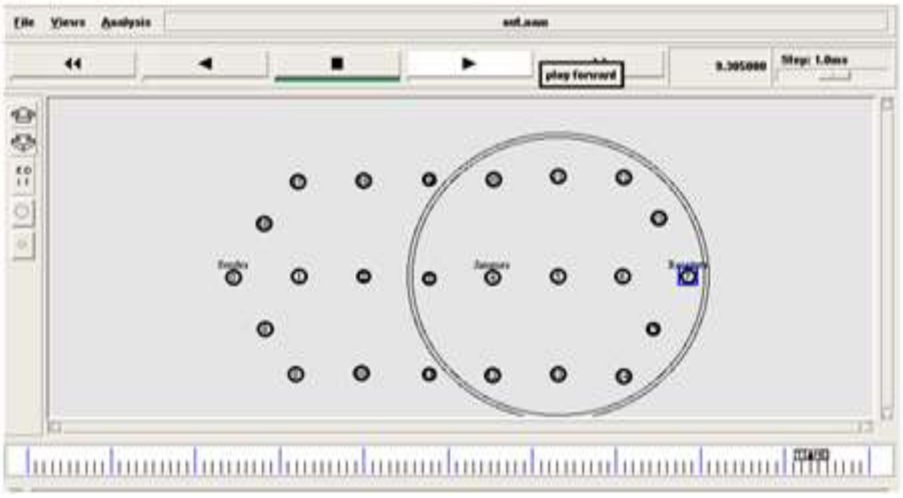

Fig.20. Scanning for Trusted nodes

GRAPH

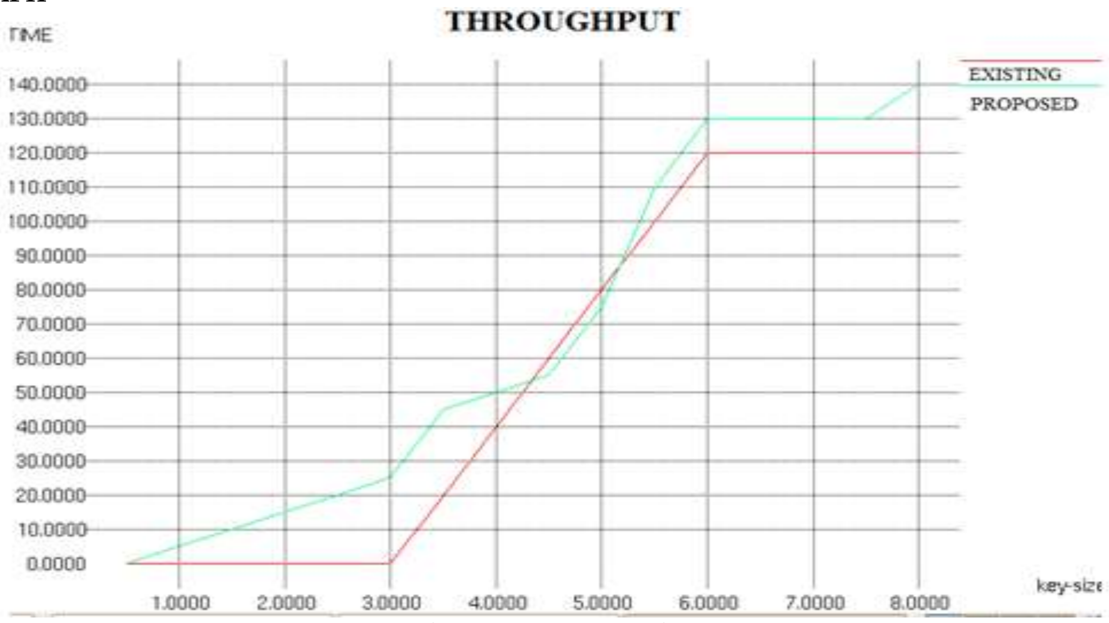

Fig.21. Performances Analysis of Throughput

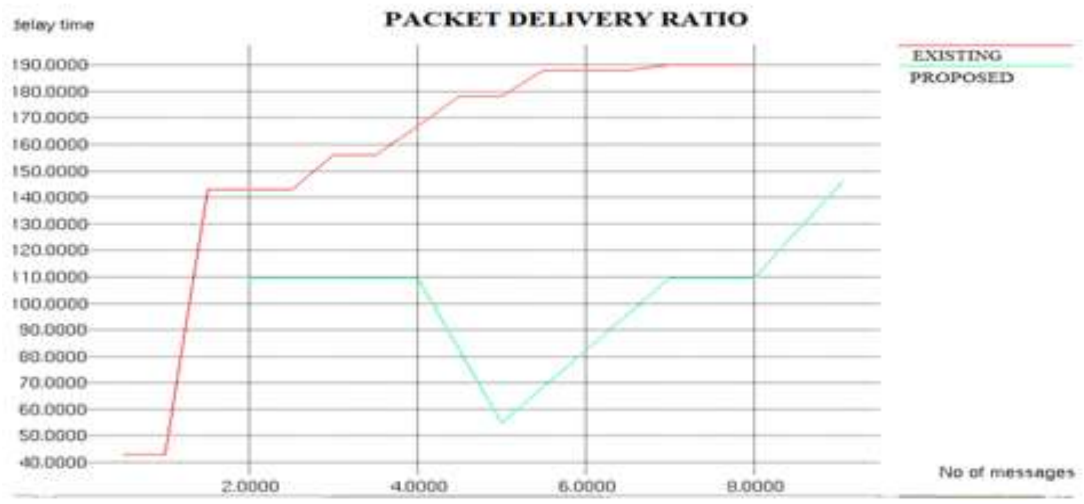

Fig.22. Packet Delivery Ratio

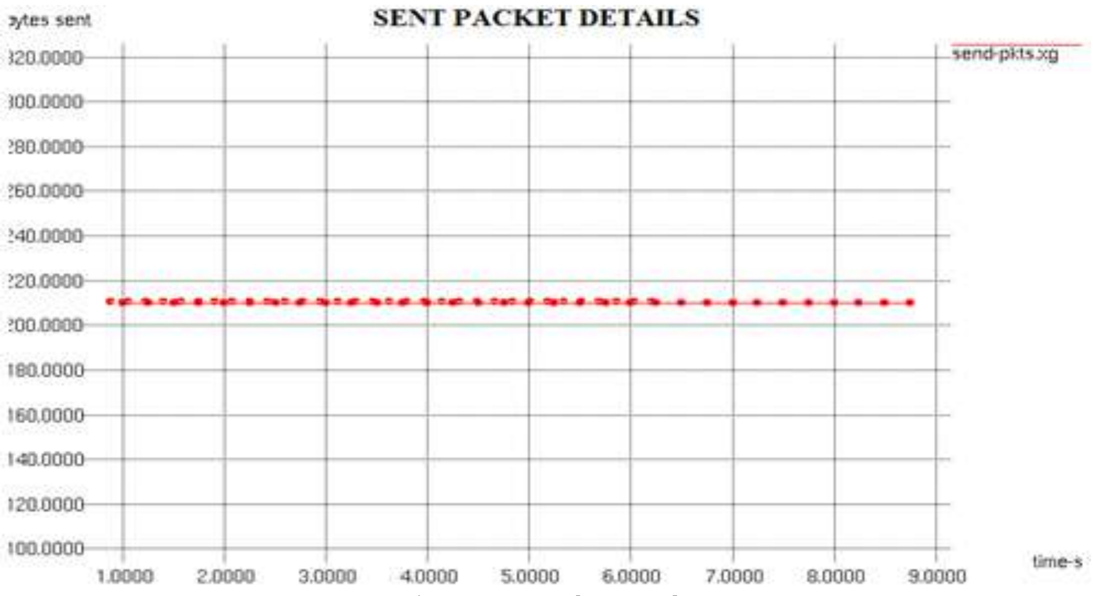

Fig.23. Sent Packet Details

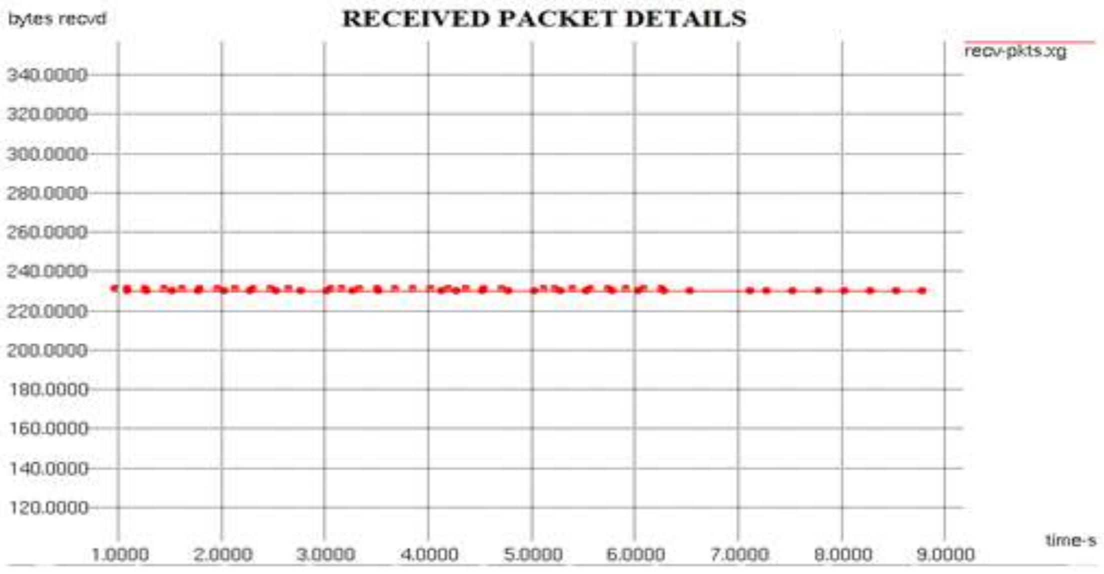

Fig.24. Received Packet Details

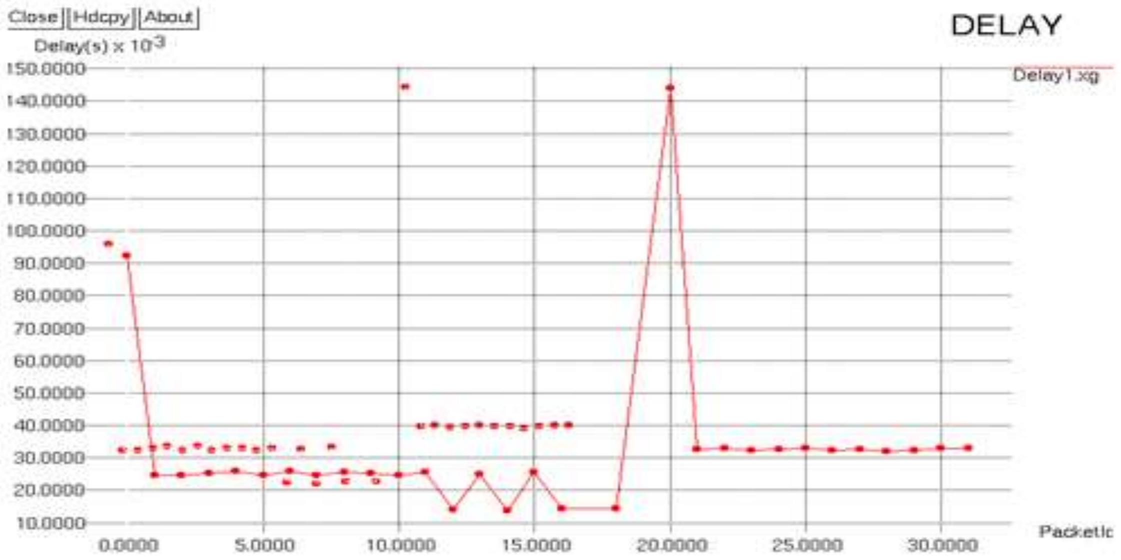

Fig.25. Delay

VI. CONCLUSION

In this paper the techniques are used to avoid the data loss and also to protect it till it reaches its destination securely. We implemented the JADE system at the application layer to achieve efficient and robust jamming detection for power networks such as power grids. The proposed JADE is a scheme for jamming-resistant broadcast communications in the presence of inside jammers. In JADE, the value of Message Invalidation Ratio probability and the number of needed samples are required to detect the location of jammer. The message invalidation ratio is used to quantify the performance of time-critical wireless applications. Therefore the alternate path is chosen for transmission when a jammer is identified in a particular path so that the packets are transmitted and received safely and securely when compared to conventional performance metrices.

