EMLTrust: An enhanced Machine Learning based Reputation System for MANETs

Rehan Akbani a, Turgay Korkmaz b,⇑, G.V. Raju c

a Bioinformatics and Computational Biology, M.D. Anderson, Houston TX, United States
b Department of Computer Science, University of Texas at San Antonio, TX, United States
c Electrical and Computer Eng., University of Texas at San Antonio, TX, United States

Article info
Article history:
Received 28 November 2010
Received in revised form 20 July 2011
Accepted 15 August 2011
Available online 24 August 2011

Keywords:
Trust management
Machine Learning
Digital credentials
Fading Memories

Abstract

Many mission critical networks including MANETs for military communications and disaster relief communications rely on node cooperation. If malicious nodes gain access to such networks they can easily launch attacks, such as spreading viruses or spam, or attacking known vulnerabilities. One way to defend against malicious nodes is to use Reputation Systems (RS) that try to predict future behavior of nodes by observing their past behavior. In this paper, we propose a Machine Learning (ML) based RS that defends against many patterns of attacks. We specifically consider the proposed RS in the context of MANETs.

After introducing a basic RS, we propose further enhancements to it to improve its performance and to deal with some of the more challenging aspects of MANETs. For instance, we consider digital signature based mechanisms that do not require trusted third parties, or servers that are always online. Another enhancement uses an algorithm called Fading Memories that allows us to look back at longer histories using fewer features. Finally, we introduce a new technique, called Dynamic Thresholds, to improve accuracies even further. We compare the performance of our RS with another RS found in the literature, called TrustGuard, and perform detailed evaluations against a variety of attacks. The results show that our RS significantly outperforms TrustGuard, even when the proportion of malicious nodes in the network is high. We also show that our scheme has very low bandwidth and computation overhead. In contrast to existing RSs designed to detect specific attacks, ML based RSs can be retrained to detect new attack patterns as well.

1. Introduction

Let us consider a military MANET in a battle field where several vehicles and soldiers are using wireless communications to exchange mission critical information and to provide various services to each other. In such a realistic MANET, it is deemed necessary to make sure that only the authorized/legitimate users/nodes can access the network resources and the services provided by the other nodes. For example, a typical MANET may have several resources including file servers, databases, web servers, etc. In addition, many nodes may provide different services as part of a larger Service Oriented Architecture (SOA) approach. In SOA, large applications are modularized into smaller services which run on heterogeneous devices. It especially makes sense to use SOA in MANETs so that large, computationally expensive applications can be implemented on resource constrained devices in a distributed fashion. But from a security standpoint, we need a mechanism to regulate access to those resources and services so that we can guard them against malicious transactions from malevolent or compromised nodes.
MANETs can be classified as open or closed. In an open MANET anyone is free to enter or leave the network (e.g., in airports and university campuses), whereas in a closed MANET only designated nodes are allowed to access the network (e.g., in a military setting). In general, it is more difficult to provide security in an open MANET since there is no restriction on who may access the network. Fortunately, the security requirements of such networks are also not very demanding since users expect public networks to be insecure. By contrast, closed networks may have very strict security requirements (e.g., in the military or in the police department). Therefore, we specifically focus on closed MANETs. Various attacks can be launched against such MANETs by outsiders and/or insiders. Often different mechanisms are needed to defend the underlying network against insiders or outsiders. In this paper, we study how to defend MANETs against malicious transactions from malevolent or compromised (insider) nodes. Defending against outsiders is out of the scope of this paper, but has been extensively investigated in our related work [2,3] and references therein.

Suppose an adversary is somehow able to join a closed MANET as a legitimate user. Such a compromise may occur in many ways. For example, an adversary may hack into and gain access to a legitimate node, or obtain the secret key of a legitimate node and assume its identity. Due to the wireless nature of MANETs, those types of attacks would be common and easier to launch. Unfortunately, there is no litmus test to enable one to verify whether an insider node is malicious or benign. We can only predict future behavior of a node by observing and analyzing its past behavior. That is the basic idea behind Reputation Systems (RSs) that have been extensively investigated in the literature [4–9]. For example, eBay uses that form of Reputation System where users leave feedback about other users [10] and Google uses PageRank where web pages are ranked for relevance by other pages [11]. RSs are also used in the context of P2P networks, large scale distributed networks, and the Internet [12,6,13–15]. In general, any network where nodes frequently transact with each other can benefit from RSs. RSs are especially warranted for mission critical networks that rely on node cooperation, such as closed MANETs for the military, emergency and disaster relief networks, and corporate networks [4–6,13].

Fig. 1 illustrates the basic steps in a typical RS. In general, a node that needs to decide whether to transact with another node or not must first gather historical data about that node (e.g., the proportion of good vs. bad transactions in the last x minutes). Then it applies a customized mathematical equation (or statistical model) to the data to produce an output score. For example, the RS in [6] is based on Eigen values from Linear Algebra, the one in [5] is based on using derivatives and integrals, and the one in [8] is based on Bayesian systems utilizing the Beta distribution. Depending on the output of the equation or model, the system then decides how to respond. In most cases, the equation or model is customized to detect specific types of malicious behavior only. For instance, the algorithm in [5] is designed to detect malicious behavior that alternates with good behavior and varies over time.

Rather than developing a separate module for each attack pattern manually, we propose the use of Machine Learning (ML) to build more flexible and dynamic RSs that can be retrained to thwart a multitude of attack patterns easily and efficiently. Specifically, we consider Support Vector Machines (SVM) and discuss how they can be used for designing RSs. The basic form of the proposed ML-based RS can be used as a general model in wired networks like the Internet. However, to be able to use it effectively in MANETs, we need to deal with several challenges unique to MANETs. For example, eBay has dedicated and trusted centralized reputation servers to collect and store reputation scores for buyers and sellers. Users can trust that (i) the scores being reported by eBay are genuine, (ii) the transactions actually did occur between the buyers and sellers, (iii) unfair scores can be challenged by users and arbitrated by eBay, and (iv) the scores have not been tampered with by Internet routers en-route from eBay to the user. All of those assumptions do not necessarily hold for MANETs due to various reasons. For instance, there is no online central authority, many nodes are limited in their computational resources, nodes may go offline at any time, and nodes are not guaranteed to be completely trustworthy [16].

