JAUES

Journal of Al Azhar University Engineering Sector

Vol. 13, No. 48, July 2018, 1042-1056

ANMS: DESIGN AND EVALUATION OF A MANET SIMULATOR FOR DIVERSIFIED ATTACKS

Hatem Salama[1], Gouda Salama[1], Khaled Badran[1] and Mohamed Zaki[2]
[1]Military Technical College, Cairo, Egypt
[2]Faculty of Engineer Al-Azhar Universty, Cairo, Egypt

ABSTRACT
In this paper as-Application Needs MANET Simulator (ANMS) is presented as a novel unique simulator that can simulate equally the two attack types. The underlying simulator consists of two levels, one for each attack type. ANMS is two-fold to cover MANET diversified attacks and it has been built up using a Unified Modeling Language (UML). Its construction starts by emphasizing the use cases [i.e. the sequential relation between the server data and the clients broadcast]. Consequently, the class diagram is built up. Each class contains its public, private member and methods, while the relations between classes express their message handling. Eventually ANMS is coded and developed. ANMS works by feeding it by MANET-under-consideration. Then the first part that embeds the Byzantine oriented consensus simulation shows whether there is a Byzantine attack. If no Byzantine attacks it announces success otherwise it announces suspected. The suspected cases are categorized to false failure, true failure or attack/malicious. The last cases are examined using the second part of ANMS (typical intrusions part) to classify the underlying attack. ANMS is subject to tremendous amount of tests. Those tests include performance evaluation, comparisons and confusion matrix. The experimental performance of ANMS confirms the fact that, it is really needed for MANETs administration and their security measurements.

Keywords: **Byzantine Attack, Consensus Algorithm, MANET Attacks, Use Cases, Class Diagram, Decision Tree**

1. INTRODUCTION

In MANET's [1-2] laptops, PCs, cellular phones, appliances with ad-hoc communication capability link together on the fly to create a network [3]. This technology is the key to solving today's most common communication problems such as having a fixed infrastructure, and centralized, organized connectivity, etc. MANET is a self-configuring network of mobile routers and associated hosts connected by wireless links. The routers (mobile devices, nodes) are free to move randomly and organize themselves arbitrarily. Thus, the network's wireless topology may change rapidly and unpredictably. In the network data moves from hop to hop till it reaches its destination. In addition the network updates and reconfigures itself to keep nodes connected. The network topology changes when a node joins in or moves out. Packet forwarding, routing, and other network operations are carried out by the individual nodes themselves [3]. Moreover, in MANETs, where each node is acting as a router with dynamically changing topology, the availability is not always guaranteed [4]. It is also not guaranteed that the path between two nodes would be free of malicious nodes (intruder nodes) [3]. The wireless links between nodes are highly susceptible to link attacks (passive eavesdropping, active interfering, etc) [5].

A mobile ad-hoc network by its nature is subject to many diversified attacks. Those attacks can be broadly classified into two categories: 1) Network intrusions and 2) Byzantine oriented attacks [7-8-9]. The first category contains typical MANET attacks [10] such as DoS, Drop attack and worm-hole. Byzantine oriented attacks mean that the compromised node can generate arbitrary data to pretend its real behavior. It is clear that each type of these attacks has its own inherent features. Accordingly, every type needs a corresponding simulator. Thus, there are no simulators that can simulate both types (typical intrusions and Byzantine attacks). Therefore, MANET [11] administrators are in real need to simulate both types. From this point ANMS gains its significance as a simulator that integrates in homogeneous and seamless manner two levels to model either Byzantine or non-Byzantine (network intrusion) attacks.

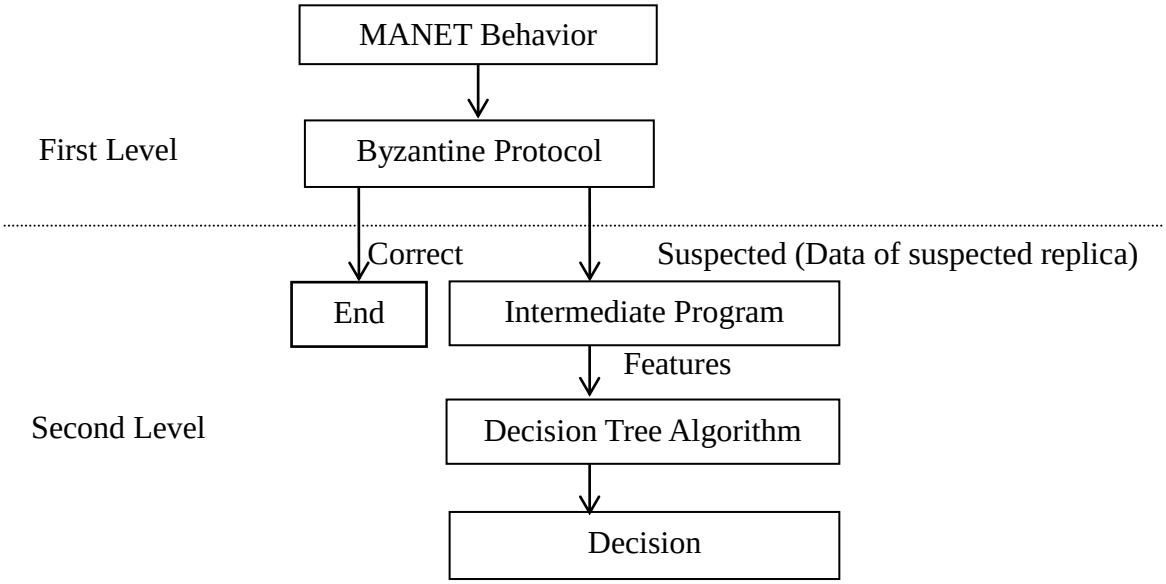

Figure (1): The simulator two levels

uction? Must be replaced

ANMS adopts a novel approach for dealing with the tw network is presented to simulator that consists of two levels, as shown in Figure (1). Throughout its first level it can decide whether or not there exist any malicious conditions. If there is not (success) the administrator will guarantee that his network is normally operating. Otherwise, ANMS announces suspicion. Such suspicion implies false failure, true failure and attack/malicious.

