Volume 80
Issue 3
May 2014
ISSN 0022-0000
ELSEVIER
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
Special Issue on Wireless Network Intrusion
Guest Editors
Al-Sakib Khan Pathan, Zubair Muhammad Fadlullah, Mostafa M. Fouda, and Hae Young Lee
Managing Editor
E. K. Blum
Editors
Alfred V. Aho
Jin-Yi Cai
Ronald Fagin
Georg Gottlob
J. Hartmanis
John Hopcroft
R. M. Karp
Raymond E. Miller
Christos Papadimitriou
Michael Rabin
Hartley Rogers, Jr.
Dana Scott
J. D. Ullman
Mihalis Yannakakis
L. A. Zadeh
Available online at www.sciencedirect.com
ScienceDirect

Journal of Computer and System Sciences 80 (2014) 554–570

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Algorithms for a distributed IDS in MANETs [☆], [☆☆]

P.M. Mafra [a, b, *], J.S. Fraga [a], A.O. Santin [c]

[a] Automation and Systems Department, PGEAS, UFSC, Caixa Postal 476, CEP 88040-900, Florianopolis, SC, Brazil
[b] Federal Institute of Santa Catarina, Sao Jose, SC, Brazil
[c] Pontifical Catholic University of Parana, Curitiba, PR, Brazil

ARTICLE INFO

Article history:
Received 30 November 2012
Received in revised form 30 April 2013
Accepted 14 June 2013
Available online 9 July 2013

Keywords:
Intrusion detection system
Mobile ad hoc networks
Distributed systems
Fault tolerance
Network security

ABSTRACT

This paper presents a set of distributed algorithms that support an Intrusion Detection System (IDS) model for Mobile Ad hoc NETworks (MANETs). The development of mobile networks has implicated the need of new IDS models in order to deal with new security issues in these communication environments. More conventional models have difficulties to deal with malicious components in MANETs. In this paper, we describe the proposed IDS model, focusing on distributed algorithms and their computational costs. The proposal employs fault tolerance techniques and cryptographic mechanisms to detect and deal with malicious or faulty nodes. The model is analyzed along with related works. Unlike studies in the references, the proposed IDS model admits intrusions and malice in their own algorithms. In this paper, we also present test results obtained with an implementation of the proposed model.

1. Introduction

The last decade has witnessed a great evolution in communication technologies and models that promote mobility and self-organization. Mobile Ad hoc Networks (MANETs) are an example of self-organized networks where there are no concentrator units (gateways) and the environment is highly dynamic with nodes joining and leaving at any time [2]. However, such networks are susceptible to a great variety of attacks. The challenge that arises is to maintain the MANET free from the activity of malicious or faulty nodes. In the face of the difficulty of avoiding the effects of malicious activities, mechanisms are necessary to at least minimize such effects. Intrusion detection systems (IDSs) can be used as one of these mechanisms [3]. However, many well-known proposals and experiences of distributed IDSs do not tolerate the presence of malicious or faulty nodes among its nodes. Most of the studies about this issue in the literature do not employ the use of cryptographic mechanisms in communications of IDS nodes, even if this communication depends on the cooperation of nodes that do not belong to the system. The key question in those IDSs is to ensure that applications in MANET environments can always evolve despite of failures, attacks by malicious entities or their own mobility.

Works in this area deal mainly with problems applied to routing protocols in MANETs [4] and some works focus on problems of malicious behavior [3–6]. These works are limited to monitoring the communications in MANETs and to propose IDS models, usually having components which centralize its decisions or information. Moreover, in the literature, the IDSs developed for MANETs usually do not have mechanisms to protect their own information and do not tolerate intrusions into its various components.

☆ This research has been supported by The National Council for Scientific and Technological Development (CNPq), Grants 310671/2012-4 and 307588/2010-6. Paulo Manoel Mafra wishes to thank the Coordination for the Improvement of Higher Level Personnel (CAPES) for the scholarship granting.
☆☆ An earlier version of this paper was published in [1].
* Corresponding author at: Automation and Systems Department, PGEAS, UFSC, Caixa Postal 476, CEP 88040-900, Florianopolis, SC, Brazil.
E-mail addresses: mafra@das.ufsc.br (P.M. Mafra), fraga@das.ufsc.br (J.S. Fraga), santin@ppgia.pucpr.br (A.O. Santin).

0022-0000/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcss.2013.06.011

The communication among entities in MANETs can be monitored by distributed components of an IDS in order to detect faulty or malicious behavior, and to improve security and reliability of such dynamic environments. In this paper we propose a secure and fully distributed IDS model for MANETs. The algorithms presented in this paper define an IDS with functions performed by sets of components (fully distributed executions) and with mechanisms and techniques for preventing and tolerating malicious activities of their own components.

The proposed system is able to deal with various faulty or malicious nodes and mobility without there being interference in the IDS's expected correctness. The proposed IDS, unlike other works, uses cryptographic mechanisms to guarantee the messages authentication between its elements. Moreover, the proposal is able to identify a large number of different attacks or variations of known attacks. The employment of distributed systems and dependability concepts in the IDS design allows modeling, within certain limits, a system less susceptible to restrictions.

This paper is organized as follows. Section 2 describes the related works and defines some parameters and attributes for being used in qualitative analyses. Section 3 introduces the organization of our distributed IDS, defining its components and their roles in the proposed model. We also describe some assumptions of our model and system dynamics. In Section 4 we present the algorithmic base to support the secure IDS. In Section 5 we describe asymptotic costs of the algorithms and results of performed tests in a simulator. Finally, in Section 6, we present our conclusions.

2. Related work

During the last decade many intrusion detection systems were proposed. In 2000, Marti et al. [7] proposed "watchdog and pathrater" mechanisms. The goal of such study was to detect nodes that do not forward packets. Excluding such nodes from routing caches will reduce their effects on MANET routing protocols. Changes were proposed to the Dynamic Source Routing (DSR) protocol [8] for including the watchdog and pathrater mechanisms. The behavior of nodes, in forwarding packets, is observed through the number of dropped packets. If a threshold value for any node is reached, the node is marked as malicious.

In 2003, Zhang, Lee and Huang [9] proposed a distributed and cooperative IDS architecture for MANETs. In that architecture, each node has an IDS agent participating in intrusion detection and response activities. These IDS agents act in a cooperative way by composing a collection of IDSs. The architecture in [9] is merely conceptual and was not implemented.

In [3] Kachirski and Guha proposed an IDS for MANETs using, in each system node, sensor agents that assume functions of data monitoring, decision making and also responding to the malicious activities. The network is divided into clusters in that work. A cluster head is defined in each cluster to route data between clusters. The cluster head is selected based on distances among nodes in the same cluster as well as on the number of neighbors of each node. Monitoring data collected by all sensor agents are merged for detecting intrusions.

Also in 2003, an IDS based on non-overlapping zones named ZBIDS was proposed by Sun, Wu and Pooch [10]. That proposition deals with the problem of cooperation among nodes where the network's nodes are grouped into zones. In ZBIDS, some nodes act as gateways to inter-zone communications. Each node must know its physical location in order to be included into a pre-established zone. For that, the authors suggested that each node must have a GPS locator. The intrusion detection method is based on Markov Chains. Study cases of ZBIDS were simulated using the network simulator GloMoSim [11].

In 2006, another IDS based on clusters was proposed by Ahmed and his colleagues for collaborative detections of intrusions [4]. Cluster compositions are formed and maintained in fixed periods of time. Each cluster has a leader which monitors all the traffic inside its cluster. This leader is also responsible for inter-cluster communications. Message exchanges in the IDS are not protected by the use of cryptography, thus making information and decisions of the IDS easily corrupted under various types of security attacks.

In [12], another model of collaborative IDS was introduced by Razak and Furnell. In the proposal, the node which detects suspect activity requests opinions from its neighbors concerning the detected activity. After analyzing each neighbor's vote, the node makes a decision and informs it to the participating nodes.

Another study introducing an IDS model was presented by Sterne and Lawler [13]. The model is also based on a node hierarchy where the lowest level collects the data and the higher levels correlate the collected data. However, that study goes further than previous cited works. The proposed model allows the detection of several malicious nodes in the IDS's own composition. However, the malicious nodes may only belong to the lower levels of the proposed hierarchy.

In the work of Rajaram and Palaniswami [14], it was described a proposition of an IDS for MANETs which includes a trust-based security protocol, taking into account a MAC-layer mechanism. The protocol provides packet authentication and confidentiality in both routing and link layers of MANETs. In the protocol's first phase, a trust-based scheme for packet forwarding is used to detect and isolate malicious nodes. It uses trust values to favor packet forwarding. When a trust counter value falls below a reliable threshold, the corresponding intermediate node is marked as malicious. In the second phase of the protocol, a link-layer security scheme was developed using the authentication and encryption CBC-X mode to provide security in the IDS messages exchange.

Another distributed cooperative IDS for MANETs was proposed by [15]. That IDS relies on local and global analysis. Each node has a local IDS engine, which runs the network-based IDS *Snort* that monitors the neighbor nodes network activity. Once a node detects a suspicious activity, it starts a distributed IDS algorithm that receives all relevant data about the intrusion detection. In this algorithm, the data received from the IDS engine as well as any other IDS alert message disseminated from other nodes will be analyzed and correlated. If there is enough evidence of intrusion, an IDS alert message will be disseminated by broadcast (flooding). If the IDS alert message is from an untrustworthy node, the IDS message will be ignored. Once a node receives an IDS alert message, intrusion prevention measures are taken and an evaluation of the node's reputation is performed, downgrading the trust level of the involved node. This evaluation is done by all network nodes that received the IDS alert message. Trust management is maintained by watching neighbor node activities: if they rebroadcast alert messages or not.

Recently, it has been proposed by Su [16] a cooperative IDS to detect suspicious activity using neighbor nodes monitoring. When the number of such suspicious packets sent by a node exceeds a threshold, the monitoring node broadcasts an alert message to all nodes in the network. The alert message is first authenticated with the *id* of the monitoring node and carries information indicating the malicious node.

