Available online at www.sciencedirect.com
ScienceDirect
Ad Hoc Networks 6 (2008) 508–523
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Collaborative techniques for intrusion detection in mobile ad-hoc networks

Ningrinla Marchang [a], Raja Datta [b,*]

[a] Department of Computer Science and Engineering, North Eastern Regional Institute of Science and Technology, Nirjuli 791109, Arunachal Pradesh, India
[b] Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, West Bengal, India

Received 14 July 2006; received in revised form 16 January 2007; accepted 10 April 2007
Available online 24 April 2007

Abstract

In this paper, we present two intrusion detection techniques for mobile ad-hoc networks, which use collaborative efforts of nodes in a neighborhood to detect a malicious node in that neighborhood. The first technique is designed for detection of malicious nodes in a neighborhood of nodes in which each pair of nodes in the neighborhood are within radio range of each other. Such a neighborhood of nodes is known as a *clique* [12]. The second technique is designed for detection of malicious nodes in a neighborhood of nodes, in which each pair of nodes may not be in radio range of each other but where there is a node among them which has all the other nodes in its one-hop vicinity. This neighborhood is identical to a *cluster* as mentioned in [12]. Both techniques use message passing between the nodes. A node called the *monitor* node initiates the detection process. Based on the messages that it receives during the detection process, each node determines the nodes it suspects to be malicious and send votes to the *monitor* node. The *monitor* node upon inspecting the votes determines the malicious nodes from among the suspected nodes. Our intrusion detection system is independent of any routing protocol. We give the proof of correctness of the first algorithm, which shows that it correctly detects the malicious nodes always when there is no message loss. We also show with the help of simulations that both the algorithms give good performance even when there are message losses arising due to unreliable channel.
© 2007 Elsevier B.V. All rights reserved.

Keywords: MANET; Intrusion detection system (IDS); Malicious node; Security; Wireless network

1. Introduction

The proliferation of mobile devices such as laptops, PDAs and mobile phones have ushered in exciting applications such as virtual classrooms, rescue missions, virtual conferences, etc. Mobile ad-hoc Networking is a technology which makes all these applications a possibility anywhere. All that is required is for a group of mobile nodes to self-configure and form a network without the need of any fixed infrastructure or a centralized controlling authority. In this network, a mobile node behaves as a host and a router at the same time.

* Corresponding author. Tel.: +91 9474065905; fax: +91 3222 282264.
E-mail addresses: ningrinla@yahoo.co.in (N. Marchang), rajadatta@ece.iitkgp.ernet.in (R. Datta).

1570-8705/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2007.04.003

Thus, this technology makes a mobile node really mobile as compared to a conventional infrastructure-based mobile network, in which case a mobile node is constrained to work within a certain radius from the infrastructure. Also a mobile ad-hoc network (MANET) could be a cost-effective solution for providing communication in areas where setting up fixed infrastructures would be an impossible task due to geographical constraints or financial inviability. However, the downside of MANET is that it is difficult to ensure a secure communication within the network.

Providing security in mobile ad-hoc networks (MANET) is a prime concern due to the need of providing protected communication between mobile nodes in a hostile environment. Early research efforts on MANET assumed a friendly and co-operative environment, which may not apply in real-life scenarios (e.g., a MANET set up in a battlefield environment). Unlike in infrastructure-based wireless networks, the unique characteristics of MANETs such as open network architecture, shared wireless medium, stringent resource constraints and highly dynamic network topology pose new security challenges. Several schemes have been proposed for secure routing protocols, intrusion detection and response systems.

One way of securing a mobile ad-hoc network at the network layer is to secure the routing protocols such that all possible attacks are prevented. Several kinds of attacks, such as routing loop attack, black hole attack, gray hole attack, partitioning, etc. are given in detail by Hu et al. [1]. Some ways through, which these attacks can be carried out are, using modification of control message fields (e.g., modifying the hop count, modifying the sequence number, etc.) and using impersonation (e.g., forge messages by spoofing sender or destination addresses). Yang et al. proposed that security solutions for MANET should provide complete protocol protection spanning the entire protocol stack [2].

However, as far as we know, no secure routing protocol proposed in the literature so far takes care of all kinds of attacks. And if it were that a secure routing protocol would come up, which takes care of all known attacks, yet, one can never say when a different kind of attack which has not been envisaged before will suddenly raise its ugly head, exploiting the weaknesses in the ever-increasingly complex systems due to design and programming errors. This would require a modification of the secure routing protocol to be able to handle this new attack. In other words, one can never claim that a prevention mechanism is foolproof. Hence, the need arises for a second wall of defense: an intrusion detection system. The idea is that in the unfortunate event of a MANET being intruded, if there exists a system for detection of such an intrusion, it could be detected as early as possible, and the MANET could be saved before any extensive harm can be done, even if it cannot be avoided altogether. Research efforts are going on to develop Intrusion Detection Systems (IDS) to detect intrusion, identify the malicious nodes, and isolate them from the rest of the network. Further, the presence of a detection system will discourage malicious nodes from attempting intrusion in future. An analogy of an intrusion detection system would be security guards posted to secure a house. A trespasser could be detected by the security guards and handed over to the police. Moreover, it is likely that the trespasser will think twice before he attempts to break in again in future.

In this paper, we have presented algorithms for detection of malicious nodes that may intrude a MANET. The first algorithm, which we call ADCLI (Algorithm for Detection in a CLIque) is for detection of malicious nodes in a *clique*, whereas the second algorithm, which we call ADCLU (Algorithm for Detection in a CLUster) is for detecting malicious nodes in a *cluster*. In both the algorithms, we have used a message passing mechanism between the group of nodes, which enables each of the nodes to determine those nodes in the group that are suspected to be malicious. Finally, a voting method is used for detecting the malicious nodes from among the suspected nodes. We give the proof of correctness of the first algorithm and also show with the help of simulation that it gives good performance even when there is packet loss (due to packet collision, unreliable channel etc.) up to 6%. Through simulation, we also show that the second algorithm is efficient even for a network where packet loss may go up to 5%. Our intrusion detection algorithms are independent of any routing protocol. Most of the earlier detection systems in the literature require a considerable amount of packets to be buffered in the process of monitoring the neighbor's traffic, which accounts for a lot of overhead for a node. Our algorithm does not need to monitor packets meant for other nodes thereby reducing this overhead.

The rest of the paper is organized as follows: In Section 2 we discuss related works on intrusion

detection. In Section 3, we describe our proposed intrusion detection algorithms along with the assumptions. We also present the algorithms formally in this section. It is followed by the proof of correctness for the first algorithm in Section 4. Section 5 gives the simulation results and conclusions are given in Section 6.

2. Related works

The following are some of the proposed techniques for intrusion detection in MANET found in the literature. Marti et al. [3] presented the watchdog and path-rater tools for detecting and mitigating routing behavior. Watchdog is an intrusion detection system running on each node in the mobile ad-hoc network. It assumes that the nodes operate in the promiscuous mode, which makes them listen to the transmissions of their one-hop neighbors. Thus by listening to its neighbors, a node can detect whether packets sent to its neighbor for forwarding have been successfully forwarded by its neighbor or not. If the neighbor is found to be behaving maliciously (crosses a threshold of accepted misbehavior), it is considered malicious and its behavior is reported to the path-rater. Examples of malicious behavior could be dropping a packet or modifying its contents before forwarding. Path-rater is also a component running on each node, which maintains behavior ratings for each node in the network. These ratings are used as metrics while choosing a path for data transmission. Watchdog has some obvious disadvantages such as watchdog can be deceived by two neighbors colluding together and the other being the need for each node to store the transmitted packets until they are forwarded by its neighbor in the route [4].

