Agent Security in MANET

6.1 Introduction

Due to its inherent dynamic nature MANET is more prone to security threats. Its open and shared communication medium invites new kinds of vulnerabilities that are not present in its wired counterpart. When a mobile agent visits a host site, it runs as a process in that host platform and hence is dependent on that platform. So securing an agent while it is visiting a host is challenging. However protecting mobile nodes and the agents they deploy is crucial for providing a reliable service. Consequently securing the nodes and hence the agents from malicious entities not only enhances MANET reliability but also improves the effective performance of the distributed application.

As has been pointed out in [88] security of a mobile agent paradigm emphasizes on protecting and preventing a mobile agent from malicious hosts’ attacks by applying cryptographic functions. But in an environment which typically undergoes continuous topology changes thereby disrupting flow of information over the existing paths, these countermeasures prove to be insufficient. Sometimes the best way to protect an agent in MANET is to prevent them from visiting/migrating through a malicious host.

Reputation is a fundamental concept that involve interaction between mutually distrust ing parties. It is established by exchanging trust opinions among the nodes. Trust can be defined as the degree of belief about the behaviour of mobile nodes. Reputation based trust model is found to suit the purpose of securing MAS and hence MANET very effectively. Some trust models are designed in this chapter for protecting the same. Based on the trust model, an agent will be asked to visit only trusted host sites in order to accomplish a task. But this approach is effective only for those (agent spawning) applications where graceful degradation of performance is possible for example service discovery. But in case of e-commerce applications where an agent is explicitly asked to visit a node (that provides the service desired) cannot be achieved if that node is suspicious. In the following section we first discuss about the possible threats to mobile agents in MANET. This is followed by description of the proposed trust models which are validated through simulation results later.
6.2 Security Issues

Before discussing about countermeasures, let us discuss about who we are securing our agents from and why. Threats in a MAS can be categorized as [113]:

- **Threats from mobile agent to host platform** - A malicious agent may attempt denial of service (DoS), damage the software and data, penetrate virus/worms and finally action repudiation [113]. It may also steal some secret information from the hosts.

- **Threats from host platform to mobile agent** - A host platform may also attack an agent in several ways, like, reveal private or sensitive action performed by mobile agent, execute agents code incorrectly, sending agent to unintended destination, cheat agent with false information and information or action repudiation [79].

- **Threats from mobile agent to mobile agent** - Finally, an agent could face threats from other agents in the system like stealing agent information, conveying false information, rendering extra messages, DoS and unauthorized access [79], [113].

Unfortunately MANET environment brings with itself much vulnerability like blackhole [44], grayhole [44] or wormhole [44] attack. Commonly used routing protocols [44] cannot prevent such attacks. In such cases, agents are either dropped by a host (blackhole or grayhole) or are forwarded elsewhere (wormhole). In either case agents will never be able to come back to their owner in due time. Thus if an agent happens to pass through such a host it will effectively be lost.

However preventing a mobile agent from visiting a malicious node solves most of the risk factors. Consequently this technique not only protects the agents but also its owner MN from communicating with the malicious nodes. Reputation based trust models can be effectively used to serve this purpose. However, reputation system also suffers from threats like strategic rater [114] or strategically malicious host [115].

- **Strategic rater** - A single or a collusive group of raters purposely provide a set of unfair feedbacks aiming to destroy (boost) a peer’s (its partner’s) reputation [114].

- **Strategically malicious host** - A host can keep its reputation above a lower bound by cooperating or defecting its partners unevenly in an acceptable range so that it can still engage itself in future interactions.
6. Agent Security in MANET

- **Whitewasher** - Entities purposely leave or join the system with a new identity in an attempt to conceal any bad reputation under their previous identity [116].

Before going into the trust model, the threat model is discussed first that shows the kinds of threats our trust model is going to deal with.

### 6.3 Threat Model and Assumptions

The threat model considered in this work allows

(i) **Host to Agent Attack** - An adversary can place malicious (wormhole/blackhole/grayhole) nodes at arbitrary places in the network. These nodes are connected through a communication channel that cannot be observed by other nodes. These nodes either kill or mislead agents in such a way that they (agents) never come back to their owners in time (within a specified time-out limit). Thus the attacker does not need to have any knowhow to fool the nodes in believing that their agents are lost due to adverse MANET conditions.

(ii) **Agent to Host Attack** - A compromised node can send malicious agents to mislead a node or it can send a number of agents to a trusted node in order to block its traffic and hence launch DoS attack.

(iii) **Agent code modification attack** - A malicious node can change any unencrypted portion of a visiting agent’s code and/or data.

We assume that

(i) distributed applications like service discovery deploys mobile agents where graceful degradation in performance (as some nodes may become malicious) is acceptable. If an agent needs to visit a particular node in MANET (as in e-commerce) and that node is corrupted then however the task cannot be completed in our model.

(ii) no mobile node can behave as Whitewasher [116].

### 6.4 Modeling Reputation Based Trust

To enforce security to the mobile nodes and its agents, we use the concept of trust that has received considerable attention in information security literature. In a way, trust and security are two sides of the same coin, because if a system is secure, it is trusted, and if it is trusted, then it is considered to be secure [117].
This observation leads us to consider security to be a desirable property of a system in a given environment, and trust as a subjective belief resulting from assessing a system and its environment. As in [117], trust is defined as a subjective quantified predictor of the expected future behaviour of a trustee according to a specific agreement elicited from the outcomes of the previous interactions, from both direct and indirect experiences. Reputation of an individual host refers to certain characteristics related to its trustworthiness. Reputation can be obtained from a set of interaction feedbacks in a mobile agent system; where mobile agents describe a visited host’s performance in fulfilling its obligations. Indirect experiences can also be considered which are gathered from other trustworthy nodes. To speed up convergence, the list of suspicious nodes may be shared among the nodes in MANET via the agents.

