Improving Self Organizing Map Performance for Network Intrusion Detection

Stefano Zanero*

Abstract

The continuous evolution of the types of attacks against computer networks suggests a paradigmatic shift from misuse based intrusion detection system to anomaly based systems. Unsupervised learning algorithms are natural candidates for this task, but while they have been successfully applied in host-based intrusion detection, network-based applications are more difficult, for a variety of reasons, including performance. We propose an architecture which implements a network-based, anomaly based intrusion detection system, which uses unsupervised learning algorithms. In this paper we describe the improvements and modifications needed in order to increase the throughput of a Self Organizing Map algorithm and make it able to handle high dimensional input data at a rate suitable for Intrusion Detection purposes at network speed.

1 Introduction and motivations

The continuous evolution of the threats against computer networks requires a paradigmatic shift from misuse based intrusion detection system to anomaly based systems. The "misuse detection" approach, which tries to define what constitutes an attack in order to detect it directly, has been widely successful. Most modern intrusion detection tools are misuse based, but they are increasingly showing the limits of this paradigm.

Whenever a new attack is discovered, the knowledge base of misuse IDSs must be updated in order to keep the systems effective. In addition, there is also an unknown number of discovered, but undisclosed, vulnerabilities (the so called "zero-days") that are not available to the experts for analysis and inclusion in the knowledge base [1]. Most attacks are also polymorph, and skilled attackers can exploit this polymorphism to evade detection [2, 3].

Since, by their own nature, Intrusion Detection Systems are intended to be a complementary security measure, which can detect the failures of other measures, the inability to detect unknown attacks or new ways to exploit an old vulnerability is an unacceptable limitation. The obvious solution seems then to implement an anomaly detection approach, modeling what is *normal* instead than what is *anomalous*. This is surprisingly similar to the earliest conceptions of what an IDS should do [4].

However, while a number of host based intrusion detection systems have been proposed and implemented, both in literature and in practice, network based anomaly detection is still an open field for research. In a previous work [5], we proposed a novel architecture for applying unsupervised learning and data mining techniques to a network based IDS. Unsupervised learning techniques are natural candidates for this type of task, but while they have been successfully applied in host based intrusion detection [6], their application to network based systems is still troublesome, mainly due to the problems of input selection, data dimensionality and throughput.

In this paper we describe the improvements and modifications we applied to a Self Organizing Map algorithm in order to increase its throughput to handle high dimensional data at a speed suitable for Network Intrusion Detection purposes. We describe various heuristics that can be used, and their effect on the accuracy of the algorithms. We also propose some performance tests that demonstrate that our heuristics do not diminish the overall effectiveness of the IDS.

The remainder of the paper is organized as follows: in Section 2 we describe the proposed architecture of our IDS; in Section 3 we describe how the curse of dimensionality affects the performance of the first stage of the architecture; in Section 4 we describe our heuristics and the associated performance improvements; in Section 5 we discuss the problem of significance of euclidean metrics in our high-dimensional space; in Section 6 we report on the experimental validation results for the improved algorithms; finally, in Section 7 we draw our conclusions and outline some future work.

2 The proposed architecture

In [5] we proposed an innovative architecture for a network based anomaly detection system, based on unsupervised learning algorithms. We chose to focus on this class of algorithms because they exhibit properties that are particularly well suited for anomaly detection, in particular the ability of detecting outliers and of building a model of "normality" without the need of

---

*Dipartimento di Elettronica e Informazione - Politecnico di Milano. E-mail: zanero@elet.polimi.it

a priori knowledge input.

If we think of the network packet flow as a stream of observations (packets), then anomaly detection is an instance of the outlier detection problem. An outlier is classically defined as follows: "an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [7]. Here, the "different mechanism" is an attacker who is trying to subvert a network service.

However, outlier detection on the flow of TCP/IP packets is not an easy task. Each packet has a variable dimension (which over an Ethernet link for instance varies between 20 and 1500 bytes), while unsupervised learning algorithms work well on multivariate data with a fixed number of features. The network and transport layer headers can be easily normalized and translated to a fixed number of features. It is important to note, however, that in the case of connection-oriented protocols (most notably TCP) the transport layer headers may need inter-correlation in order to be fully deciphered.