The proposals concerning IDS for MANETs show that the majority of the proposed approaches are capable of identifying few types of attacks or some problems of routing protocols in these networks [6]. Some aforementioned IDS architectures assume a hierarchical component stratification by introducing the idea of clusters [3,4,13]. Sharma, Khandelwal and Singh [17] proposed a new clustering layer for MANETs through an efficient distributed algorithm that uses location metrics for cluster composition. In the work of Ahmed, Samad and Mahmood [4] a time period was established for cluster composition and leaders election through a voting process.

Most of the above mentioned works do not deal with the joining and leaving of nodes or even the node mobility within the composition of the IDS. In related works, we were unable to find convincing results of tests on simulations or implementations that describe the dynamic features of MANETs. Moreover, just some of the cited proposals [9,13,14] considered the use of encryption mechanisms to ensure the properties of authenticity, confidentiality and integrity of messages exchanged between IDS nodes. Proposals which use clustering mechanisms centralize in the cluster leaders operations of the distributed decision and inter-cluster communication. Thus, these leaders must be reliable. Finally, the proposals in the related works do not consider malicious behavior in IDS components or limit it to the lower nodes of the proposed hierarchies.

3. A fault tolerant and secure distributed IDS for MANETs

In this section, it will be described our experiences in developing the model of a fully distributed IDS for MANETs. Communication protocols for MANETs rely on the collaboration of user devices that compound the nodes of such spontaneous arranged networks. As such, in the proposal, it is also assumed a distributed and collaborative nodes environment. However, this collaboration is not limited only to the communications: all network nodes participate in the intrusion detection system.

In addition to the distribution, the proposed IDS model must assume a hierarchical topology in order to meet the diverse IDS functions. Two classes of nodes are distinguished in these spontaneous arranged networks: the "leader nodes" which perform higher level functions such as analysis; and the "collector nodes" that assume the IDS's lower level functionalities like collecting data for future analysis.

In the proposed approach, f denotes a threshold for the anomalies occurrence. While f is not surpassed, the IDS and its distributed functions will continue presenting the correct and expected behavior. Therefore, f value delimits the cluster number of occurrences for failed and corrupted nodes and path disconnections due to nodes' mobility. For maintaining node connectivity in the IDS, we also assume that all the network nodes are connected to at least $2f+1$ neighbors. Thus, the proposed algorithms should be reliable even in presence of at most f failures, intrusions or node disconnections.

3.1. Characterization of the IDS model

The hierarchical topology of the model introduces the idea of clusters, as well as in the works in the literature [3,18,10,4–6]. However, in this proposed model, we consider each cluster as having various *leaders*. These leaders compose what we denominate as *cluster leadership* (or just *leadership*). The leaders are chosen by considering its connectivity and available energy. Any node can be chosen as a leader. The collecting nodes send a summary of the data collected from the MANET to the leaders. The leadership, which defines a cluster domain, is constituted by at least $2f+1$ leaders – without this leaders' quorum, the leadership and the corresponding cluster no longer exist.

The collector nodes constitute the largest part of the cluster components. In order to belong to a cluster, a collector node needs to be connected with at least $f+1$ leaders of a cluster.[1] The data gathered by a collector node always are related to its neighborhood. In other words, collector nodes capture data from each neighbor and store this information in internal tables for subsequent dissemination in the cluster leadership. The information may be, for example, the quantity of received and sent packets by the neighbor node i.

The leader nodes analyze the data sent by the collector nodes of the same cluster. Based on the analysis of these monitoring data, they make their own decisions concerning malicious nodes. These partial decisions are in turn registered in local lists (suspect lists) which are later shared, compared, and synchronized with other leaders of the corresponding

[1] Each cluster node i needs to have at least $f+1$ disjoint routes to the cluster leadership. Disjoint routes are considered paths in the network that start in the considered node and arrive in nodes of leadership. These routes have no shared nodes, except the first one (node i).

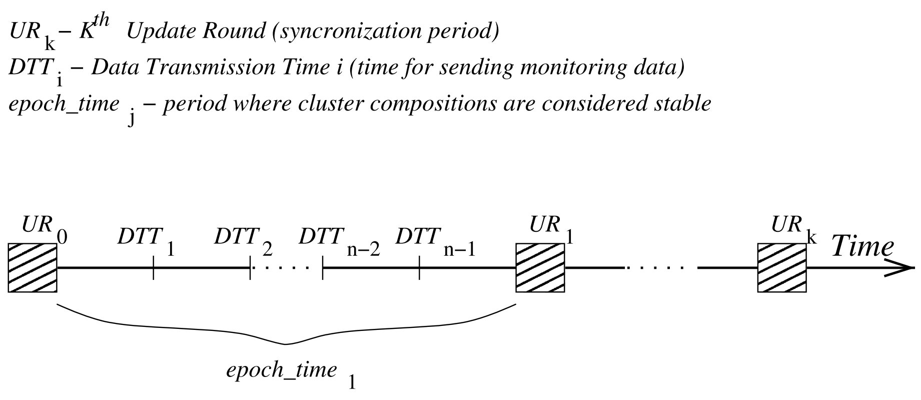
UR_k – K^{th} Update Round (syncronization period)
DTT_i – Data Transmission Time i (time for sending monitoring data)
$epoch\_time_j$ – period where cluster compositions are considered stable
UR_0
DTT_1
DTT_2
DTT_{n-2}
DTT_{n-1}
UR_1
UR_k
Time
$epoch\_time_1$

Fig. 1. Time slots used between update rounds.

cluster. Since the leadership is composed for at least $2f + 1$ leaders, the same decision of $f + 1$ leaders about a node malicious behavior of a given node is enough for excluding it from the IDS composition.

3.2. IDS assumptions

Some assumptions are taken into account in the proposed distributed IDS in order to ensure that our algorithms work as expected:

Assumption 1 *(Connectivity). All cluster nodes have at least $2f + 1$ neighbors and $f + 1$ disjoint routes to the leadership. The leadership composition is not characterized by disconnected graphs.*

Assumption 2. *Each node must have a pair of asymmetric keys to take part in a cluster of a distributed IDS. The node's public key is available as a certificate, signed by a trusted entity of the system.*[2]

Assumption 3. *The leadership and, consequently, the corresponding cluster cease to exist when a leadership has less than $2f + 1$ leaders.*

Assumption 4. *The clocks of the IDS components behave according to a monotonic function.*

Although the clocks may have differences among granularities and drift rates, they do not need to be periodically synchronized. The only assumed requirement in the IDS model is that clocks increase continuously in relation to time.

3.3. The system dynamics

Changes are usual in spontaneous arranged networks (MANETs). Random joining and leaving in the system (*churn*), energy containments – that can cause sporadic failures of components – and node mobility are determinants for changes in the composition and topology of these networks. Therefore, the dynamic characteristics of the MANETs should be taken into account when modeling a distributed IDS for these mobile networks. In other words, the IDS should be self-organized.

For controlling dynamic changes and, therefore, adapt our distributed IDS model to the system changes, we introduced some time notions in the model that help to capture dynamic changes in the system and make the behavior of the algorithms suitable even when changes occur. The time periods that define the synchronization for the model components are: *Epoch*, *Update Rounds* (URs) and *Data Transmission Time* (DTT).

An *epoch* corresponds to a period of time that succeeds other *epoch*; during the *epoch* a cluster "freezes" its composition: changes or requests for changing which may occur in an epoch are not considered in the cluster composition at the same period (see Fig. 1). Possible changes are taken into account in cluster composition in the next epoch.

At the end of each epoch, the cluster components should synchronize with each other. Thus, the *Update Round* (UR) is initiated. URs define a time period for cluster leaders to exchange information in order to update their knowledge concerning the current composition of the cluster. Therefore, changes or changing requests that occurred in the cluster at the last epoch are considered in the subsequent period of synchronization (an update round). Based on changes, the composition for the new epoch is then defined.

IDS decisions taken during the update rounds are based on collected monitoring data. It is supposed that collectors (or sensors) must have sent periodical summaries of collected data to the cluster leadership in the previous epoch. These data

[2] A trusted entity of the system could be an administration committee or any other entity responsible for the spontaneous arranged network.

are sent at the end of a period called *Data Transmission Time* (DTT). Each leader merges and analyzes the data received from collectors at the end of each DTT. In each epoch, monitoring data are collected several times. DTT period is assumed to occur n times in each epoch ($DTT = epoch/n$).

During an epoch, the composition and topology of a cluster does not change. Requests for leaving or joining the cluster in an epoch are only taken into account (and implemented) in the next synchronization period (next UR). Even the effects of faults, mobility and possible malicious actions that occur in an epoch period are only counted during the next UR. A node that tries to leave the cluster in a disorderly manner, during an epoch period, without waiting for the next update round, is considered faulty. All events mentioned above determine a new composition for the new epoch.

4. Algorithmic support for the IDS

We developed a set of distributed algorithms to support an infrastructure that allows the IDS to tolerate up to f malicious nodes.

4.1. Data analysis

The local analyses take into account the monitoring data sent from neighbors of a node, and are implemented based on the system Octopus-IIDS system [19]. Each leader makes the analysis of each network node based on the data that it received from the collectors. If the analyses of $f+1$ nodes point to node i as suspect, then it is considered suspect by the corresponding leader.

Local analyses are done by all the leaders present in a leadership. In the next update round an encrypted and authenticated message is exchanged among leaders. Global decisions are performed in the leadership at each update round by comparing these local analyses. Whether $f+1$ leaders agreeing upon the analysis results, these results will be considered by the leadership. As the leadership is composed by at least $2f+1$ leaders, it eventually decides for a result among the analyses about each cluster node, even in the presence of f malicious leaders.

4.2. Data dissemination

A leader only belongs to a cluster and to the cluster leadership: if it has disjoint routes to at least $f+1$ leaders of this cluster (Assumption 1). In order to reach a cluster leadership, monitoring data must be sent by using a dissemination primitive. For sending these messages to the leadership, it was proposed Algorithm 1 which is based in a reliable multicast presented in [20], adapted from [1].