6.4.1 Modeling MAS on MANET

An important use of mobile agent may be to collect data from network, like in service discovery [6] in MANET. Even e-commerce applications where the agents are expected to visit all nodes in the network [118] may also be considered here. Let us take an example where people may want to share real time data among them in a meeting. Here the people in the meeting have established an ad hoc network to get and stay connected with each other and the communicating devices (i.e. Laptop/PDA) used by them have the ability to process an agent code, that means all the nodes can act as potential hosts for the agents. Now, it is assumed that person 1 (MN_A) wants to gather some data from other members and has launched a mobile agent for that purpose. That agent will move from one node to another, collect data and at last return with the collected data to its owner, that is, the node that launched it. Thus an agent starts its journey from a given owner and moves from one node to another at its will. The owner provides a Priority List (PL) to the agent which contains a list of node ids that are most beneficial migration sites (for the application that deployed that particular agent). So, an agent always tries to visit those nodes from the priority list as its first preference. But this movement is successful if the two nodes are connected through a path having desired link quality and the routing table is properly updated to reflect the path. So, a probability measure is associated with the movement. Thus, if an agent residing at node MN_A decides to move to node MN_B (connected to MN_A) then the agent successfully moves to MN_B with probability p_t. Here p_t denotes the problem of unpredictable background noise level affecting link quality or lack of updation of routing table in a highly volatile MANET. For example, noise level may increase due to heavy rainfall.
6.4.2 Basic Trust Model

Every node assigns a reputation value to the other nodes of the network and updates it from the feedbacks of its agents. A mobile node gives a priority list (PL) of nodes to the agents (that they may visit) it deploys. The PL contains those node ids which are most beneficial for this agent as they may contain meaningful information for the application (that deployed the agent). An agent migrates through the network to visit the nodes mentioned in PL and reports back to its owner. From this feedback the owner updates reputation values of the nodes in PL and is able to detect malicious node(s). If any malicious node is detected the owner broadcasts this information to the network which helps others in updating their view of MANET.

The data structures needed to describe the model are listed in Table 6.1. Initially

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL&lt;sub&gt;ij&lt;/sub&gt;</td>
<td>It has two fields: nodeId and trustLevel (unvisited 0; suspected -1; trusted +1)</td>
</tr>
<tr>
<td>Trust Threshold</td>
<td>k (a positive integer)</td>
</tr>
<tr>
<td>Default trust level</td>
<td>TS (&gt; k)</td>
</tr>
<tr>
<td>Trust level view of the MANET at MN&lt;sub&gt;i&lt;/sub&gt;</td>
<td>(Trust level&lt;sub&gt;1&lt;/sub&gt;, Trust level&lt;sub&gt;2&lt;/sub&gt;, Trust level&lt;sub&gt;3&lt;/sub&gt;,...)&lt;sub&gt;i&lt;/sub&gt;, where Trust level&lt;sub&gt;1&lt;/sub&gt; represents the trust value assigned to MN&lt;sub&gt;1&lt;/sub&gt; by the current node</td>
</tr>
</tbody>
</table>

The priority lists (PL) of all agents contain unvisited node ids represented by 0 trust level corresponding to every node id in their PL. The workflow can be divided into two parts:

- Algorithm executed by the agents (AgentCode() in Algorithm 11) - collects direct information about the nodes visited by the agent;

- Algorithm executed by the nodes (MNCode() in Algorithm 12) - updates the trust level view of MANET maintained by a node from the direct information submitted by the agents it spawns and from any broadcast message received by the node (indirect information). This in turn affects the route taken by newer agents.

In this basic trust model it is assumed that there is a reliable routing protocol in place to ensure that the agents once deployed eventually come back to their respective owners. Thus host to agent attack (described in the threat model in Section 6.3) is not considered here. However it is taken care of in the next section (Section 6.4.3).
In our model the trust value of a node is incremented slowly but decremented rapidly at the owner from agent’s feedback. This is done to avoid agent migration to compromised sites under any circumstances.

As can be seen, agents in our system work like watchdogs [87]. Their feedbacks (stored in the status of the PL) work like direct information [87] for the reputation system working at the nodes (in Algorithm 12). The broadcast messages from a trusted node (in Algorithm 12) to others act as indirect information [87] to the recipients. The reputation system at each node updates its view of the network (based on these information) and accordingly guides (providing PL) the agents it deploys.

Algorithm 11: AgentCode(). An algorithm for collecting node status from the PL.

Input: PL containing unvisited nodes provided by the owner
Output: PL showing the agent’s experience with the nodes in PL

begin
1 while task given to the agent is not completed
2 Move to a yet unvisited host from the PL provided.
3 if that destination falls in the same cluster as it is now residing then
4 The agent moves to the new destination with probability p.
5 Before processing, take hashcode of the agent’s own code and data.
6 if the hashcode matches with the one stored in a secured way in the agent’s data then
7 Share information regarding the visited nodes with the host platform.
8 Set the trust value of this node to +1 in the PL.
9 Gather information needed by the owner application.
10 Update the computed results.
11 Compute hashcode of the code and updated data and store it in a secured way.
12 else
13 Put the trust level of this node to -1 in its PL and retract back to owner.
14 /* inference: most likely agent data has been changed.*/
15 Retract back to the owner.
end

Behavior of the agent can be altered sometimes by code manipulation. However the control flow of agent execution cannot be fully protected; hence the host can easily deduce information about the state of the agent. Besides, the host platform may also
Algorithm 12 : MNCodex(). An algorithm for updating trust based reputation at the nodes.

**Input:** Initial position, initial speed and maximum acceleration of each node

**Output:** The updated trust level view of MANET

```plaintext
begin
1. Input network configuration.
2. for t=t_0 to T do
   3. Software reliability of agent r_i(t) is calculated.
   4. if this node fails then stop.
   5. Use srmMANET() in Algorithm 7 to find out connected components.
   6. if an agent comes to this site/node (MN_j)
      7. if the visit frequency of agent from a particular node (MN_k) reaches threshold
         The agent is killed. // preventing DoS attack
         A message is broadcast stating MN_k to be compromised.
         Delete the current trust level view stored at MN_j.
      8. else update the trust level of the nodes.
         if the agent is found to trust a node (MN_k) // +ve trust level in PL
            Increment the trust value of (MN_k) in the trust level view of MN_j by 0.5.
         if the agent is found to suspect a node (MN_k) // -ve trust level in PL
            Kill that agent.
            Decrease the trust level of its owner.
            Learn not to migrate an agent via this node.
      9. if an agent owned by this node comes back containing at most one suspected node in its PL
         Update the results.
         Update the trust level view of the network according to the agent’s PL // Increase by 0.5 or decrease by 1
         if a node is found to be suspected
            Learn to avoid the existing route followed by the agents.
            Kill the agent.
      10. if the resulting trust level for any node falls below TrustThreshold
          Advertise that node to be suspected to the rest of the nodes.
      11. if a message regarding suspected node is received from a trusted sender
          Relevant information is updated depending on how much the receiver trusts the sender.
      12. The PL for each agent is formed and kept with the owners.
      13. Deploy agent, if needed by the application that follows AgentCode() listed in Algorithm 11.
end
```
unnecessarily hold a time-critical agent [119]. Since agent’s code is decided by the
owners and is not expected to be changed en’route, it is feasible to sign the code. In
step 5 of AgentCode() (in Algorithm 11) hashcode is calculated for the agent’s own code
and is stored in a secured way. The part of agent code meant for computing hashcode
(i.e., hashcode computation algorithm like MD5 [120]) of itself along with data is signed
by the agent’s owner. Moreover this is also encrypted using public key cryptography in
order to hide it in transit. Thus each agent carries the following

\[ \text{ENCRYPT}_{\text{publicKey}}[\text{SIGNATURE}_{\text{owner}}(\text{code for hashcode computation}) + \text{hashcode}] + \text{code of agent + data(except hashcode)} \]