On the contrary, the data carried by the packet (the payload) cannot be easily translated into a fixed set of features, since each different application layer protocol would require its own set of features, increasing complexity and decreasing generality. In addition it would require a full reconstruction of traffic sessions, which would expose the IDS to reconstruction problems, possibly leading to attack windows [2].

In addition, the computational complexity of unsupervised learning algorithms scales up steeply with the number of considered features, and the detection capabilities decrease correspondingly (this is known as *the curse of dimensionality*). Only a few algorithms can be optimized to treat data with many thousands of dimensions, but only in the case that they are sparse (for instance, a word/document incidence matrix in a document classification and retrieval problem [8]). We are instead dealing with dense data.

Most of the existing researches on the use of unsupervised learning algorithms for network intrusion detection purposes solve this problem by discarding the payload and retaining only the information in the packet header (e.g. [9, 10, 11, 12]) . This is clearly not an optimal solution, since it leads to an unacceptable information loss: most attacks, in fact, are detectable only by analyzing the payload of a packet, not the headers alone. These algorithms show nevertheless interesting, albeit obviously limited, intrusion detection properties.

In order to solve this problem, we developed the concept of a two-tier intrusion detection system (shown in Figure 1), which allows us to retain at least part of the information related to the payload content. In the first tier of the system, an unsupervised clustering algorithm classifies the payload of the packets, observing one packet at a time and "compressing" it into a single byte of information. This classification can be added to the information decoded from the packet header (or to a subset of this information), and passed on to the second tier. On most networks, the traffic belongs to a relatively small number of services and protocols, regularly used, and a good learning algorithm can map it onto a relatively small number of clusters.

The second tier algorithm instead takes into consideration the anomalies, both in each single packet and in a sequence of packets. It is worth noting that most of the solutions proposed by previous researchers in order to analyze the sequence of data extracted by the packet headers could be used as a second tier algorithm, complemented by our first tier of unsupervised clustering.

3 The curse of dimensionality in the first stage

The first tier algorithm receives in input the payload of a TCP or UDP over IP packet: on an Ethernet segment this means up to 1460 bytes of data. Its role is to classify such information in a "sensible" way, which means that it should, in principle, keep as much information as possible for the second tier algorithm about the "similarity" between packets. Obviously, since our end goal is to detect intrusions, the classification should show mainly the property to separate packets with anomalous or malformed payload from normal packets, and should also divide the payloads reflecting the divisions between protocols as closely as possible. "The grouping of similar objects from a given set of inputs" [13] is obviously a typical clustering problem.

We have shown [5] that a Self Organizing Map (SOM) algorithm [14] is indeed able to sensibly cluster payload data, discovering interesting information in an unsupervised manner. Our research is the first attempt to cluster packet payloads to obtain meaningful results. A previous research showed that neural algorithms can recognize protocols automatically [15], while another paper later independently confirmed that the payload of the packets indeed shows some interesting statistical properties [16].