To deal with the challenges of MANETs while further improving performance, we enhance our core SVM based RS with various mechanisms. To guard against fake transactions and dishonest/incorrect feedback, we propose a digital signature based scheme that does not need online trusted third parties. Using extensive simulations, we demonstrate the efficiency and effectiveness of the proposed core SVM approach, and compare it against two other algorithms found in the literature, namely TrustGuard Naive and TrustGuard TVM [5]. We consider TrustGuard because it has been shown to perform very well compared to eBay’s Reputation System. We simulate five different attack scenarios and show that our approach outperforms TrustGuard in all five scenarios, including when there is oscillating or steady behavior, collusive or non-collusive behavior. Our scheme can achieve high accuracy and correctly predict good vs. malicious nodes, even when the proportion of malicious nodes in the network is very high. The ROC curves show that the improvement of SVM over TrustGuard is statistically significant, as their 95% confidence intervals do not overlap each other. We also show that SVM has the same bandwidth overhead as TrustGuard Naive while having much less overhead than TrustGuard TVM.

We propose two further enhancements to improve the performance of our core SVM based RS. First, we consider how to look back at longer histories using only a few features. That enhancement forces the adversary to behave well for longer periods of time in order to boost its reputation score. We evaluated its performance and showed that it was much better at detecting malicious behavior that varied over longer periods. Second, we introduce an algorithm called Dynamic Thresholds that further improves the
accuracy of SVM by dynamically shifting the SVM decision boundary based on the proportion of malicious nodes in the network. Through simulations, we show that Dynamic Thresholds can significantly reduce classification error and improve performance in detecting malicious nodes.

Researchers have also investigated monitoring based trust management (TM) systems to deal with attacks against insiders. In monitoring based approach, each node’s wireless traffic is monitored by its neighbors and conclusions are drawn based on it [17,18]. Many monitoring based systems have been proposed in the literature [19–23]. The principle behind their operation is that traffic from a node is classified as legitimate or illegitimate by its neighbors. Examples of illegitimate traffic include known attack signatures, viruses and worms, or anomalous behavior. A node that is behaving well and communicating mostly legitimate traffic is deemed to be trustworthy. Such a node accumulates “good credit” points through its good behavior and slowly gains access to increasingly sensitive resources. That model has a very fast response time. A misbehaving node can quickly be detected and its traffic can rapidly be blocked. That is because all the information gathering and decision making is done within a node’s one hop neighborhood. Any detected malicious traffic cannot pass beyond that one hop neighborhood. However, the most serious disadvantage of using monitoring based trust management systems is that they have been shown to raise too many false positives due to noise. According to [24], traditionally simulated noise models do not mimic the behavior of observed noise patterns in an actual setting, leading to optimistic results in simulations. In experiments done on MANET testbeds, it was shown that monitoring based systems do not work well as they raise too many false alarms [25]. Therefore, in our research, we do not consider using monitoring based systems. Instead we just focus on Reputation Systems (RSs) and assume that the underlying network routing protocols will allow any two nodes to communicate with each other as long as they are connected. In other words, our proposed mechanisms actually work above the network layer and do not deal with routing (which by itself is a major problem and has been extensively investigated in the literature).

The rest of this paper is organized as follows. In Section 2, we define RSs in more detail and justify why Support Vector Machines (SVM) are suitable for them. In Section 3, we describe the three principle challenges associated with designing any RS and explain how our core SVM based RS tries to solve them. In Section 4, we evaluate our core RS and compare it against another RS mentioned in the literature, called TrustGuard. In Section 5, we describe our first enhancement to the core RS using Fading Memories and digital signatures, and evaluate its performance. In Section 6, we describe our second enhancement to the core RS with Dynamic Thresholds, and evaluate its performance. Finally, we conclude the paper and give some directions for future research in Section 7.

2. Basic Machine Learning approach

Based on the general framework in Fig. 1, we can redefine the problem of designing an RS into one of finding the optimal set of input features and equations (steps 1 and 2 in Fig. 1) that allow us to distinguish between malicious and benign nodes with high accuracy. Machine Learning (ML) is of particular significance in this context since many ML algorithms are able to determine and approximate the optimal equation needed to classify a given set of data. We envision the problem of RS as a time series prediction problem, which states: Given the values of the dependent variable at times \((t, t - 1, t - 2, \ldots, t - n)\), predict the value of the variable at time \((t + 1)\) [17,18]. The dependent variable in this case is the proportion of good transactions conducted by a node in a given time slot. Predicting this variable at time \((t + 1)\) gives us the probability that the node will behave well if we choose to transact with it at time \((t + 1)\). Therefore, we opted to use Support Vector Machines (SVM) as our ML algorithm because it has been shown to successfully approximate mathematical functions [26] and make time series predictions [27]. In our scheme, we build SVM models against different types of malicious behaviors offline, and then upload those models to the nodes in the network. The nodes can use those models to classify new nodes and predict if a new node is malicious or not. Constructing models is computationally expensive so it is done offline, possibly by a third party. However, the classification step is not very expensive and can be done on the node in real time. When a new type of attack is discovered, a new model can be constructed against it. This is similar to how anti-virus systems work where the anti-virus is developed offline and then uploaded to clients. Similarly, in our scheme the vendor of the RS might update its subscribers with SVM models against new attacks.

An implied assumption is that after a transaction has taken place, a node can determine if the transaction was good or bad with a high probability. This is true in many cases, such as in commercial transactions on eBay, as well as in file downloads (where a corrupted or virus infected file would be considered bad), or in providing network services [26,27]. Another assumption is that the feedback can be reliably transmitted without being tampered with. This can be accomplished by digitally signing feedback messages. These assumptions are made by many researchers in the field [4–6] and we also make the same assumptions in our study. However, a few transactions might be incorrectly labeled good or bad. SVM can handle fair amounts of such “noise” in the data set [26].

