Self-Organizing Maps versus Growing Neural Gas in Detecting Data Outliers for Security Applications

Zorana Banković, David Fraga, Juan Carlos Vallejo, and José M. Moya

ETSI Telecomunicación, Univ. Politécnica de Madrid,
Av. Complutense 30, 28040 Madrid, Spain
{zorana,dfraga,jcvallejo,josem}@die.upm.es

Abstract. Our previous work has demonstrated that clustering-based outlier detection approach offers numerous advantages for detecting attacks in Wireless Sensor Networks, above all adaptability and the possibility to detect unknown attacks. In this work we provide a comparison of Self-organizing maps (SOM) and Growing Neural Gas (GNG) used for this purpose. Our results reveal that GNG is superior to SOM when it comes to the level of presence of anomalous data during the training, as GNG is capable of detecting the attack even with small portion of normal data during the training, while SOM need the majority of the training data to be normal in order to detect it. On the other hand, after both being trained with normal data, SOM performs somewhat better as the attack becomes more aggressive, i.e. it exhibits higher detection rate, although both are capable of detecting the attack in each case.

Keywords: Self-organizing maps, growing neural gas, outliers, wireless sensor networks.

1 Introduction

In our previous work [1] we have demonstrated that clustering-based outlier detection approach offers numerous advantages for detecting attacks in Wireless Sensor Networks (WSN), such as high adaptability, flexibility, possibility to detect unknown attacks, no restrictions on training data, etc. Regarding clustering, the possibilities are to deploy techniques such as *k*-means, *k-NN*, but also the topology-preserving competitive methods, such as Self-organizing maps (SOM) [2], or Growing Neural Gas (GNG) [3].

The topology preserving techniques are very convenient for our application, since one of the main parameters that reveal the presence of outliers is the average distance of a cluster to its closest neighbors. In the case of topology preserving techniques it is very well known which the closest clusters are, thus making the calculation of the mentioned parameter straightforward. Furthermore, the fact that both techniques use exponentially decaying learning rate makes them less susceptible to the issue of poor initialization. As *k*-means suffers from this problem, their advantage to it is obvious. On the other hand, *k-NN* has been discarded from the start due to its high memory

© Springer-Verlag Berlin Heidelberg 2012

consumption also during the testing process, which is unacceptable in resource limited environments such as WSNs.

SOM and GNG mainly differ in the fact that the size of SOM is fixed from the start, while the size of GNG grows during the training. Fixed size can be a limitation, as it is not possible to know the optimal number of clusters from the start. In order to overcome this issue in the case of SOM, we have deployed genetic algorithm (GA) that in essence searches for the optimal clustering. However, GA consumes lots of resources, which limits its application in WSNs. For this reason, we have implemented GNG in order to overcome the issues of both SOM and GA.

The rest of the work is organized as follows. Section 2 provides some basic information on the previous work in WSN security. Section 3 gives more detail to the description of the problem and on the implemented algorithms, while Section 4 provides experimental evaluation. Finally, conclusions are drawn in Section 5.

2 Previous Work

Recently few solutions that deploy machine learning techniques appeared [7], [10]. Among these solutions we can also find a few anomaly based solutions [8], [9], [11] that claim of having the possibility to detect unknown attacks. They uphold the idea that machine learning techniques offer higher level of flexibility and adaptability to the changes of the environment, as retraining can be done automatically.

However, the feature sets they deploy mostly include those features that capture the properties of known attacks, i.e. those that are known to change under the influence of an attacker, or are known to be weak spots. This is their major deficiency, as relying on these features only the known attacks or their variations can be detected. Furthermore, it assumes that an attacker can exploit only the known vulnerabilities, but general experience is that vulnerability is detected after being exploited by an adversary. Some of them assume that the feature sets can be expanded [7], yet this has to be done through a human intervention.