There are multiple reasons for choosing a SOM for this purpose. A SOM is a hard-competitive, neural based algorithm, which is capable to map a high-dimensional input space onto a low-dimensional (usually bi-dimensional) neuron space. The algorithm is robust with regard to the choice of the number of classes to divide the data into, and is also resistant to the presence of outliers in the training data, which is a desirable property: in real-world situations, the training data could already contain attacks or anomalies and the algorithm must be capable of learning regular patterns
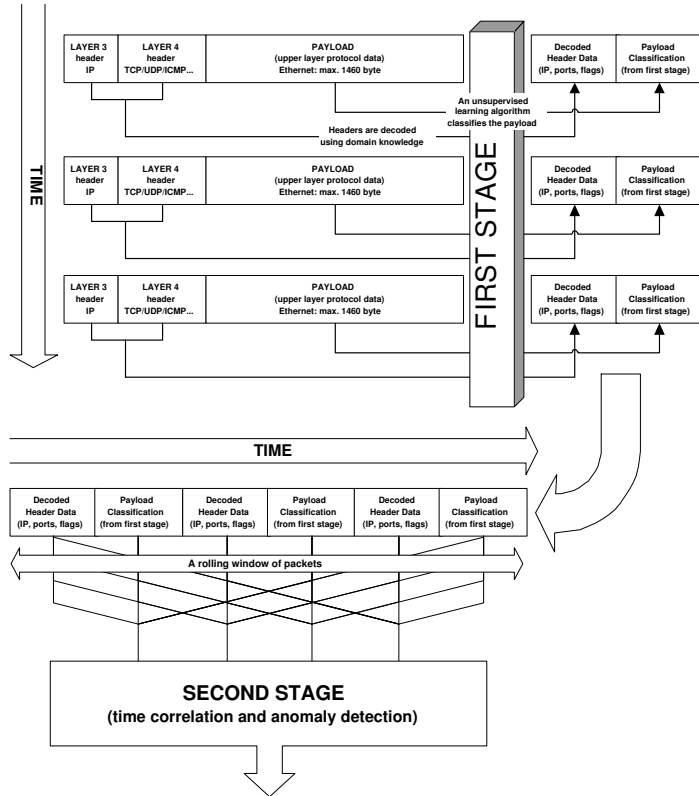
LAYER 3 header IP | LAYER 4 header TCP/UDP/ICMP... | PAYLOAD (upper layer protocol data) Ethernet: max. 1460 byte

Decoded Header Data (IP, ports, flags) | Payload Classification (from first stage)

An unsupervised learning algorithm classifies the payload

Headers are decoded using domain knowledge

TIME

FIRST STAGE

LAYER 3 header IP | LAYER 4 header TCP/UDP/ICMP... | PAYLOAD (upper layer protocol data) Ethernet: max. 1460 byte

Decoded Header Data (IP, ports, flags) | Payload Classification (from first stage)

LAYER 3 header IP | LAYER 4 header TCP/UDP/ICMP... | PAYLOAD (upper layer protocol data) Ethernet: max. 1460 byte

Decoded Header Data (IP, ports, flags) | Payload Classification (from first stage)

TIME

Decoded Header Data (IP, ports, flags) | Payload Classification (from first stage) | Decoded Header Data (IP, ports, flags) | Payload Classification (from first stage) | Decoded Header Data (IP, ports, flags) | Payload Classification (from first stage)

A rolling window of packets

SECOND STAGE
(time correlation and anomaly detection)

Figure 1: The two-tier architecture of the IDS, comprising a tier of unsupervised clustering followed by a tier of outlier detection.

out of a "dirty" training set. In addition, we have compared various algorithms and shown that the SOM had the best performance trade-off between speed and classification quality.

Unluckily, the curse of dimensionality hits heavily against the first tier. The second tier is not a problem, being required to handle a multivariate time series with a comparably small number of features (up to a maximum of about 30). There are alternative algorithms for clustering which are much faster in the learning phase than SOM; for example, the well known K-means algorithm is one of the fastest. But at runtime even K-means is not more efficient than a SOM, so we cannot solve the problem by choosing a different algorithm.

A traditional approach to the problem would use dimension reduction techniques such as dimension scaling algorithms [17] or Principal Component Analysis [18]. But our experiments demonstrated that they are quite ineffective in this particular situation, since by their nature they tend to "compress" outliers onto normal data, and this is exactly the opposite of what we want to achieve.

4 Improving the performance of the SOM algorithm

Since no alternative solution was viable, we developed various approximate techniques to speed up the SOM algorithm. The reference machine for our tests is an Athlon-XP 3200 based computer with 1 GB of DDR RAM, running GNU/Linux with a 2.6 kernel. All the tests, unless otherwise stated, refer to a SOM with square topology, and a size in the space of neurons of 10×10. The test are conducted on TCP packets, as they constitute over 85% of Internet traffic.