3. Building the Core SVM based Reputation System

If all the nodes in a network gave honest and correct feedback about the transactions they conducted, then it would be trivial to spot malicious nodes since all the good nodes would have 100% positive feedback, whereas the malicious nodes would not. But in reality, this is not the case and we have to deal with three principle challenges:

1. Dishonest feedback given by malicious nodes against other nodes they have transacted with.
2. Incorrect feedback from legitimate nodes by mistake.
3. Fake feedback given by malicious nodes about transactions that never really occurred.
In Section 3.1, we discuss the key factors in an SVM based RS and explain how we take them into account to tackle problems 1 and 2. Since SVM cannot detect if a feedback was fake, we need another mechanism to deal with problem 3. Accordingly, we propose a digital signature based mechanism in Section 3.2. We distinguish between inadvertent (problem 2) and deliberate false feedback (problem 3) by assuming that the proportion of dishonest to honest feedback given by malicious nodes is much higher than the proportion of incorrect to correct feedback given by legitimate nodes. However, if malicious nodes reduce the proportion of dishonest feedback to match those of incorrect feedback, we have still succeeded in our goal of reducing malicious behavior.

3.1. Factors in building the classifier

There are many factors to consider in building the classifier that will be used to distinguish malicious nodes from good nodes. The following factors should be taken into account:

1. **Feature Selection**: The features used to train and test the classifier must be the same in the training as well as the testing set.
2. **Proportion of malicious nodes**: The proportion of good vs. bad nodes in the data set should not be the same in the training and testing sets. This relates to the degree of imbalance in the data sets.
3. **Size of data set**: The number of instances in the training and testing sets.
4. **Evaluation methodology**: The method used to evaluate the performance of the classifier, such as train/test split, n-fold cross validation, and leave one out.
5. **Evaluation metrics**: The metrics used to evaluate the classifier, such as accuracy, precision (specificity), and recall (sensitivity).
6. **Kernel used**: Which kernel should be used for SVM?

The last factor is only valid for Kernel Machines, such as SVM, while the other factors are valid for all types of ML classifiers and will be described next in more detail.

3.1.1. Feature Selection

Feature Selection is a critical step in constructing the classifier. Using too few features might not provide sufficient information to the classifier. On the other hand, it is common knowledge in ML that increasing the number of features increases the accuracy up to a certain point. After that, increasing the number of features results in a degradation in performance. This phenomenon is called the "Curse of Dimensionality" [26]. Therefore, the number of features should be selected carefully.

To construct our features, we divided time into regular intervals called time slots. The network administrator can choose and fix a time slot from a few minutes to a few hours long, or the system can dynamically determine a time slot depending on how frequently nodes in the network transact on average. The features in our experiments consist of the proportion of positive vs. negative feedback assigned to a node during a given time slot by the nodes it has transacted with. To collect features for a test node, we need to query all the nodes in the network and ask them to provide us any feedback they have about the node for a given slot. The fraction of positive feedbacks versus total feedbacks for that slot forms a single feature. Each time slot then corresponds to one feature. This is in accordance with [5], and is also based on features used in time series prediction problems [27]. We can vary the number of features by varying the number of time slots used. We tried different numbers of time slots and observed that 15 time slots provided a good trade-off between accuracy and computational cost; thus, we use 15 time slots for our core SVM. In Section 5, we present an algorithm using Fading Memories that will enable us to look further back in time using only a few features.

3.1.2. Proportion of malicious nodes

In a real world setting, we would not know the true proportion of malicious nodes vs. good nodes in the network. The ratio, or degree of imbalance, could vary from zero to a very large degree. To account for this, we use multiple test sets with different ratios in our experiments. We would expect the accuracy to deteriorate as the ratio of imbalance becomes larger. In Section 6, we present our second enhancement that uses Dynamic Thresholds based on estimated imbalance to improve accuracy even further.

Next, we need to consider the imbalance ratio for the training set. In an actual setting, we would not know the proportion of malicious nodes in the network, so the testing should be done with varying imbalance ratios. However, the training set can only have one imbalance ratio since we need to build just one SVM model. We used a malicious node proportion of about 60% since that gave us the best results.

3.1.3. Size of data set

The overall goal is to try to estimate the performance of our RS in the "real world" where an infinite number of transactions will occur. Obviously, we cannot consider infinite number of transactions in simulations. So we have to use simulations with a limited number of transactions. The question then becomes how many transactions (or instances) in our sample are good enough to give reasonable estimates. If the training data set is too small, the classifier will not have enough instances to properly train itself and accuracy will suffer due to overfitting. A small test data set can lead to wide variations between the test error and the true error. Of course, larger and larger sample sizes will give us better and better estimates of the infinite "real world" data set. However, using very large data sets increases computational costs without necessarily increasing accuracy. We should note that a general rule of thumb in Machine Learning is that the number of instances in the training set must be at least 10 times the number of features to obtain reasonably accurate metrics [28]. Since we are using 15 features, we need at least 150 instances to get accurate metrics. Accordingly, we conducted some preliminary experiments with different sizes and determined that 1000 instances for the training set and another 1000 instances for the test set achieve a good trade-off between computational cost and accuracy. We then decided...
to go with 1000 instances, which effectively diminish the “size of the data set” as a possible source of error in the estimate while being computationally feasible.

### 3.1.4. Evaluation methodology

Several traditional ways of evaluating the classifier exist in Machine Learning. In one common approach, two independent sets of data are used: one for training the classifier and another one for testing it. Following this approach, we used two independently constructed data sets in our experiments. Other common techniques for testing include n-fold cross validation and leave-one-out, where a randomly chosen portion of the data is used for training and the remainder of the data is used for testing. This procedure is repeated many times. These techniques usually provide a better estimate of the true error than using simple training and test sets. However, since the proportion of good vs. bad nodes needs to be varied in our test set while keeping the training set constant, cross validation is not the appropriate choice for our experiments.

### 3.1.5. Evaluation metrics

Many ubiquitous evaluation metrics are used in ML to evaluate classifiers. The most common metric is accuracy, which simply measures the fraction of correct predictions. Other common metrics include precision and recall (or sensitivity) [29]. Graphs such as ROC curves are based on these metrics. The metrics are defined below:

\[
\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}
\]

\[
\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}
\]

\[
\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}
\]

In our experiments, we have used accuracy as our evaluation metric as it is also used in [5]. In the future, we will also use precision and recall to evaluate the classifier.

### 3.1.6. Kernel used

Unfortunately, there is no easy way to pre-compute which is the best kernel to use for a given data set [30]. As a result, for most applications we have to use trial and error to evaluate commonly used kernels and determine the most appropriate one. Common kernels include polynomial kernels of varying degrees, exponential kernels, and Radial Basis Function (RBF) kernels.