Here code of agent refers to the purpose for which it is deployed by its owner. Upon
reaching a host site an agent takes its own hash code to check if it is attacked in transit
or by the current site. Once authenticated it executes the application code and updates
the results in its data. Then takes a new hashcode and replaces the old one. If in the
mean time the agent finds anything suspicious, it marks that (-1 trust level in PL) and
returns back to its owner so that the owner may update its trust level accordingly. To
save network bandwidth and improve performance of MAS, killing of the agent at the
suspicious host site cannot be suggested. Thus when any agent comes back to its owner
with at most one node id in its PL having a negative status (step 18 of MNCod() in
Algorithm 12), the owner node learns about a suspected area in the network. If no
suspicous activities are detected at the host site, the agent moves to a new host site
according to the task given [120].

It is quite reasonable to assume that influence (if any) of any strategic group of raters
on a node (agent owner) could eventually be nullified by counting direct experiences of
the agents. In reality it may be difficult that a strategically malicious host maintains the
same trust rating about itself at all the nodes in MANET. This is because by collecting
indirect feedbacks from neighbors the trusted community of nodes can eventually identify
those strategically malicious ones.

Thus the direct and indirect information can be effectively combined using Dempster-
Shafer belief theory as is discussed in the following section.

### 6.4.3 Extended Trust Model

The main focus of this work is to prevent the nodes and agents (deployed by them) in a
MANET from the effect of network layer attacks like blackhole [41] or wormhole [41] in
which the affected agents do not come back to their owner. Since in MANET nodes can
only have some knowledge about others it is impossible to know with certainty whether
a host is malicious or not; but an opinion can be formed about it, which translates into
degrees of belief (how much trustworthy the host is) or disbelief (how much suspicious the host is) as well as uncertainty in case both belief and disbelief are lacking. This can be expressed mathematically [13] as

\[ b + d + u = 1 \]  

(6.1)

Here \( b, d, u \) designate belief, disbelief and uncertainty respectively. The design of our reputation system is shown in Figure 6.1. It focuses on how to exploit the collected information to quantify the reputation of a node to ensure that an agent never falls into a blackhole, grayhole [41] or even wormhole trap. To quantify trust, parameters (\( b, d \) and \( u \)) are updated from direct observations (agent’s experience at different nodes) and indirect observations (feedback from neighbouring nodes and others, collected via agents). Both observations are combined towards quantifying trust from \( b, d \) and \( u \). Aging is also considered in the process that accounts for network dynamics. In addition, digital signature may be used to prevent or at least detect any attempt to change static code of the agent [120]. The agents will have the following information as input:

- **PriorityList of agent j.** It has two fields - node id and trust level of that node as perceived by the owner (unvisited 0; suspected -1; trusted +1; recent visit by an agent of same owner +2).

- **Suspected node list for agent j.** It has two fields - node id and provider node id
from which the agent comes to know about this suspicious node. Provider node id is optional and is meaningful only when the provider node is not the owner of agent j.

A node MN_i updates its observation about any other nodes from agent feedback as follows:

1. An agent visits MN_i: When an agent visits this node, the suspected node list of the agent gets shared with that of the node.

2. An agent owned by MN_i comes back: When an agent comes back to its owner, the agent’s feedback is updated as direct observation of its owner.

The way direct and indirect observation get updated in any node’s view is described below.

6.4.3.1 Direct Observation

An agent migrates from one node to the other according to its priority list. Upon reaching at a host, an agent checks for authenticity of agent code as mentioned in Section 6.4.2, and then executes code of agent. If the host is successful in fulfilling the agent’s obligations, the agent gives a positive feedback about the host to its owner. An agent retracts back to its owner if all the nodes in its priority list are either visited or suspected. The feedback of the agents about the nodes visited is considered to be direct observation at its owner. An agent gives negative feedback about a node if authenticity about agent’s code could not be checked. An agent may be attacked in transit but it may only be detected once it reaches a host site and checks itself. Also a node may act as a good host site for an agent (for some time) and behave maliciously with others (later on). Thus there is an uncertainty associated with the agent’s observation.

We assume that an agent eventually finds its owner whenever it needs. Here we utilize Beta(α,β) distribution as in [120], [121] due to similarity in characteristics. α_{ij} represents the number of good transactions between the agents deployed by owner i and MN_j. Thus for each positive feedback from agents, α_{ij} is incremented. Otherwise β_{ij} is incremented. To deal with uncertainty (as mentioned above), an approach proposed in [87], leveraging on the Dempster-Shafer Belief Theory [30] is adopted here to quantify the uncertainty of some random variables. Thus the uncertainty (u_{ij}) in predicting the nature of MN_j by MN_i is [87]:

\[
\begin{align*}
  u_{ij} &= \frac{12 \times \alpha_{ij} \times \beta_{ij}}{(\alpha_{ij} + \beta_{ij})^2 \times (1 + \alpha_{ij} + \beta_{ij})} \\
  \end{align*}
\]
For each positive feedback from agents, $\alpha_{ij}$ is incremented as follows

$$\alpha_{ij(new)} = w \times \alpha_{ij(old)} + (1 - w) \times p^k_j$$

(6.3)

Where $p^k_j$ represents agent$_k$’s observation about MN$_j$. Here weighted average is taken, where $w$ ($0 < w < 1$) represents absolute trust on each agent’s observation as this observation may change from time to time taking care of network dynamics. Also a malicious host may behave rationally for some time to gain trust from its peers. To tackle this, $w$ should be close to 1. Again $w$ behaves as the aging factor for values close to 0. Moreover, it may so happen that an agent successfully visits a number of hosts before falling into a trap. So, every node maintains last Z nos of owner ids that sent agents to this node. Thus if agent$_k$ while visiting MN$_j$ finds its owner id in Z (indicating some agent from the same owner has recently visited this node) then its (agent$_k$) owner further increments $p^k_j$ in equation 6.3 accordingly.