The second level of ANMS is a classifier that classifies the network conditions as false failure, true failure or attack/malicious. In this paper, the main concern is to investigate the design approach that has been employed using UML. The design process begins by starting the network use cases. Accordingly the UML class diagram is laid out, from which ANMS code is developed and heavily tested. Such tests confirm:

i. Crossing the gap that exists between typical simulators (eg. NS3, Glomosim, Omnet++) [12] and MANET Byzantine consensus (eg. Tangaroa [13], Turquois [14], BFT-CUP [15] , GCAP [16]).
ii. The unique capabilities of the simulator are handling both 1) Typical intrusion and 2) Byzantine oriented attack.
iii. The superiority of ANMS performance.

This paper is organized as follows. Section 2 discusses the related work while section 3 is an elaboration of the design methodology that starts with use cases and consequently the class diagram is given. Section 4 emphasize ANMS performance and investigates its capabilities. Section 5 is the paper conclusion.

2. RELATED WORK

There are many different network simulators available [17], it is extremely difficult to choose an appropriate tool for performance testing without the complete analysis of existing tools

Investigating MANETs is achievable by resorting either to software-based simulators or to experimentation networks (test-beds). Most researchers favor simulators as the expense of test-beds. What prevent (or at least hinder) the use of real-size test-beds are their cost and their inherent lack of flexibility [12].

This becomes particularly impeding as the size of the experimented network grows. Software-based simulation then turns out to be a viable alternative and a widely used solution

Test-beds suffer from several drawbacks. More precisely, the cost of the hardware (one node is several hundred euros) coupled with the difficulty of managing applications in terms of deployment, monitoring, etc. over such test-beds makes that only a few test-beds could be built up to now.

Because of the complex nature of the MANETs, their simulation is a very challenging issue. Simulators rely on various techniques for improving their accuracy, speed, scalability, usability, etc. Examples of these simulators are:

i. NS-2: It is an open source simulator for wired, wireless, Ad-Hoc and sensor networks.

ii. OMNET++: It is an open source program that simulates discrete events for wired, wireless, Ad-Hoc and sensor networks.

iii. Glomosim: It is an open source and simulates wired, wireless, Ad-Hoc and sensor networks.

For these simulators it is difficult to model the malicious conditions of different attacks, since such attacks are executed throughout unknown procedures.

Geetha et al [8] have proposed security measures to mitigate MANET Byzantine attacks [18-19-20]. In their paper, they claimed that they can provide tools against both Byzantine and non-Byzantine attacks. However they relied on an oversimplified approach in which they considered that the node which can send arbitrary messages to other network nodes is Byzantine. Thus, they ignored entirely the Byzantine consensus [21] which guarantees the safety of the network servers.

3. DESIGN AND DEVELOPMENT OF ANMS

Here, UML is used as a de facto design approach in software engineering, therefore we begin with elaborating ANMS use cases.

3.1 Use cases

The use cases provide a clear relation between ANMS and the corresponding actors. Moreover, the sequence diagrams of different attacks are demonstrated. Those diagrams are represented from the victim view point while the attacker view point, that has been discussed in [22], is ignored. Such use cases are classified to Byzantine oriented and non-Byzantine cases (attacks), however, ANMS starts by accomplishing leader selection.

3.1.1 Leader Selection

The MANET nodes are moving randomly at random distances that are calculated at each term (time interval). If the distance between any two nodes is increased over a particular maximum allowed distance then no connection can be established even the two nodes are not faulty. Thus, in MANETs, by their nature, every node will be connected by a connection or more to other nodes. The system counts and records these connections for each node. On the basis of the node mobility network groups (i.e. two nodes or more) may be formed. For a not faulty node in a group with maximum number of connections it might be selected as a leader for this group. In Figure (2) clarifies the groups and the leaders, as in the figure we have two groups G1 and G2, node 1 is the leader of group G1 and node 9 is the leader of G2. Leaders nodes are 1, 9. Such

leader(s) is assumed to be connected to the server cluster, and communicated with it to obtain the current data.

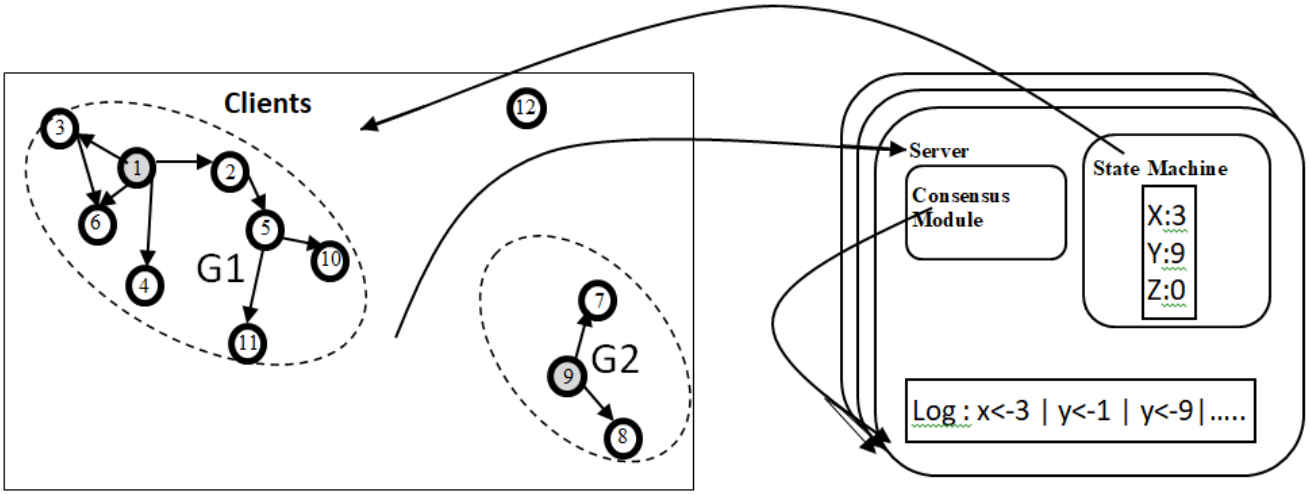

Figure (2) MANET clients and servers in two

3.1.2 Byzantine oriented Attack

A Byzantine attack means that a compromised node may act as a leader that can receive the current data from the cluster server and send arbitrary messages to other nodes. For clarification the following illustrative cases are considered:

1- Normal case i.e. neither failure nor attack.
2- True failure because of node inability to send
3- True failure because of node inability to receive
4- True failure because of node inability to receive but the following nodes are aware with the current data
5- True failure because of node inability to send but the following nodes are aware with the current data

In the first case we assume all nodes are up and running and node1 is acting as a leader. It can receive the current data from the cluster server. That node has four connections (to deserve leadership) and it sends the new messages to all its connected nodes as shown in Figure (3a). Consequently, its connected nodes are aware and can send the new messages to their descendent nodes.

In the second case we have node 1 is a leader with high count of 3 connections and node 2 received the new messages to send them to node 3 which has a problem in sending. Consequently the last node cannot send the new messages to node 4. Then, node 4 has the old messages which means that is not aware with the up to date messages and this corrupts the data consistency in the descendent nodes as shown in Figure (3b).

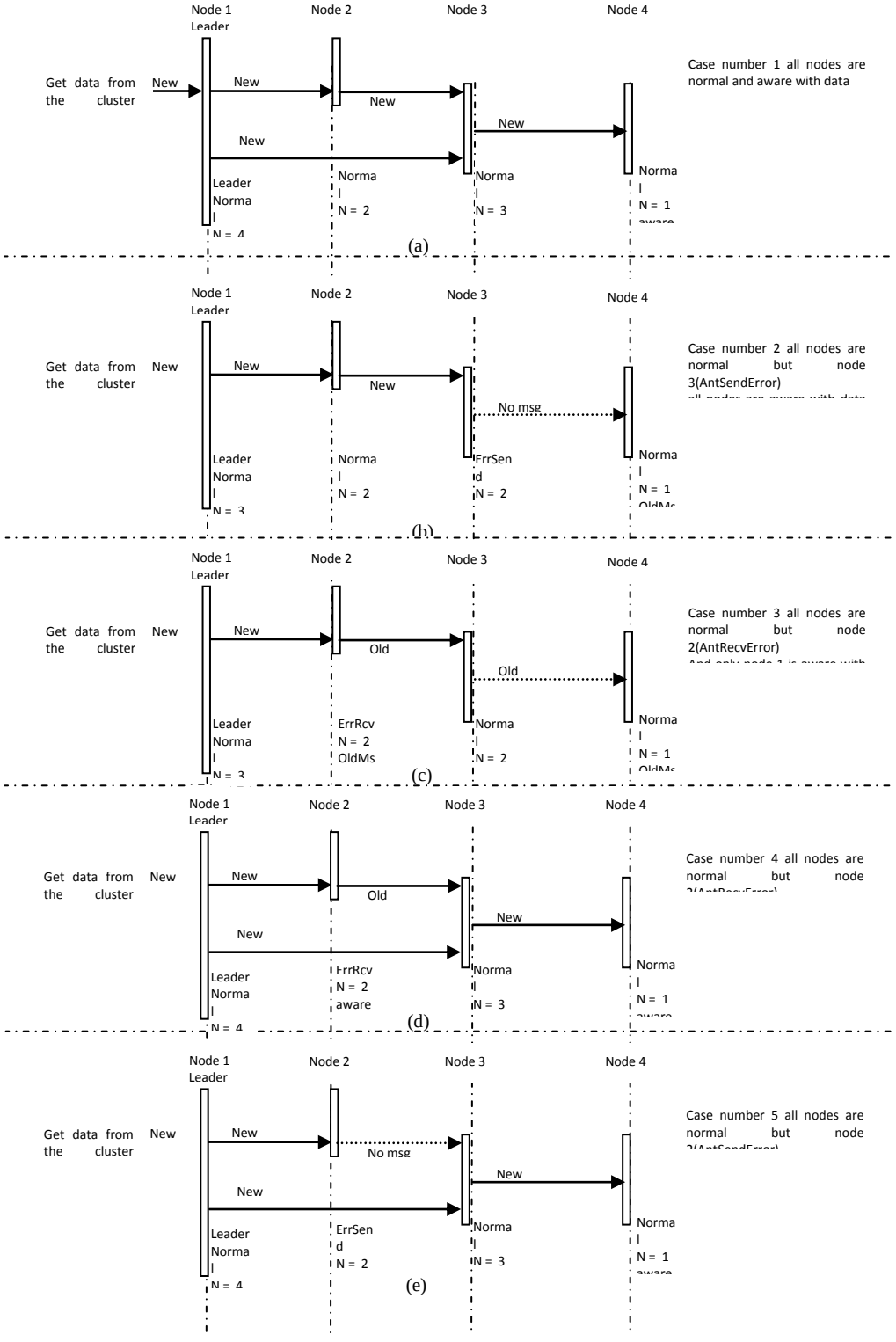

Figure (3): Sequence diagram of Byzantine cases
a Normal case i.e. neither failure nor attack
b True failure because of node inability to send
c True failure because of node inability to receive
d True failure because of node inability to receive but the following nodes are aware with the current data
e True failure because of node inability to send but the following nodes are aware with the current data

In the third case we have node 1 is a leader with high count of 3 connections and node 2 cannot receive the new messages because it has a problem in receiving, so it sends the old messages. Consequently, nodes 3 and 4 have the old messages which mean that those nodes are not aware with the up to date messages as shown in Figure (3c).