The data used for training and testing the prototype are subsets of the "1998 DARPA IDS Evaluation dataset", which is well commented and described by a master's thesis [19]. In [20] there is a detailed analysis of the shortcomings of this traffic sample set, and we share many of the author's observations: no detail is available on the generation methods, there is no evidence that the traffic is actually realistic, and that spurious packets, so common on the Internet today, are not taken into account. On the other hand, whenever we need to test the capability of our prototype of detecting attacks mixed in background data, we need to do this under test condi-

tions, with clearly labeled background data, and in spite of its shortcomings the DARPA dataset fulfills this function very well. However, we positively validated most of our results using also smaller dumps collected on our own internal network.

As we can see from the first line of values in Table 1, the throughput of an implementation of the Kohonen algorithm on our hardware and software configuration is on average of 3400 packets per second, which is not acceptable for an IDS monitoring a modern network.

We tried to develop heuristics for speeding up the computation, introducing minimal errors in the classification. The idea behind our heuristic is simple. Let N be the number of classes, and d the number of dimensions of the data. At runtime, the SOM algorithm consists simply of N evaluations of the distance function: in our test implementation, an euclidean distance function over d dimensions. Since the number of computations is $N \cdot d$, in order to speed up the computation we can try to reduce d by applying any dimensionality reduction technique: this, as we said before, cannot be done meaningfully via dimensionality reduction techniques. However, since just a few packets contain a high number of bytes of payload, we can try to use just the first $d' < d$ dimensions. Further experimental evaluation would then of course be required in order to understand if the "reduced" payloads carry the same information value as the complete packets.

If we do not want to reduce d, we must try to reduce the number of evaluations N. A way to do this is to pre-compute a grouping of the N centroids of the classes in $K < N$ super-clusters, and then to select the winning neuron in a two-step procedure. First, we determine which of the super-clusters the observation belongs to; and then we evaluate the distance function just over the $N' < N$ neurons belonging to the winning super-cluster. The algorithm is heuristic, since it can happen that the best matching neuron is not in the best matching super-cluster, but as we will see the error rate is very low. Obviously the best performance gain with this heuristic happens if each of the K super-clusters is formed by $\sim N/K$ neurons, since the average number of computations becomes $d \cdot (K + N/K)$ which has a minimum for $n = \sqrt{N}$. If the clusters are not balanced the worst case computational cost is higher, and this leads to a lower overall throughput. For smaller values of K the algorithm would be on average slower, and the error rate statistically would be slightly lower.

To form the super-clusters, a first naïve idea would be to exploit the map structure, which tends to keep "close" to each other the neurons which are close in the map space. However, this does not work very well experimentally, probably because of the high dimensionality of the feature space, causing a 35% error rate with $N = 3$, and even 60% with $N = 10$. Thus we resort to a K-means approach.

However, we must overcome two different issues in doing this. A first issue has to do with the nature of K-means, which is inherently initialization dependent, and prone to create very unbalanced cluster. Experimentally, with $N = 100$, using $K \geq 4$ does not create a balanced structure of clusters, unless we correct the randomness of the algorithm. Some authors proposed, in order to eliminate these weaknesses, the "global K-means" algorithm [21], which repeats K-means with all the possible initializations. We use a different and faster approach, by using the algorithm a fixed number m of times, and choosing the distribution in classes which minimizes the average expected number of operations, roughly approximating the probability that an observation falls into the i-th super-cluster as proportional to the fraction N_i/N (where N_i is the number of neurons in the i-th super-cluster). In Table 1 we refer to our variant of the K-means algorithm as "K-means+", and the column labeled "Crossv." reports the parameter m (number of runs of the K-means algorithm).