In our experiments, the linear kernel (polynomial kernel of degree 1) has shown good results and increasing the degree of the kernel did not significantly increase performance. In general, it is desirable to use the lowest degree kernel that gives good results to save on computational costs. Exponential and RBF kernels have also been tried and they did not significantly increase performance beyond the linear kernel either.

### 3.2. Guarding against fake feedback

Fake feedback means that a node gives feedback about a transaction which never actually occurred. So, for instance, node $x$ may provide some feedback (either good or bad) about another node $y$, when in reality $y$ never transacted with $x$. A malicious node may try to submit several bad feedbacks about another node in order to reduce its reputation. To guard against that, TrustGuard [5] proposed a mechanism for binding transactions to “transaction proofs,” such that (i) a proof cannot be forged and (ii) is always exchanged atomically. Those proofs ensure that every feedback is tied to a transaction that actually took place. To ensure atomicity, TrustGuard uses trusted third parties that must come online frequently to resolve disputes.

In our work, we also consider the transaction proofs that cannot be forged, but we relax the restriction of atomicity. This is because we would like to eliminate the need for a trusted third party, which may not be available in MANETs. Relying on a third party in itself provides a window of opportunity to adversaries when the third party goes offline, as pointed out in [5] itself.

In our implementation, each node has a pair of public and private keys. A client contacts a server and requests service. The server responds and it either denies the request or commits to providing the service. If it commits, the server will send the client a certificate to sign. The certificate has the following fields:

\[
\text{(Server ID, Client ID, Time Slot ID, Transaction Number)}
\]

\[
\text{(Cert.)}
\]

The server and client IDs are self-explanatory (e.g. IP addresses of each). The Time Slot ID is the time stamp of when the current time slot began. If, for instance, a time slot is 30 min long, the client should check that the time slot ID should be no more than 30 min prior to the current time. All the nodes in the network must be aware of how long the time slots are and when a new time slot starts. The Transaction Number is the number of transactions that have occurred (including the current one) between the client and the server in the current time slot. All of these fields are verified by the client and then signed. Note that each node only needs to keep track of the transactions that it has conducted within the current time slot. A node will never need to sign a certificate for a previous time slot.

After verification, the client sends the signed certificate back to the server. The server verifies the signature and then signs the same certificate with its own private key and sends it to the client. Then it provides the requested service. In this way, both the client and the server end up with copies of the certificate signed by each other. In the future, if a node $z$ asks either of them to provide feedback about the other, it will provide the feedback and present the signed certificate. In this way, $z$ can verify the number of transactions that actually occurred between the two nodes in the given time slot, and the amount of feedback expected for that time slot.

We realize that because of the lack of exchange atomicity, after receiving the signed certificate from the client a malicious server might refuse to provide the client with a certificate signed by itself. In that case, the server will get only one opportunity to give bad feedback about the client. In addition, the client will know that the server is malicious and not transact with it in the future.
continues to do this with other nodes, several nodes will quickly realize that the server is malicious. Furthermore, since no transactions were actually completed with those nodes, the server would not achieve its goals of conducting many malicious transactions. On the other hand, if the server was to go ahead and complete the transaction, the client would not know for sure if the server is malicious and might transact with it in the future again. This will give the server more than one opportunity to give bad feedback about the client and conduct many malicious transactions. Therefore, we argue that it is in the malicious server’s interest to complete a transaction, so we do not need to enforce exchange atomicity and employ trusted third parties that must remain online frequently.

4. Evaluations of the Core SVM based Reputation System

4.1. Simulation setup

Our proposed mechanisms work above the network layer and do not deal with MANET routing (which by itself is a major problem and has been extensively investigated in the literature). We assume that the underlying network protocols will allow any two nodes to communicate with each other as long as the underlying topology is connected. If that is not the case, and any two nodes are unable to communicate with each other, then that means the network has been split into two or more disjoint subnetworks. In that case each subnetwork can be treated as an independent network, making the simulations appropriate for those (sub) networks. For the cases where nodes are able to communicate sometimes, and unable to communicate at other times because of mobility, we ignore unsuccessful communications since no transactions or feedbacks are occurring. We only consider successful transactions that lead to feedback. We assume that several such successful transactions occur between several pairs of nodes in any given time slot. That assumption is valid as long as a time slot is defined to be long enough (several minutes for instance), and the network is not so unreliable as to be disconnected most of the time.

We consider a relatively moderate size of network consisting of 1000 nodes and any node may want to interact with any other node through the underlying routing protocol (so they do not need to be direct neighbors). Time was divided into slots and in each time slot, several transactions were conducted between two randomly chosen pairs of nodes. Each node would then label the transaction as good or bad and store that label. The label may or may not reflect the true observation of a node, i.e., a node may lie about a transaction and give dishonest feedback (problem 1 described in Section 3).

**Good behavior:** Good behavior is characterized as a node conducting a normal transaction and giving honest feedback about it.

**Bad behavior:** Bad behavior is characterized as a node conducting a malicious transaction and/or giving dishonest feedback.

In addition, we introduced a random error of 5% to account for the fact that a node may incorrectly detect a transaction and mistakenly label it good or bad. This corresponds to problem 2 described in Section 3.

The simulation was allowed to run for several time slots and then data about each node was gathered. To gather data about a node \( x \), all of the other nodes in the network were queried and asked to give information about \( x \) within a certain number of past time slots. The total number of good and bad transactions conducted by \( x \) in a given time slot were accumulated and the proportion of positive feedback was computed. This computation was repeated for each time slot of interest. In this way a concise, aggregate historical record of \( x \) was obtained. The correct label of malicious or benign was assigned to \( x \) by us, based on its role in the simulation, for testing purposes only. The following attack scenarios were tested.

4.2. Attack scenarios

In each attack scenario, all the good nodes behave well consistently throughout the simulation, however the behavior of malicious nodes varies with the attack type as described below.

**Attack 1:** This is the simplest attack scenario in which all the malicious nodes consistently behave maliciously. These nodes do not collude amongst each other.

**Attack 2:** In this scenario, the behavior of malicious nodes oscillates between good and bad at regular intervals. The aim of the malicious node is to boost its reputation by being good first, and then use its high reputation to conduct malicious transactions. As a result, its reputation would decrease again, so it will oscillate into good behavior once again to boost its reputation. Again there is no collusion between the nodes.