Manikopoulus and Ling [5] presented an architecture for mobile ad-hoc network security where an intrusion detection system (IDS) runs on every node. This IDS collects local data from its host node and neighboring nodes within its communication range, processes raw data and periodically broadcasts to its neighborhood classifying normal or abnormal behavior based on processed data from its host and neighbor nodes. Another proposed intrusion detection scheme, which is based on the principle of misuse detection that can accurately match signatures of known attacks is presented in [6] by Nadkarni and Mishra. Partwardan et al. proposed an intrusion detection scheme based on anomalous behavior of neighboring nodes [7]. Each node monitors particular traffic activity within its radio range. All locally detected intrusions are maintained in an audit log. Once local audit data is collected, it can be processed using some algorithm to detect ongoing attacks from the collected data. Zhang and Lee [8] examined the vulnerabilities of a wireless ad-hoc network, the need for an intrusion detection to supplement a secure routing mechanism, and the reason why detection methods available for the wired environment are not applicable directly in a wireless environment. They also proposed an intrusion detection system which is both cooperative and distributed.

Zhang et al. [9] developed an architecture for intrusion detection which is distributed and cooperative. They also presented how anomaly detection could be done by using a classifier which is trained using normal data to predict what is normally the next event given the previous sequence of events. Deviation from the predicted event would mean that there is an intrusion. Anantvalee and Wu [11] extensively surveyed on various intrusion detection techniques and also gave a comparison among these techniques. Albers et al. [13] gave an IDS architecture with the use of mobile agents, which is both distributed and cooperative. A Local Intrusion Detection System (LIDS) sits on every node, which detects intrusion locally. However, a LIDS can cooperate with other LIDS for global detection. Kachirski and Guha [14] also used mobile agents to develop a multisensor intrusion detection system. The system consists of three main agents: monitoring agent, action agent and decision agent, each taking care of a functionality thereby distributing the workload. Monitoring agents are of two types: the network monitoring agent and the host-based monitoring agent. The action agent sits on every node and takes care of initiating a response after an anomaly is detected. The network is logically divided into clusters, each with a clusterhead. The network monitoring agent and the detection agent are run on the clusterhead. The network monitoring agent captures and monitors packets passing through the network within its radio range. When the local detection cannot make a decision on its own, it reports to the decision agent, which uses the packet-monitoring results that comes from the network-monitoring agent to decide whether it is malicious or not.

Buchegger and LeBoudec [15] proposed the CONFIDANT (Cooperation of Nodes, Fairness in Dynamic Ad-hoc NeTworks) protocol which

makes misbehavior unattractive. This protocol is similar to Watchdog and Pathrater. However, apart from monitoring malicious behavior within its radio range as in Watchdog, a node also processes information from trusted nodes to detect a misbehaving node. When a node concludes from its observations that another node is malicious, it informs the path manager, which removes all paths containing the misbehaving node. Moreover, it also sends ALARM message about this misbehaving node to other trusted nodes. Michiardi and Molva [16] proposed CORE, a mechanism based on reputation to detect selfish nodes and enforce cooperation among them. CORE can be said to have two components, the monitoring system and the reputation system as in CONFIDANT. For the reputation system, it maintains several tables for each node, one table for each function such as routing discovery or forwarding packets performed by the node and also a table for accumulated values for each node. Negative rating is given to a node only from direct observation when the node does not cooperate, which eventually results in decreased reputation of the node. However, positive rating is given from both direct observation and positive reports from other nodes, which results in increased reputation. When a request comes from a node, if the overall reputation of the node is negative, it is rejected thus isolating it from the network. Bansal and Baker [17] proposed OCEAN (Observation-based Cooperation Enforcement in Ad-hoc Networks), an extension to the DSR protocol. It also uses a monitoring system and a reputation system as in the above mechanisms. The difference of OCEAN from the other protocols that use both these systems is that it relies only on its own observations. This prevents unwanted conclusions that may result from false accusations.

The algorithms proposed in this paper are based on anomaly detection as in [3,7,10]. This is different from misuse detection as proposed in [5,6]. However, our algorithms use collaborative efforts of nodes in the neighborhood to detect a malicious node. This makes them more tolerant to factors such as packet collision. The proposed algorithms also takes care of colluding nodes, which find little or no mention in earlier works. As in [14], to run our algorithms, the network is divided into clusters, and the algorithms can be run in each cluster, with the clusterhead as the *monitor* node. As far as we know this is the first work that uses a collaborative message passing mechanism to detect malicious nodes.

3. The algorithms

In this section we propose the two algorithms for detection of malicious nodes in a MANET. By a malicious node, we mean a node, which does not follow expected behavior. A malicious node may try to launch several kinds of attacks as mentioned in the previous section. Most of these attacks are accomplished by modifying a message before forwarding or simply not forwarding a message, which it is supposed to forward. While developing both our algorithms, we have assumed that a malicious node will exhibit these two properties during its lifetime. We try to detect this unexpected behavior and finally nail the malicious nodes exhibiting these behaviors.

The ADCLI and ADCLU algorithms can be used in different *cliques* (*clusters*). Each *clique* (*cluster*) will come up with information about the malicious nodes, if any. It can then use this information for isolating these malicious nodes from itself. Moreover, this information may be sent to other *cliques* (*clusters*), so that they can also isolate the malicious nodes from themselves. In other words, information about the malicious nodes may be used in routing decisions. The node that initiates the ADCLI (ADCLU) algorithm is called the *monitor* node. There exist several algorithms for grouping the nodes in a MANET into clusters (cliques are known as 1-hop clusters), which are known as clustering algorithms. These algorithms specially assign a node in a cluster as the clusterhead. Two examples of such clustering algorithms are [18,19]. Clustering is generally done for hierarchical routing. In a MANET in which clustering is already being done for routing purposes, our algorithms can be executed on the existing clusters with the clusterheads as the *monitor* nodes. Even otherwise, any existing clustering algorithm can initially be applied to divide the network into clusters, and then, our algorithms can be used on those clusters with the clusterheads as the *monitor* nodes. If a *monitor* node remains the *monitor* node for a long time, its battery power may get depleted more rapidly than that of the other members of the cluster because of the overhead of being the *monitor* node. This can be well taken care of by employing clustering algorithms which elects a new clusterhead from time to time depending on the power consumption, etc. of the members so that there is a balance in energy levels of all the nodes in the cluster. One such algorithm is proposed in [18].

Our algorithms detect malicious nodes in a clique or a cluster at a particular instant of time. However, it would be detrimental to punish a node solely depending on its action at one point of time, taking into consideration our existing imperfect networks, which suffer from packet collisions, congestion, etc. So, our algorithms could be run randomly throughout the lifetime of the network. And the behavior of a node could be seen for a past specified duration of time, before any punishment be meted out. This will also help nodes who had past malicious records to correct themselves and be accepted again into the network. Besides, in case there is congestion, because of which a node may be dropping packets, the algorithm will not work properly and so, it may be aborted assuming a mechanism is there to detect congestion in the network.

3.1. The ADCLI algorithm

This algorithm can be used to detect malicious nodes in a set of nodes such that each pair of nodes in the set are within the radio range of each other (Fig. 1). This set of nodes is generally known as a *clique*. Two nodes within radio range of each other may be represented by an edge between the nodes. An example of a set of nodes in a mobile ad-hoc network represented thus is given in Fig. 2.

To present the algorithm we make the following assumptions: Once a message is received by a malicious node, it forwards wrong messages to at least half of the other nodes. Without loss of generality we also assume that the initiating node of this algorithm i.e., the *monitor* node is not malicious and when the *monitor* node initiates the detection process by sending out a message to the other nodes, the malicious nodes have no way of knowing that a detection algorithm is in progress. This assumption is required because, if a malicious node is able to detect that a detection algorithm has been initiated, it may try to behave non-maliciously and try to avoid detection. A malicious node in almost all probability will not try to initiate the algorithm as it may in any case pass wrong information to other nodes even without any intrusion detection system.