An agent may not come back to its owner in time due to network latency or presence of wormhole or blackhole. In such cases, to detect the exact cause, the owner divides the task into n subtasks (value of n depends on network bandwidth) and deploys n (part) agents. As these (part) agents have smaller trails (lesser number of nodes to visit) they are expected to come back faster. Moreover due to smaller trails the probability of getting lost at a malicious node is also reduced. In this way the trusted or malicious part of the neighbourhood can also be detected with more precision. If an agent$_k$ does not come back, its owner$_i$ increments $\beta_{ij}$ for all j nodes (that an agent is asked to visit) as follows

$$\beta_{ij(new)} = w \times \beta_{ij(old)} + (1 - w) \times q^k_j$$

(6.4)

Here $q^k_j$ represents the probability of misbehaviour at MN$_j$ with agent$_k$. An agent while visiting a host site may also share and update its suspected list with the host. Any updation to the list will be considered as indirect observation at the agent owner and is used to update the owner’s current view of the network. This is done to prevent a node from having any deceptive information. If an agent$_k$ finds that it would not be able to retract back in time i.e there could be time-out at its owner, it kills itself. In this way, agents can detect that it is lost and hence kills itself.

The values of $\alpha_{ij}$ and $\beta_{ij}$ are input to the reputation system that maps these to a tuple ($b_{ij}$, $d_{ij}$, $u_{ij}$). Here $b_{ij}$ gives MN$_i$’s belief in MN$_j$’s behavior as safe host site for agents deployed by MN$_i$. Similarly $d_{ij}$ indicates MN$_i$’s disbelief and $u_{ij}$ reflects MN$_i$’s uncertainty at predicting MN$_j$ as a safe host site for its agents. Here $u_{ij}$ is calculated using equation 6.2. Consequently following equation 6.1, total certainty (= (1-$u_{ij}$)) is
divided into $b_{ij}$ and $d_{ij}$ according to their proportion of supporting evidence as follows (initial observation follows from [87]):

\[
\begin{align*}
    b_{ij} &= \frac{\alpha_{ij}}{\alpha_{ij} + \beta_{ij}} (1 - u_{ij}), \text{ initially} \\
    &= \frac{\alpha_{ij}}{\alpha_{ij} + \beta_{ij}} (1 - u_{ij}) \times w_1 + b_{ij}(t - \Delta t) \times w_2 \times w_1, \text{ otherwise} \\
\end{align*}
\]

\[
\begin{align*}
    d_{ij} &= \frac{\beta_{ij}}{\alpha_{ij} + \beta_{ij}} (1 - u_{ij}), \text{ initially} \\
    &= \frac{\beta_{ij}}{\alpha_{ij} + \beta_{ij}} (1 - u_{ij}) \times w_1 + d_{ij}(t - \Delta t) \times w_2 \times w_1, \text{ otherwise} \\
\end{align*}
\]

Averaging (weighted) is needed to reflect n part agents’ behavior in the same tuple ($b_{ij}, d_{ij}, u_{ij}$). New observation ($b_{ij}$ in terms of $\alpha_{ij}, \beta_{ij}, u_{ij}$) is given a weight of $w_1$ and old observation ($b_{ij}(t - \Delta t)$ and $d_{ij}(t - \Delta t)$) is given a weight of $w_2$. Thus old values of $b_{ij}$ and $d_{ij}$ are given lesser weights ($w_2 < w_1$) than recent values to represent aging.

In this way with the help of Dempster-Shafer Belief Theory [30] uncertainty can significantly be reduced even though perfect accuracy may not be achieved.

6.4.3.2 Indirect Observation

For faster convergence of trust view, nodes share information about malicious node/s among each other via the agents. A node is suspected (not marked as malicious yet) if its $b < u < d$. This information indirectly influences a node’s view of the network. The influence is indirect as an agent suspects a node based on another (preferably trusted) node’s observation without ever visiting that node. This second-hand information helps a node to cope with long delays and frequent partitions (formation of disconnected clusters) which are characteristics of MANET. Let $b_{ij}|^{ij}$ represent belief (b) of MN$_i$ on MN$_j$ while taking indirect observation from MN$_j$. So this parameter depends on two factors-

1. MN$_i$’s belief on MN$_j$ and
2. MN$_j$’s observation about MN$_i$ that it shares (suspicious node list) with the agents of owner$_i$. 

118
Thus following the approaches proposed in [13] \((b_{ij}^i, d_{ij}^i, u_{ij}^i)\) can be formulated as

\[
b_{ij}^i = b_j^i \times b_i^j \tag{6.7}
\]

\[
d_{ij}^i = d_j^i \times d_i^j \tag{6.8}
\]

\[
u_{ij}^i = u_j^i \times u_i^j + d_i^j + u_j^i \tag{6.9}
\]

Here \(b_j^i\) = trust threshold, \(d_j^i\) = 1-trust threshold and \(u_j^i\) = 0. Trust threshold is fixed and is input to the reputation system. If any node’s trust value goes below trust threshold, it is treated as malicious (trust calculation is detailed later). It may be noted that MN_i’s disbelief in MN_j’s observation becomes an uncertainty in prediction about MN_l. Also MN_i’s uncertainty on MN_j amounts to the uncertainty of MN_i in predicting MN_l’s future behavior.

If MN_i enters a new network and gets trapped by its very neighbours then no agents deployed will come back. Then MN_i will prefer to wait till it moves. Moreover if a significant number of agents do not come back indicating wormhole or blackhole trap in transit, MN_i will prefer to request and collect information about suspicious nodes from its trusted neighborhood. The neighbors respond with their final observation \((b_j^i, d_j^i, u_j^i)\) about the nodes they suspect. Thus a node predicts about the future behavior of a node taking indirect feedbacks from all agents that retracted back to the owner recently \((\Delta t)\) and/or from response received from neighbours. MN_i updates its view \((b, d, u)\) as follows [13]

\[
b_{i:l} = \sum_{k \in S} b_{ijk} \tag{6.10}
\]

\[
d_{i:l} = \sum_{k \in S} d_{ijk} \tag{6.11}
\]

\[
u_{i:l} = \sum_{k \in S} \left( b_{ik}^j \times b_{jk}^i + d_{ik}^j + u_{ik}^j \right) / |S| \tag{6.12}
\]

Here \(b_{i:l}\) represents the indirect belief of MN_i about MN_l. S denotes the set of nodes that shared its view of the network (that MN_i received) with the agents deployed by MN_i.