Algorithm 1 Dissemination primitive

```
1:  Init
2:    $Received_i \leftarrow \{\}$;

3:  upon Disseminate($msg_j$, $Leadership_j$) at $node_j$ do          % node $j$ is any node
4:  for all $i \in Leadership_j$ do                % for all $i$ known and reachable by $j$
5:    send ⟨DISSEMINATION, $msg_j$⟩ to $node_i$
6:  end for

7:  upon receive(⟨DISSEMINATION, $msg_j$⟩) at $node_i$ do{                  % received at node $i$
8:    if (⟨DISSEMINATION, $msg_j$⟩ $\notin Received_i$) then{
9:      $Received_i \leftarrow Received_i \cup \{$⟨DISSEMINATION, $msg_j$⟩$\}$;
10:     Disseminate($msg_j$, $Leadership_i$)               % Dissemination in $Leadership_i$
11:     deliverDisseminate($msg_j$); }}                      % $msg_j$ is locally delivered
```

Algorithm 1 describes the steps for disseminating messages in the leadership. In this algorithm, node i sends the message msg_i (line 5) to leaders who match i's knowledge about cluster leadership ($Leadership_i$). When receives msg_i, each leader k resends the message to her own leadership knowledge ($Leadership_k$, line 10) and then delivers it locally (line 11).

We have assumed that the leadership does not compose disconnected graphs (Assumption 1). Thus, considering timing conditions in message routing and that we also have assumed a limit f for malicious activities and path disconnections (disconnections due to nodes' mobility), a message sent by *Disseminate*() (Algorithm 1) reaches a correct leader. In this way, under these favorable conditions of connectivity, failure and timing, all the leaders of the cluster have a high probability of also receiving this message.

4.3. Cryptographic mechanisms

Leaders which compose the leadership in a cluster use secure channels to exchange information and alerts among themselves. Also, collectors make use of secure channels to send monitoring data to the leadership of the cluster. These secure channels are based on cryptographic techniques for authenticating, encrypting and sending IDS messages (in both cases: collector-to-leader and leader-to-leader communications). It is also assumed that communication for exchanging messages which do not belong to the IDS occurs in an insecure way. Symmetric encryption and session keys are used to encrypt messages in the secure channels.

Each node participating in the cluster must have an asymmetric key pair (asymmetric encryption). The public key of each node is signed by the management committee of the system (trusted entity), generating a certificate (*cred*) for node's authentication. The exchange of certificates allows participants, using the corresponding public keys to establish session keys necessary to secure channels.

4.3.1. Leadership authentications

Cluster leadership should also authenticate its messages in many activities of the proposed model. Leadership authentications are founded on threshold signature scheme (TSS) [21]. In that threshold scheme, a trusted distributor generates n partial keys $(PK_1, \dots, PK_n)$, n verifying keys $(VK_1, \dots, VK_n)$, the group verification key VK and the leadership's public key E_l. The last one is used to validate leadership signatures. Furthermore, the distributor sends those keys to n different leader nodes. Thus, each leader i receives its partial key PK_i and its verification key VK_i. The public key E_l and the group verification key VK are available to any node of the system.[3]

In the process of generating a leadership signature of data, each leader node i generates a partial signature sig_i of *data*. Subsequently, a combiner receives at least t valid partial signatures $(sig_1, sig_2, \dots, sig_t)$[4] and generates the final signature *Sign* of data through these t partial signatures (using the function *threshCombineS*()). A fundamental property of threshold schemes is the impossibility of generating valid signatures from less than t partial signatures. The threshold scheme used for leadership authentications is based on the following primitives [21]:

- **threshSign**$(PK_i, VK_i, VK, data)$: function used by a leader i to generate the partial signature of *data*, i.e., sig_i.
- **threshVerifyS**$(data, sig_i, VK_i, VK, E_l)$: function to verify if the partial signature sig_i, presented by node i, is valid.
- **threshCombineS**$(data, sig[1, \dots, t], E_l)$: this function performs the combination of t valid partial signatures $(sig[1, \dots, t])$ to obtain the leadership signature *Sign*.
- **verifiedSignature**$(data, Sign, E_l)$: function applied to verify the validity of leadership signature *Sign*. It can be used to verify node's credentials.

Algorithm 2 DSignMessage(*data*, *group*)

```
 1: Init
 2:     $buffSign_i, partial\_sigs_i \leftarrow \emptyset$

 3: upon $DSignMessage(msg, Leadership_c)$ at $node_c$ do{                    % coordinator asks for signature
 4:     $Sign_c \leftarrow SignMessage(\langle DSIGN, id_c, tUR_c, msg, cred_c\rangle, D_c)$    % coordinator signs the message
 5:     $dSignMsg_c \leftarrow \langle DSIGN, id_c, tUR_c, msg, Sign_c, cred_c\rangle$
 6:     $Disseminate(dSignMsg_c, Leadership_c)$}

 7: upon $deliverDisseminate(dSignMsg_c)$ at $node_i$: $id_i \in clusterLeaders$ do{
 8:     if $(dSignMsg_c \notin buffSign_i) \wedge (dSignMsg_c.tUR = tUR_i)$ then{
 9:         if $ValidateSign(dSignMsg_c.Sign, dSignMsg_c.cred)$ then{
10:             $buffSign_i \leftarrow buffSign_i \cup \{dSignMsg_c\}$
11:             $sig_i \leftarrow threshSign(PK_i, VK_i, VK, dSignMsg_c.msg)$
12:             $repDsignMsg_i \leftarrow \langle REPDSIGN, id_i, dSignMsg_c.msg, sig_i, VK_i\rangle$
13:             send $repDsignMsg_i$ to $node_c$}}}                          % partial signature is sent to coordinator

14: upon $receive(repDsignMsg_k)$ at $node_c$ do{
15:     if $threshVerifyS(msg, sig_k, VK_k, VK, E_l) \wedge (sig_k \notin partial\_sigs_c)$ then{
16:         $partial\_sigs_c \leftarrow partial\_sigs_c \cup \{sig_k\}$
17:         if $|partial\_sigs_c| \geqslant f + 1$ then{                    % coordinator has at least $f + 1$ partial signatures
18:             $Sign_l \leftarrow threshCombineS(msg, partial\_sigs_c, E_l)$    % $c$ generates the complete signature
19:             return $Sign_l$}}}
```

Algorithm 2 is based on the primitives defined above to perform leadership's distributed signature (*DSignMessage*(*data*, *group*)) of messages. In this algorithm, a leader node coordinates the distributed signature of a message. The coordinator disseminates a message ($dSignMsg_c$, line 6 in Algorithm 2) which proposes the distributed signature of *msg*. Upon receiving $dSignMsg_c$ (line 7), a leader verifies if this message has already been received and if it is not an old one (line 8), aiming to prevent replay attack. Then, it generates the partial signature (sig_i) through the function *threshSign*() (line 11). These partial signatures are sent to the coordinator which executes the function *threshVerifyS*() to check the partial signature validity (line 15 in Algorithm 2). If the partial signature is valid and the number of valid signatures received is greater than $f + 1$ then the coordinator executes the function *threshCombineS*() to generate the signature $Sign_l$ of the message *msg*.

[3] Also, the verification keys $(VK_1, \dots, VK_n)$ are easily obtained from the partial key (PK_i) and the group verification key VK that is available and known by all nodes of the IDS.

[4] In order to ensure the inviolability of the authentication scheme of the cluster leadership, it is necessary that the limit f does not exceed t of the threshold scheme $(f < t < n)$. In this way, we assumed $t = f + 1$ partial keys in the proposed IDS model.

The function *ValidateSign*() used by Algorithm 2 (line 9) validates the leader's signature of a message by using the *cred* (its public key) of each leader.

4.3.2. Updates of partial keys in leadership

It is necessary to perform partial keys' updates before each new epoch, due to the node's joining and leaving from a cluster. As a result, the leader nodes for the next epoch can obtain their new partial keys.

A transition of the current epoch scheme (n,t)-TSS to a scheme (n',t')-TSS corresponding to the new epoch is performed by the partial keys update process, according to the definitions presented by [22]. In the transition process, each leader node of epoch n generates n' shares and proofs of its partial key (for each leader of the new epoch) and sends these shares to leaders of the new epoch n'. Each leader of the new schema should get at least t valid shares to generate its new partial key.

A fundamental requirement for this protocol is that new partial keys $(PK'_1,\dots,PK'_{n'})$ should be compatible. In other words, it is necessary that the shares will be generated by the same set of at least t leaders from (n,t)-TSS (the threshold scheme of current epoch, in the configuration with n components) [22]. Therefore, the proposed algorithm for updating partial keys is based on the following primitives:

- **th_share**(PK_i, j, n'): function used by node i to generate the share $\widehat{k}_{ji}$ from his partial key PK_i. This share is used by j to generate its new partial key.
- **th_generateProof**(i,r): this function is used by node i to create the proof set $\{e_{ri}\colon\ 0 \leqslant r < n'\}$ which is used to certify the validity of the generated parts (shares $\widehat{k}_{ji}$) of partial key.
- **th_verifyShare**$(\widehat{k}_{ji}, \{e_{0i}, e_{1i}, \dots, e_{(n'-1)i}\})$: function used by node j to verify the validity of share $\widehat{k}_{ji}$ generated by i. The proof set $\{e_{ri}\colon\ 0 \leqslant r < n'\}$ is used in this verification.
- **th_combineShares**$(\widehat{k}_{i1}, \widehat{k}_{i2}, \dots, \widehat{k}_{it}, \mathit{qualified})$: function employed by node i to obtain its new partial key PK'_i from the combination of t valid shares and proofs generated from partial keys of the current epoch. The *qualified* list is formed by identifiers (*ids*) of nodes whose shares are correct and may be used in this function to create partial keys of the new epoch.
- **th_generateVK**(PK_i, VK): function used for node i to generate its new verification key VK'_i from its partial key PK'_i and the group verification key VK.

Algorithm 3 is used in the proposed IDS model to generate a new partial key for each leader node of the new leadership (for the new epoch). The purpose of this algorithm is to keep the same pair of asymmetric keys (E_l and D_l) of the leadership while updating the set of private partial keys $(PK'_r\colon\ 1 \leqslant r \leqslant n')$ and verification keys $(VK'_r\colon\ 1 \leqslant r \leqslant n')$ that are used for signing messages in the new leadership. The algorithm starts when a coordinator disseminates a message $\mathit{updateKeys}_c$ (lines from 7 to 9, Algorithm 3). Each leader node i, upon receiving this message, verifies its validity and generates the shares $(\widehat{k}_{ji})$ and proof set $(\{e_{ri}\colon\ 0 \leqslant r < n'\})$ for the new partial keys (PK'_j) for all leaders j of the next epoch $(\mathit{clusterLeaders}^{e+1})$ (lines from 10 to 18). After, node i disseminates to the leadership of the new epoch the shares (encrypted with the corresponding leaders' public keys) and the respective proofs in the signed message shares_i (lines from 19 to 20).