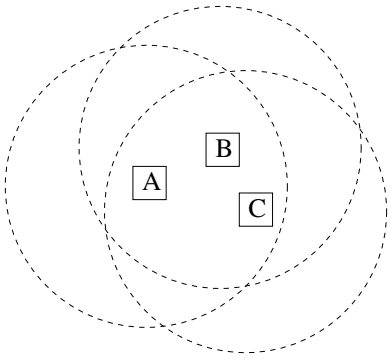

Fig. 1. Three mobile nodes within radio range of each other.

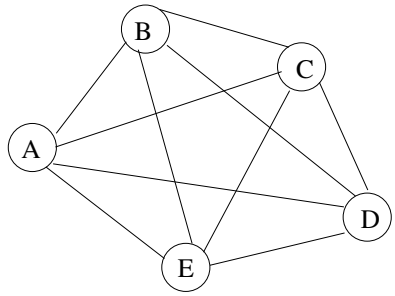

Fig. 2. A set of nodes in a MANET (a clique): an edge between two nodes denotes they are within radio range of each other.

We now present the *Intrusion Detection Algorithm* that can detect at most k malicious nodes in a set of n nodes, which are within radio range of each other where $n \geqslant 4k + 1$.

Step 1. The monitor node, M sends the message RIGHT to the other $n - 1$ nodes asking them to forward the message in turn to the other $n - 2$ nodes.

Step 2. Upon receiving the message RIGHT, node i (for each i such that node $i \neq$ monitor node), relays the message to the other $n - 2$ nodes (Here a malicious node may either refuse to forward the message or forward a message other than the message RIGHT, which it received in step 1).

Step 3. The monitor node then sends a MALICIOUS-VOTE-REQUEST message to all the other $n - 1$ nodes.

Step 4. On receipt of a MALICIOUS-VOTE-REQUEST message from the monitor node, each of the $n - 1$ nodes does the following:For each i and each j ($j \neq i$), let M_j be the message node i received from node j in step 2 (if node i does not receive any message from j or if it receives a message different from RIGHT, M_j is assigned default message WRONG). If $M_j \neq$ RIGHT, mark the node j as a suspected node and send node j to the monitor node. (i.e., a vote for node j being a suspected node is sent to the monitor node.)

Step 5. On receipt of the votes in step 4, the monitor node does the following:

i. Accept a maximum of k distinct votes from each of the nodes (By distinct votes, it means that the monitor node can accept at most one vote about a suspected node from any node).
ii. Mark the nodes, $(D_1, D_2, \ldots, D_m)$ with at least $k+1$ votes as malicious nodes. Let m be the number of malicious nodes.
iii. If the number of detected nodes is more than k (i.e., $m > k$), detection process has failed (When detection fails, it means that there were more than k malicious nodes).

Step 1 is used to send the message used for detection by the *monitor* node. In step 2, if a node is not malicious, it will faithfully forward the message, RIGHT. However if a node is malicious, it may act maliciously and do either of the following:

Case (a): Not forward the message at all to some or all of the $n-2$ nodes.
Case (b): Modify the message and send the modified message to some or all of the $n-2$ nodes.

In step 3, after waiting for step 2 to be completed, the *monitor* node sends a MALICIOUS-VOTE-REQUEST message requesting the other $n-1$ nodes to send in information (votes) about nodes, which they suspect as malicious. The assumption is that until a node receives this vote request message from the *monitor* node, it has no idea that a detection algorithm is in progress. By the time a malicious node receives this message, it has already acted maliciously in step 2 exhibiting its maliciousness, and it cannot escape detection. This assumption is valid and can be ensured by using a mechanism such as making the message sent by the *monitor* node in step 1 (i.e, RIGHT) look like a regular packet, so that no node will suspect it. However, this packet should not be of an important packet type, which may lead to security problems. A node is thus fooled and thinks that the *monitor* node (the monitor node is like any ordinary node) has requested it to forward some message to the other nodes.

In step 4, the $n-1$ nodes send information about suspected nodes to the *monitor* node. Suspected nodes are those nodes that acted maliciously. At this stage we call such nodes suspected nodes and not malicious (or detected) nodes because some suspected nodes may not actually be malicious. This situation can arise in the case when a malicious node lies about some other nodes and send these nodes as suspected nodes to the *monitor* node. (Please remember that a malicious node cannot be trusted to behave non-maliciously at any stage.)

In step 5, the *monitor* node counts the votes to finally nail the detected nodes. The algorithm can detect at most k malicious nodes. If there are more, the detection process is said to have failed.

3.1.1. Illustrative examples

To understand how this algorithm works, we consider the case when there are 5 nodes out of which one node is malicious, i.e., $k=1$, $n=5$. Fig. 3 illustrates the messages passed between the nodes during the first two steps of the ADCLI algorithm. Node 0 is the malicious node and node 1 is the *monitor node*. In step 1, node 1 sends out a message RIGHT (represented by solid lines labeled *R*), to the other four nodes 0, 2, 3 and 4. In step 2, each of these four nodes in turn relays the message to the other three nodes. Nodes 2, 3, and 4 being non-malicious relays the message RIGHT (represented by dashed lines labeled *R*), to other nodes. Node 0 being malicious relays message WRONG (represented by dotted lines labeled *W*), to nodes 2 and 3, whereas it sends message RIGHT to node 4. Fig. 4 illustrates the messages received by the $n-1$ nodes during step 1 and step 2 of the algo-

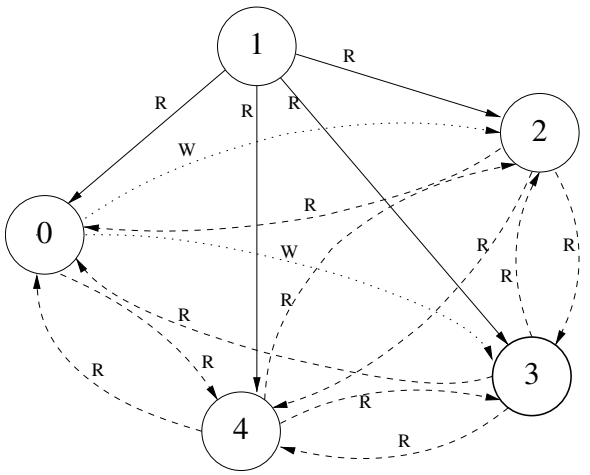

Fig. 3. Message received during execution of the ADCLI algorithm for $n=5$ and $k=1$; node 0 is malicious; node 1 is the monitor node.

	0	1	2	3	4	Suspected nodes	Nodes sent to monitor
0	–	R	R	R	R	–	4
2	W	R	–	R	R	0	0
3	W	R	R	–	R	0	0
4	R	R	R	R	–	–	–

Fig. 4. Matrix showing messages passed corresponding to Fig. 3.

rithm, and the votes sent by them in response to the MALICIOUS-VOTE-REQUEST message sent out at step 3. A row denotes the messages received by a node. As for example, the first row of the matrix denotes that node 0 received messages (R, R, R, R) from nodes 1, 2, 3 and 4 respectively. Similarly, the second row denotes that node 2 received messages (W, R, R, R) from nodes 0, 1, 3 and 4, respectively, and so on. In step 3, the *monitor* node sends out a MALICIOUS-VOTE-REQUEST message to all the four nodes. In response to that, nodes 2 and 3 send one vote each of node 0 to the *monitor* node. The lone malicious node 0 might choose to send a vote to the *monitor* node or might not do so. We consider these two cases:

Case (a): The lone malicious node 0 sends no vote to the *monitor* node:
In this case, the *monitor* node in step 5 will receive the votes: $(0, 0)$. Hence it will detect node 0 as the malicious node as it receives at least $k + 1$ votes.
Case (b): The lone malicious node 0 sends maliciously at most k votes of non-malicious node(s) to the *monitor* node:
In this case (Fig. 4), a vote for non-malicious node 4 (that node 4 is a suspected note) is sent by node 0. The *monitor* node in step 5 will receive the votes: $(0, 0, 4)$. However, it will detect node 0 as the malicious node as it is the only node, which receives at least $k + 1$ votes. Hence, in either case, node 0 is detected correctly.