**Attack 3:** This attack is similar to attack 2, except that now the malicious nodes collude with each other. Every time they happen to transact with each other, they recognize each other and leave positive feedback to boost each others’ scores. The nodes might recognize each other, for instance, if they belong to the same owner or colluding groups of owners.

**Attack 4:** This attack is more severe than attack 3 because this time whenever malicious nodes recognize each other, not only do they leave positive feedback about each other, but they also conduct further transactions with each other to leave even more positive feedback. But of course, there is a limit to the number of fake transactions they can conduct without being caught as obviously fake. In our simulations we conduct a random number of fake transactions, up to a maximum of ten, within one time slot.

**Attack 5:** In this attack scenario we combined all four types of malicious nodes described above. A quarter of all the malicious nodes behave as in attack 1, another quarter behave as in attack 2 and so on. In a real world setting we would not know which, if any, attack was being launched by any given node, so the performance of the RS in this attack scenario would tell us what would happen if all the attacks were conducted simultaneously.
4.3. Experiments and results

We evaluated our core SVM based RS against two other algorithms, TrustGuard Naive and TrustGuard TVM (Trust Value based credibility measure) [5]. Each of the five attack scenarios described above were tested. We set the same parameters for TrustGuard that their authors used in their paper. TrustGuard’s authors have shown that it performs very well compared to eBay’s Reputation System, which is commonly used as a benchmark in the literature for RSs. Therefore, we decided to directly compare our performance with TrustGuard, instead of eBay.

We collected data going back 15 time slots for each simulation run. For oscillating behavior, the period of oscillations was kept at less than 15 to ensure it was distinguishable from legitimate behavior. In the next section, we propose a system to overcome this limitation by looking at many more timeslots and compressing their data into a few features using Fading Memories. For SVM, a separate set of training data was also generated and SVM was trained on it using the Weka Machine Learning software [28]. The training data had a fixed proportion of malicious nodes (about 60%). For each node, its transaction history for the last 15 slots was fed into each RS. Then, using the output of the RS, a determination was made about whether the node was malicious or benign. For SVM this was done by looking at the distance between the test node and the decision boundary. If this distance was greater than a threshold, the node was considered benign. Larger thresholds result in fewer false positives, but also fewer true positives. This might be desirable in critical applications where we want to be sure that a node that is given access to a resource is indeed good, even if that means denying access to some legitimate nodes. We discuss thresholds in greater detail in the next section, when we introduce Dynamic Thresholds. TrustGuard also outputs a score that can be compared against a threshold and access can be granted if the score is greater than the fixed threshold.

4.3.1. Classification error

In the first set of experiments, the thresholds were fixed at their midpoint values so that the results were not artificially biased either towards increasing true positives (lower thresholds) or decreasing false positives (higher thresholds). Since the range of thresholds for SVM is \((-\infty, \infty)\), its threshold was set to 0. The range for TrustGuard is [0, 1], so its threshold was set to 0.5. Then the percentage of malicious nodes in the network was varied. The proportion of nodes that were misclassified, or the classification error, was measured. The results for each attack type are illustrated in Figs. 2–6. The results show that SVM significantly outperforms TrustGuard’s Naive and TVM algorithms for all attack types, even if the proportion of malicious nodes is very large (i.e., 80%). The difference is especially stark in attacks 2–5, when the attacker’s behavior oscillates between good and bad. It is also interesting to note that, with the exception of attack 1, there is not much difference between TrustGuard’s TVM and Naive algorithms, even though TVM is much more complex.

4.3.2. ROC curves

We generated ROC curves for all three RSs. ROC curves are commonly used in Machine Learning to evaluate classifiers, irrespective of the thresholds used. The curve is obtained by varying the threshold, so that we can compare how the true positive rate varies with the false positive rate. The area under the ROC curve shows how good a classifier is. Classifiers with larger areas under the curve are better.

![Classification Error for Attack 1](image-url)

Fig. 2. Classification error vs. proportion of malicious nodes for attack 1.
The ideal ROC curve is an upside down L-shaped curve, containing the point (0, 1) that corresponds to a 100% true positive rate and a 0% false positive rate.

Each point on the curve was obtained by running 30 simulations with different random number seeds, and then taking their mean. Confidence Intervals of 95% were taken around each point to ensure that the curve of SVM did not overlap with that of TrustGuard (the confidence intervals are too small to be visible on the graphs). The results are shown in Figs. 7–11, along with the diagonal random “Guessing” line. The results show that SVM outperforms TrustGuard, regardless of the thresholds used. The area
under the curve is greater for SVM than TrustGuard in all cases.

4.3.3. Bandwidth and computation overhead

Next we measured the bandwidth overhead involved in using SVM vs. TrustGuard. This overhead is due to passing feedback messages between the nodes. Therefore, we consider the average number of feedback messages exchanged to classify one node as the bandwidth overhead. Fig. 12 depicts the bandwidth overhead for three mechanisms. The results show that the overhead is the same for SVM and TrustGuard Naive (around 30), whereas the overhead for

---

Fig. 5. Classification error vs. proportion of malicious nodes for attack 4.

Fig. 6. Classification error vs. proportion of malicious nodes for attack 5.
TrustGuard TVM is much higher (around 960), since TVM traverses one more level to obtain further feedback about nodes that gave the original feedback.

Computation overhead is divided into two parts, namely online and offline. For online computations, TrustGuard and SVM mainly have the same overhead as they process the same amount of incoming feedback messages and compute trust scores based on a mathematical formula and ML model, respectively. To create and train the underlying ML model, SVM requires some offline computation too, which is not needed in TrustGuard. The offline computation is relatively time consuming. However, since it is done only once
at the RS vendor’s servers, it does not use scarce resources during live operation of a MANET.

5. Enhancing SVM with Fading Memories (SVM-FM)

As described in Section 3.1.1, each feature used in our SVM classifier corresponds to one time slot. This means that for n features, we can only look back at the behavior of a node going back n time slots. However, this may allow an adversary to have a fresh start because any behavior before that will be “forgotten,” so the adversary only needs to behave well for n time slots to get a clean slate. We would like our security system to be able to look back farther into the past so that it is not easy for an adversary to behave

Fig. 9. ROC curves for attack 3.

Fig. 10. ROC curves for attack 4.
well for a short period of time and then wipe its record clean. However, we cannot make $n$, the number of SVM features, arbitrarily large for several reasons:

- Using too many features may result in the well known Machine Learning problem of the “Curse of Dimensionality,” which causes accuracy to decrease as $n$ increases beyond a certain threshold [26].