Upon receiving a shares_j message, each node i decrypts it, verifies its validity, and saves it (lines from 22 to 24 and 28). If the share $\widehat{k}_{ij}$ is not valid, the sender (node j) is included in the list $\mathit{suspect\_List}_i$ and the receiving node (node i) disseminates a message $\mathit{invalidKey}_i$ with the received share and proofs, informing that the share and proofs are invalid (lines from 25 to 27).

When a node i receives an $\mathit{invalidKey}_k$ message, it verifies the validity of the indicated share and, if it is invalid, i includes the node that generated the share in its own suspect list (lines from 36 to 37).

The coordinator must indicate which shares should be used in partial keys generation. All partial keys must be composed by the shares that were generated from the same set of leaders. The coordinator indicates which shares should be used through the *qualified* list. Using the shares received from leaders included in the *qualified* list, each leader performs the operation *th_combineShares*() to generate its partial key (PK'_i) for the next epoch. Also, with the new partial PK'_i and VK (group verification key), each leader of the new leadership calculates its new verification key (VK'_i) by using the *th_generateVK*() operation (line 41). Thus, it must have at least $f+1$ correct leaders that participate in the current epoch for performing this algorithm to ensure the necessary shares to compose the partial keys for the next epoch's leaders.

4.4. Nodes' synchronization

In order to deal with the dynamic aspects of the MANET and to ensure the correction of the collecting data for the detection process, it was necessary to define time periods for nodes' synchronization. As the IDS model works essentially with time periods, clock synchronization is not necessary to start operations to keep as close as possible the cluster nodes' view of the IDS status.

Node's synchronization is reached by using Algorithm 4. In this algorithm, the end of each epoch is controlled using local timers (parameter d; line 4, Algorithm 4). When period d expires, a leader node starts sending a *sync* message to the leadership (lines from 4 to 8). Upon receiving a *sync*, node i verifies the message validity, saves it and checks how many *sync*

Algorithm 3 UpdatePartialKeys(id_c, $clusterLeaders^e$, $clusterLeaders^{e+1}$)

```
 1: Init
 2:     $n' \leftarrow 0$                                                    % size of new leadership
 3:     $key\_List_i, V\_List_i, received_i, received\_VK'_i, qualified \leftarrow \emptyset$
 4:     $proofShares_i \leftarrow \emptyset$

 5: upon $UpdatePartialKeys(id_c, clusterLeaders^e, clusterLeaders^{e+1}, Old\_Leadership_c)$ at $node_c$ do{
 6:     $n' \leftarrow |clusterLeaders^{e+1}|$
 7:     $Sign_c \leftarrow SignMessage(\langle UPDATE\_KEYS, clusterLeaders^{e+1}, tUR_c, id_c, n', cred_c\rangle, D_c)$
 8:     $updateKeys_c \leftarrow \langle UPDATE\_KEYS, clusterLeaders^{e+1}, tUR_c, id_c, n', Sign_c, cred_c\rangle$
 9:     $Disseminate(updateKeys_c, Old\_Leadership_c)\}$

10: upon $deliverDisseminate(updateKeys_c)$ at $node_i$: $id_i \in clusterLeaders^e$ do{
11:     if $(updateKeys_c.id = c) \wedge ValidateSign(updateKeys_c.Sign, updateKeys_c.cred)$ then{
12:         if $(updateKeys_c.tUR = tUR_i) \wedge (updateKeys_c \notin received_i)$ then{
13:             $received_i \leftarrow received_i \cup \{updateKeys_c\}$
14:                 for all $id_j \in updateKeys_c.clusterLeaders^{e+1}$ do{
15:                     $\widehat{k}_{ji} \leftarrow th\_share(PK_i, id_j, updateKeys_c.n')$
16:                     $proofShares_i.encrypted\_\widehat{k}_{ji} \leftarrow encrypt(\widehat{k}_{ji}, cred_j.E)\}$
17:                 for all $r$: $0 \leqslant r < updateKeys_c.n'$ do{
18:                     $proofShares_i.e_{ri} \leftarrow th\_generateProof(id_i, r)\}$      % the proof of share is generated
19:                 $shares_i \leftarrow \langle SHARES, id_i, proofShares_i, n', tUR_i, Sign_i, cred_i\rangle$
20:                 $Disseminate(\langle shares_i, id_i\rangle, Leadership_i)\}\}\}$        % node $i$ disseminates shares and proofs

21: upon $deliverDisseminate(\langle shares_j, id_j\rangle)$ at $node_i$: $id_i \in clusterLeaders^{e+1}$ do{
22:     if $(shares_j.tUR = tUR_i) \wedge ValidateSign(shares_j.Sign, shares_j.cert)$ then{
23:         $\widehat{k}_{ij} \leftarrow decrypt(proofShares_j.encrypted\_\widehat{k}_{ij}, D_i)$      % the received shares are decrypted
24:         if $\neg(th\_verifyShare(\widehat{k}_{ij}, proofShares_j.e_{0j}, \ldots, proofShares_j.e_{(n'-1)j}))$ then{
25:             $suspect\_List_i \leftarrow suspect\_List_i \cup \{\langle id_j, tUR_i\rangle\}$
26:             $invalidKey_i \leftarrow \langle INVALID\_KEY, id_i, tUR_i, \langle proofShares_j, id_j\rangle, Sign_i, cred_i\rangle$
27:             $Disseminate(invalidKey_i, Leadership_i)\}$
28:         else $key\_List_i \leftarrow key\_List_i \cup \{\langle\widehat{k}_{ij}, id_j\rangle\}$
29:         if $(id_i = c) \wedge (|key\_List_i| \geqslant f + 1)$ then{
30:             for all $id_k \in key\_List_i$ do{
31:                 $qualified \leftarrow qualified \cup \{id_k\}\}$              % the list of qualified leaders is generated
32:             $qualifLid_c \leftarrow \langle QUALIF\_LID, id_c, tUR_c, qualified, Sign_c, cred_c\rangle$
33:             $Disseminate(qualifLid_c, Leadership_c)\}\}\}$

34: upon $deliverDissiminate(\langle invalidKey_k\rangle)$ at $node_i$: $id_i \in clusterLeaders^{e+1}$ do{
35:     if $(invalidKey_k.tUR = tUR_i) \wedge ValidateSign(invalidKey_k.Sign, invalidKey_k.cred)$ then{
36:         if $\neg(th\_verifyShare(\widehat{k}_{km}, proofShares_m.e_{0m}, \ldots, proofShares_m.e_{(n-1)m}))$ then{
37:             $suspect\_List_i \leftarrow suspect\_List_i \cup \{\langle id_m, tUR_i\rangle\}\}\}\}$

38: upon $deliverDisseminate(qualifLid_c)$ at $node_i$: $id_i \in clusterLeaders^{e+1}$ do{
39:     if $(qualifLid_c.tUR = tUR_i) \wedge ValidateSign(qualifLid_c.Sign, qualifLid_c.cred)$ then{
40:         $PK'_i \leftarrow th\_combineShares(key\_List_i, qualifLid_c.qualified)$
41:         $VK'_i \leftarrow th\_generateVK(PK'_i, VK)\}\}$
```

Algorithm 4 Synchronization(tUR_i) at $node_i$

```
 1: Init
 2:     $received_i, Sync_i \leftarrow \emptyset$                                % buffers of messages
 3:     $\delta \leftarrow$ time()                                           % timer is activated

 4: upon ((time() $- \delta) \geqslant d$) do{                                % at the end of period $d$
 5:     if $(id_i \in clusterLeaders)$ then{
 6:         $Sign_i \leftarrow SignMessage(\langle SYNC, tUR_i, id_i, cred_i\rangle, D_i)$       % signs the sync message
 7:         $sync_i \leftarrow \langle SYNC, tUR_i, id_i, Sign_i, cred_i\rangle$
 8:         $Disseminate(sync_i, Leadership_i)\}\}$                        % $i$ disseminates sync

 9: upon $deliverDisseminate(sync_k)$ do{
10:     if $(sync_k.tUR = tUR_i) \wedge ValidateSign(sync_k.Sign, sync_k.cred)$ then{
11:         if $(sync_k \notin Sync_i)$ then{
12:             $Sync_i \leftarrow Sync_i \cup \{sync_k\}$                      % $sync_k$ is saved in $Sync_i$
13:             if $(|Sync_i| \geqslant f + 1) \wedge (id_i \in clusterLeaders)$ then{
14:                 $Disseminate(sync_k, Leadership_i)$                    % call other leaders to synchronize
15:                 $UR(tUR_i)\}\}\}\}$                                     % starts UR synchronized
```

messages it has received from different leaders (lines from 9 to 12). If the number of *syncs* is greater than $f + 1$, then node i sends again a *sync* message to force those nodes that still have not received $f + 1$ messages to get in the synchronization period (lines from 13 and 14). After this last message is sent, node i starts participating in the Update Round (UR).

The processing during Data Transmission Time (DTT) and Update Rounds (URs) require specific algorithms, presented in the following.

4.5. Data transmission times DDTs

At the end of a data transmission times, collector nodes must send a summary of data collected about neighbor nodes. Algorithm 5 (function *DTT*()) describes the activities of collector and leader nodes at the end of this period.