In the fourth case node 1 is the leader with high count of 4 connections and node 2 cannot receive the new messages because it has a problem in receiving, so node 2 sends the old messages to node 3, but node 1 has a connection with node 3 and sends the new messages to it. Consequently node 3 becomes aware and sends the new messages to the descendent nodes as shown in Figure (3d).

In the fifth case node 1 is the leader with high count of 4 connections and node 2 can receive the new messages but it cannot send them to node 3 because it has a problem in sending, but node 1 has a connection with node 3 and sends the new messages to it. Consequently node 3 becomes aware and sends the new messages to the descendent nodes as shown in Figure (3e).

3.1.3 Intrusion Attack

A major strength for ANMS is its capability to simulate typical intrusion attacks, also. In what follows we explain 4 attack cases, as shown in Figure (4).

For clarification the following illustrative cases are considered:

1- DoS attack.
2- Overflow sending causes Drop attack
3- Node under Noise attack
4- Jamming attack to all the nodes

In the first case, node 1 is the leader. It sends the new message to its connected nodes (e.g. node 2). However, node 2 is so aggressive that it sends over-flow messages to his descendent nodes (e.g. node 3). Thus, node 3 becomes aware but cannot pass the message to the next level nodes because of the DoS flooding, as shown in Figure (4a). Consequently, nodes 4 and 5 are not aware and they are carrying only the old messages.

In the second case node 2 also has the over-sending problem, but it continues to send to itself, which affects the process of passing new messages. Then, node 3 suffers from a drop attack as shown in as shown in Figure (4b). Consequently, nodes 3,4 and 5 are not aware with new messages.

In the third case node 3 is under noise attack, so the node cannot receive or send any messages, which means that it looks like a hole. Consequently, nodes 4 and 5 are not aware with new messages as shown in Figure (4c).

In the fourth case all nodes are under noise attack (all nodes cannot send or receive), this case called a Jamming attack on all the nodes. Really there is no leader as shown in Figure (4d).

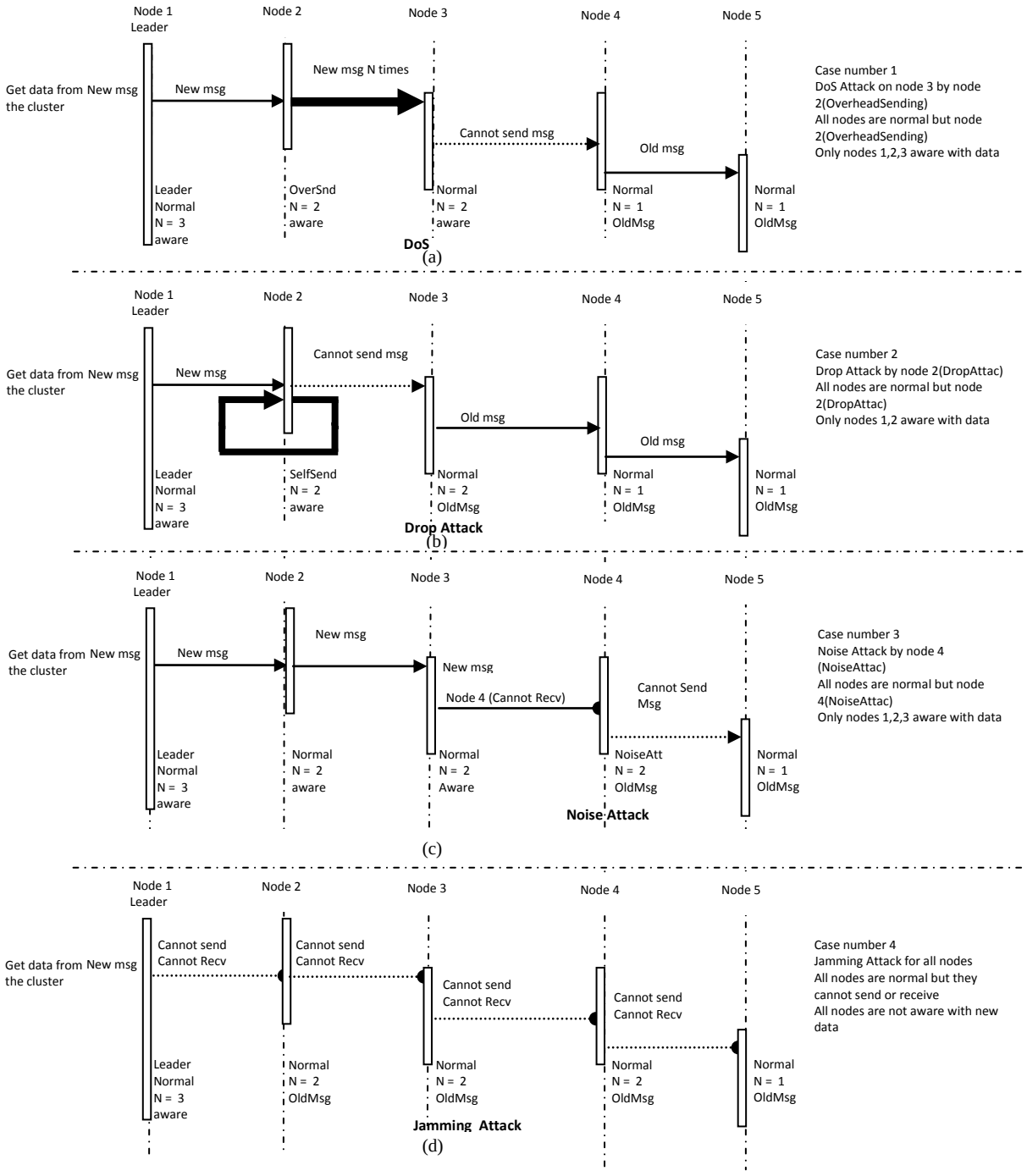

Figure (4): Sequence diagram of Intrusion cases
a DoS attack.
b Overflow sending causes Drop attack
c Node under Noise attack
d Jamming attack to all the nodes

3.2 Class Diagram

Here, the class diagram provides a static view for ANMS, where each class consists of: 1) Class name, 2) Attributes and 3) Methods. The attributes and methods may be either public or private. The classes, as such, are based on the use cases of section 3.1.