- Storage requirements increase as $n$ increases because the nodes have to store $n$ features.

- Computational requirements increase as the SVM models become more complex. It takes longer to build new SVM models as well as to classify new nodes [26].

Ideally, we would like to use fewer features while retaining the ability to look back further into the past. In
other words, a feature should be able to summarize the data for several time slots into one value. To accomplish that goal, we build on the concepts presented in the Fading Memories algorithm of TrustGuard [5] and enhance them for use with SVM and digital signatures.

Fig. 13 illustrates the basic idea of Fading Memories. As explained in [5], the idea is to aggregate data over intervals of exponentially increasing length in the past \((k^0, k^1, \ldots, k^{m-1})\) into m values (for some integer \(k > 0\)). For more recent data, the interval size is smaller so fewer time slots are aggregated together resulting in greater precision. For older data, larger intervals are used so more time slots are aggregated together resulting in lower precision but greater compression. This enables us to retain more detailed information about the recent past, while storing less detailed information about older data. The value of \(k\) can be chosen so as to allow a tradeoff between compression and precision. We chose a value of 2 for simplicity.

More specifically, for a given node \(x\) during time slot \((t + 1)\), feature \(F_x^{t+1}[0]\) is initialized to \(p_x^t\), the proportion of positive feedbacks given to \(x\) during the previous time slot \(t\) (Eq. (1)). The remaining features are computed using Eq. (2). In TrustGuard all the features are updated after each time slot.

\[
F_x^{t+1}[0] = p_x^t \\
F_x^{t+1}[j] = \frac{(F_x^t[j] + (2^j - 1) + F_x^t[j - 1])}{2^j} \quad \text{for } j > 0
\]

The advantage of Fading Memories (FM) is that it allows us to look back further in time to evaluate a given node’s behavior. Without FM, an adversary would only need to behave well for a short period of time before its history is erased and then it can start behaving maliciously again. It can oscillate its behavior between good and bad indefinitely, as long as its period of good behavior is greater than the history we are looking at. Fading Memories is an attempt to elongate the history size so as to force the adversary into choosing longer periods of good behavior.

We were interested in finding out if Fading Memories would work with SVM, so we compared the performance of SVM with Fading Memories (SVM-FM) to SVM without Fading Memories.

5.1. Evaluating SVM with Fading Memories (SVM-FM)

5.1.1. Small fixed periods

Fading Memories (FM) aggregates data in several time slots resulting in lower precision as compared to one feature representing one time slot. As a result, we would expect the performance of SVM with FM to be worse than SVM without FM when dealing with small oscillation periods. We conducted experiments in order to test this. We generated data sets using simulations of Attack 5 in a network of 1000 nodes, similar to those described in the previous section. Once again, all the good nodes behaved well consistently, while malicious nodes oscillated their behavior with fixed periods of size \(n\), where \(n\) was chosen to be 8. They behaved well for four time slots, then maliciously for four time slots. The number of features used in both, SVM-FM and SVM without FM is also \(n\).

As described in Section 3.1, each feature in SVM without FM corresponded to the proportion of positive versus negative feedback in one time slot, whereas each feature in SVM-FM was computed using Eq. (2) for the new time slots. At the end of the simulation the features were collected and used in building and testing SVM models. The training and testing data sets were distinct and were generated using different random number seeds. The percentage of malicious nodes in the training data set was fixed at 50%. In the testing data sets, the percentage was varied and the classification error was plotted against it. The results are summarized in Fig. 14. They show that the change in error with FM is insignificant compared to that without FM. The difference between them is not statistically significant, so Fading Memories is a viable option for use with SVM even when the period of oscillation is small.

5.1.2. Long variable periods

The real benefit of SVM with FM is apparent when we vary the period of oscillation. To illustrate this, another set of experiments were conducted where the data sets were generated as before, but the period of oscillation for each malicious node was randomly selected up to a maximum of 250 time slots for good behavior and 250 time slots for bad behavior. Given eight features and \(k = 2\), with FM we can look back at \(2^8 - 1 = 255\) time slots, whereas we can only look back at the last eight time slots without FM. As a result we would expect a significant improvement in the classification error with FM. We tested this through simulations and the results are plotted for each of the five attack patterns in Figs. 15–19. The results clearly show the advantage of using SVM-FM when the periods of oscillation are long. The error with SVM-FM is much lower than the error without FM, especially at smaller percentages of malicious nodes. In attack 1, there are no oscillations and the nodes have consistent behavior. By reducing the error for longer periods, we can successfully force the adversary to behave well for longer periods of time.

As seen in Figs. 15–19, the error keeps rising as the percentage of malicious nodes increases until a point where almost all the good nodes are being classified as bad. This happens when malicious nodes overwhelm the network and leave too much negative feedback. SVM concludes that almost every node is malicious, which happens at around
70%. After 70%, virtually all the nodes are classified as bad. The error then becomes proportional to the percentage of good nodes in the network. As the percentage of good nodes decreases, so does the error, which explains the dip in error. We combat this phenomenon using Dynamic Thresholds as discussed in Section 6.

Fig. 14. Classification error vs. percentage of malicious nodes for small, fixed periods of oscillation for SVM with FM and SVM without FM.

Fig. 15. Classification error vs. percentage of malicious nodes for large, variable periods of oscillation for SVM with FM (FM error) and SVM without FM (Orig error). Attack 1 has no oscillations.
5.2. Fading Memories enhanced with digital signatures

In Section 5, we showed that by using Fading Memories we can successfully compress the data in several time slots into a few features that SVM can use. This helps us in using longer histories without needing extra storage to store those histories. However, we run into a problem when we try to use Fading Memories along with the Digital Signature scheme presented in Section 3.2. The digital signature scheme is needed to protect against fake feedback, so that one can verify that a given feedback pertains to a transaction that actually took place. As mentioned in

Fig. 16. Classification error vs. percentage of malicious nodes for large, variable periods of oscillation in attack 2.

Fig. 17. Classification error vs. percentage of malicious nodes for large, variable periods of oscillation in attack 3.
Section 3.2, at the beginning of every transaction the client and the server exchange digital certificates. The certificate fields are:

\[ (\text{Server ID, Client ID, Time Slot ID, Transaction Number}) \]

\[ (\text{Cert.1}) \]

Transaction number tells us how many transactions took place in the given time slot. If a new transaction occurs in the same time slot, the transaction number is incremented by one. This in turn corresponds to the amount of feedback that one of the nodes, let's say the client, can legitimately give about the other node (the server) for that time.