Thus, regarding previous work on WSN security, we can say that the main deficiencies of the known solutions are: the scope of attacks they can detect is limited and their adaptation has to be performed through human interaction. Thus, our aim is to provide a machine learning based solution that does not suffer from these issues, i.e. a solution that would be capable of detecting wide range of attacks, including the previously unseen ones, which would also be adaptable automatically.

3 Problem Definition

In order to provide uninterrupted network operation in WSNs, core network protocols (aggregation, routing and time synchronization) have to be secured. Regarding the attacks on the aggregation protocol, we assume that they demonstrate themselves in skewed aggregated values, which can be the result of either a number of skewed sensed values, or a compromised aggregated node. The assumption is very reasonable, having in mind that the main objective of these attacks is to provide wrong picture of the observed phenomenon.

On the other hand, in time critical systems it is mandatory to receive information within certain time window. If the attacker manages to introduce delays or desynchronize clock signal in various nodes, the received critical information will not be up to date, which can destabilize the system. Also, if the received information is not up to date, the aggregated value will be skewed, as it will also be out of date. For these reasons, and given the existing redundancy in WSNs, we believe that these attacks can be detected as temporal and/or spatial inconsistencies of sensed values.

Regarding attacks on routing protocols [4], we assume that they will introduce new and different paths than those that have been seen before. Here we have attacks whose main objective is to compromise the routing protocol, and they usually do it by spoofing or altering the data stored in the routing tables of the nodes. Thus, the resulting routing paths will be different from those used in a normal situation. In the case of wormhole for example, two nodes that are not within each other's radio range result in consecutive routing hops in routing paths, which is not possible in a normal situation. From these examples we can see that the assumption about the attacks resulting in routing paths different from those that appear in normal situation is reasonable. Thus, in this case we want to detect temporal inconsistencies in paths used by each node.

3.1 Feature Extraction and Formation of Model

Following the idea of temporal and/or spatial inconsistency in the presence of attackers, we want to provide the model of the data that would capture these properties and allow us to deploy machine learning.

For the case of sensed values, we follow the idea presented in our previous work [1] based on extracted *n*-grams and their frequencies within different time windows. For the purpose of illustration, we will give a short example for a sensor that detects presence. Let the sensor give the following output during the time window of size 20: 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0. If we fix the *n*-gram size on 3, we extract all the sequences of size 3 each time moving one position forward. In this way we can observe the following sequences and the number of their occurrences within the time window: 111 – occurs 6 times, 110 – 2, 100 – 2, 000 – 6, 001 – 1, 011 – 1. Thus, we can assign them the following sequences: 111 – 0.33, 110 – 0.11, 100 – 0.11, 000 – 0.33, 001 – 0.05, 011 – 0.05. In our model, the sequences are the features and their frequencies are the corresponding feature values. Thus, the sum of the feature values is always equal to 1. This characterization is performed in predefined moments of time and takes the established amount of previous data, e.g. we can perform the characterization after every 20 time periods based on previous 40 values.

In a similar fashion, we form features for spatial characterization. The first step is to establish vicinities of nodes that historically have been giving consistent information. In this way, an *n*-gram for spatial characterization in a moment of time is made of the sensor outputs from that very moment. For example, if sensors S1, S2, S3 that belong to the same group each give the following output: 1 1 1 0 during four time epochs, we characterize them with the following set of *n*-grams (each *n*-gram contains at the first position the value of S1, the value of S2 at the second and the value of S3 at the third at a certain time epoch): 111 – occurs 3 times, 000 – occurs once, thus the feature value of each *n*-gram is: 111 – 0.75, 000 – 0.25, i.e. the frequencies within the observed period of time.