In Figure (5) each node has its location that determines the possible connections with other nodes. Location also may be in obstacle area which causes the ban of connection with the other nodes outside this obstacle area. Normal nodes are the nodes which have no malfunctioned conditions, not in obstacle area, not under attack and not compromised, otherwise the node is abnormal. As shown in Figure (3), the root is the class node and its private attributes (written in Latin) are: *XPos*, *YPos*, *Isleader*, *canSend* = true, *canReceive* = true and it contains one method only; *createNode*(). The class location is associated with the class node. Its public methods are *getX*() and *getY*() while it has no attributes. Such node class has two children, namely, normal node and abnormal node. Here, we are concerned only with abnormal nodes that are allocated on two levels: Typical intrusion and Byzantine oriented, as shown in Figure (5). Noise class is a subclass of node with private attributes: *isNoisy* = true, *canSend* = false and *canReceive* = false. However, for that class we don't care about the values of other node attributes. Jamming class is a subclass of node with private attributes: *isJamming* = true, *isNoisy* = true, *canSend* = false and *canReceive* = false. When Jamming is applied all ANMS nodes are faulty. Overflow class is a subclass of node with private attributes: *isOverFlow* = true, *canSend* = true and *canReceive* = true, that class send the message N times which affect the receiving nodes under type of DoS because it cannot send any message while DoS class exhausts his connecting bandwidth in replying to its sender then it cannot send the message to its receiving class.

The Byzantine oriented classes are Obstacle Area, Antenna Error, Zero Connections. The Antenna error only is considered as example (case) in use cases. The obstacle area class contains inObstacleArea = true. Thus, that class can only communicate with nodes that exist in the obstacle area. The Zero connection class contains connectionCount = 0, which means that the underlying class is either in the obstacle area lonely or connectionless.

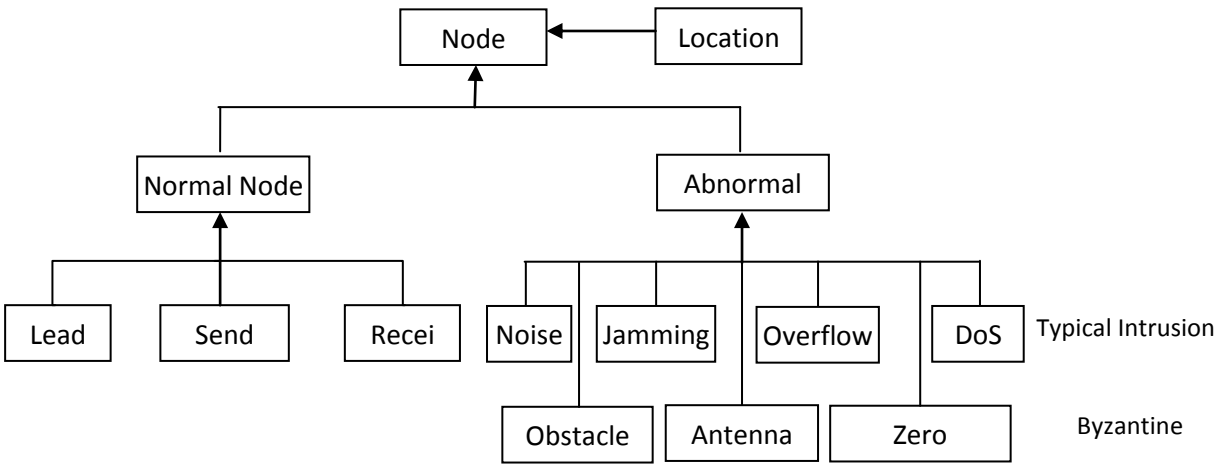

Figure (5): Class diagram of ANMS

3.3 ANMS Functionality

ANMS starts from its main in which the maneuvering area, number of nodes, and the parameters of the obstacle area and the number of terms are initially determined. The system works iteratively in loops. In each loop the program runs the relevant methods the first method to be executed is *setLocation*() which is responsible for

changing all nodes locations randomly assuming that all MANET nodes should be in the underlying area. The nodes locations are changed every term as long as such nodes are not stopped. Accordingly, the method of *setNodesDistances*() compute the distances between every pair of nodes. As long as the distance is less than a predefined threshold the connection between and two nodes can be determined using the method *setNodesConnections*(). When a node has no connections then, its attribute *isZeroConnection* = true. Class *setConditions*() contains the methods that choose nodes, condition and number of terms (life of this condition), all are reset randomly. The pseudo-code is emphasized in Appendix A. It contains five methods in addition to the *main*().

4. Implementation and Performance Evaluation

The implantation of ANMS is taken place experimentally and its performance is evaluated and compared with other similar products. In fact the performance evaluation is carried out on two levels, the first (high) one is concerned with the Byzantine consensus while the second (low) level is interested in defending typical intrusions, as shown in Figure (1) .

4.1 Performance of First Level

The first level utilizes and simulates the ideas of Raft [23] to solve the Byzantine consensus problem that may be raised between ANMS servers. Accordingly, a strong leader is employed. Thus, log entries only flow from the leader to other servers. This simplifies the management of the replicated log and makes ANMS easier to understand. Moreover Raft, consequently, ANMS uses randomized timers to elect leaders. This adds only a small amount of mechanism to the heartbeats already required for any consensus algorithm, to solve conflicts simply and rapidly. The main characteristics of that level are pointed out for 10000 Operational Cases (OC) in Table (1). In this table 5035 OCs suffer neither Byzantine disagreement nor malicious attack, while 1901 are suspected, false failure OCs but actually they are not attacks, according to the classification of the second level (section 4.2). Moreover, 2131 OCs are true failure but the second level indicates that they are not attacks. The rest of the table emphasizes typical client intrusions.