Fig. 18. Classification error vs. percentage of malicious nodes for large, variable periods of oscillation in attack 4.

Fig. 19. Classification error vs. percentage of malicious nodes for large, variable periods of oscillation in attack 5.
slot. But when we start aggregating the data in different time slots in Fading Memories, we run into the problem of trying to aggregate two or more digital certificates into one. As the client moves into a new time slot \( t + 1 \), it must aggregate the data in time slot \( t \) with the data in time slot \( t - 1 \), and continue aggregating the data in time slots further back (Fig. 13). However, it is unable to combine the digital certificate for \( t \) with the certificate for \( t - 1 \) to get a single certificate because the new certificate would not have a valid signature by the server. As a result, it would be forced to store the certificates for each time slot individually, defeating the purpose of using Fading Memories.

To circumvent this problem, we propose a new protocol for the digital signature scheme. Suppose \( n \) features are being used by SVM and the system is currently in time slot \( t \). The client and the server will transact for the first time and exchange certificates similar to Cert. (1). If they transact again within the same time slot, they will update this certificate by incrementing the number of transactions and re-signing it. Now let’s assume that at some point in the future, \( t + x \), the client again transacts with the server. This time, the client and the server both update their certificates by moving along the time line and using the update step of Fading Memories (Eq. (2)). This will allow them to aggregate all the previous time slots according to the Fading Memories formula and then insert the current time slot as a new feature. Since both the client and the server use the same formula, both will end up with the same results about the aggregated number of transactions in previous time slots. Then each will present this new certificate for the other to sign. This new certificate will have the following fields:

\[
\text{(Server ID, Client ID, Time Slot ID, } N_1, N_2, \ldots, N_n \text{)}
\]

(Cert.2)

where \( N_i \) is the aggregate number of transactions that took place for feature slot \( i \) (not time slot), beginning at Time Slot ID. Although this increases the size of each certificate by \( n - 1 \) fields, note that only one certificate now needs to be stored by the client for all the transactions it has conducted with the server. In the original scheme, it stored one certificate per time slot, so we have considerable savings in storage space with this new scheme.

The certificate is updated whenever a new transaction occurs with the server, so the certificate can tell us the number of transactions in each previous feature slot. Of course, this certificate may be requested at any point in the future after the given time slot has expired. In that case, the receiving node will start at time slot ID and update the number of transactions and their feedback data until the present time slot, using (Eq. (2)).

Using this new protocol, we can use digital signatures to detect fake feedback while still using Fading Memories to look at longer histories without increasing storage requirements.

6. Introducing Dynamic Thresholds with SVM

Figs. 20–22 compare SVM to TrustGuard Naive and TVM. Fading Memories (FMs) are enabled in all. We also included the biased SVM with FM to see how it compares to other mechanisms. We noticed that in Figs. 21 and 22, the reason why the default SVM model performed poorly when the malicious nodes percentage was high in attacks 3 and 5 was because it was misclassifying the positive (non-malicious) instances. Therefore, we increased the threshold \( b \) in the linear SVM equation, so that those instances close to the SVM boundary are classified as positive. The equation of a linear SVM is [26]:

\[
w \cdot x + b \geq 0 \quad \text{for positive class} \\
w \cdot x + b < 0 \quad \text{for negative class,}
\]

where \( x \) is the instance vector, \( w \) is the normal to the SVM hyperplane, and \( b \) is a bias threshold. By increasing \( b \), we effectively trade false negatives with false positives which results in an advantage whenever false negatives are much higher in number than false positives. The results of this tradeoff are shown in Figs. 21 and 22 (attacks 3 and 5).

As expected, the bias pays off at higher proportions of malicious nodes when there are more false negatives than false positives. However, it costs us when the proportion of malicious nodes is small since there are more false positives than false negatives. This increases the error for small proportions. This observation led us to the idea that if we could know the proportion of malicious nodes in the network, we could adjust the bias threshold accordingly to improve accuracy. To achieve this goal, we propose a scheme called “Dynamic Threshold” that tries to estimate the proportions of malicious nodes and then decreases the threshold in case of fewer malicious nodes while increasing it in case of more malicious nodes.

To begin with, we used a brute force trial and error approach and discovered what the ideal thresholds were for given proportions of malicious nodes. The ideal threshold is defined as that threshold which maximizes accuracy. We conducted several experiments where the threshold was varied and the threshold that gave the minimum error was noted for each proportion of malicious nodes. For our tests, we used the model that was trained using attack 5 since it combines all of the other four attack patterns. We then plotted the ideal thresholds versus the proportions of malicious nodes, as seen in Fig. 23. The graph is a straight line, so once the data points are plotted, it is easy to compute the equation of the line (by computing the slope and intercept). The equation will be different for different SVM models, so we need to determine at least two data points (but preferably more) by the brute force approach. Using that equation, we can then calculate the appropriate threshold for any percentage of malicious nodes.

Fig. 24 shows the reduction in error achieved using the optimum thresholds versus the default threshold of zero. The results clearly show that Dynamic Thresholds are very useful for significantly reducing the error. However, the challenge is that in a real world setting, we do not know the proportion of malicious nodes in the network and therefore, we cannot decide what threshold to use. To overcome this, we propose estimating the proportion of malicious nodes through sampling.
The idea is that a new node that joins the network would initially use the default threshold of zero. It would conduct transactions as usual and estimate the proportion of malicious nodes from all the nodes it has interacted with. A node is considered malicious if either the Reputation System classifies it as malicious, or if a transaction is conducted with the node and the transaction is deemed to be malicious. Once a large enough sample is collected...
by the node, it can use that sample to estimate the proportion of malicious nodes in the network and then dynamically adjust its threshold to improve the RS's accuracy.