The same principle is followed for characterizing routes that a node has been using to send its sensed data to the sink. Each routing hop adds its ID to the message that is further forwarded, so the sink gets the information about the routing path together with the message. Each sensor has its own model and each feature, i.e. n-gram in the model consists of a predefined number of successive hops used in routing information coming from the node. For example, if during the characterization time, the node has used the following paths for routing its data to the sink: A-B-C-S – 3 times, A-D-E-F-S – 2 times, A-B-E-F-S – 1 time (A – the node that is sending the data, B, C, … - other nodes in the network, S- sink), we can characterize the routing with the following n-grams (n=3): ABC, BCS, ADE, DEF, EFS, ABE and BEF. In all of the routes, the *n*-gram ABC occurs 3 times, BCS – 3, ADE – 2, DEF - 2, EFS – 3, ABE – 1, BEF – 1. The total number of *n*-grams is 15, so dividing the values given above with 15, we get the frequencies of each *n*-gram which are the values that we assign to out features, i.e. *n*-grams.

It is important to notice that the extracted feature vectors will not be of the same size, so we are not able to use standard distance functions. For this reason, we calculate distance using the approach presented in [5], which calculates distance between sequences.

3.2 Attack Detection

We treat attacks as data outliers and deploy clustering techniques. There are two possible approaches for detecting outliers using clustering techniques [6] depending on the following two possibilities: detecting outlying clusters or detecting outlying data that belong to non-outlying clusters.

The important points necessary for the understanding of the principle is the deployed distance function [5], which is equivalent to Manhattan distance after making the following assumption: the feature that does not exist in the first vector while exists in the second (and vice versa) actually exists with the value equal to 0, since we can say that it occurs with 0 frequency. In this way, we get two vectors of the same size and the distance between the centre and an input is between 0 (when they are formed of the same features with the same feature values) and 2 (when the features with the values greater than 0 are completely different). In the same way, if the set of the features of one is the subset of the feature set of the other, the distance will be between 0 and 1.

In during the testing, different *n*-grams occur in an input, that can happen when the node starts sending data significantly different than before or starts using different routes to send the data, the distance, which is the *QE* value defined previously, between it and its corresponding centre will be greater than 1. This can serve as evidence of abnormal activities happening in the node or in its routing paths. It is also a typical case when the training is performed with clean data.

3.3 Recovery from Attacks

Every sensor node is being examined by agents that execute clustering algorithms and reside on nodes in its vicinity and listen to its communication. The agents are trained separately. The system of agents is coupled with a reputation system where each node has its reputation value that basically reflects the level of confidence that others have in it based on its previous behavior. In our proposal, the output of an agent affects on the reputation system in the way that it assigns lower reputation to the nodes where it detects abnormal activities and vice versa. We further advocate avoiding any kind of interaction with the low-reputation nodes: to discard any data or request coming from these nodes or to avoid taking them as a routing hop. In this way, compromised nodes remain isolated from the network and have no role in its further performance.

In this work the reputation is calculated in the following way. For the reasons explained in the previous chapter, the value (`rep`) for updating overall reputation based on *QE* is calculated in the following way:

```
if (QE<1) rep = 1; else rep=1-QE/2;
```

There are two functions for updating the overall reputation of the node, depending whether the current reputation is below or above the established threshold that distinguishes normal and anomalous behavior. If the current reputation is above the threshold and the node starts behaving suspiciously, its reputation will fall quickly. On the other hand, if the reputation is lower than the established threshold, and the node starts behaving properly, it will need to behave properly for some time until it reaches the threshold in order to “redeem” itself. In order to achieve this, we use the function x+log(1.2*x) because it provides what we want to accomplish: if x is higher than 0.5, the output rises quickly, so the reputation rises; if x is around 0.5, the output is around 0, so the reputation will not change its value significantly; if x is smaller than 0.4, the output falls below 0. Finally, the reputation is updated in the following way:

```
if (last_reputation[node]>threshold)
new_reputation[node]=last_reputation[node]+rep+log(1.2*rep);
else
new_reputation[node]=last_reputation[node]+0.05*(rep+log(1.2*rep));
```

If the final value falls out from the [0, 1] range, it is rounded to 0 if it is lower than 0 or to 1 in the opposite case.