### 6.4.3.3 Combining Direct and Indirect Observation

After collecting first-hand and second-hand information from the agents/trusted neighborhood, a node attempts to integrate these to come to a unified conclusion about
future behavior of the nodes. Thus the comprehensive belief \( b_{ji}(f) \), disbelief \( d_{ji}(f) \) and uncertainty \( u_{ji}(f) \) of MN\(_i\) on MN\(_j\) are derived from the following equations as in [13]

\[
\begin{align*}
    b_{ji}(f) &= \phi_1 \times b_{ij} + \phi_2 \times b_{ij} \\
    d_{ji}(f) &= \phi_1 \times d_{ij} + \phi_2 \times d_{ij} \\
    u_{ji}(f) &= 1 - b_{ji}(f) - d_{ji}(f)
\end{align*}
\] (6.13-6.15)

Where

\[
\begin{align*}
    \phi_1 &= \frac{\gamma \times u_{ij}}{(1 - \gamma) \times u_{ij} + \gamma \times u_{ij} - 0.5 \times u_{ij} \times u_{ij}} \\
    \phi_2 &= \frac{(1 - \gamma) \times u_{ij}}{(1 - \gamma) \times u_{ij} + \gamma \times u_{ij} - 0.5 \times u_{ij} \times u_{ij}}
\end{align*}
\] (6.16-6.17)

Here \( \gamma \) (0 < \( \gamma \) < 1) indicates a node’s confidence on the agents it deployed. Larger values of \( \gamma \) (\( >0.5 \)) means a node tends to trust its agents whereas smaller values (\( <0.5 \)) indicates that a node tends to trust others’ recommendations. Now, trust can be quantified from the comprehensive belief, disbelief and uncertainty as in [117],[87]

\[
T_{ij} = b_{ij}(f) + \sigma \times u_{ij}(f)
\] (6.18)

where \( \sigma \) is relative atomicity based on the principle of indifference. Here the possibility that an agent’s visit to a host will be safe or unsafe indicates two mutually exclusive and collectively exhaustive states. The principle of indifference states that if all (say n) possibilities are indistinguishable except for their names, then each possibility should be assigned a probability equal to \( 1/n \). Among the total uncertainty associated with an agent’s visit, there is a 50% chance that the agent will be safe. Thus \( \sigma \) is taken to be 0.5. But this parameter can be tuned more accurately such that for higher values of disbelief, there is a possibility that \( \sigma <0.5 \).

Consequently, depending on the trust values calculated from equation 6.18 and the safety requirement of the applications (running at the nodes) that deploys agents, an owner decides an agent’s task route and may ask it to avoid suspicious host sites.

An important use of mobile agents is to collect data from a network, like service discovery [6] or clustering in MANET [60] or e-commerce applications [118] etc. An agent starts its journey from a given owner and moves from one node to another at its will. The owner provides a PL to the agent which contains a list of node ids that are most beneficial migration sites (for the application that deployed that particular agent). A Suspicious Node List (SL) is also given that indicates potential blackhole or
wormhole points. An agent shares and updates its knowledge about suspicious nodes after reaching a trusted site. So, an agent will always try to visit nodes from (Priority List-updated Suspicious Node List) set. But this movement is successful if the two nodes are connected according to two-ray propagation model and there is no transient error in the environment and received signal can be distinguished from background noise. Thus, an agent residing at node MN\textsubscript{A} moves to node MN\textsubscript{B} (connected to MN\textsubscript{A}) with probability \( p_t \).

This proposed approach is used in our agent based system in MANET, with the data structures mentioned in Table 6.2.

Table 6.2: Data Structures for Extended Trust Model

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL(_j)</td>
<td>two fields: nodeId and trustLevel (unvisited 0; suspected -1; trusted +1; recent visit by an agent of same owner +2)</td>
</tr>
<tr>
<td>SL(_j)</td>
<td>two fields: nodeId and optional providerId if not given by the owner of agent(_j)</td>
</tr>
<tr>
<td>( \alpha, \beta )</td>
<td>positive integers to be kept at node</td>
</tr>
<tr>
<td>( C_{agent-id} )</td>
<td>number of part agents sent for the lost agent designated by 'agent-id'</td>
</tr>
<tr>
<td>( X )</td>
<td>maximum number of malicious nodes in the network</td>
</tr>
<tr>
<td>( T_{agent-id} )</td>
<td>maximum time an agent can be enroute</td>
</tr>
<tr>
<td>Trust Threshold</td>
<td>same as in Table 6.1</td>
</tr>
<tr>
<td>Default trust level</td>
<td>same as in Table 6.1</td>
</tr>
<tr>
<td>Trust level view at MN(_i)</td>
<td>same as in Table 6.1</td>
</tr>
</tbody>
</table>

Initially the PLs of all agents have the default trust value TS corresponding to every node id. So, MN\(_i\)'s view of the network will be \((TS_1, TS_2,)_i\). An owner divides its task and assigns it to the agents it spawns. Thus the PLs assigned to the agents of one owner contain disjoint set of node ids. We assume that the maximum number of malicious nodes (\(X\)) in the network is known. In the worst case it can be set to N. Assuming the PL of the agents to be unique, maximum X agents may get trapped at the X malicious nodes. But if more than X agents are lost at some point, clearly there are in-transit attacks or poor connectivity between the nodes. This is also taken care-of in the proposed algorithm.

The workflow can be divided into two parts

**Input:** PL containing unvisited nodes, SL containing malicious nodes

**Output:** PL and SL showing the agent’s experience with the nodes

begin
1. while task given to the agent is not completed
2. Move to an agent site i from PL if status(i)=0 and site i is not in the appended SL.
3. if destination site falls in the same cluster as it is now residing
4. The agent moves to the new destination with probability p.
5. Before processing, hashcode is used to detect any attempt to change agent’s code/data by the node.
6. if hashcode matches
7. Gather information needed by the application that deployed this agent.
8. Update computed results.
9. Hash code should also be computed to take care of updated data.
10. Share and update SL (if any) with this host
11. Appended entry (if any) is marked by the id of this host.
12. else go to next step. // inference: most likely agent’s visit was not safe
13. Retract back to the owner.
end

- **Computation/Action in mobile node:** An evolutionary algorithm based on MC simulation, is running at the nodes that takes its input from SecureAgentCode() listed as Algorithm 13 and any message received from trusted neighbors (second hand information) to update the distributed trust model and hence the node’s trust level view of the network. This updation is done according to UpdateTrust() algorithm listed as Algorithm 15. The trust level view in turn affects the route taken by newer agents. This is summarized as SecureMNCODE() in Algorithm 14. Here algorithm performance (in terms of false negatives etc.) can be estimated by repeating steps 2-30 Q times following MC simulation.