The above example shows how the algorithm works when there is a lone malicious node. That the algorithm works even for the case when more than one malicious node collude with each other is illustrated by the following example. Figs. 5 and 6

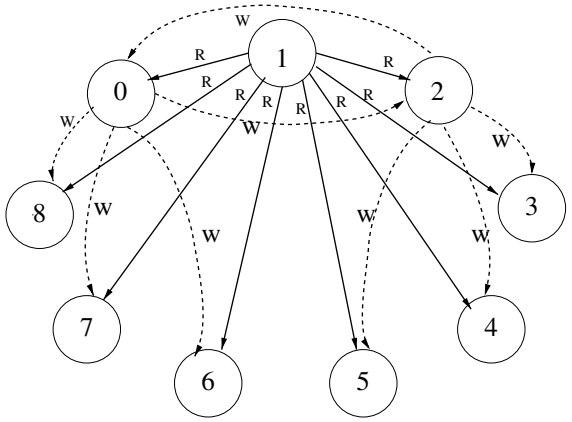

Fig. 5. Messages received during execution of the ADCLI algorithm for $n = 9$ and $k = 2$; nodes 0 and 2 are malicious; node 1 is the monitor node.

	0	1	2	3	4	5	6	7	8	Suspected nodes	Nodes sent to monitor
0	–	R	W	R	**R**	R	R	R	R	2	**3** and 4
2	W	R	–	R	R	R	R	R	R	0	**3** and 4
3	W	R	R	–	R	R	R	R	R	0	0
4	W	R	**R**	R	–	R	R	R	R	0	0
5	W	R	R	R	R	–	R	R	**R**	0	0
6	R	R	W	R	R	R	–	R	R	2	2
7	R	R	W	R	R	R	R	–	R	**2**	2
8	R	R	W	R	R	R	R	R	–	2	2

Fig. 6. Matrix showing messages passed corresponding to Fig. 5.

illustrate how the algorithm works for the situation when $n = 9$ and $k = 2$. Here, the *monitor* node is node 1 and nodes 0 and 2 are the malicious nodes. As in the previous example, in step 1, node 1 sends out a message RIGHT (represented by solid lines labeled R), to the other eight nodes 0, 2, 3, 4, 5, 6, 7 and 8. In step 2, each of these eight nodes in turn relays the message to the other seven nodes. Nodes 3, 4, 5, 6, 7 and 8 being non-malicious relays the message RIGHT (not shown in the figure for readability), to other nodes. Node 0 being malicious relays message WRONG (represented by dotted lines labeled W), to nodes 2, 6, 7 and 8 (at least half of the other seven nodes). It relays the message RIGHT (not shown in figure) to the rest of the nodes (3, 4, and 5). Similarly, Node 2 being malicious relays message WRONG (represented by dotted lines labeled W), to nodes 0, 3, 4 and 5 (at least half of the other seven nodes). It relays the message

RIGHT (not shown in figure) to the rest of the nodes (6, 7, and 8). Fig. 6 shows the messages received by the nodes during step 1 and step 2, and the votes sent by them in response to the MALICIOUS-VOTE-REQUEST message sent out at step 3. Assume that nodes 0 and 2 have colluded with each other not to report on each other. In addition, they have agreed to send votes reporting that nodes 3 and 4 are suspect. Notice that when in fact, node 0 and node 2 have each other in the suspected nodes list, they have sent out wrong votes about nodes 3 and 4. The maximum votes that will be accepted by the *monitor* node is k according to the algorithm. So, even if they were to send out more votes, only two distinct votes will be accepted. However, when the votes are added up the *monitor* node will detect nodes 0 and 2 as the malicious nodes as they are the only nodes, for which the *monitor* node receives at least $k+1$ (i.e., 3) votes. Hence, even when two nodes collude, the algorithm detects correctly. The proof of correctness of the algorithm shows that even when all the malicious nodes collude with each other, the algorithm still works correctly.

3.2. The ADCLU algorithm

The ADCLU algorithm can be used to detect malicious nodes in a set of nodes, which forms a *cluster*, which is defined as a neighborhood of nodes in which there a node, which has all the other nodes as its 1-hop neighbors (Fig. 7). Unlike in a *clique*, each pair of nodes may not be within radio range of each other. To present the algorithm we make the following assumptions: The wireless links between the nodes are bi-directional. When the *monitor* node initiates the detection process, the malicious nodes have no way of knowing that a detection algorithm is in progress as in the case of the ADCLI algorithm.

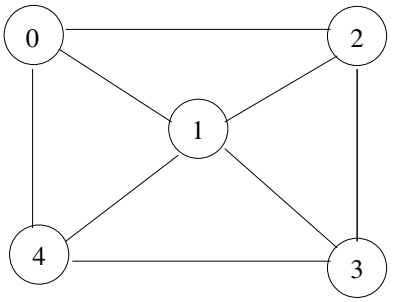

Fig. 7. A neighborhood (cluster) in a MANET consisting of 5 nodes: an edge between two nodes denotes they are within radio range of each other.

We now present the *ADCLU Intrusion Detection Algorithm*.

Step 1. The monitor node, M broadcasts the message RIGHT to its neighbor nodes asking them to further broadcast the message in their neighborhood.
M → broadcast: (RIGHT)

Step 2. Upon receiving the message RIGHT, each neighbor, B of M further broadcast the message in its neighborhood
B → broadcast: (X) ($X = RIGHT$ if B is not malicious, $X \neq RIGHT$ if B is malicious)

Step 3. The monitor node, M then broadcasts a MALICIOUS-VOTE-REQUEST message in its neighborhood.
M → broadcast: (MALICIOUS-VOTE-REQUEST)

Step 4. On receipt of a MALICIOUS-VOTE-REQUEST message from M, each neighbor, B of M does the following:Let P_A be the message node B received from node A in step 2 (if node B does not receive any message from A or if it receives a message different from RIGHT, P_A is assigned default message WRONG.)
If $P_A \neq$ RIGHT, then B sends a vote for node A being a suspected node to M.
B → M: (VOTE; A)

Step 5. On receipt of the votes in step 4, the monitor node does the following:

i. Accept only distinct votes from each of the nodes (By distinct votes, we mean that the monitor node can accept at most one vote about a suspected node from any node).

ii. Let N_A be the number of votes received for node A. If $N_A \geqslant k$, mark node A as malicious. (The monitor node also gives its vote. k is the threshold value.)

The basic difference between the ADCLI algorithm and the ADCLU algorithm is that the ADCLI algorithm uses unicast for message passing, whereas the ADCLU algorithm uses broadcast for message passing.

In this algorithm the *monitor* node sets a trap by sending a junk message which a malicious node (say m) might drop or forward after modifying it to its neighbors. When it does that, the neighbors of m,

which are also neighbors of the *monitor* node come to suspect that node. The identity of this suspected node is then reported by the neighbors to the *monitor* node, which in turn after calculating the number of such reports finally detects the malicious nodes. In this regard a threshold value (k) is maintained by the *monitor* node, such that a suspected node is marked actually malicious by the *monitor* node if it receives negative reports from at least k of its neighbors. The value of k depends on the number of neighbors a node can have in the network and also on whether we are using a strict or a lenient detection measure. A high value of k would mean that a strict measure (requiring more votes for detection) has been used and a low value would mean a lenient measure (requiring less votes for detection) has been used in the detection process.

Step 1 of the algorithm is used to broadcast the message used for detection by the *monitor* node. This message is limited to only two hops. That is after the neighbors broadcast in step 2, it dies. In step 2, if a node is not malicious, it will faithfully broadcast the message, RIGHT. However if a node is malicious, it may act maliciously and do either of the following:

Case (a) : Not broadcast the message at all.
Case (b) : Modify the message and broadcast it.