However, if during the testing of temporal coherence, we get normal data different from those that the clustering algorithms saw during the training, it is possible to get high *QE* value as well. On the other hand, the spatial coherence should not detect any anomalies. Thus, the final reputation will fall only if both spatial and temporal algorithms detect anomalies. This is implemented in the following way:

```
if (value_rep <  threshold)
{     if ( space_rep <  threshold) result =    value_rep;
       else result = 1 - value_rep; }
else result = value_rep;
```

where `value_rep` is the reputation assigned by the algorithms for temporal characterization and `space_rep` is the reputation assigned by the algorithms for spatial characterization.

Concerning the detection of routing protocol anomalies, the explained approach can tell us if there is something suspicious in routing paths of a certain node. Yet, it order to find out the nodes that are the origin of the attack, we need to add one more step. In the second step, if the reputation of the routes calculated in the previous step is lower then the established threshold, the hops that participated in bad routes will be added to the global list of bad nodes, or if they already exist, the number of their appearance in bad routes is increased. The similar principle is performed for the correct nodes. For each node, let the number of its appearances in bad routes be *nBad* and the number of its appearances in good routes be *nGood*. Finally, if *nGood* is greater than *nBad*, the node keeps its reputation value, and in the opposite case, it is assigned the following reputation value: *nGood* / (*nGood* + *nBad*). In this way, as the bad node spreads its malicious behavior, its reputation will gradually decrease.

3.4 Developed Techniques

Both SOM [2] and GNG [3] algorithm follow the standard steps. The only problem-specific point is the centre, i.e. node representation and updating. Each centre is implemented as a collection whose size can be changed on the fly and whose elements are the *n*-grams defined in the previous text with assigned occurrence or frequency. The adjustment of nodes (that belong to the map area to be adjusted) is performed in the following way:

- If an *n*-gram of the input instance $v(t)$ exists in the node, its $\varphi(x)$ is modified according to the centre update[2, 3];
- If an *n*-gram of the instance $v(t)$ does not exist in the cluster centre, the *n*-gram is added to the centre with occurrence equal to 1.

4 Experimental Evaluation

The proposed algorithm has been tested on the reputation systems simulator developed by our research group and designed using the C++ programming language. The network consists of a number of distributed nodes. The reputation can be calculated in various ways, which are the implementation of the class *ReputationServer*. The time in the simulator is measured in time ticks, where one tick is equivalent to one sensing period in WSNs. In this work the algorithms have been tested in the presence of the Sybil attack[4], where the compromised node pretends to have multiple IDs, either false, i.e. *fabricated*, or impersonated from other legitimate nodes, i.e. *stolen IDs*. The attacker can affect on many aspects of network (aggregation, routing, etc.).

In the following experiments we will present the performance of the approach in various scenarios, varying the attack strength (Fig. 1) while training with clean data, and varying the starting point of the attack (Fig. 2) while keeping the attack strength. There will be two typical situations: in the first case the attack will start after the end of training, so the training will be performed with "clean" data, while in the second case we will have the situations where the training data contains the traces of attacks as well. The scenario is based on 200 entities that can take one of the possible 2000 positions.

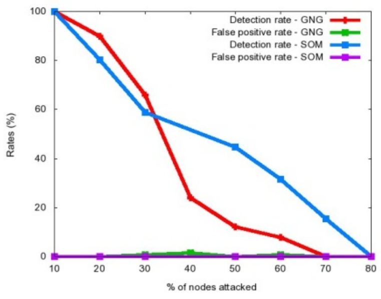

Fig. 1. Detection and false positive rate vs. attack strength

We can observe from Fig. 1 that GNG exhibits higher detection rate than SOM when they deal with the attack of lower strengths (up to 30% of nodes attacked), while for the stronger attacks SOM is clearly the winner. This is probably the result of performing the training with clean data, during which GNG results in having fewer clusters than SOM, thus not being able to detect more subtle differences. On the other hand, when training with a mixture of clean and unclean data, GNG results in having more clusters, thus in this case it is the one being capable of detecting more subtle differences. This is demonstrated in Fig. 2, where we can observe that SOM is capable of isolating the attack if down to 60% of data during the training is normal, while GNG goes down to 5%. Also, the time of isolating the attack of GNG is much lower than the isolation time of SOM. Thus, in this case GNG is clearly the winner.