- **Functions of the agents:** The steps that an agent follows is summarized as SecureAgentCode() listed in Algorithm 13. This algorithm takes care of collecting first hand information about a node’s (whom it is visiting) trust and second hand information about the nodes whom the hosts (visited by the agent) suspect.

However fault tolerance of the nodes is not considered in this work. Here a node failure is treated to be irrecoverable. It is assumed that a host eventually detects a malicious node. Creation of part agents will be continued unless all of them come back or decision
Algorithm 14 SecureMNCODE(). An algorithm for updating trust based reputation at the nodes.

**Input:** Initial position, initial speed and maximum acceleration of each node  
**Output:** The updated trust level view of the MANET

```plaintext
begin
1    Input network configuration.
2    for t=t_0 to T do
3        Find software reliability of agents.
4        Simulate mobility and find connected components
5          using srmmMANET() listed in Algorithm 7.
6        if an agent (not owned) comes to this site/node (MN_j)
7            Kill the agent if it is found to suspect a node (MN_k).
8        else allow the agent to compute
9            Looking at the agent’s SL update indirect observation using
10               equations 6.10 through 6.12.
11            Share own (if nonempty) SL with the current visitor.
12        if an agent owned by this node comes back having at most
13            one suspected node in its PL then
14            Call UpdateTrust() listed in Algorithm 15.
15        if an agent does not come back and time out occurs
16            Divide the job of that agent into n parts and spawn n agents
17                which carry n priority sub lists.
18            Start T_{agent-id} timer for these n new agents.
19            Set C_{agent-id} to n.
20        if a part agent comes back
21            Decrease C_{agent-id} by one.
22            Call UpdateTrust() method listed in Algorithm 15.
23        if a T_{agent-id} expires
24            Find its corresponding C_{agent-id}.
25                if C_{agent-id} > X then
26                    Deploy the lost agents again asking them
27                        to follow different route.
28                else if 0 < C_{agent-id} < X then
29                    Ask recommendation from trusted neighborhood
30                        about the ones from PLs of C_{agent-id} lost agents.
31            Receive information from trusted neighborhood and update
32                the indirect observation following equations 6.10 through 6.12.
33            Hence update final (b,d,u) using equations 6.13 through 6.17.
34            Compute trust for each node following equation 6.18.
35        if the resulting trust of any node < Trust threshold demanded
36            by the deployer application then
37            append the node id to SL.
38        Deploy the agents with the SL (which nodes to avoid) and
39                a PL (which nodes to follow).
end
```
Algorithm 15 UpdateTrust(). An algorithm for updating observation at the nodes from agent feedbacks.

**Input:** Agent feedback about nodes in PL

**Output:** The updated trust level view of the MANET

```
begin
1. Update the results.
2. Update direct observation of this node.
3. if a node is found to be trusted then
   4. $\alpha$ is incremented according to equation 6.3.
   else
   5. $\beta$ is updated according to equation 6.4.
   6. Learn to avoid the existing route towards this node.
   7. Update values of $b_{ij}$, $d_{ij}$ and $u_{ij}$ using equations 6.5, 6.6 and 6.2 respectively $\forall j$ nodes visited by the agent spawned by MN$_{i}$.
9. Update indirect observation of this node.
10. if any new entry is found in the suspected node list then
11. Update this information depending on how much the owner trusts the information provider according to equations 6.7 through 6.12
12. Kill the agent

return
```

can be made about the nodes found in the priority lists of missing agents.

As can be seen, agents in our system migrate and collect feedback about the trust-worthiness of the nodes they visit. So, they work like watchdogs [87]. The reputation system at the nodes updates its view of the network based on the first hand and second hand information and accordingly guides (providing priority list and suspected node list) the agents the node deploys.

### 6.5 Experimental Results

The simulation is carried out in java and can run in any platform. MANET environment is programmed according to our description in Section 4.2.2 of Chapter 4. The nodes follow the SRMM and hence the network topology changes dynamically. For simplicity, in our simulation the PL tells the agents which nodes to visit. After visiting all the nodes from the PL successfully, the agent moves back to its owner. The experimental results of the two reputation models described in earlier sections are shown below.
6.5.1 Results for Basic Trust Model

We have taken an instance where there are six nodes in the network. Two mobile agents are deployed by four different owners (nodes). Agents 0 and 1 start their journey from nodes MN₀ and MN₁ respectively and roam around the network to accomplish its task. Thus an application (for example service discovery) running on MN₁ deploys agent 1. Our job is to protect the agents from malicious hosts and to kill a compromised agent as soon as possible. Initially all nodes are initialized with a default trust level. The agents are provided with a given PL by their respective owners (agent 0 needs to visit MN₁ and MN₂ whereas agent 1 needs to visit MN₂ and MN₃). For example, visiting nodes MN₁ and MN₂ will be most beneficial for agent 0 and so on. The nodes are taken close enough so that they form a connected network initially as shown in Figure 6.2(a). Every 3 seconds the positions of the nodes are updated according to SRMM. The simulation is carried out for 120 seconds. The network topology at 3 successive time instants is shown.
in Figure 6.2 (a, b and c). Agents are also shown in Figure 6.2 by callouts along with a numeral to indicate agent ids. The dotted ones (callouts) represent the starting position and the bold ones (callouts) represent end point of their journey at that time instant. The status of their PL is also shown in the figure. MN\(_3\) is assumed to be malicious in a sense that it tries to change the data carried by an agent.