A second, more difficult issue, is how to deal with the training phase. During the training phase the neurons change their position, so theoretically we should repeat the K-means algorithm once for each training step. We can avoid to do so, and fix an arbitrary update frequency, a number of step after which we will recalculate the position of the centroid. As an additional attempt to reduce the cost of the K-means step, we decided to initialize the position of the K centroids to the same position they held before, even if this could lead the convergence to a local optimum, creating a non-optimal clustering. Our tests showed that in each case the cumulative approximations introduced by the algorithm make the training very unstable, leading to results which are not compatible with the ones obtained by normal training, and in which the properties of outlier resilience and robustness of the SOM algorithm are impaired. We are working to find a way to overcome these problems without sacrificing the throughput gain, but for now, the only way to speedup the throughput is to lower the number of dimensions.

In Table 1 we report the runtime throughput and the error rate of the algorithm, evaluated in packets per second, depending on different combination of the parameters, namely the number of bytes considered for each packet, the usage of an heuristic, the parameter K for the K-means algorithm used in the heuristic and the use of cross-validation repetitions. The results have been validated over multiple "days" of the DARPA dataset.

Max bytes per packet	Heuristics	K	Crossv.	Packets/sec.	Error %
1460	None	-	-	3464.81	-
1460	K-means	10	No	8724.65	0.8
1460	K-means+	5	10	5485.95	0.4
1460	K-means+	10	10	10649.20	0.8
800	None	-	-	4764.11	-
800	K-means+	5	10	9528.26	0.5
800	K-means+	10	10	15407.36	1.0
400	None	-	-	8400.45	-
400	K-means+	5	10	28965.84	0.6
400	K-means+	10	10	30172.65	1.2
200	None	-	-	10494.87	-
200	K-means+	5	10	51724.70	0.8
200	K-means+	10	10	65831.45	2.3

Table 1: Throughput and errors during runtime phase, calculated over multiple runs of the algorithm over different days of the DARPA dataset.

In order to evaluate the results, we refer to a well known study of the statistical properties of Internet traffic [22]. Analyzing the traffic flowing through an Internet Exchange datacenter, they show that approximately 85% of the traffic is constituted by TCP packets, and that a large proportion of TCP packets are 40 bytes long acknowledgments which carry no payload (30% to 40% of the total TCP traffic). Zero-size UDP packets, on the contrary, are almost non-existant. Since the first stage analyzes only packets with a non-null payload, almost 30% of the total traffic on the wire will not even enter it. The average size of a TCP packet is 471 bytes, of a UDP packet 157, and the overall average is approximately 420 bytes. It is also known from theoretical modeling and practical experience that an Ethernet network offers approximately 2/3 of its nominal capacity as its peak capacity. This means that a saturated 10 Mbps Ethernet LAN carries about 2.000 packets per second. Other statistics suggest that this value could be higher, up to 2.500 pps.

From Table 1, we can see that the original SOM algorithm, considering the full payload of 1460 maximum bytes per packet, with no heuristics, operates at a speed that is acceptable for use on a 10 Mb/s Ethernet network, but insufficient for a 100 MB/s network. However, using the K-means algorithm with 10 classes and no cross-validation, we obtain a much higher throughput (more than three times higher than the original one) but also a 0.7% error rate. Introducing K-means+ and crossvalidation, we obtain a better tradeoff between throughput and error rate, improving the former without compromising the latter. A speed of 10.500 packets/second is enough to handle a normal 100 Mbps link (considering also the presence of empty packets).

If necessary, performance could also be improved by reducing the number of bytes of the payload. This could evidently impact heavily against the recognition capabilities of the algorithm.

5 Choosing a meaningful metric

In recent researches, the effect of the curse of dimensionality on the concept of "distance metrics" has been studied in detail. In high dimensional spaces such as the one we are considering, the data becomes very sparse. Recent research results [23, 24] show that in high dimensional spaces the concept of proximity and distance may not be meaningful, even qualitatively.

Let $Dmax_d$ be the maximum distance of a query point to the points in a d-dimensional dataset, and $Dmin_d$ the minimum distance, and let X_d be the random variable describing the data points. It has been shown, under broad conditions, that if

$$lim_{d \to \infty} var \left(\frac{\parallel X_d \parallel}{E[\parallel X_d \parallel]} \right)$$

then

$$\frac{Dmax_d - Dmin_d}{Dmin_d}$$

This means, plainly, that in high dimensional space the difference between the distance of a query point to the farthest and to the nearest point in the dataset tends to be of a smaller order of magnitude than the minimum distance: in other words, the nearest neighbor identification is unstable and does not give much information.