We conducted experiments to determine what a good sample size would be before adjusting the threshold. We ran simulations with different proportions of malicious nodes in the network. Then we randomly sampled the nodes and determined which of them were malicious. Based on the proportion of malicious nodes in our random sample, we estimated the actual proportion in the entire

![Fig. 22. Classification error vs. malicious nodes percentage for different algorithms with FM in attack 5.](image)

![Fig. 23. SVM's best threshold vs. malicious nodes percentage for attack 5.](image)
network. Next, we used the estimated proportion to decide which threshold to use. Finally, we obtained the RS error over all the nodes using that threshold. The sample size was varied and the error plotted against it. The results are plotted in Fig. 25. The results show that the error is erratic and unstable until a sample size of about 20. After 20 samples, a fairly good estimate of the actual proportion can be obtained. We therefore recommend that nodes should obtain a sample of at least 20 before adjusting their thresholds.

Fig. 24. SVM error under the default threshold and under the best thresholds.

Fig. 25. Dynamic Thresholds error vs. number of Samples taken for different proportions of malicious nodes in the network.
**Fig. 26** shows the reduction in error when Dynamic Thresholds are put into practice. Sample sizes of 20 and 25 were used. These samples were randomly collected and classified, based on a node’s interactions with other nodes, and used to estimate the proportion of malicious nodes in the network. The threshold was adjusted based on the estimated proportions. The results show a significant reduction in error even at high proportions. Once the threshold has been adjusted, the sample is discarded and a fresh sample is started. In this way, the node can continuously monitor the proportion of malicious nodes and adjust its threshold as the proportion changes.

### 7. Conclusions and future work

In this paper, we proposed a Machine Learning based Reputation System (RS) that can be used to guard against malicious insider nodes in mission critical networks. The nodes can try to attack the network by conducting malicious transactions, or spreading viruses and worms, or attacking known vulnerabilities. Although there is no way of knowing whether a future transaction will be malicious or not, we can design and use an RS to predict the future behavior of a node by observing its past behavior.

We discussed how and why using Machine Learning (ML), and especially Support Vector Machines (SVM), can provide a good approach for designing RSs. We then proposed our SVM based core RS along with various mechanisms to deal with the challenges posed by MANETs. Through extensive simulations, we evaluated our core SVM based RS and compared it against two other algorithms found in the literature, called TrustGuard Naive and TrustGuard TVM. We showed that our model achieves high accuracy and correctly predicts good vs. malicious nodes under various attack scenarios, whether there is oscillating or steady behavior, collusive or non-collusive behavior, or even when the proportion of malicious nodes in the network is very high. The ROC curves showed that the improvement of SVM over TrustGuard is statistically significant, as their 95% confidence intervals do not overlap each other. We also showed that SVM has the same bandwidth overhead as TrustGuard Naive, but much less overhead than TrustGuard TVM. Then we enhanced the core SVM based RS with Fading Memories and modified our digital signature scheme so that we could look back at longer histories without using too many features. We evaluated its performance and showed that it was much better at detecting malicious behavior that varied over longer periods. Finally, we introduced a new technique (namely, Dynamic Thresholds), which dynamically adjusts the SVM threshold based on the estimated proportion of malicious nodes in the network. We showed that Dynamic Thresholds can significantly reduce the classification error.

Overall, we believe that the proposed SVM based RS, along with various enhancements, provides a flexible and extensible model for designing and implementing RSs for mission critical closed MANETs under certain assumptions. Clearly, the proposed solution is not the ultimate one or the complete one to address all the problems. Rather, it is an important step towards understanding the key issues and developing necessary mechanisms in a Reputation...
System for MANETs. Accordingly, it is open to further improvements. For example, as future work, we plan to enhance our classifier by taking into account the reliability of the agent providing the feedback. This is done in TrustGuard’s Personalized Similarity Measure (PSM) algorithm, however the overhead involved in PSM is similar to TVM since we have to recursively collect further feedback about agents that gave us the original feedback. In the future, we will study and try to minimize the overhead associated with collecting these features. We will also consider new attacks and train our model to detect such new attacks. We also need to relax some of the assumptions and further investigate how to integrate our proposed mechanisms into routing and possibly below layers for better performance in highly mobile networks.

Acknowledgements

This work is supported in part by the National Science Foundation and under Grant CRI-0551501 and by DoD Infrastructure Support Program for HBCU/MI under Grant 54477-CI-ISP (UNCLASSIFIED). The authors thank the anonymous reviewers for their careful evaluations of the paper and for the comments that helped us improve the quality of the paper. We would also like to thank Justin Chester, Drew Wichmann, and Fatma T. Arslan Korkmaz for their help in proofreading the paper.

References


Rehan Akbani received a B.Sc. degree in Computer Science with Honors in 2002 from the University of Texas at San Antonio. He graduated Summa Cum Laude. He received his M.Sc. degree in Computer Science in 2005. He received his PhD in August 2009 from the University of Texas at San Antonio. He is currently a postdoctoral fellow in the Dept. of Bioinformatics and Computational Biology at the University of Texas MD Anderson Cancer Center, Houston, TX. His research interests include bioinformatics, MANETs and wireless security, trust management, cryptography and Machine Learning.
Turgay Korkmaz received the B.Sc. degree with the first ranking from Computer Science and Engineering at Hacettepe University, Ankara, Turkey, in 1994, and two M.Sc. degrees from Computer Engineering at Bilkent University, Ankara, and Computer and Information Science at Syracuse University, Syracuse, NY, in 1996 and 1997, respectively. In December 2001, Dr. Korkmaz received his PhD degree from Elec. and Computer Eng. at University of Arizona, under the supervision of Dr. Marwan Krunz. In January 2002, he joined the University of Texas at San Antonio, where he is currently an Associate Professor of Computer Science department. Dr. Korkmaz works in the area of computer networks. Specifically, he is interested in quality-of-services (QoS)-based networking issues in both wireline and wireless networks. He is also working on network security, network measurement and modeling and Internet related technologies.

G.V.S. Raju is currently Peter T. Flawn Distinguished Professor at The University of Texas at San Antonio (UTSA). He is a Life Fellow of IEEE and he received MILLENNIUM MEDAL from IEEE in 2000. Over the years he was an investigator, co-PI and PI to multi-million-dollar grants from NASA, DoD, FAA, DoE, NSF and industry. With this extramural funding, Dr. Raju was able to conduct basic theory and applied research in the areas of Computer Communications Networks, networks security, wireless communications and intelligent systems. He is the author of numerous technical publications in refereed journals and peer-reviewed conference proceedings. He is listed in American Men and Women in Science, Who’s Who in America, the Dictionary of International Biography, and Outstanding Intellectuals of 21st century.