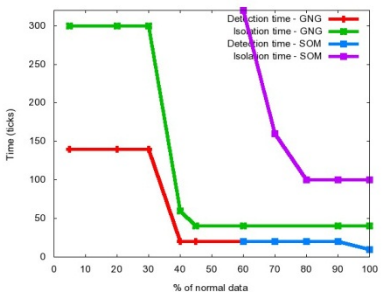

Fig. 2. Detection and isolation times vs. % of normal data during the training

5 Conclusions

In this work we have provided the comparison of SOM and GNG algorithms for clustering-based detection of outliers in its application for security of WSNs. Our results reveal that GNG is superior to SOM when it comes to the level of presence of anomalous data during the training, as GNG is capable of detecting the attack even with small portion of normal data during the training (down to 5%), while SOM need the majority of the training data to be normal (down to 60%) in order to be able to detect it. On the other hand, after both being trained with normal data, SOM performs somewhat better as the attack becomes more aggressive, i.e. it exhibits higher detection rate, although both are capable of detecting the attack in each case.

Thus, based on the presented results, in a general (unknown) case GNG would be more appropriate. However, a combination of both techniques could improve the performances of GNG in certain scenarios. For this reason, in the future we will dedicate our efforts towards obtaining such combination.

Acknowledgments. This work was funded by the Spanish Ministry of Science and Innovation, under Research Grant AMILCAR TEC2009-14595-C02-01.

References

1. Banković, Z., Moya, J.M., Araujo, A., Fraga, D., Vallejo, J.C., de Goyeneche, J.M.: Distributed Intrusion Detection System for WSNs based on a Reputation System coupled with Kernel Self-Organizing Maps. Int. Comp. Aided Design 17(2), 87–102 (2010)
2. Haykin, S.: Neural networks - A comprehensive foundation, 2nd edn. Prentice-Hall (1999)
3. Fritzke, B.: Growing Neural Gas Network Learns Topologies. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems, vol. 7, pp. 625–632. MIT Press, Cambridge (1995)
4. Roosta, T.G.: Attacks and Defenses on Ubiquitous Sensor Networks, Ph. D. Dissertation. University of California at Berkeley (2008)
5. Rieck, K., Laskov, P.: Linear Time Computation of Similarity for Sequential Data. J. Mach. Learn. Res. 9, 23–48
6. Muñoz, A., Muruzábal, J.: Self-Organizing Maps for Outlier Detection. Neurocomputing 18(1-3), 33–60 (1998)
7. Krontiris, I., Giannetsos, T., Dimitriou, T.: LIDeA: A Distributed Lightweight Intrusion Detection Architecture for Sensor Networks. In: 4th International Conference on Security and Privacy for Communication Networks. ACM (2008)
8. Onat, I., Miri, A.: A Real-Time Node-Based Traffic Anomaly Detection Algorithm for Wireless Sensor Networks. In: Systems Communications, pp. 422–427. IEEE Press (2005)
9. Kaplantzis, S., Shilton, A., Mani, N.: Sekercioglu, Y.A.:Detecting Selective Forwarding Attacks in WSNs using Support Vector Machines. In: Proc. Conf. Int. Sensors, Sensor Networks and Inf., pp. 335–340. IEEE Press (2007)
10. Wallenta, C., Kim, J., Bentley, P.J., Hailes, S.: Detecting Interest Cache Poisoning in Sensor Networks using an Artificial Immune Algorithm. Appl. Intell. 32, 1–26 (2010)
11. Loo, C.E., Ng, M.Y., Leckie, C., Palaniswami, M.: Intrusion Detection for Routing Attacks in Sensor Networks. Int. J. of Dist. Sens. Net. 2(4), 313–332 (2006)