Most of the hypotheses used in the demonstration do not hold for our type of variables. We have experimentally observed that in our setup the described ef-
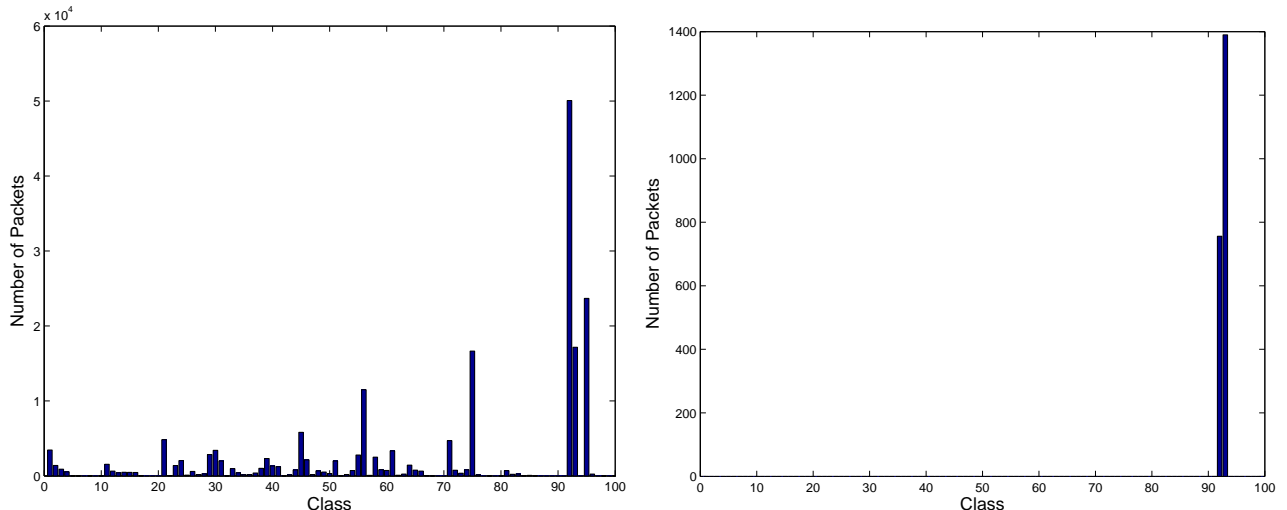
Figure 2: Comparison between the classification of a window of traffic and the traffic destined to port 21/TCP by a 10x10 SOM

fect does not happen: most points are extremely well characterized into dense and compact clusters. In order to better understand if this condition applied to our dataset, we recursively filtered out the most compact clusters and the "farthest" centroids, and analyzed the results, and in each case the difference between $Dmin$ and $Dmax$ was still significant. We thus concluded that the effect observed in the cited articles does not apply to our particular situation, probably because we are working in a compact region where the maximum possible distance between two different points is $\sqrt{255^2 \times 1460}$.

It has also been reported that in high dimensional spaces the L_1 metric, or "Manhattan distance", behaves considerably better than the usual euclidean metric we applied [24]. In [25] distance metrics with a fractional index $f\epsilon(0, 1)$ are also proposed.

On the basis of these suggestions, we explored the application of different distance metrics and their effects on the classification of packets. However, in our particular application the use of these alternate distance seems to lump all the data in a few cluster, diminishing the overall recognition capabilities of the algorithm.

6 Recognition capabilities of the modified algorithm

In order to evaluate the recognition capabilities of the new algorithm, we must see if it can still usefully characterize traffic payloads for different protocols, and detect anomalous attack payloads from normal payloads. In Figure 2 we present a demonstration of the recognition capabilities of a 10×10 Self Organizing Map that creates a division of the data in 100 clusters. The network was trained for 10.000 epochs on TCP packet payloads. The histograms represent the number of packets (on y-axis) present in each cluster (on x-axis). Here and in the following, for graphical reasons, the number of packets on y-axis may be differently scaled in the various graphs. In Figure 2 we suppressed from the output the representation of classes 90 and 93, which are the most crowded and less characterized clusters in the classification, for better display.