To ensure the successful deployment of pervasive wireless networks, it is crucial to localize jammers. The future work is mainly focus on interesting issues arises on multi-channel networks, channel switching while the jammer has high energy costs when jamming more channels. Another issue is to finding alternatives for modeling lack of knowledge for attacker and the network. The issue of multiple potentially cooperating attackers is worth the future attention.

ACKNOWLEDGEMENT

We thank SRM Valliammai Engineering College for providing research resource to complete the task successfully.

REFERENCES

[1]. Abdel Rahman. M, Rahbari. H and Krunz. M (2012), “Adaptive frequency hopping algorithms for multicast rendezvous in DSA networks,” in *Proc. IEEE DYSPAN Symp., pp. 517–528.*

[2]. Adamy. D (2001), *EW 101: A First Course in Electronic Warfare*. Norwood, MA, USA: Artech House.

[3]. Baird. L. C, Bahn. W. L, Collins. M. D, Carlisle. M. C and Butler. S. C (2007), “Keyless jam resistance,” in *Proc. IEEE Workshop Inf. Assurance United States Military Acad*.

[4]. Bahl. P, Chandra. R and Dunagan. J (2004), “SSCH: slotted seeded channel hopping for capacity improvement in IEEE 802.11 ad-hoc wireless networks,” in *Proc. MOBICOM Conf*., pp. 216–230.

[5]. Bayraktaroglu. E, King. C, Liu. X, Noubir. G, Rajaraman. R and Thapa. B (Jun. 2013), “Performance of IEEE 802.11 under jamming ” *IEEE INFOCOM Proceedings.*

[6]. Bian. K, Park. J and Chen. R (2009), “A quorum-based framework for establishing control channels in dynamic spectrum access networks,” in *Proc. MOBICOM Conf.*, pp. 25–36.

[7]. Chan. A, Liu. X, Noubir. G and Thapa. B (2007), “Broadcast control channel jamming: Resilience and identification of traitors,” in *Proc. ISIT Conf.*, pp. 2496–2500.

[8]. Chaporkar. P, Kar. K, Luo. X and Sarkar. S (Feb. 2008), “Throughput and fairness guarantees through maximal scheduling in wireless networks,” *IEEE Trans. Inf. Theory*, vol. 54, no. 2, pp. 572–594.

[9]. Chiang. J. T and Hu. Y. C (2008), “Dynamic jamming mitigation for wireless broadcast networks,” in

Proc. INFOCOM Conf., pp. 1211–1219.

[10]. Liu. S, Lazos. L and Krunz. M (2011), "Thwarting inside jamming attacks on wireless broadcast communications," in *Proc. 4th ACM WiSec Conf.*, pp. 29–40.

[11]. Liu. S, Lazos. L and Krunz. M (2011), "On the Robustness of IEEE 802.11 Rate Adaptation Algorithms against smart jamming", in *Proc. 4th ACM WiSec Conf.*, pp. 29–40.

[12]. Mingyan Li, Iordanis Koutsopoulos, Radha Poovendran (Jun. 2013), "Optimal jamming attacks and Network defense policies in wireless sensor networks," in *Proc. IEEE Workshop Inf. Assurance United States Military Acad.*

[13]. Nadeem Sufyan, Nazar Abbass Saquib, Muhammad Zia (2013), "Detection of Jamming Attacks in 802.11b Wireless Networks", in *EURASIP Journal on Wireless communication and Networking.*

[14]. Sisi Liu, Loukas Lazos and Marwan Krunz (May/Jun. 2015) "Time-Delayed Broadcasting for Defeating Inside Jammers," in *Proc. IEEE Transactions On Dependable And Secure Computing*, Vol. 12, No. 3.

[15]. Wei Chen, Yingzhou Zhang, Yuanchun Wei (2013), "The feasibility of launching and detecting jamming attacks in wireless networks" in *Proc. IEEE DYSPAN Symp., pp. 517–528.*

[16]. Zhang. Y, Yu. G, Li. Q, Wang. H, Zhu. H and Wang. B (Jun. 2014), "Channel hopping-based communication rendezvous in cognitive radio networks," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 889–902.