In step 3, after waiting for step 2 to be completed, the *monitor* node sends a MALICIOUS-VOTE-REQUEST message requesting the nodes in its neighborhood to send in information (votes) about nodes, which they suspect as malicious. The assumption is that until a node receives this vote request message from the *monitor* node, it has no idea that a detection algorithm is in progress. By the time a malicious node receives this message, it has already acted maliciously in step 2 exhibiting its maliciousness, and it cannot escape detection. This assumption is valid and can be ensured by using a mechanism such as making the message sent by the *monitor* node in step 1 (i.e, RIGHT) look like any ordinary data packet, so that no node will suspect it.

In step 4, the neighbor nodes of the *monitor* node send information about suspected nodes to the *monitor* node. Suspected nodes are those nodes that acted maliciously. At this stage we call such nodes suspected nodes and not malicious (or detected) nodes because only after the votes are gathered and the number of votes cross some threshold value will the node be said to be malicious.

In step 5, the *monitor* node counts the distinct votes from other nodes and detects the nodes with k or more votes, from among the suspected nodes, as malicious nodes.

The overhead of our algorithm is the amount of messages passed during the detection process. However, since broadcast mechanism is used, it is not as high as would be otherwise. For example, for the neighborhood of Fig. 7, where node 1 is the *monitor* node only ten messages need to be passed: one broadcast message by the *monitor* node at step 1, four broadcast messages by the four neighbors at step 2, one broadcast message by the *monitor* node in step 2, and finally four messages by the four neighbors while sending their votes in step 4.

3.2.1. Illustrative example

To understand how the ADCLU algorithm works, we consider a neighborhood (cluster) in which there are 5 nodes out of which node 1 is the *monitor* node (Fig. 7). Let us assume that node 0 is malicious and the threshold, $k = 2$. Fig. 8 illustrates the messages passed between the nodes during the first two steps of the ADCLU algorithm. Node 0 is the malicious node and node 1 is the *monitor* node. In step 1, node 1 broadcasts a message RIGHT (represented by solid lines labeled *R*), to its four neighbor nodes 0, 2, 3 and 4. In step 2, each of these four nodes in turn broadcasts the message to its neighbor nodes. Nodes 2, 3, and 4 being non-malicious broadcasts the message RIGHT (represented by dashed lines labeled *R*), to its neighbor

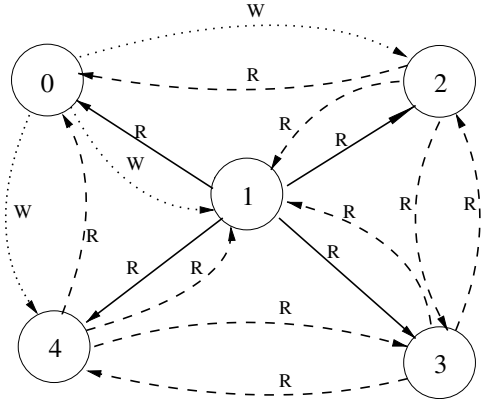

Fig. 8. Messages received during execution of the ADCLU algorithm for a neighborhood of 5 nodes; node 0 is malicious; node 1 is the monitor node.

	0	1	2	3	4	Suspected nodes	Nodes sent to monitor
0	–	R	R	–	R	–	–
2	W	R	–	R	–	0	0
3	–	R	R	–	R	–	–
4	W	**R**	–	R	–	0	0
1	W	–	R	R	R	0	

Fig. 9. Matrix showing messages received corresponding to Fig. 8.

nodes. Node 0 being malicious broadcasts the message WRONG (represented by dotted lines labeled *W*), to its neighbor nodes, 1, 2 and 4. Fig. 9 shows the messages received by each node during step 1 and step 2 of the algorithm, and the votes sent by them in response to the MALICIOUS-VOTE-REQUEST message sent out at step 3. As for example, the first row of the matrix denotes that node 0 received messages (*R*,*R*,*R*) from nodes 1, 2 and 4 respectively. Similarly, the second row denotes that node 2 received messages (*W*,*R*,*R*) from nodes 0, 1 and 3 respectively, and so on. In step 3 of the algorithm, the *monitor* node 1 broadcasts a MALICIOUS-VOTE-REQUEST message to all its neighbor nodes. In response to that nodes 2 and node 4 in step 4 send one vote each of node 0 to the *monitor* node because it received a WRONG message from node 0. In step 5, the *monitor* node counts the votes it received for each neighbor. Here we see that it received 2 votes for node 0 (from node 2 and node 4) and adding its own 1 vote for node 0, the count is 3 votes for node 0. The value of k is 2, and since the count is greater than the value of k, node 0 is detected as malicious.

3.2.2. Threshold value setting

As can be seen, the effectiveness of the ADCLU algorithm hinges on the threshold value k. Thus the value of k has to be set judiciously. Too high a value of k would lead to the algorithm being too strict and hence may result in malicious nodes remaining undetected. On the other hand, too low a value of k would result in the algorithm ending up detecting nodes which are not malicious as malicious (a high false positive rate). The threshold value, k can be set automatically by the *monitor* node assuming that it knows the topology of its cluster. This can be achieved by asking for the neighbor tables from its member nodes periodically.

Assuming a reliable communication between any two neighbor nodes, the value of k can be set such that the algorithm even takes care of colluding nodes. Colluding nodes are those nodes who have agreed with each other not to report (send votes) about each other to the *monitor* node. We assume only malicious nodes can be colluding nodes. We also assume that colluding nodes may go a step further in behaving maliciously and agree upon to send votes about a non-malicious node to the *monitor* node, thus incriminating it. Let the minimum number of neighbors in the cluster any member node can have be denoted by d. Let c be the expected number of colluding nodes. We consider two cases:

Case (a): No colluding nodes are present. Then, $k = d$. In this scenario, in case a node is malicious, all its neighbor nodes (whose count will be $\geqslant d$) will report (send vote about it) to the *monitor* node. Hence, it will be detected.

Case (b): Colluding nodes are present and it is expected that there are c of them. Then, $k = d - (c - 1)$, when $k > c$. Say, a node is malicious and also it colludes with other malicious nodes. Assuming colluding nodes are neighbors of each other, $c - 1$ is the number of votes that may not be reported by its colluding neighbors. Hence, the algorithm must try to detect from votes contributed by other non-colluding neighbors.This value of k works only when $k > c$. This condition is required so as to take care of the case when the colluding nodes try to incriminate a non-malicious node. For instance, in a cluster where $d = 3$ and $c = 2$, k would be set to 2, i.e., the *monitor* node requires a minimum of 2 votes for a node to be detected as malicious. Assuming that the two colluding nodes send a vote each about a non-malicious node to the *monitor* node, it will unfortunately be detected as malicious since it accrued number of votes equal to k. Thus the above expression for k also gives an upper bound on the number of colluding nodes that can be successfully taken care of.

4. Proof of correctness for the ADCLI algorithm

Theorem. *For a set of n nodes within the radio range of each other, where there are at most* $k(k > 0)$ *malicious nodes, the algorithm successfully detects the malicious nodes if* $n \geqslant 4k + 1$.

Proof. Two cases may arise. They are:

Case (a) : k malicious nodes are present.
Case (b) : less than k malicious nodes are present.

Initially we prove *case* (a) and then prove *case* (b) for each of two scenarios explained below.

In step 4 of the ADCLI algorithm, a node i marks node j as a suspected node if it receives from node j a message M_j such that $M_j \neq$ RIGHT. That is node j relays a message that is not originally sent by the *monitor* node (The original message (i.e., RIGHT) was sent to the node by the *monitor* node in step 1). As we have already assumed that the *monitor* node is not malicious, the k malicious nodes are among the other $n - 1$ nodes. These k malicious nodes forward messages other than RIGHT to at least half of the other $n - 2$ nodes, i.e., $\lceil (n-2)/2 \rceil$ nodes. Here we arrive at two scenarios.