On the left handside, we can see the classification of a whole window of traffic. On the right handside, we can see how the network classifies the subset of the packets with the destination port set to 21/TCP (FTP service command channel). It can be observed how all the packets fall in a narrow group of classes, demonstrating a strong, unsupervised characterization of the protocol.

In Figure 3 we present the result of the same test using the modified algorithm for runtime recognition. Also in this case, we can see the same strong characterization of the protocol (the similarity between the graphs is striking, but not surprising, since the error rate is approximately 1%). The same situation happens for all the cases we examined and compared, granting that the protocol characterization property is well preserved by the heuristics.

The capability to detect anomalous packets is also preserved. We analyzed how the SOM classifies packets from the attacks contained in the DARPA datasets. For example, let us discuss the case of a race condition and buffer overflow bug in the "ps" command, which is exploited over a perfectly legitimate telnet connection. 99.76 % of packets destined to TCP port 23 fall in classes
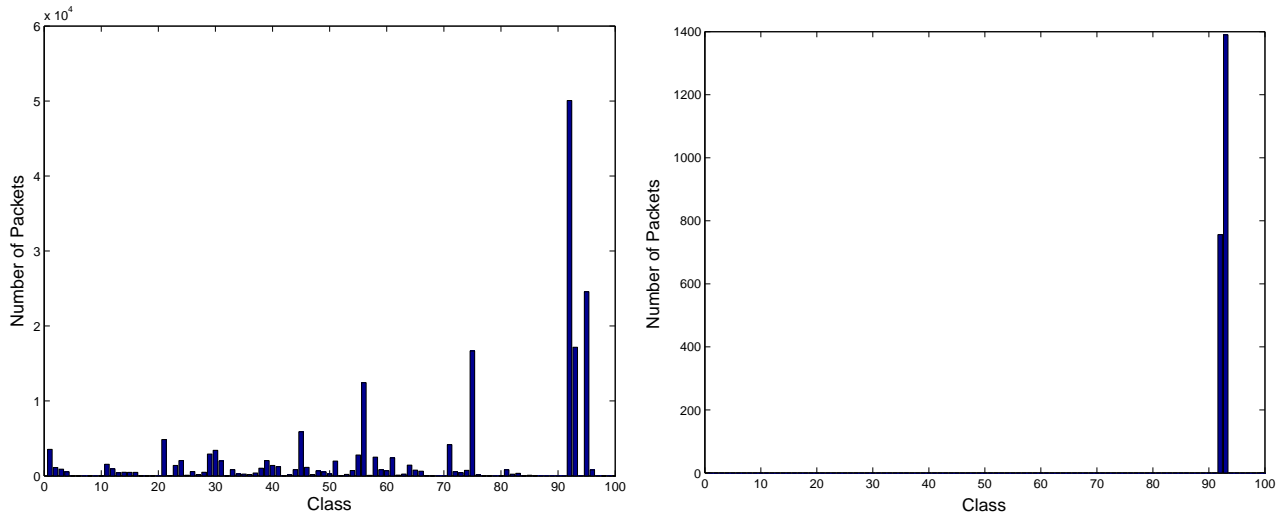
Figure 3: Comparison between the classification of a window of traffic and the traffic destined to port 21/TCP by a 10x10 SOM with our modified algorithm.

91 and 95, and all of them fall between class 90 and 95. The packets containing the attack fall instead in classes 45, 54, 55, 65, 71, 73 and 82, which are not normally associated with DPORT 23. This happens consistently over each instance of the attack.