Algorithm 5 DTT(tUR_i) at $node_i$

```
1: Init
2:     $\gamma \leftarrow$ time()
3:     $Collected\_data_i, suspects\_List_i \leftarrow \emptyset$
4:     $tDTT_i \leftarrow 1$
5:     $timeout \leftarrow p/3$                                   % initial timeout is established in 1/3 of p

6: upon ((time() $- \gamma) \geqslant p$) at $node_i$: $id_i \in Collectors_i$ do{
7:     $monitoring\_data_i \leftarrow \langle MONITORING\_DATA_i, tUR_i, tDTT_i, id_i, data_i, Sign_i, cred_i\rangle$
8:     $Disseminate(monitoring\_data_i, Leadership_i)$
9:     $tDTT_i \leftarrow tDTT_i + 1$
10:    $\gamma \leftarrow$ time()}

11: upon $deliverDisseminate(monitoring\_data_k)$ at $node_i$: $id_i \in clusterLeaders$ do{
12:    if $(monitoring\_data_k.tUR = tUR_i) \wedge (monitoring\_data_k.tDTT = tDTT_i)$ then{
13:        if $ValidateSign(monitoring\_data_k.Sign, monitoring\_data_k.cred)$ then{
14:            $Collected\_data_i \leftarrow Collected\_data_i \cup \{monitoring\_data_k\}$
15:            if $(|Collected\_data_i| \geqslant |Collectors_i| - f) \vee (($time() $- tDTT_i * p) \geqslant timeout)$ then{
16:                $reachedTimeout \leftarrow ($time() $- tTt_i * p)$
17:                for all $(id_n \in Cluster(Leadership_i))$ do{
18:                    $analysis_i^n \leftarrow Data\_Analysis(Collected\_data_i, id_n)$
19:                    if $(analysis_i^n = is\_suspect)$ then{
20:                        $suspects\_List_i \leftarrow suspects\_List_i \cup \{id_n\}\}\}$
21:                $tDTT_i \leftarrow tDTT_i + 1$
22:                $timeout \leftarrow EstimatedTime(timeout, reachedTimeout)$       % a new timeout is calculated
23:                $Collected\_data_i, suspects\_List_i \leftarrow \emptyset\}\}\}\}$
```

In the *DTT*() algorithm, when p period expires, each collector node sends a summary ($monitoring\_data_i$) of collected data to the leadership (lines from 6 to 10, Algorithm 5). Upon receiving a $monitoring\_data_k$ message, the leader node i verifies if the message is not old (by comparing the time values of *tDTT* and *tUR*) and if the message signature is valid. Therefore, if the message is correct, node i saves $monitoring\_data_k$ in the corresponding buffer (lines from 11 to 14). If the number of received messages in the $Collect\_data_i$ buffer is greater or equal to the number of correct collectors ($|Collectors_i| - f$) or if *timeout* has been reached (line 15), then the leader node stores the reached timeout and starts analyzing each cluster node by checking the received data (lines from 16 to 18). If a node is considered suspect by most of its neighbors ($f + 1$ at least), it is inserted into the list of suspects ($suspects\_List_i$) by leader node i (lines from 19 to 20).

With the purpose of estimating an adequate timeout to ensure that most of the $monitoring\_data_k$ messages sent by collectors reach leader nodes, it was adopted an adaptive timeout for adjusting values over time. The adaptive timeout is based on the mechanism used in TCP protocol proposed by Jacobson [23]. This adaptive timeout is implemented by using the *EstimatedTime*() function, in which the observed error from last timeout (the difference between calculated and reached timeouts) is used to calculate the value of the next timeout (an estimated value).

4.6. Update rounds

An Update Round (UR) is defined as a period of time where the cluster nodes update their views about the cluster status. A view is composed by a set of lists (list of malicious nodes, list of cluster leaders, list of cluster collectors, etc.) and represents the IDS state during an epoch. During these URs, the information about nodes participating of the IDS and suspected nodes is shared among cluster leaders. Also, during an UR, joining and leaving nodes are considered for composing the new epoch view.

Algorithm 6 describes the activities executed by leader nodes in UR periods. When a leader node i starts to execute in an UR, it first sends a $list_i$ message with its list of suspect nodes (generated from DTTs of the last epoch) to the remaining leaders through the protocol *Disseminate*() (line 12, Algorithm 6).

At receiving a $list_k$, through primitive *deliverDisseminate*, leader node i verifies if the message is not outdated and is a correct message (by comparing values of epoch counters and verifying message signature, line 14). In case of the message validity confirmed, $list_k$ is then saved in buffer $Group\_suspects_i$ (line 15). If the number of valid messages saved in this buffer is greater or equal to $2f + 1$, then leader node i executes the *identify_Suspects* function, which makes an analysis of each cluster node and reports which are suspects according to at least $f + 1$ leaders. Nodes indicated as suspects are then included into the list of malicious nodes (*malicious_List*, line 17). Thereafter, node i disseminates a message requesting the election of a leader to coordinate the update of the current view of the cluster status.

In line 18, leader i assumes the initial value of the election counter (*int*), which is defined in order to indicate the number of rounds in the process of choosing a coordinator. Under favorable environment conditions, this number of rounds

Algorithm 6 UR(tUR_i)

```
1:  Var
2:      $Leadership_i$
3:      $Collectors_i$
4:      $clusterLeaders$                                   % list of cluster leaders
5:      $cert_i$
6:      $suspect\_List_i$
7:      $tUR_i$                                            % epoch counter
8:      $view$
9:      $count\_msg_i^{int}$

10: upon $UR()$ at $node_i$: $id_i \in clusterLeaders$ do{
11:     $list_i \leftarrow \langle LIST, tUR_i, id_i, suspect\_List_i, Sign_i, cred_i \rangle$
12:         $Disseminate(list_i, Leadership_i)$}

13: upon $deliverDisseminate(list_k)$ at $node_i$: $id_i \in clusterLeaders$ do{
14:     if $(list_k.tUR = tUR_i) \wedge ValidateSign(list_k.Sign, list_k.cred)$ then{
15:         $Group\_suspects_i \leftarrow Group\_suspects_i \cup \{list_k\}$
16:         if$|Group\_suspects_i| \geqslant 2f + 1$ then{          % if received at least $2f + 1$ messages
17:             $malicious\_List \leftarrow identify\_Suspects(Group\_suspects_i, tUR_i)$
18:             $int \leftarrow 1$                         % node $i$ starts election counter
19:             $new\_c_i \leftarrow \langle NEW\_COORDINATOR, tUR_i, id_i, int, Sign_i, cred_i \rangle$
20:             $Disseminate(new\_c_i, Leadership_i)$}}}

21: upon $deliverDisseminate(new\_c_k)$ at $node_i$: $id_i \in clusterLeaders$ do{
22:     if $(new\_c_k \notin count\_msg_i^{int})$ then{
23:         if $(new\_c_k.tUR = tUR_i) \wedge ValidateSign(new\_c_k.Sign, new\_c_k.cred)$ then{
24:             $count\_msg_i^{int} \leftarrow count\_msg_i^{int} \cup \{new\_c_k\}$
25:             if $|count\_msg_i^{int}| \geqslant (\lfloor |clusterLeaders|/2 \rfloor + 1)$ then{   % majority of leaders are reached
26:                 $c \leftarrow elect\_Coordinator(clusterLeaders, int)$      % $c$ is the new coordinator
27:                 $\alpha \leftarrow$ time()                 % timer is activated
28:                 $deadlineUR \leftarrow InitialValue$
29:                 if $(c = id_i)$ then{
30:                     $view \leftarrow new\_view(inactive\_nodes_i, leaving\_nodes_i, cert_i, malicious\_List, tUR_i)$
31:                     $view_c \leftarrow \langle VIEW, tUR_c, id_c, view \rangle$
32:                     $gbroadcast(view_c)$}}}}}          % broadcasts the new view

33: upon $deliver\_gbroadcast(view_c)$ at $node_i$ do{
34:     if $(view_c.tUR = tUR_i \wedge ValidateSign(view.Sign, view.cred))$ then{
35:         if $VerifiedValidity(view_c.view)$ then{
36:             $Viz_i \leftarrow (neighborsUpdate() \cap view.Col \cup view.Lid)$
37:             for all $id_j \in Viz_i$ do{
38:                 $routeCache_i \leftarrow routeDiscovery(id_j)$
39:                 $neighbors\_List_i \leftarrow neighbors\_List_i \cup \{id_j\}$}
40:             $malicious\_List \leftarrow view.black\_List$
41:             $Collectors_i \leftarrow view.Col$
42:             if $(id_i \in clusterLeaders)$ then{
43:                 $Old\_clusterLeaders \leftarrow clusterLeaders$
44:                 $clusterLeaders \leftarrow view.Lid$      % the list of leaders for next epoch is updated
45:                 $Old\_Leadership_i \leftarrow Leadership_i$
46:                 $Leadership_i \leftarrow RouteUpdate(clusterLeaders)$
47:                 $Keys_i \leftarrow UpdatePartialKeys(c, Old\_clusterLeaders, clusterLeaders, Old\_Leadership_i)$
48:                 $count\_msg_i^{int} \leftarrow \emptyset$
49:                 $\alpha \leftarrow \perp$
50:                 $tUR_i \leftarrow tUR_i + 1$
51:                 $Synchronization(tUR_i)$}
52:             else{
53:                 $Leadership_i \leftarrow RouteUpdate(view.Lid)$
54:                 $tUR_i \leftarrow tUR_i + 1$
55:                 $tDTT_i \leftarrow 1$
56:                 $DTT(tUR_i)$}}}}

57: upon ((time() $- \alpha) \geqslant deadlineUR)$ at $node_i$: $id_i \in clusterLeaders$ do{
58:     if $|count\_msg_i^{int}| < (\lfloor |clusterLeaders|/2 \rfloor + 1)$ then{
59:         $reachedDeadline \leftarrow$ (time() $- \alpha)$
60:         $deadlineUR \leftarrow EstimatedTime(deadlineUR, reachedDeadline)$   % new deadline is estimated
61:         $int \leftarrow int + 1$
62:         $count\_msg_i^{int}, count\_msg_i^{int-1} \leftarrow \emptyset$
63:         $new\_c_i \leftarrow \langle NEW\_COORDINATOR, tUR_i, id_i, int, Sign_i, cred_i \rangle$
64:         $Disseminate(new\_c_i, Leadership_i)$
65:         $\alpha \leftarrow$ time()}}
```

is limited for having a new coordinator chosen ($int \leqslant f + 1$). When running the lines 19 and 20 of Algorithm 6, a leader node i prepares and disseminates a new_c_i message to the current leadership ($clusterLeaders^e$). This message requests the election of a new coordinator and carries out the current values of epoch (tUR) and election (int) counters and i's credential ($cred_i$) and signature ($Sign_i$).