<table>
<thead>
<tr>
<th>MN(_0)</th>
<th>MN(_2)</th>
<th>MN(_3)</th>
<th>MN(_4)</th>
<th>MN(_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 6.4: Default Values of Simulation Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobility Model</td>
<td>SRMM</td>
</tr>
<tr>
<td>M</td>
<td>20</td>
</tr>
<tr>
<td>N</td>
<td>25</td>
</tr>
<tr>
<td>Trust View default</td>
<td>5</td>
</tr>
<tr>
<td>TrustThreshold</td>
<td>3</td>
</tr>
<tr>
<td>Minimum required signal power</td>
<td>18dBm</td>
</tr>
<tr>
<td>Length of priority list</td>
<td>0.5N</td>
</tr>
<tr>
<td>Q</td>
<td>100</td>
</tr>
</tbody>
</table>

In time instant \(t\) (say) both agent 0 and agent 1 migrate to MN\(_2\) (Figure 6.2(a)), do necessary computations, compare hash code, update results, take new hash code (AgentCode() in Algorithm 11 in Section 6.4.2) and then move towards the next node in its PL. Consequently agent 0 migrates to MN\(_1\), computes and compares hash code and shares its belief of MN\(_2\) (Figure 6.2(b)). Thus trust level for MN\(_2\) at MN\(_1\) gets increased by 0.5. In the mean time agent 1 suspects MN\(_3\) and decides to move back without computing results at MN\(_3\). Thus in the next time instant MN\(_1\) finds agent 1 back and decreases the trust level of MN\(_3\) before creating a new agent (Figure 6.2(c)). Due to transient faults agent 0 could not make a successful migration this time and stays back at MN\(_1\). After this time, the trust view of MN\(_1\) is shown in Table 6.3. The trust value for MN\(_2\) is increased twice by 0.5 (by visiting agent 0 and then by agent 1) thus making it 6. The trust value for MN\(_3\) is decreased by 1 as is indicated by the PL of agent 1. The process goes on and as the trust value of MN\(_3\) reaches below 4 at MN\(_1\), it broadcasts a message which in turn updates the trust view about MN\(_3\) at MN\(_0\). This
6. Agent Security in MANET

process goes on and eventually all the nodes get a consistent view of the network only if they create or are visited by the agents.

Figure 6.3: (a) System performance in noisy environments; (b) System performance as threat to the agents increases

We have done a series of experiments to show the robustness of the proposed algorithm. The default values for the experiments are shown in Table 6.4. Any change in these values for individual experiments is explicitly mentioned.

First the effect of the inherent background noise is shown in Figure 6.3(a). As background noise increases, stronger signal becomes necessary for successful transmission. This increases the minimum required $p_r$ thereby decreasing coverage area of a node. Thus it takes longer times to detect compromised nodes in the network. As more nodes in the network become compromised, more time is needed to detect them. This difference in time increases further with increase in background noise. Thus for a highly noisy environment, longer time is needed before the trust views of the nodes in MANET reach a steady state.

If the background noise level is kept at a particular value and hence the minimum required $p_r$ and vary the number of malicious nodes in the network, the time to detect all of them increases even further. By AgentCode() algorithm (Algorithm 11), whenever an agent finds a suspected node, it comes back to its owner without discovering the MANET any further. This strategy saves bandwidth but makes detection of other malicious nodes in the network a time consuming task. Hence Figure 6.3(b) shows that as number of compromised nodes increases, the time to detect all of them increases even further.

But if the size of MANET is increased keeping other parameters fixed, then the result is shown in Figure 6.4(a). Here the percentage of trusted nodes is kept almost fixed to 84%. Correspondingly the PL of the agents also becomes larger (Table 6.4). Thus with larger networks, because of inherent mobility, the network becomes partitioned into a
number of components making the movement of the agents in some parts of the network impossible. As agent migration gets delayed, the process of trust calculation and sharing is also hampered. Thus for larger MANETs longer time is necessary to find all suspicious nodes.

A new metric called the ratio of agents passed is introduced in this work that is defined as follows

$$\text{Ratio of Agents Passed}(t) = \frac{\text{No. of agents going through malicious nodes till time } t}{\text{Total no. of agents deployed till time } t}$$ (6.19)

It is assumed that MN$_2$ and/or MN$_3$ are compromised where MN$_2$ behaved maliciously from the beginning but MN$_3$ is compromised during simulation (see Figure 6.4(b)). It is observed that after a certain time (513 units) the ratio of agents passed becomes independent of the reference point when a node becomes compromised. This explains the robustness of the proposed basic trust model (Section 6.4.2) as all malicious nodes can eventually be detected by the trusted ones.

### 6.5.2 Results for Extended Trust Model

We have taken an instance where there are 6 nodes and 3 agents in the network. The connectivity graph, agents and their corresponding PL are shown in Figure 6.5. Due to smooth movement of the nodes (according to SRMM) no drastic change can be observed in the connectivity graph in subsequent time instants. In this example MN$_4$ is treated as a malicious node that can launch routing attack that prevents visitor agents from coming back to their owners. As can be observed agents 1, 2 and 3 eventually get stuck at MN$_4$. Thus according to SecureMNC() listed in Algorithm 14 time out ($6 \times$ average
Figure 6.5: MANET configuration with 6 nodes and corresponding agent locations: (a) Initially; (b) After 3 seconds; (c) In next 3 seconds indicating end of the agents’ journey

propagation delay) occurs and the owners MN1, MN2 and MN3 subdivide the PL of their lost agents into two unequal parts and spawn two agents accordingly. The division is carried out according to a factor that is initialized to 0.5 but is decreased by 0.15 each time an agent gets lost till it reaches 0.2 (in an order to group all suspicious nodes in one sublist). For example when an agent with 10 nodes in its PL goes missing, the PL is divided equally (ratio is 0.5) into 2 sublists having \((0.5 \times 10) = 5\) nodes each and is given to two part agents. If both part agents go missing, the PL is subdivided into 2 again with ratio \((0.5-0.15)=0.35\). Thus one PL contains \((10 \times 0.35) = 3\) nodes and rest \((7\) nodes) are assigned to the other part agent. Here nodes with \((d_{ij}(f) - b_{ij}(f)) > \epsilon = 0.0028\) are put in the smaller sublist. Thus in this simulation example, agent 1 now needs to visit MN2 and MN3 while agent 4 visits MN4. Moreover, while visiting MN2 and MN3, agent 1 finds that some agent from the same owner (MN1) has recently visited these nodes. This observation is reflected in the status \((=2\) instead of \(1\)) of agent 1’s PL. Clearly this time direct observation \((b_{12} = 0.2, d_{12} = 0.11, u_{12} = 0.7\)) of agent 1 gets reflected in the
Table 6.5: Default Values of Simulation Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRMM Mobility Model</td>
<td>20</td>
</tr>
<tr>
<td>N</td>
<td>25</td>
</tr>
<tr>
<td>Trust View default(b,d,u)</td>
<td>(0,0,1)</td>
</tr>
<tr>
<td>TrustThreshold</td>
<td>0.49</td>
</tr>
<tr>
<td>Minimum required signal power</td>
<td>18dBm</td>
</tr>
<tr>
<td>PL Length</td>
<td>10</td>
</tr>
<tr>
<td>Time</td>
<td>80min</td>
</tr>
<tr>
<td>Q</td>
<td>100</td>
</tr>
</tbody>
</table>

final observation\(b_1^{2(f)}=0.05298,d_1^{2(f)}=0.02796,\ u_1^{2(f)}=0.91906\) of its owner’s (MN₁) trust view. This process goes on. While updating direct observation in equation 6.5 and 6.6, for simplicity (major change is not expected in simulation time=80min) old and new values are given equal weights. As soon as trust value of any node goes below 0.49, that node is declared to be malicious and is appended in the suspected list of agents (removed from its PL as well) spawned by the detector node. Thus the nodes try to overcome routing attacks without any overhead of control messages.