Table (1) Performance of ANMS two levels

Operational Cases (OC)	Count of OC	Ratio	Level 1 results	Level 2 results
			Is-Suspected	Is-Attack
Normal	5035	50.3%	No	No
False Failure	1901	19%	Yes	No
True Failure	2131	21.3%	Yes	No
Jamming	178	1.8%	Yes	Yes
Noise	23	0.2%	Yes	Yes
Drop	555	5.6%	Yes	Yes
DoS	177	1.8%	Yes	Yes

4.2 Performance of Second Level

The second level performs intrusion detection in MANETs using a decision tree classifier which is based on C4.5 algorithm. Upon prototyping ANMS that tree can be obtained and illustrated in appendix B. Also the corresponding classifier performance is pointed out, Table (2). In this table the true positive, TP-rate, false positive, FP-rate, precision, recall and F-measure are recorded. In addition the corresponding confusion matrix is illustrated in Table (3). These metrics are defined by the following equations.

Accuracy = (TP + TN) /(TP + FP + FN + TN) (1)

Precision = (TP)/(TP + FP) (2)

Recall =(TP) /(TP + FN) (3)

True Negative Rate = (TN) / (TN + FP) (4)

True positive rate TPR = TP / (TP + FN) (5)

True Negative rate TNR = TN /(TN +FP) (6)

F-Measure = 2TP /(2TP + FP + FN) (7)

Where,
TP : True detected attack
TN : True detected non attack
FP : Non attack detected as attack
FN : Attack detected as non-attack

The classifier has a structure of pruned binary tree that utilizes C4.5 algorithm [24]. This tree -structured classifier is attractive because of the fact that the most informative nodes are the nearest to the root.

Table (2) Decision tree parameters

Class	TP-Rate	FP-Rate	Precision	Recall	F-Measure
False Failure	**0.97**	**0.056**	**0.945**	**0.97**	**0.957**
True Failure	**0.912**	**0.009**	**0.983**	**0.912**	**0.946**
Jamming	**0.974**	**0**	**0.957**	**0.974**	**0.966**
Noise	**0.784**	**0**	**0.978**	**0.784**	**0.871**
Drop	**0.987**	**0.01**	**0.899**	**0.987**	**0.941**
DoS	**0.956**	**0.007**	**0.852**	**0.956**	**0.901**

Table (3) Confusion matrix of the decision tree

Class	False Failure	True Failure	Jamming	Noise	Drop	DoS
False Failure	**13930**	**151**	**0**	**0**	**135**	**140**
True Failure	**722**	**9582**	**10**	**2**	**137**	**57**
Jamming	**0**	**6**	**224**	**0**	**0**	**0**
Noise	**14**	**3**	**0**	**91**	**5**	**3**
Drop	**27**	**5**	**0**	**0**	**2456**	**0**
DoS	**51**	**2**	**0**	**0**	**0**	**1152**

4.3 Comparative Study

There is no one product that can be compared with the two levels of ANMS. Therefore, two products are chosen for comparison, where every single product is compared to its corresponding level. The first level is compared with Turquois [14]. Such comparison is pointed out in Table (4) for a MANET with n-nodes from which f nodes may fail.

Table (4) ANMS Comparative Study

Aspect of comparison	Turquois	ANMS
Theoretical foundation	Basics of Byzantine general problem	Raft based with extensions for MANETs
Scope of applicability	Immune for Byzantine attacks	Immune for general MANET attacks
Allowed failed nodes	Failure of f < n/3 and momentary break down communications	Failure of f < n/3 and momentary break down
Differentiation between accidental and malicious failure	Partial differentiation	Complete differentiation by making use of suspected cases
Intrusion tolerant	Yes	Yes
Energy saving	By efficient utilization of the broadcasting media	By handling the attacks/failures on two levels
Combination of Byzantine and dynamic Omission faults	Yes	Yes
Inexpensive hashing	For authentication instead of public key cryptography	Incremental hashing to ensure integrity

The second level, as shown in Figure (1), is compared with [25] that can investigate the process of intrusion detection in MANET using classification algorithm. Figures (6) shows the performance of ANMS C4.5 decision tree classifier relative to the Gaussian Mixture Model (GMM), the naïve Bayes model and the Support Vector Machine (SVM) model and the Multi-Layer Perceptron (MLP) algorithms [25].

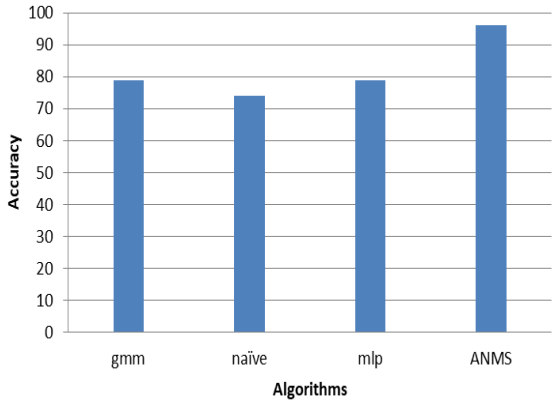

Figure (6): Accuracy comparison

5. CONCLUSION

This paper presents ANMS as a MANET-under-attack simulator. For such simulator the attacks are broadly categorized into two categories, namely, Byzantine attacks and typical MANET intrusions. Accordingly, ANMS has been built up on two levels. The first (high) level mitigates the Byzantine attacks. It makes use of solving the Byzantine consensus problem to guarantee that all the network servers are running in non-faulty conditions i.e. the unknown nodes are not malicious this level of the simulator has the advantage that it is Raft based and it takes into account the structure-less ad-hoc operational conditions. The second

(low) level is devoted to detect the typical MANET intrusions using a decision tree classifier which utilizes C4.5 algorithm to determine the intrusion class.