Scenario I: The k malicious nodes forward messages other than RIGHT only to non-malicious nodes (From the receiver point of view, a malicious node not forwarding the message RIGHT is equivalent to forwarding the message WRONG). This scenario is depicted in Figs. 3 and 4 for $k = 1$ and $n = 5$. As per our assumption, each of the k malicious nodes sends out at least $\lceil (n-2)/2 \rceil$ many number of WRONG messages to non-malicious nodes. So the non-malicious nodes will be able to collectively suspect each malicious node at least $\lceil (n-2)/2 \rceil$ times. In other words, there will be at least $\lceil (n-2)/2 \rceil$ many votes for each of the k malicious nodes at the *monitor* node. As for example in Fig. 4, the non-malicious node 2 and node 3 send a vote each saying that node 0 is suspected. Hence, collectively the *monitor* node receives 2 votes for node 0 that it is suspected to be malicious.

On the other hand, the malicious nodes also may try to send some votes maliciously to the *monitor* node. In Fig. 4 for example, the malicious node 0 sends node 4 as a suspected node to the *monitor* node. At most k such votes from malicious nodes are considered by the *monitor* node even if more are received as per our assumption. Thus for every malicious node, let x be the number of votes (saying that it is a suspected node) sent by the non-malicious nodes to the *monitor* node. Hence, $x \geqslant \lceil (n-2)/2 \rceil$. For any non-malicious node, let y be the maximum number of votes (saying that it is a suspected node) that can be sent by the malicious nodes maliciously to the *monitor* node. We see that $y = k$ (when all of k malicious nodes decide to vote it as a suspected node). As $n \geqslant 4k + 1$, it is easy to conclude that $x > y$. In step 5, we see that only the suspected nodes with at least $k + 1$ votes are detected as malicious. Hence, only the actual malicious nodes, which have x many votes each will be detected since $x > k$. For any non-malicious node wrongly suspected by a malicious node, each will have a maximum of k votes and so, will not be detected.

Moreover for the case when less than k malicious nodes are present, the algorithm works correctly. Let j where $(j < k)$ be the number of malicious nodes present. In this case also, the number of votes sent to the *monitor* node for each malicious node will be at least $\lceil (n-2)/2 \rceil$, which is greater than k, whereas the maximum number of votes that can be sent to the *monitor* node for a non-malicious node maliciously suspected by the malicious nodes is only j (all malicious nodes deciding to send a vote for a non-malicious node). And $j < k$. Hence, only the actual malicious nodes will be detected as they only will have votes $\geqslant k + 1$.

Scenario II: The k malicious nodes forward the messages other than RIGHT to malicious as well as non-malicious nodes. This scenario is depicted in Figs. 5 and 6 for $k = 2$ and $n = 9$. As per our assumption, each of the k malicious nodes sends out at least $\lceil (n-2)/2 \rceil$ many number of WRONG messages to non-malicious nodes as well as to the other malicious nodes. As for example in Fig. 6 (column 1), node 0 which is a malicious node relays out message WRONG to four nodes of which node 2 is malicious, whereas nodes 3, 4 and 5 are not. A non-malicious node on receiving a WRONG message from a malicious node behaves (suspects) correctly in that it sends a vote for this malicious node to the *monitor* node. However, when a malicious node receives a WRONG message from another malicious node, it may report correctly about this malicious node (which results in a situation similar to scenario I), or otherwise act maliciously while sending votes to the *monitor* node. It could act maliciously by resorting to either of the two actions:

Case (*i*): It may not report about this malicious node to the *monitor* node (Two nodes colluding with each other may not report about each other).

Case (*ii*): Still worse, not only it may not report about this malicious node but also may maliciously send at most k non-malicious nodes in its place as suspected nodes (The *monitor* node receives only a maximum of k votes from a node).

A malicious node may resort to the above actions to defeat the detection process. However, our algorithm takes care of this even in the worst case. In the worst case, any of the k malicious nodes, say node i may send WRONG messages to the other $k-1$ malicious nodes and these $k-1$ malicious nodes may resort to one of the above actions. Hence, $k-1$ votes for node i (saying that node i is a suspected node) will not be sent to the *monitor* node in either of the above cases. Thus for each of the k malicious nodes, say node i, the votes for node i (saying that i is a suspected node) that the *monitor* node receives eventually, if any are the ones contributed by non-malicious nodes. Let u represent the number of these votes and $e =$ total number of votes generated for malicious node i (depending on the WRONG messages sent out by node i to malicious as well as non-malicious nodes). Also let f be the maximum number of votes for node i not reported by other colluding malicious nodes. Therefore $u = e - f$ and $e \geqslant \lceil (n-2)/2 \rceil$. And, $f = k-1$, since there can be a maximum of $k-1$ colluding malicious nodes other than node i. Substituting the values of e and f, we get $u \geqslant \lceil (n-2)/2 \rceil - (k-1)$. Substituting the minimum value of n (i.e., $n = 4k+1$), we get $u \geqslant k+1$.

Now, for any non-malicious node maliciously suspected by a malicious node, there can be a maximum of v ($v = k$) votes (All of k malicious nodes deciding to send it as a suspected node). We find that $u > v$. In step 5, we see that only the suspected nodes with at least $k+1$ votes are detected as malicious. Hence, only the actual malicious nodes, which have u many votes each will be detected at the *monitor* node, since $u \geqslant k+1$. For any non-malicious node wrongly suspected by a malicious node, each will have a maximum of k votes and so, will not be detected.

Moreover for the case when less than k malicious nodes are present, the algorithm works correctly. Let $j (j < k)$ be the number of malicious nodes present. In this case, the number of votes sent to the *monitor* node for each malicious node will be greater than k, whereas the maximum number of votes that can be sent to the *monitor* node for a non-malicious node maliciously suspected by the malicious nodes is only j where $j < k$. Hence, only actual malicious nodes will be detected as they only will have greater than or equal to $k+1$ votes. Moreover, since $u = v + 1$, where $v = k$, and the algorithm detects only suspected nodes with at least $k+1$ votes, $4k+1$ is the minimum value that n can have such that at most k malicious nodes will always be correctly detected. Please note that if the minimum value of n is decreased by 1 (i.e., $n = 4k$), the value of u becomes equal to the value of v and the detection process fails.

Hence, the algorithm correctly detects k malicious nodes in a system of n nodes where $n \geqslant 4k+1$. □

5. Simulation results

5.1. ADCLI algorithm

The ADCLI algorithm was simulated using the NS-2 Simulator. Various realistic radio ranges were taken where the nodes move according to the waypoint mobility model with a maximum speed of 10 m/s. The pause time used was 2 s and the routing protocol used was AODV. Message loss was considered by random selection of messages at various steps of the algorithm except at the first step. The detection process was aborted if packet loss happened at step 1. The malicious nodes were selected at random and were made to drop or modify all the messages that they were to forward. In view of our algorithm, they send WRONG messages.

The simulation was done for different values of n and k. We used different movement spaces for different values of n. For different percentage of collisions, twenty random runs were performed for each simulation scenario, i.e., for each value of n. The percentage of collisions is the percentage calculated from the maximum total number of messages that will be received during the execution of the algorithm. It may be noted that if no collision occurs the maximum total number of messages are passed and received successfully. However, when collisions are assumed, some messages may never reach a recipient due to collision.

Due to space constraint we show here the results for two values of n i.e., for $n = 5$ and $n = 21$. Figs.

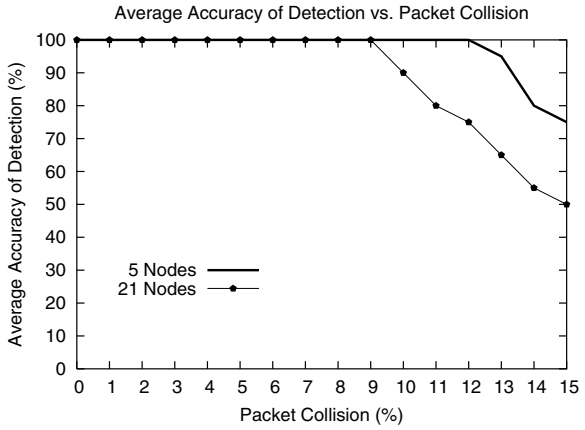

Fig. 10. ADCLI algorithm: Average accuracy of detection vs. packet collision for two neighborhood sizes.