We have done a series of experiments to validate the extended trust model (Section 6.4.3) and found some interesting results. For simplicity whenever an agent goes missing, 2 agents are spawned as is explained in the example. The simulation parameters are summarized in Table 6.5. Any change to it is explicitly mentioned. Experiments are done to show timely variation of the ratio of agents passed while the MANET has
3 malicious nodes (MN\textsubscript{3}, MN\textsubscript{4} and MN\textsubscript{5}) and 5 malicious nodes (MN\textsubscript{3}, MN\textsubscript{4}, MN\textsubscript{5}, MN\textsubscript{11} and MN\textsubscript{13}). Results plotted in Figure 6.6(a) clearly show that agents gradually overcome network hostility. This is evident from the steady slope of the curve especially after T=8min. As number of malicious nodes increases more agents are affected but eventually the agents detect them by trust calculation. Since the curve well stabilizes at around 80 min with 3 malicious nodes, simulation time is kept at 80 min in our experiments.

Also the variation of the ratio (equation 6.19) with number of nodes (N) is shown in Figure 6.6(b) while MN\textsubscript{3} and MN\textsubscript{4} launch blackhole/wormhole attack. It can be observed that as the network gets bigger, the ratio gradually declines (due to increased amount of indirect observation) and eventually (N=35 onwards) reaches a steady state. Also more number of agents (M=20) implies richer direct observation resulting in even faster convergence of trust. Arrival of steady state for both M=10 and 20 indicates the scalability of our scheme with moderate accuracy.

Another metric called ratio of successful agents is defined as follows

\[
\text{Ratio of Successful Agents}(t) = \frac{\text{No. of agents came back to owner till time } t}{\text{Total no. of agents deployed till time } t} \quad (6.20)
\]

All agents deployed by an owner may not come back within stipulated time as some of them may be rerouted by malicious nodes (wormhole), engulfed by them (blackhole) or lost due to network partitioning. Agent code/data may get modified also that can be detected by the owner (by generating and checking hashcode [120]). Considering MN\textsubscript{3} and MN\textsubscript{4} to be malicious nodes, the effect of MANET size on agent success is found. Momentary drop in agent success can be observed when M, N and L values are almost comparable in Figure 6.7(a). But as MANET becomes bigger, agents manage to provide a steady success rate. Both figures 6.6(b) and 6.7(a) confirm the fact that bigger networks are not detrimental for agent success if the level of hostility remains same.

The next experiment again introduces another metric called the node success ratio defined as

\[
\text{Node Success Ratio}(t) = \frac{\text{Nodes that can prevent their agents from attacks till time } t}{\text{Total number of nodes working till time } t} \quad (6.21)
\]

The value of this ratio depends on successful detection and subsequent deletion of malicious nodes from PL. The variation of this ratio with the number of agents deployed is indicated in Figure 6.7(b). It is seen that with 50 agents, up to 3 malicious nodes can be successfully detected within 80 minutes and no nodes in that case will be sending
Figure 6.7: (a) Variation of agent success rate with no. of nodes (N); (b) Variation of node’s success ratio with number of agents (M)

their agents to the malicious nodes (MN_3, MN_4 or MN_5). Also it can be observed that all curves reach a local maxima when number of agents is approximately equal to number of nodes (=25) (> L). This is because at this point all nodes get the direct observation from agents, that is, agents tend to cover the entire network.

In the next experiment the proposed model is tested with increasing MAS size. A metric called ratio of false negatives is introduced that is defined as follows

\[
\text{Ratio of False Negatives} = \frac{\text{No. of undetected malicious nodes}}{\text{Total No. of malicious nodes}}
\] (6.22)

Figure 6.8: Success of the reputation model proposed in detecting malicious nodes

Let us presume we know the total number of malicious nodes (X) in the network at a time instant. So it is checked if the proposed algorithm can successfully detect all malicious nodes. The results show (Figure 6.8) that as more nodes participate for some job and hence deploy agents (which in turn also gains direct experience) to traverse
various parts of the network more malicious nodes are eventually detected. Thus for
greater MAS size the ratio ultimately drops to 0 indicating successful detection of all
malicious nodes. For bigger network more agents are needed to achieve the same value
of false negatives. Interestingly with $M=2 \times N$, the ratio of false negative hits 0. It
also portrays correctness of our algorithm as all malicious nodes can be detected by
deploying sufficient number of agents. Here it is assumed that the bandwidth provided
by the underlying network well support those agent migration.

6.6 Conclusion

This chapter provides possible strategies for securing mobile agents in MANET against
possible threats of modification of agent data and/or code by compromised nodes. It
attempts to find distributed trust model for the network so that each trusted node may
eventually get a consistent trust level view of the network and hence prevent agents
deployed by them from visiting compromised nodes any further. Here the concept of
hashing is utilized to detect possible modification of an agent’s data and code. Our
model provides methodology to secure not only the agents, but also the agent owners
(nodes). It provides prime security services like integrity, authenticity. The agent owners
are given the responsibility of killing malicious agents and creating new ones. If any
node is found to be malicious, its entry is removed from the PL of new agents. The
schemes presented in this chapter enable an agent to share information about MANET
with the nodes it trusts, helping MNs update their trust levels. Modification of agent’s
code and/or data in transit is also detected eventually. On detection, a node broadcasts
this information. The nodes only listen to (and update trust level) broadcast messages
from the senders they trust. More a node trusts the sender more impact this broadcast
can make to the receiver’s trust level view.

Even if agents get lost by routing layer attacks like blackhole, grayhole or wormhole
in MANET causing agents to be lost in transit, even then also the proposed trust mech-
anisms enable the agents and their owners to eventually learn and avoid the malicious
ones.

The models are validated and results are shown in the previous section. It can be
observed that for larger MANET longer time is necessary to detect all compromised
nodes. But the proposed trust mechanisms are found to detect any change in node
behaviour even when the change occurs in runtime.

A node may dynamically change behaviour not only because it is attacked by some
adversary. A node may start behaving selfishly because of device and network constraints.
6. Agent Security in MANET

In the next chapter these issues are addressed in order to make agents more dependable.