As a software product ANMS is designed using the de-facto UML approach. Consequently the use cases and their corresponding sequence diagram are investigated. Next, the class diagram is given and utilized to develop the ANMS code.

The fact that ANMS crosses the gap between Byzantine and non-Byzantine attack simulators has been confirmed throughout its prototyping. Therefore, ANMS can be used with the following essential advantages.

1 - It is based on a sound theoretical foundation for solving the Byzantine general problem. Consequently the success of the system servers ensures that they are free of any malicious attack.
2 - It reduces the number of faulty operations by passing them from the first level to the second one, thus decreasing the false alarms.
3 - It is suitable for source constrained MANETs as it saves energy by reducing the number of faults to be processed.
4 - It takes into consideration the unreliable communications between MANET nodes by providing messages repetition.

Appendix A
Pseudo Code of ANMS

ANMS code consists of a main() and five methods. In the main() we set conditions, locations, distance, connections and decisions. Also it increments the terms (time intervals).

```
Main(){
        Create networkNodes at  location(x,y)
        Number of terms = nTerms
        Number of nodes = nNodes
        Define the maneuvering area: A
        Define the obstacle area: Obs  A

        While terms <= nTerms {
                If Terms / Interval = 0,
                        setConditions of nodes

                setLocation of nodes
                setNodesDistances of nodes
                setNodesConnections of nodes
                setDecision of nodes
                Terms ++
        }
}
setConditions (nodes){          //      Method for setting random conditions
        For i =1 to nNodes{
                ii = rand (nNodes)
                If AntSendErr[ii] = 0 then AntSendErr[ii] = rand(xTerms)
                ii = rand (nNodes)
                If AntRcvErr[ii] = 0 then AntRcvErr [ii] = rand(xTerms)
                ii = rand (xNodes)
                If CanNotSendErr[ii] = 0 then CanNotSendErr[ii] = rand(xTerms)
                If CanNotRecvErr[ii] = 0 then CanNotRecvErr[ii] = rand(xTerms)
        }
}
setLocation(nodes){
        For i = 1 to nNodes{
                Nodes[i].x = Nodes[i].x + rand(displacement)*random(direction)
                Nodes[i].y = Nodes[i].y + rand(displacement)*random(direction)
                If Nodes[i] not in A then repeat till be in A
                Else If Nodes[i] Obs, nodes[i].obstacle = true
        }
}
setNodesDistances(nodes){
        For i = 1 to nNodes{
                For ii = 1 to nNodes{
                        nodesDistances[i,ii],     // Calculate distances between nodes
                }
```

```
        }
    }
    setNodesConnections (nodes[n]){
        For i = 1 to nNodes{
            For ii = 1 to nNodes{
                If nodesDistances[i,ii] < distanceThreshould, nodesConnection[i,ii] = true
                Else    nodesConnection[i,ii] = false
            }
        }
    }
    SetNodesDecision(nodes){
        --      Depending of the conditions decision will be one of the next (false failure, true failure, DoS, Drop, …
        If nodes[i].obstacle = true number of terms < termThreshold, false failure
        Else if nodes[i].obstacle = true number of terms  > termThreshold, true failure
        If AntSendErr[i] number of terms  < termThreshold, false failure
        Else if number of terms  > termThreshold, true failure
        If CanNotSendErr [i] > 0 && CanNotRecvErr[i] >0, noise
        ……
    }
```

Appendix B
Decision Pruned Tree

That pruned tree acts as classifier which has been implemented using WEKA [26]. Because of the number of the available examples is not large enough a 10-fold cross validation is employed.

```
Test mode:    10-fold cross-validation
=== Classifier model (full training set) ===
J48 pruned tree
------------------
traficdrop <= 0
|   traficoverflow <= 0
|   |   obs_area <= 0
|   |   |   zeroconcections <= 0
|   |   |   |   canRecv <= 0
|   |   |   |   |   canSend <= 0: 2 (7902.0)
|   |   |   |   |   canSend > 0: 4 (185.0/8.0)
|   |   |   |   canRecv > 0
|   |   |   |   |   antsendsw_fail <= 0: 5 (84.0)
|   |   |   |   |   antsendsw_fail > 0: 2 (2.0)
|   |   |   zeroconcections > 0
|   |   |   |   antrecvhw_fail <= 0
|   |   |   |   |   antsendhw_fail <= 0
|   |   |   |   |   |   antrecvsw_fail <= 0
|   |   |   |   |   |   |   antsendbattery_fail <= 0
|   |   |   |   |   |   |   |   antrecvbattery_fail <= 0
|   |   |   |   |   |   |   |   |   antsendsw_fail <= 0
|   |   |   |   |   |   |   |   |   |   canSend <= 0: 1 (3482.0/7.0)
|   |   |   |   |   |   |   |   |   |   canSend > 0: 4 (37.0)
|   |   |   |   |   |   |   |   |   antsendsw_fail > 0: 2 (163.0)
|   |   |   |   |   |   |   |   antrecvbattery_fail > 0: 2 (177.0/2.0)
|   |   |   |   |   |   |   antsendbattery_fail > 0: 2 (186.0)
|   |   |   |   |   |   antrecvsw_fail > 0: 2 (227.0/3.0)
|   |   |   |   |   antsendhw_fail > 0: 2 (260.0)
|   |   |   |   antrecvhw_fail > 0: 2 (311.0)
|   |   obs_area > 0
|   |   |   antsendbattery_fail <= 0
|   |   |   |   antrecvsw_fail <= 0
|   |   |   |   |   antsendhw_fail <= 0: 1 (9551.0/368.0)
|   |   |   |   |   antsendhw_fail > 0
|   |   |   |   |   |   antrecvhw_fail <= 0
```

```
|  |  |  |  |  |  |  antrecvbattery_fail <= 0
|  |  |  |  |  |  |  |  antsendsw_fail <= 0: 1 (401.0/118.0)
|  |  |  |  |  |  |  |  antsendsw_fail > 0: 2 (11.0/1.0)
|  |  |  |  |  |  |  antrecvbattery_fail > 0: 1 (24.0)
|  |  |  |  |  |  antrecvhw_fail > 0: 2 (43.0/1.0)
|  |  |  |  antrecvsw_fail > 0
|  |  |  |  |  antrecvbattery_fail <= 0
|  |  |  |  |  |  antsendhw_fail <= 0
|  |  |  |  |  |  |  antrecvhw_fail <= 0
|  |  |  |  |  |  |  |  zeroconcections <= 0: 2 (266.0/119.0)
|  |  |  |  |  |  |  |  zeroconcections > 0: 1 (86.0/33.0)
|  |  |  |  |  |  |  antrecvhw_fail > 0
|  |  |  |  |  |  |  |  zeroconcections <= 0: 1 (79.0/27.0)
|  |  |  |  |  |  |  |  zeroconcections > 0: 2 (6.0/2.0)
|  |  |  |  |  |  antsendhw_fail > 0: 1 (108.0/10.0)
|  |  |  |  |  antrecvbattery_fail > 0: 1 (46.0)
|  |  |  antsendbattery_fail > 0
```