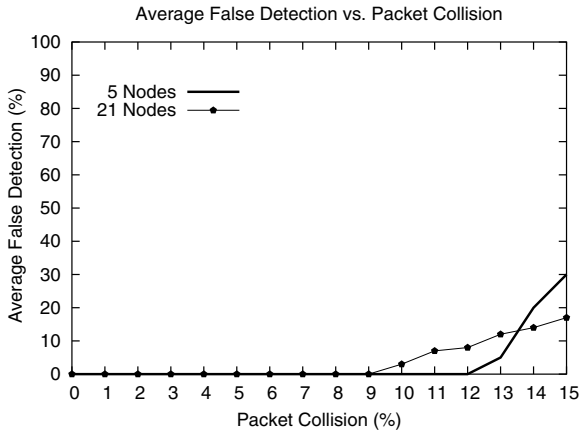

Fig. 11. ADCLI algorithm: Average false detection vs. packet collision for two neighborhood sizes.

10 and 11 give the results for two neighborhoods, one of 5 nodes in which there is 1 malicious node, the movement space being 250 m × 250 m. The other neighborhood is of 21 nodes, in which there are 5 malicious nodes; the movement space is taken as 500 m × 500 m. The average accuracy of detection for the neighborhood of 5 nodes is 100% even for the case when the collision of packets is as high as 12%. False positives (false detection of non-malicious nodes) were seen to appear only when the percentage of collision is more than 12 (Fig. 11). Similarly, for the neighborhood of 21 nodes, the accuracy of detection (Fig. 10) decreases as the percentage of collision increases as expected. However, it is noticed that as compared to the scenario of a neighborhood size of 5 nodes the decrease is more rapid. This is due to the reason that for the same percentage of collision the number of collisions is more in a neighborhood of size 21 than in a neighborhood of size 5. More number of collisions would mean higher loss of messages passed between the nodes and hence more adverse affect on the correct working of the algorithm. The average false detection is found to be nil for collisions up to 9% (Fig. 11).

5.2. ADCLU algorithm

The algorithm was simulated with the help of an Intel Xeon 2.8 GHz machine using threads. It was run for two different neighborhood sizes, one in which the number of nodes is 5 and the other in which the number of nodes is 8. Three different topologies were taken for each of the neighborhood sizes and the results given are averages of the three. For different percentage of collisions, ten random runs were performed for each topology for each neighborhood size. So, a total of 1080 runs of the algorithm were performed. The percentage of collisions is the percentage calculated from the maximum total number of messages generated during the execution of the algorithm. It may be noted that in the scenario where there is no collision, all the messages generated by the algorithm are received. However, when collisions were assumed, some messages never reached a recipient due to collision. These collisions can happen at any stage during the running of the algorithm. If a collision occurs to a message meant for a neighbor during execution of step 1, the neighbor does not receive the message broadcast, and accordingly cannot participate during voting at step 4. Care has been taken so that the collisions are randomly and uniformly distributed at all stages of the algorithm. Figs. 12 and 13 give the results for a neighborhood of 5 nodes and a threshold (k) value of 2. We have chosen the threshold value as 2 because we have chosen the topologies of each neighborhood such that each of the nodes in the neighborhood is a 1-hop neighbor (within radio range) of at least 2 other nodes. Fig. 12 shows that till 5% collision the algorithm is completely successful. The accuracy of detection is also high even for the case when the collision is within 10%. Fig. 13 shows very few false positives (detection of non-malicious nodes) and that too above 10% collision percentage in some cases.

Figs. 14 and 15 give the results for a neighborhood of 8 nodes and a threshold (k) value of 2.

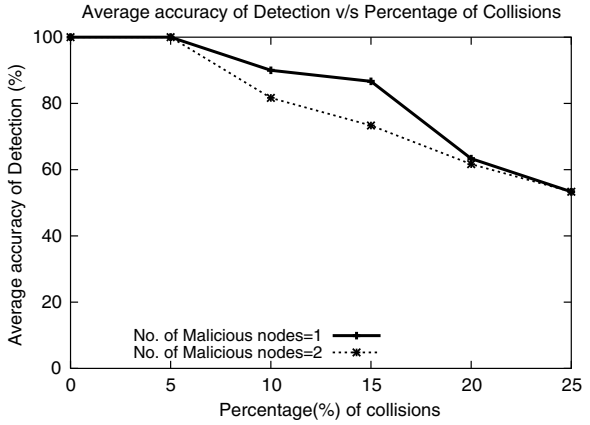

Fig. 12. ADCLU algorithm: Average accuracy of detection for a neighborhood size of 5 nodes and a threshold value of 2.

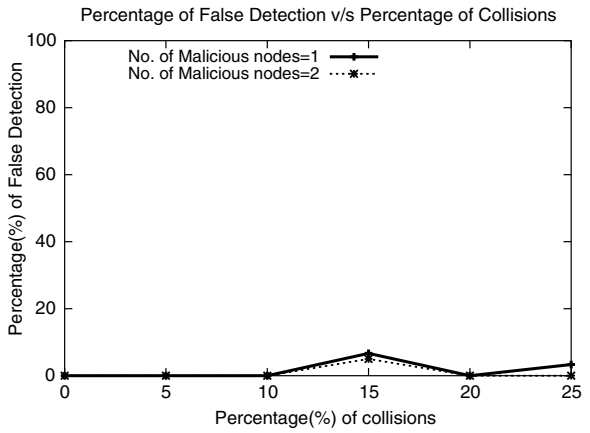

Fig. 13. ADCLU algorithm: Average false detection for a neighborhood size of 5 nodes and a threshold value of 2.

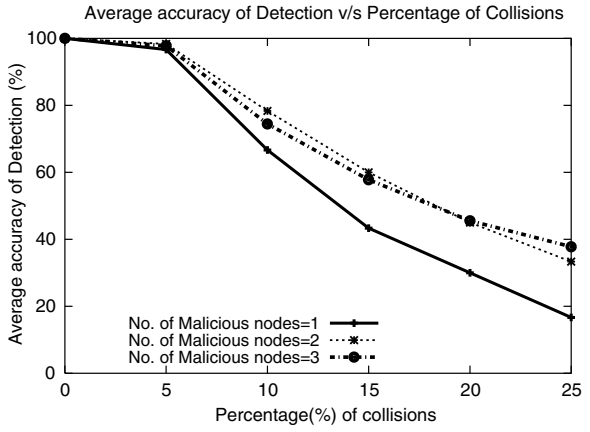

Fig. 14. ADCLU algorithm: Average accuracy of detection for a neighborhood size of 8 nodes and a threshold value of 2.

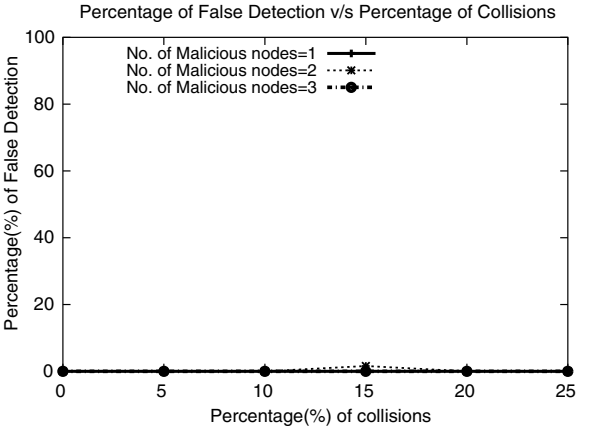

Fig. 15. ADCLU algorithm: Average false detection for a neighborhood size of 8 nodes and a threshold value of 2.