When node i receives a new_c_k message requesting for a new coordinator, it first checks the message validity. If new_c_k is correct and current, node i stores this message in the buffer $count\_msg_i^{int}$ (lines from 22 to 24, Algorithm 6). If the reception number of new_c_k messages ($|count\_msg_i^{int}|$), reaches the limit $\lfloor|clusterLeaders^e|/2\rfloor + 1$, then it ensures the necessary quorum for election (knowing that $\lfloor|clusterLeaders^e|/2\rfloor + 1 \geqslant f + 1$). Given this condition, a leader is elected coordinator (lines 25 and 26). This election is based on identifiers' order of leader nodes in $clusterLeaders^e$. In line 27, a timer for controlling timeouts is activated, thus establishing a time period in which the coordinator must fulfill its attributions during the Update Round (UR).

The elected coordinator processes the joining and leaving requests from cluster nodes and generates a new cluster view including nodes which have solicited joining in the current epoch, as well as the removal of malicious nodes and those which have requested leaving during the current epoch (lines from 29 to 30). The new cluster view is thus sent in a signed message by using the communication primitive *gbroadcast* (which provokes a flooding) to all network nodes, including nodes in other clusters (lines from 31 to 32). The function $new\_view()$ is used to generate a new cluster view which will determine the node composition for the next epoch (line 30, Algorithm 6).[5]

Upon receiving the message with the new cluster view of the network ($view_c$), each cluster node verifies the correction of the message (line 34). After, line 35, the function which checks the validity of the received $view_c$ is executed. Once the validity of the received $view_c$ is confirmed, it is implemented (through the updating of the local lists on the receiving node). Next, the node updates its list of neighbors, line 36 (using the $neighborsUpdate()$ function). In the following, the neighbor routes are established (lines from 37 to 39). The primitive $routeDiscovery()$ is part of the network support offered in the DSR protocol. The final list obtained ($neighbors\_List$), in line 39, corresponds in fact to the set of $2f + 1$ neighbors to which the node has established routes. Malicious nodes list ($malicious\_List$) and collectors list ($Collectors_i$) are then updated based on the decision obtained through messages exchanged during UR, it is centered on the chosen coordinator and on the view it disseminated (lines 40 and 41).

Each leader node i begins to instantiate the new cluster view, saving the list of leaders ($clusterLeaders^e$) from the current view in *Old_clusterLeaders*. Thus, a new list of leaders ($clusterLeaders^{e+1}$ for the next epoch) is uploaded based on the received view (lines 43 and 44, Algorithm 6). The routes from leader i to the leadership in the current view ($Leadership_i$) are saved (line 45), and the function $RouteUpdate()$ is executed using as argument the next epoch's list of leaders ($clusterLeaders^{e+1}$), line 46. This function defines the new routes from the leader node i ($Leadership_i$) to the new leadership ($clusterLeaders^{e+1}$).

The leader node executes the $UpdatePartialKeys()$ function (Algorithm 3), which updates partial keys for the new leadership, line 47, Algorithm 6. The arguments to this function call are: the coordinator (c), the current leadership list (i.e., $clusterLeaders^e$ saved in *Old_clusterLeaders*), the next leadership list ($clusterLeaders^{e+1}$), and the leadership routes in the current epoch (saved in $Old\_Leadership_i$). Then, the buffer $count\_msg^{int}$ is initiated, the timeout is deactivated, and the epoch counter (tUR_i) is incremented in order to point out to the new epoch (lines from 48 to 50). Finally, leader i executes the *Synchronization* function (Algorithm 4) in order to start a new epoch. Collector nodes conclude their updates for the new cluster view, creating their routes towards new leadership, determining the value of their epoch counters for the new epoch (tUR_i), initiating the counter of data transmission time which is used for sending data collections ($tDTT_i$), and finally calling the $DTT()$ function (lines from 53 to 56, Algorithm 5).

The thread which is defined between lines 57 and 65 of Algorithm 6, controls the coordinator timeout that is used to limit the activities of calculating and coordinating of the new view instantiation. This thread is executed among leaders of the current epoch. A leader node i defines the new timeout value for the *deadlineUR* (line 60) executing the $EstimatedTime()$ function with the used timeout (*deadlineUR*) and the period reached in the timer (*reachedDeadline*) as arguments. The election counter int is incremented to the new election round and the buffers are reset (line 62). Further, the leader node disseminates a new message requesting a new coordinator, line 64. Finally, the timer used in the timeout control is initiated, line 65.

The *deadlineUR* also establishes limits for controlling bad behavior of the current coordinator. For example, if the coordinator does not generate a new view or if its validation fails, or even if the coordinator suddenly leaves the network, the remaining leaders can request the election of another leader as coordinator and the IDS continues to operate the new epoch's instantiation.

A node does not instantiate a view established by the coordinator when the $view_c$ message (lines from 33 to 56) or the request for a new coordinator election (lines from 21 and 32 of Algorithm 6) did not reach the considered node. If any of these situations happen, the node would not be considered by the remaining nodes (its tUR will be different from the remaining nodes). When this node receives the view of another epoch with epoch count larger than tUR, it will consider that it is outdated and need to send a join request to the IDS composition.

[5] The new cluster view includes information about collectors, leaders, and malicious nodes lists, as well as the cluster's certificate lists and the corresponding value of the epoch counter (tUR). These data will be applied to all cluster nodes and will be valid during the next epoch.

4.6.1. Initial update round (UR_0)

The initial update round (UR_0) is the period in which the system is composed: the cluster and its leadership are defined, and the IDS starts to operate. In this phase, with the assistance of the network administrator, the public key of the leadership is created, as well as each leader's partial keys. In the final stage of this period, the data transmission time ($tDTT$) and the update rounds (tUR) counters are initialized in each node of the cluster.

4.7. Route discovery

The discovery and updating of routes are essential for the correct operation of the IDS and communications in the MANET described in our model. The DSR routing protocol [8] was adopted for routing messages in the system. DSR is an on-demand routing protocol widely used in MANETs with a local routing cache. We adapted the DSR algorithm for updating local route caches by inserting $f+1$ disjoint routes to the leadership in each demand to routes for the leadership.

Algorithm 7 RouteUpdate(*clusterLeaders*) at *node*$_i$

```
1: Init
2:     $Routes_i \leftarrow \emptyset$
3:     $RouteCache_i \leftarrow \emptyset$
4:     $LeadersList_i \leftarrow \emptyset$

5: upon $RouteUpdate(clusterLeaders)$ at $node_i$ do{
6:     $LeadersList_i \leftarrow clusterLeaders$
7:     while $LeadersList_i \neq 0$ do{
8:         $id_l \leftarrow random(LeadersList_i)$                    % a leader is randomly chosen
9:         $RouteCache_i \leftarrow RouteDiscovery(id_l)$             % gets all routes from $i$ to $l$
10:        $LeadersList_i \leftarrow LeadersList_i \setminus \{id_l\}$    % removes $l$ from $LeadersList_i$
11:        for all $r_{il} \in RouteCache_i$ do{
12:            if $disjointRoute(r_{il}, Routes_i)$ then{
13:                $Routes_i \leftarrow Routes_i \cup r_{il}$
14:                if $id_i \in Collectors_i$ then{
15:                    if $|Routes_i| \geqslant f+1$ then{
16:                        $RouteCache_i \leftarrow Routes_i$           % updates DSR local cache
17:                        return $Routes_i$}}                   % returns disjoint routes from collector $i$ to leadership
18:                break }}}                                     % a disjoint route from $i$ to $l$ was found
19:    $RouteCache_i \leftarrow Routes_i$                             % updates DSR local cache
20:    return $Routes_i$}
```

The adapted protocol is presented in Algorithm 7 where all existing routes from node i to a leader l are obtained through the Route Discovery mechanism of the DSR. If the node i which is running this algorithm is a leader, it attempts to get the maximum possible number of disjoint routes to the leadership ($f+1$ at least). Otherwise (node i is a collector), it gets just $f+1$ disjoint routes.[6] The *Disseminate*() algorithm depends on these disjoint routes to spread a message in the leadership.

4.8. IDS nodes joining and leaving

Node identification process in the network should be secure enough for not allowing the creation of multiple identities to nodes, thus avoiding attacks like Sybil [24]. In the proposed architecture, node identification is based on certificates. A certifying authority (CA) known and trusted by the network nodes, generates a certificate with the public key of a node i when it joins the system ($cred_i$).[7] The CA role in our model is assumed by an entity that manages the ad hoc network.[8] With this model, we can identify the user and associate her to her equipment. Therefore, it cannot have users with multiple devices on the network.

Algorithm 8 presents the steps involved in joining a new node in a network cluster. When joining, a new node i needs to do the prospection of 0 10 0 42 its neighbors (through the function *neighborsUpdate*()), building its list of neighbors and sending a request for joining the network. This request is done by using the message REQIN ($reqin_i$), sent to each neighbor (lines from 6 to 13, Algorithm 8). When a neighbor j receives a message $reqin_i$, it verifies the message signature and if the sender (node i) is not in the malicious list. If the verification succeeds, it disseminates the message to the leadership and saves a copy of it in the buffer of *inactive_nodes* (lines from 14 to 19).

The dissemination of this message should reach the cluster leadership informing its identification (the identity id_i of node i) and its credential (its public key in a certificate format). The new node i will join the IDS composition as it receives the new view of the leadership. The view received (line 20, Algorithm 8) has verified its validity. If it succeeds, the cluster view is instantiated assuming the role assigned to node i by the *leadership*[9] (lines from 20 to 24). After that, the new node

[6] Collector nodes need to reach just one correct leader that will disseminate the message in the leadership using Algorithm 1 (*Disseminate*() function).
[7] These certificates must have the public key and attributes of the user and also the MAC address of his mobile device.
[8] This certifying entity will not necessarily need to be an official PKI. It may be a system administration committee or an administrator, etc.
[9] The role of a new node is defined based on the needs for leaders to maintain the cluster, node's connectivity and available energy.