REFERENCES

[1] M.Dahiya, “MANET’s: Security Attacks and Securing Routing Protocols”, Advances in Wireless and Mobile Communications Volume 10, Number 4, 2017, pp.693-697.

[2] S.Singh and R.Singh, “A Review on Detection and Isolation of Selective Packet Drop Attack in MANET”, IJCSN International Journal of Computer Science and Network, Volume 6, Issue 3, June 2017, pp.395-399.

[3] A. Mishra, and K. Nadkarni, “Security in MANETs”, The handbook of wireless Ad-Hoc networks, 2002.

[4] L. Ertau, D. Ibrahim, “Evaluation of Secure Routing Protocols in Mobile Ad Hoc Networks (MANETs)”, California State University USA, 2004 .

[5] G.Santos and M.Correia , "Efficient Byzantine Fault-Tolerance", IEEE Transactions on Computers, VOL. 62, NO. 1, Jan 2013.

[6] V. Athira, G. Jisha, "Network Layer Attacks and Protection in MANETA Survey", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3) ,2014.

[7] N.Agrawal and K.Kumar, N.Joshi, “Performance evaluation of byzantine rushing attack in ad-hoc network”, International Journal of Computer Applications (0975 – 8887) Volume 123 – No.6, Aug 2015.

[8] A. Geetha, N. Sreenath, "Byzantine Attacks and its Security Measures in Mobile Ad-hoc Networks", Int'l Journal of Computing, Communications & Instrumentation Engg. (IJCCIE) Vol. 3, Issue 1,2016.

[9] C.Guntewar and V.Sahare, “A Review on Byzantine Attack Detection and Prevention Using Game Theory”, (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (1) , 2015, pp. 749-752.

[10] S. Sevil, C. John Andrew, "Intrusion Detection in Mobile Ad Hoc Networks", Guide to Wireless Ad Hoc Networks, Computer Communications and Networks, 2009.

[11] R.Sivakami, G.Kadhar, " A radical block byzantine attacks in mobile ad-hoc network ", Wireless Personal Communications: An International Journal archive Volume 87 Issue 2, March 2016, pp. 485-497.

[12] L Hogie, P. Bouvry, F. Guinand, "An Overview of MANETs Simulation", 2005, URL www.elsevier.nl/locate/entcs .

[13] C.Copeland and H. Zhong, "Tangaroa: a Byzantine Fault Tolerant Raft", 2013,

[14] H.Moniz, N. Ferreira Neves, and M. Correia, "Turquois: Byzantine Consensus in Wireless Ad-Hoc Networks", University of Lisboa Portugal, 2009.

[15] A.Eduardo, P.Alchieri and others, "Byzantine Consensus with Unknown Participants", OPODIS 2008, LNCS 5401, pp. 22–40, 2008.

[16] Mao-Lun Chiang, Shu-Ching Wang, and Lin-Yu Tseng3, "The Anatomy Study of Consensus Agreement in MANETs", 2005, URL.

[17] S.Mallapur, S.Patil, "Survey on Simulation Tools for Mobile Ad-Hoc Networks", International Journal of Computer Networks and Wireless Communications (IJCNWC), Vol.2, No.2, April 2012.

[18] S.Duan, S.Peisert, And K.Levitt, "hBFT: Speculative Byzantine Fault Tolerance with Minimum Cost ", IEEE Transactions on Dependable and Secure Computing Volume: 12, Issue: 1, Feb 2015.

[19] H.LeBlanc and Et.Al, "Resilient Asymptotic Consensus in Robust Networks", IEEE Journal on Selected Areas in Communications Volume: 31, Issue: 4, April 2013, pp.766-781.

[20] L.Tseng and N.Vaidya, "Byzantine Consensus in Directed Graphs", US army Research Office grant, Feb 2013.

[21] C.Cachin and M.Vukolic, "Blockchain Consensus Protocols in the Wild", IBM Research - Zurich, Jul 2017.

[22] M.Desai and S.Nagtilak, “A Survey of Network Layer Attacks in MANET Using Sequence Diagram”, International Journal of Innovative Reasearch in Computer and Communication Engineering Vol.4 Issue 10, October 2016

[23] D.Ongaro and J.Ousterhout, “In Search of an Understandable Consensus Algorithm”, Stanford University, 2014.

[24] H.Chauhan and A.Chauhan, “Implementation of decision tree algorithm c4.5”, International Journal of Scientific and Research Publications, Volume 3, Issue 10, Oct 013.

[25] A.Mitrokotsa and C.Dimitrakakis, "Intrusion detection in MANET using classification algorithms:The effects of cost and model selection", Ad-Hoc networks, Vol. 11, 2013 .

[26] R.Bouckaert and Et.Al, "WEKA Manual for Ver 3-7-8", Jan 21 2013