A similar, albeit less defined, situation happens in the case of a buffer overflow in the "sendmail" MTA daemon. The packets destined to port 25 are less characterized, but over 90 % of them fall into 7 classes. The attack packets fall instead into three different classes that contain less than 3% of the normal packets destined to port 25. This helps us to understand that an important requirement for the second stage detection algorithm will be to keep track of anomaly scores in the recent past.

In order to test the algorithms on newer attacks, we ran the same SOM on the packet captures of some FTP server attacks (a format string wu-ftpd bug exploit, a globbing denial-of-service, a buffer overflow attack). In each case the anomalous payloads fall outside of the narrow characterization we have seen in Figures 2 and 3. The results we presented in [5] are thus preserved while using the modified, faster algorithm.

7 Conclusions and future work

We have described the challenges we met while implementing an innovative model of anomaly based network intrusion detection system, completely based on unsupervised learning techniques. We have described the overall architecture of the system and how the curse of dimensionality requires an appropriate resolution for a working implementation of the first stage of unsuper-

vised clustering. By the means of various techniques, we improved the runtime efficiency of the algorithm, obtaining a throughput rate almost three times higher than the original one, if we are willing to accept a misclassification rate of about 0.8%, and twice as high than the original one with a very small misclassification rate of 0.4%, without truncating the number of the bytes of the payload examined by the algorithm. We have studied how these errors affect the algorithm detection capabilities, and concluded that our heuristically modified implementation works as well as the original version of the SOM. Having thus solved most of the challenges for the design of the first tier, our future work will focus on the choice and implementation of the second tier of learning, and on the empirical evaluation of the IDS under practical workloads.

References

[1] Stefano Zanero. Detecting 0-day attacks with learning

intrusion detection systems. In *Blackhat Briefings USA 2004*, 2004.

[2] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical Report T2R-0Y6, Secure Networks, Calgary, Canada, 1998.

[3] Giovanni Vigna, William Robertson, and Davide Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 21–30. ACM Press, 2004.

[4] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, J. P. Anderson Co., Ft. Washington, Pennsylvania, Apr 1980.

[5] Stefano Zanero and Sergio Savaresi. Unsupervised learning techniques for an intrusion detection system. In *Proceedings of the 14th Symposium on Applied Computing, ACM SAC 2004*, 2004.

[6] C. Kruegel, D. Mutz, F. Valeur, and G. Vigna. On the detection of anomalous system call arguments. In *Proceedings of ESORICS 2003*, Oct. 2003.

[7] D. Hawkins. *Identification of Outliers*. Chapman and Hall, London, 1980.

[8] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[9] M.V. Mahoney and P.K. Chan. Detecting novel attacks by identifying anomalous network packet headers. Technical Report CS-2001-2, Florida Institute of Technology, 2001.

[10] Dit-Yan Yeung and Calvin Chow. Parzen-window network intrusion detectors. In *Proceedings of the 16th International Conference on Pattern Recognition*, volume 4, pages 385–388, aug 2002.

[11] K. Labib and R. Vemuri. NSOM: A real-time network-based intrusion detection system using self-organizing maps. Technical report, Dept. of Applied Science, University of California, Davis, 2002.

[12] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Trans. on Information and System Security*, 2(3):295–331, 1999.

[13] J. A. Hartigan. *Clustering Algorithms*. Wiley, 1975.

[14] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berling, 3 edition, 2001.

[15] K.M.C. Tan and B.S. Collie. Detection and classification of TCP/IP network services. In *Proc. of the Computer Security Applications Conf.*, pages 99–107, 1997.

[16] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *RAID Symposium*, September 2004.

[17] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling*. Monographs on Statistics and Applied Probability. Chapman & Hall, 1995.

[18] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[19] K. Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Massachussets Institute of Technology, 1998.

[20] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[21] A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2), 2003.

[22] S. McCreary and K. Claffy. Trends in wide area ip traffic patterns - a view from ames internet exchange. In *Proceedings of ITC'2000*, 2000.

[23] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.

[24] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the nearest neighbor in high dimensional spaces? In *The VLDB Journal*, pages 506–515, 2000.

[25] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*, 1973, 2001.