The average accuracy of detection decreases as the percentage of collision increases. However, it is noticed that as compared to the scenario of a neighborhood size of 5 (Fig. 12), the decrease is more rapid. This is because for the same percentage of collision the number of collisions is more in a neighborhood of size 8 than in a neighborhood of size 5. And as collision can happen at any stage of the algorithm, more number of collisions would result in more adverse effect on the correct working of the algorithm. The average false detection is found to be almost NIL (Fig. 15). The above results show that, even if due to increased number of collisions the algorithm does not detect all the malicious nodes correctly, the false detection is minimal. If the collision level (or message loss) is higher than 5% in a network, the algorithm may be run more than once for better result.

6. Conclusions

In this paper we presented two new algorithms for intrusion detection in mobile ad-hoc networks. The algorithms use collaborative effort from a group of nodes for determining the malicious nodes by voting. Messages are passed between the nodes and depending on the messages received, these nodes determine suspected nodes (nodes that are suspected to be malicious). These suspected nodes (votes) are eventually sent to the *monitor* node (the initiator of the detection algorithm). At the *monitor* node, the suspected nodes that receive at least a minimum number of votes are finally detected as malicious nodes. Thus instead of giving the sole

authority to a single node to decide about the maliciousness of another node, the algorithm works in such a way that a group of nodes together make this decision.

The ADCLU algorithm can be used for cliques as well as for clusters, whereas ADCLI can be used only for cliques. Both algorithms use a message passing mechanism, but with slightly different approaches. As mentioned earlier, the proper setting of the threshold value, k is crucial for the effectiveness of the ADCLU algorithm. However, the ADCLI algorithm is more robust in the sense that its effectiveness does not depend on the judicious setting of a parameter. The proof of correctness of the ADCLI algorithm shows that assuming a reliable communication between neighbor nodes, if the algorithm is run on a clique having n nodes, such that $n \geqslant 4k + 1$, then at most k malicious nodes present in the clique will be successfully detected.

To summarize, it can be seen from the simulation results that the ADCLI algorithm indeed detects the malicious nodes successfully with a high percentage of accuracy when at most k malicious nodes are present in a set of $n(n \geqslant 4k + 1)$ nodes even when there is a realistic percentage of packet collision (message destruction). Besides, average false detection is also minimal in such a scenario. However for the cases where more than k malicious nodes are present, the result may be unpredictable. The proof of correctness shows that the algorithm works correctly at all times for a reliable channel. In case of the ADCLU algorithm, our simulation results show that the algorithm works well even in an unreliable channel where the percentage of collision is around 5%.

Acknowledgement

This work was done as part of a Project No. 12(44)/05-IRSD of the Department of Information Technology, Government of India.

References

[1] Y. Hu, A. Perrig, D.B. Johnson, Ariadne: A secure On-Demand Routing Protocol for Ad-hoc Networks, Mobicom 2002, September 23–26, 2002, Atlanta, Georgia, USA.

[2] H. Yang, H. Luo, F. Ye, L. Zhang, Security in mobile ad-hoc networks: challenges and solutions, IEEE Wireless Communications (2004).

[3] S. Marti, T.J. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in a mobile ad-hoc environment, in: Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking, August 2000.

[4] S. Gupte, M. Singhal, Secure routing in mobile wireless ad-hoc networks, Ad Hoc Networks 1 (2003) 151–174.

[5] C. Manikopoulos, Li Ling, Architecture of the mobile ad-hoc network security (MANS) system, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, vol. 4, October 2003, pp. 3122–3127.

[6] K. Nadkarni, A. Mishra, Intrusion detection in MANETs – the second wall of defense, in: Proceedings of the IEEE Industrial Electronics Society Conference'2003, Roanoke, Virginia, USA, November 2–6, 2003, pp. 1235–1239.

[7] A. Partwardan, J. Parker, A. Joshi, M. Iorga, T. Karygiannis, Secure routing and intrusion detection in ad-hoc networks, in: Proceedings of Third IEEE International Conference on Pervasive Computing and Communications, Hawaii Island, Hawaii, March 8–12, 2005.

[8] Y. Zhang, W. Lee, Intrusion Detection in Wireless Ad-hoc Networks, Mobicom 2000, August 6–11, 2000, Boston, Massachusetts, USA.

[9] Y. Zhang, W. Lee, Y. Huang, Intrusion Detection Techniques for Mobile Wireless Networks, ACM WINET.

[10] Y. Huang, W. Fan, W. Lee, P. Yu, Cross-feature analysis for detecting ad-hoc routing anomalies, in: Proceedings of 23rd International Conference on Distributed Computing Systems, Providence, RI, May 2003.

[11] Tiranuch Anantvalee, Jie Wu, A survey on intrusion detection in mobile ad hoc networks, in: Y. Xiao, X. Shen, D.-Z. Du (Eds.), Wireless/Mobile Network Security, Springer, 2006, pp. 170–196.

[12] Yi-an Huang, Wenke Lee, A cooperative intrusion detection system for ad hoc networks, in: Proceedings of the ACM Workshop on Security in Ad Hoc and Sensor Networks (SASN'03), October, 2003, pp. 135–147.

[13] P. Albers, O. Camp, J.-M. Percher, B. Jouga, L. Me, R. Puttini, Security in ad hoc networks: a general intrusion detection architecture enhancing trust based approaches, in: Proceedings of First International Workshop on Wireless Information Systems (WIS-2002).

[14] O. Kachirski, R. Guha, Effective intrusion detection using multiple sensors in wireless ad hoc networks, in: Proceedings of 36th Annual Hawaii International Conference on System sciences (HICSS'03), January 2003, p. 57.1.

[15] S. Buchegger, J. Le Boudec, Performance analysis of the CONFIDANT protocol Cooperation of nodes – fairness in dynamic ad-hoc networks, in: Proceedings Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02), June 2002, pp. 226–336.

[16] P. Michiardi, R. Molva, Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks, in: Proceedings of Sixth IFIP Communication and Multimedia Security Conference (CMS'02), September 2002.

[17] S. Bansal, M. Baker, Observation-based cooperation enforcement in ad hoc networks, Research Report cs. NI/0307012, Stanford University.

[18] M. Chatterjee, S.K. Das, D. Turgut, WCA: a weighted clustering algorithm for mobile ad hoc networks, Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks) 5 (2002) 193–204.

[19] P. Krishna, N.H. Vaidya, M. Chatterjee, D.K. Pradhan, A cluster-based approach for routing in dynamic networks,

ACM SIGCOMM Computer Communication Review (1997) 49–65.

Ningrinla Marchang received her B.Tech. degree in Computer Science and Engineering from North Eastern Regional Institute of Science and Technology (NERIST), Itanagar, Arunachal Pradesh, India in 1993, and M.Tech degree in Computer Science and Engineering from Indian Institute of Technology (I.I.T.), Delhi, India in 1995.

From 1995 to 1996, she worked as a research engineer in the department of Computer Science and Engineering in Indian Institute of Technology, Delhi. From 1996 to 2001, she taught in the Department of Computer Applications in Sathyabama Deemed University, Chennai, India. Since 2001, she has been a faculty member of NERIST, Itanagar, India where she is a lecturer in the Department of Computer Science and Engineering. Her research interests include Computer Networks and Natural Language Processing.

Raja Datta received his B.E. degree in Electronics and Telecommunication Engineering from Regional Engineering College (presently National Institute of Technology), Silchar, India in 1988. He received the M.Tech degree in Computer Engineering and the Ph.D. degree in Computer Science and Engineering both from the Indian Institute of Technology Kharagpur, India.

From 1990 to February 2007 he worked as a faculty member of North Eastern Regional Institute of Science and Technology (NERIST), Itanagar, Arunachal Pradesh, India. Presently he is an Assistant Professor in the Department of Electronics and Electrical Communication Engineering (E&ECE) at the Indian Institute of Technology Kharagpur. His research interests include WDM networks, distributed processing and mobile ad-hoc networks.