Algorithm 8 Joining of node i into the cluster

```
1: Var
2: $clusterLeaders_i$
3: $Collectors_i$
4: $role_i \leftarrow null$
5: Init
6: $Viz_i \leftarrow neighborsUpdate()$
7: for all $id_j \in Viz_i$ do{
8:     $routeCache_i \leftarrow routeDiscovery(id_j)$
9:     $neighbors\_List_i \leftarrow neighbors\_List_i \cup \{id_j\}$}
10: $Sign_i \leftarrow SignMessage(\langle REQIN, id_i, cred_i \rangle, D_i)$
11: $reqin_i \leftarrow \langle REQIN, id_i, Sign_i, cred_i \rangle$
12: for all $id_j \in neighbors\_List_i$ do{
13:     $send\ reqin_i\ to\ id_j$}
14: upon receive$(reqin_i) \vee deliverDisseminate(reqin_i)\ at\ node_j$ do{
15:     if $ValidateSign(reqin_i.Sign, reqin_i.cred)$ then{
16:         if $(id_i \notin malicious\_List)$ then{
17:             $Disseminate(reqin_i, Leadership_j)$
18:             if $(id_j \in Leadership_j) \wedge (reqin_i \notin inactive\_nodes_j)$ then{
19:                 $inactive\_nodes_j \leftarrow inactive\_nodes_j \cup \{reqin_i\}$}}}}
20: upon $deliver\_gbroadcast(view_c)$ at $node_i$ do{
21:     if $ValidateSign(view_c.Sign, view_c.cred)$ then{
22:         $malicious\_List \leftarrow view_c.black\_List$
23:         $Collectors_i \leftarrow view_c.Col$
24:         $clusterLeaders \leftarrow view_c.Lid$
25:         $Viz_i \leftarrow (neighborsUpdate() \cap view_c.Col \cup view_c.Lid)$
26:         for all $id_j \in Viz_i$ do{
27:             $routeCache_i \leftarrow routeDiscovery(id_j)$
28:             $neighbors\_List_i \leftarrow neighbors\_List_i \cup \{id_j\}$}
29:         $Leadership_i \leftarrow RouteUpdate(view_c.Lid)$
30:         $tUR_i \leftarrow view_c.tUR + 1$
31:         if $id_i \in clusterLeaders$ then{
32:             $role_i = leader$
33:             $\alpha \leftarrow \perp$
34:             $int \leftarrow \perp$
35:             $Synchronization(tUR_i)$}
36:         else if $(id_i \in view.Col)$ then{
37:             $role_i = colector$
38:             $DTT(tUR_i)$}}}
```

updates its list of neighbors and routes to the leadership, its counters of data transmission time ($tDTT$) and update round (tUR), executes the synchronization algorithm and starts a data transmission time (lines from 25 to 38).

Algorithm 9 Leaving of node i

```
1: upon $leaving(tUR_i)$ do{
2:     $Sign_i \leftarrow SignMessage(\langle REQOUT, id_i, tUR_i, cred_i \rangle, D_i)$
3:     $reqout_i \leftarrow \langle REQOUT, id_i, tUR_i, Sign_i, cred_i \rangle$
4:     $Disseminate(reqout_i, Leadership_i)$}                          % $i$ disseminates $reqout_i$
5: upon $deliverDisseminate(reqout_i)$ at $node_k$: $id_k \in clusterLeaders$ do{
6:     if $(reqout_i.id \notin malicious\_List) \wedge (reqout_i.tUR = tUR_k)$ then{
7:         if $(reqout_i \notin inactive\_nodes_k)$ then{
8:             $leaving\_nodes_k \leftarrow leaving\_nodes_k \cup \{reqout_i\}$}}}    % node $k$ waits for a leadership response during next UR
9: upon $deliver\_gbroadcast(view_c)$ at $node_i$ do{
10:     if $ValidateSign(view_c.Sign, view_c.cred)$ then{             % signature and credential of $c$ are valid
11:         if $VerifiedValidity(view_c) \wedge (view_c.tUR = tUR_i)$ then{
12:             $node_i\ leaves\ the\ system$}}}
```

Algorithm 9 presents the steps involved in the procedure for a node leaving the IDS. In this algorithm, the node i which wants to leave the network, disseminates a message REQOUT ($reqout_i$) to the cluster leadership (lines from 2 to 4, Algorithm 9). When a leader k receives the message, it verifies if the sender is not a malicious node and if the message is not old (outdated), adding the message to a list of leaving nodes ($leaving\_nodes$) (lines from 6 to 8). In the next UR, when the node wanting to quit the network receives the new view, it checks the view signature and can then leave the system. The IDS's joinings and leavings are always processed during periods of Update Rounds (URs). If a node leaves the IDS disrespecting this procedure, in the middle of an epoch, the system will consider it as a malicious node.

Table 1
Communication costs.

Joining of a node in the network	$(l^2 - l)/2$
Data Transmission Time (DTT)	$nl^2 - l^3$
Update Round (UR)	$n^2 + 6l^3 + ln$

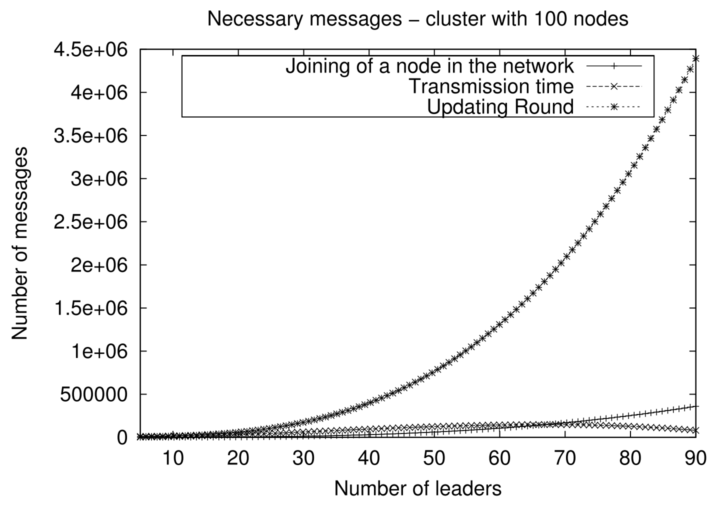

Fig. 2. Tradeoff between demanding messages and amount of leaders.

5. Results and corresponding analysis

This section analyzes the model's communication costs and presents results obtained based on simulation tests.

5.1. Communication costs

The communication costs of the proposed algorithms depend mainly on the size of the cluster and its leadership. As such, the costs were computed in terms of issued messages for each algorithm. Table 1 summarizes these costs, in which: n is the number of nodes and l is the number of leaders in the cluster. We adopted the values of the limit f being defined by $f = (l/2) - 1$. In other words, the number of malicious or faulty nodes is one half of the cluster's leadership size minus one; this determination establishes the limits for the IDS's proper operation.

In the *joining* procedure (Algorithm 8), a node sends a request to its neighbors ($f + 1$ messages). Upon receiving the request, each neighbor node retransmits the request to the other leaders of the leadership using the protocol *Disseminate*(). The cost of this protocol is given by $D = (l^2)$. Thus, the total cost for a node joining the network is then $D * (f + 1)$ or $(l^3 - l^2)/2$, where $f = (l - 1)/2$. In order to calculate the costs for *Data Transmission Time* (DTT), each collecting node $(n - l)$ sends a message with monitoring data to the leaders using the *Disseminate*() primitive. Thus, the total costs are at the order of $(n - l) * (l^2)$ or $(nl^2 - l^3)$.

To calculate the costs of an *update round*, it is necessary to consider that each leader disseminates, to the leadership, a message containing the result of its analysis concerning each of the cluster's node. The message cost to do it is around $l * D$ or l^3. Also, when receiving this message, each leader disseminates another message demanding a new coordinator, with costs of $l * D$ or l^3. In the sequence, the coordinator generates the new view and sends it to all cluster nodes through a broadcast that generates messages at the order of n^2. Next, each node i updates its routes leading to the new leadership with costs of ln. Algorithm 3 (*UpdatePartialKeys*()) is then executed by the leaders (with costs of $l * 4D$ or $4l^3$). Thus, the total cost of Algorithm 6 (*UR*()) is then $n^2 + 6l^3 + ln$ messages.

Fig. 2 presents an example of estimated costs (in terms of messages) to the joining operation, data transmission time and update round. Considering the communication costs obtained, it is always possible to find more adequate values for the number of leaders in a cluster. In a 100-node MANET arrangement, merely one cluster, the adequate ratio between leaders and collectors should be approximately 20 leaders per 100 nodes. In Fig. 2, one can observe that when more than 30 leaders are present in the MANET, the number of necessary messages during the *Update Round* (UR) grows exponentially. A similar behavior is not noticed when one considers the algorithms of *Data Transmission Time* (DTT) and joining operation. These algorithms, in terms of the size assumed for the leadership, are not dependent on message costs like the *update round*, for instance.

Whenever possible, in the proposed algorithms, it is used the primitive *Disseminate*() for sending messages to the leadership, with a cost of l^2 for spreading messages to the leadership. One could use the broadcast primitive, which has complexity of n^2 messages, but such cost would always be greater when *Disseminate*() is used, since $n > l$.

Table 2
Tests' results.

Observed feature	0% mal.	10% mal.	20% mal.	30% mal.	20% surp. f	30% surp. f
Loss of messages	6.52%	13.56%	20.63%	29.67%	29.69%	35.43%
Disseminated *msg*	100%	100%	100%	99.99%	99.68%	98.03%
Detection rate	100%	100%	100%	96.03%	95.08%	91.26%

Table 3
Tests' results without malicious nodes.

Observed feature	5% lea.	10% lea.	15% lea.	20% lea.
Loss of messages	7.03%	7.54%	14.31%	15.01%
Disseminated *msg*	100%	100%	93.66%	92.06%
Detection rate	100%	100%	97.13%	97%

Table 4
Tests' results with 10% of malicious nodes.

Observed feature	5% lea.	10% lea.	15% lea.	20% lea.
Loss of messages	15.91%	16.55%	21.63%	23.12%
Disseminated *msg*	100%	100%	93.37%	93.05%
Detection rate	100%	100%	97.07%	96.73%

5.2. *Implementation and tests*

For verifying the features of the proposed IDS model, we developed simulation tests for their propositions. Also, some tests were developed for verifying the thresholds that the IDS should deal with.

The implementations and simulations were performed using the simulator Omnet++ (version 4.1) with a Mixim (version 1.1) wireless network module, considering a 300×400 meter rectangular area. The period for data transmission time was established as 60 seconds, and each epoch's time was set at 300 seconds. Those parameters were adopted based on tests executed for verifying the simulator's limitations. Based in these values, the IDS is not saturated with messages during data transmission times and update rounds. The total simulation time was limited to 6010 seconds (corresponding to 20 URs). Such total time was considered as enough for observing the proposed model's behavior. Node's mobility rate was assumed at 2.0 mps,[10] which is similar to the parameters employed in [25].

Malicious activity was defined to be nodes which perform random actions on messages being routed. In other words, sometimes malicious nodes take part by correctly forwarding messages in the MANET routes and, sometimes, they do not cooperate in such task. This behavior was simulated by following a uniform probability distribution, in which 80% of the messages were not relayed by the malicious nodes. We chose this type of behavior because it is more difficult to detect it than a node which simply discards all the messages that it receives, or than nodes which retransmit only to malicious nodes (routing messages merely to a list of nodes which are also corrupted).

In the aforementioned tests, first, it was measured the rate of message losses in collector-to-leader and leader-to-leader communications (*Loss of messages*). The corresponding results showed the MANET's behavior in terms of failures. Next, we quantified the amount of message delivery in the leadership when the *Disseminate*() protocol is used (Disseminated *msg*). In other words, we observed whether a message sent using the sender's leadership knowledge (i.e., the set of disjoint routes of the sender node) is delivered to the leadership when it reaches at least one correct leader. Finally, we verified the detection rate of malicious nodes in the MANET (*Detection rate*).

Three tests were performed to get measurement data. The first test used 100 nodes, while the second and third tests began with 100 nodes, however, the rates of nodes' joining and leaving are considered to provoke dynamic changes in the MANET's node composition.

In the first test, three distinct scenarios regarding malicious nodes were observed, (i) without malicious nodes, (ii) considering 10%, 20%, and 30% of malicious nodes but respecting the f limit, and (iii) assuming 20% and 30% of malicious nodes, but keeping the leadership at 10% (the number of malicious nodes do not respect the f limit). The results of these tests are presented in Table 2. The mentioned tests showed that even with the loss of messages caused by MANET problems, node's mobility, or malicious activities, the message dissemination rate among leaders remained very close to 100% (even when the f limit was surpassed), yielding a high malicious activity detection rate. One can notice that even when the f limit supported by the proposed model is exceeded (Table 2, 20% surp. f and 30% surp. f), the system continues to work, with a very high message delivery rate and a very low false negative detection rate.

In the second and third simulated tests, two distinct scenarios were observed: without malicious nodes and considering 10% of the malicious nodes (respecting the $2f+1$ leader limit in leadership but do not respecting the $2f+1$ neighborhood limit) respectively. In these tests, we adopted a fixed number of nodes joining at rate of 10% on each update round, but making the leaving rate (lea.) equal to 5%, 10%, 15%, and 20%. The test results, presented in Table 3 and Table 4, showed the IDS's behavior regarding the nodes joining and leaving.

[10] Meters per second.

Evaluating the tests we were able to observe, as expected, that IDS message loss was greater with 10% malicious nodes than without any malicious node. However, the rate of messages spread throughout the leadership was similar (considering 10% maliciousness and without such activity). We also observed that with a leaving rate greater than its joining rate, some messages were not widespread among the leadership. Such situation may be explained by a low MANET density of nodes over a time. In such a case, the nodes have fewer connections. As a consequence, some messages may not reach any correct leader (the probability of disseminated messages decreases with MANET density reduction). Additionally, we observed that the detection rate follows the observed behavior for message dissemination rate. Therefore, when some messages are not spread throughout leadership, the detection system will not perform the correct detection. Finally, we could consider the set of tests produced satisfactory with results which evidence the limits and effectiveness of our proposals.

6. Concluding remarks

The node's mobility associated with the absence of a communication infrastructure, demands collaborative communication protocols for spontaneously arranged networks like MANETs. In such environments, it is hard to maintain and to guarantee security at message levels that are relayed using MANET's nodes (users' devices). Malicious activities may prejudice service access or even disrupt the MANET's composition.

This paper focuses on the development of an IDS model for dynamic environments. In the proposed model, we have centered our efforts on identifying malicious activity in MANET communications and in services available on this network. A number of distributed algorithms were introduced in order to support this dynamic and distributed IDS model, which continues to work even under a certain malicious activity rate.

The IDS described in this paper, unlike other related models, may adapt itself to MANET's topological and nodes composition changes. To deal with the dynamic features of the MANET, the deployment of distributed IDS was divided into time periods called *epoch*, where the IDS status is considered unchanged. Moreover, nodes requests for joining and leaving the IDS and nodes failures are only processed during synchronization periods, identified as *update round*.

The proposed IDS model innovates by considering intrusion and fault occurrences in their own components. Which means the model was conceived taking into account that malicious behavior can be present in certain IDS's components.

This proposal is centered on a hierarchical IDS model for malicious behavior detection in MANETs. But unlike other approaches which make use of hierarchies, in our approach, there is no dependence on a single node for developing the IDS functionality. Thus, the proposed IDS is able to identify and circumvent malicious nodes, since each MANET node is connected to at least $f+1$ correct neighbors (nodes that have no malicious activity). Furthermore, each cluster view needs to have a leader composition (cluster *leadership*) of at least $f+1$ correct leaders.

The tests simulation showed that in some scenarios, even when the f limit is exceeded by the number of faulty and malicious nodes, the IDS continues to work correctly. Thus, the simulation developed based on the model showed the viability of the proposal.

An important aspect should be noticed in the proposal's assumptions, that while the f limit for faulty or malicious nodes is not exceeded, the proposed model behaves with certain determinism in IDS detections and decisions (detections were around 100%, without false positives). Unfortunately, we were not able to find other related works in IDS for MANET with simulation or real environment tests' measurements for comparing with the results obtained in our proposal.

References

[1] P.M. Mafra, J.S. Fraga, A.O. Santin, Distributed algorithms for the creation of a new distributed IDS in MANETs, in: Proceedings of the 5th International Conference on Internet and Distributed Computing Systems, IDCS'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 29–42.

[2] D. Djenouri, L. Khelladi, A. Badache, A survey of security issues in mobile ad hoc and sensor networks, IEEE Communications Surveys and Tutorials 7 (2005) 2–28.

[3] O. Kachirski, R. Guha, Effective intrusion detection using multiple sensors in wireless ad hoc networks, in: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03), Track 2, vol. 2, IEEE Computer Society, 2003, pp. 1–8.

[4] E. Ahmed, K. Samad, W. Mahmood, Cluster-based intrusion detection architecture for mobile ad hoc networks, in: Asia Pacific Information Technology Security Conference, AUSCERT2006, Australia, 2006, pp. 1–11.

[5] L. Bononi, C. Tacconi, Intrusion detection for secure clustering and routing in mobile multi-hop wireless networks, International Journal of Information Security 6 (6) (2007) 379–392, http://dx.doi.org/10.1007/s10207-007-0035-9.

[6] A. Nadeem, M. Howarth, Protection of manets from a range of attacks using an intrusion detection and prevention system, Mobile Computing technologies of Telecommunication System Journal (2011) 1–12.

[7] S. Marti, T.J. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, in: The 6th ICMCN, 2000, pp. 255–265.

[8] D.B. Johnson, D.A. Maltz, J. Broch, DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, Addison–Wesley, 2001.

[9] Y. Zhang, W. Lee, Y. Huang, Intrusion detection techniques for mobile wireless networks, Wireless Networks 9 (5) (2003).

[10] B. Sun, K. Wu, U. Pooch, Alert aggregation in mobile ad hoc networks, in: ACM Workshop on Wireless Security (WiSe), 2003, pp. 69–78.

[11] X. Zeng, R. Bagrodia, M. Gerla, GloMoSim: a library for parallel simulation of large-scale wireless networks, in: The 12th Workshop on Parallel and Distributed Simulations, Banff, Canada, 1998, pp. 154–161.

[12] S.A. Razak, S.M. Furnell, Friend-assisted intrusion detection and response mechanisms for mobile ad hoc networks, Ad Hoc Networks 6 (7) (2008) 1151–1167.

[13] D. Sterne, G. Lawler, A dynamic intrusion detection hierarchy for manets, in: Sarnoff Symposium, 2009, IEEE, 2009, pp. 1–8.

[14] A. Rajaram, S. Palaniswami, Malicious node detection system for mobile ad hoc networks, IJCSIT International Journal of Computer Science and Information Technologies 2 (1) (2010) 77–85.

[15] S. Mutlu, G. Yilmaz, A distributed cooperative trust based intrusion detection framework for manets, in: The Seventh International Conference on Networking and Services, Venice, Italy, 2011, pp. 292–298.

[16] M.-Y. Su, Prevention of selective black hole attacks on mobile ad hoc networks through intrusion detection systems, Computer Communications 34 (1) (2011) 107–117.
[17] K. Sharma, N. Khandelwal, S.K. Singh, New proposed classic cluster layer architecture for mobile adhoc network (cclam), International Journal of Computer Science and Security 6 (2012) 94–102.
[18] D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C. Tseng, T. Bowen, K. Levitt, J. Rowe, A general cooperative intrusion detection architecture for manets, in: Proceedings of the 3rd IEEE IWIA, 2005, pp. 57–70.
[19] P.M. Mafra, V. Moll, J.S. Fraga, A.O. Santin, Octopus-IIDS: An anomaly based intelligent intrusion detection system, in: ISCC, Italy, 2010, pp. 405–410.
[20] T. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM 43 (2) (1996) 225–267.
[21] V. Shoup, Practical threshold signatures, in: B. Preneel (Ed.), Advances in Cryptology – EUROCRYPT 2000, in: Lecture Notes in Computer Science, vol. 1807, Springer, 2000, pp. 207–220.
[22] T.M. Wong, C. Wang, J.M. Wing, Verifiable secret redistribution for archive systems, in: Proceedings of the 1st International IEEE Security in Storage Workshop, 2002, pp. 94–105.
[23] V. Jacobson, Congestion avoidance and control, SIGCOMM Computer Communication Review 18 (4) (1988) 314–329, http://dx.doi.org/10.1145/52325.52356.
[24] A. Vora, M. Nesterenko, S. Tixeuil, S. Delaët, Universe detectors for sybil defense in ad hoc wireless networks, arXiv:0805.0087.
[25] J.-H. Böse, Atomic transaction processing in mobile ad-hoc networks, Master's thesis, Freie Universität, Berlin, 2009.