

Network Intrusion Detection Using a Hardware-Based Restricted Coulomb Energy Algorithm on a Cognitive Processor

Mahdi H. Moghaddam and Ricardo A. Calix

Purdue University Calumet, CITG, 2200 169th Street, Hammond, IN, 46323-2094

{mahdi.moghaddam, ricardo.calix}@purduecal.edu

Abstract

The current state of the art in intrusion detection systems mainly relies on heuristic rules called signatures to detect intrusions to a network environment. The downside of signature based approaches is that they can only detect previously known attacks. Since no signature exists for new attacks, other approaches need to be considered. Here, machine learning algorithms may be beneficial. Additionally, at the network level, intrusion detection system performance is very important. Therefore, fast and efficient machine learning implementations are needed. In this study, a parallel hardware based implementation of the KNN and RCE classifiers will be analyzed to get a better understanding of the advantages and disadvantages of hardware based machine learning for network intrusion detection.

Introduction

The worldwide market for embedded systems will have sustained growth at a rate of 7% over the next 5 years and it is expected that the market will reach \$158.6 billion by 2015 (BCC research, 2012). This new set of devices will add to the earlier explosion of data caused by standard devices such as PCs, tables, and smart phones. It is expected that the so called internet of things will eventually generate most of the data in the digital universe. This growth in digital data, as a side effect, increases the threat to information systems and the need to defend them. Hourly down time costs can vary significantly for a square foot of data center space and can range from \$8,500 to \$210,000 (Savitz, 2012). Therefore, preventing downtime and dealing with cyber-attacks is a big concern for companies.

There are several techniques that can be used to prevent cyber-attacks. In particular, intrusion detection systems

can be helpful in preventing attacks or raising alerts when networks are under attack. The issue with current network intrusion detection systems is their inability to detect unknown attacks. IDS systems require a knowledge base of known attacks called signatures that must be matched against incoming network traffic to detect attacks. However, if the attack is not known, there is very little that the system can do. Therefore, research in the field has proposed that machine learning approaches can help to address these problems. In particular, machine learning can be used to develop models to try to detect unknown attacks. Most implementations of IDS systems focus on approaches at the software level. IDSs at the software level, however, may suffer from low performance speeds. Using standard CPU architectures and standard RAM memory may slow down processing of an IDS system. In the context of network speeds, a software based machine learning NIDS system may not be adequate. To address this issue, this study proposes the use of a hardware based machine learning cognitive processor for network intrusion detection. A KNN algorithm and an RCE algorithm are used to perform classification and measure performance of the IDS system.

Literature Review

An Intrusion detection system is a hardware or software system that, by monitoring any system or network, can detect malicious cyber activities. There are different kinds of IDS systems such as host based IDSs and network based IDSs (Scarfone, 2007). With respect to the detection methods used, two main types are used which are statistical anomaly-based IDS systems and signature-based IDS systems. A signature based IDS monitors network traffic and compares that traffic to a database of known attacks. This method is similar to the method most anti-virus

systems use. However, signature based IDS systems are not able to address novel attacks since there is no record in the database of them. This will result in a time gap between the first appearance of the attack and the time when the system can finally detect that attack (Barman 2012). One of the most popular signature-based IDS system is Snort. Snort is a packet sniffer based on the c-based libpcap library. It monitors packet traffic and performs pattern matching against its knowledgebase. It can generate alarms in real time that are sent to the system administrator (Roesch, 1999).

Anomaly-based detection methods, on the other hand, try to detect abnormal traffic which has differences in characteristics to normal traffic. This method needs to define normal traffic first, and then identifies variations from the normal model. Hence, it can detect traffic that did not generate from a user's normal activities (Axelsson, 2000). In other words, anomaly detection tries to measure the acceptable or usual behavior of a user and then shows potential exceptions (Kumar, 1995). The downside of anomaly detection is that not all abnormal traffic is necessarily an attack. This approach can therefore result in many false alarms. Normal traffic detected as abnormal traffic is known as false positives and abnormal traffic that is detected as normal traffic is referred to as false negatives. A third category for IDS systems involves pattern matching using supervised learning. In this case, 2 or more classes can be identified using trained classification models. This study focuses on supervised learning for network intrusion detection.

Software-based machine learning

In 1999, Lee et al. introduced a data mining framework in order to build an intrusion detection system. Their idea was to extract lots of features from the network data and then apply machine learning methods in order to learn the behavior of abnormal and normal traffic. They used the DARPA dataset for their experiments (Tavallae et al. 2009). The DARPA dataset includes samples of abnormal and normal behavior in the network. Sabhnani and Serpen (2003) proposed an approach that learns multiple classifier models for each class. Results of their work showed that some algorithms provide better results for a given attack. Hence a multi-classifier model is more suitable in order to deal with different attack categories. Mukkamala et al. (2002) suggest that by using neural networks and SVM it is possible to create an IDS. Like most of the ML techniques, they first extracted patterns of user's experiences within the system and then created a classifier by using those patterns. That classifier can be further used in order to detect anomalies. The SVM was trained using an RBF kernel. The accuracy of this method was 99.5%. Their

neural network model achieved similar results with an accuracy of 99.25%.

Hardware-based machine learning

Intrusion detection systems are highly dependent on their hardware architecture. Therefore, some researchers have focused on ways to improve IDS performance by analyzing various aspects of the hardware implementation. Most researchers use field programmable gate arrays (FPGA) as a preferred hardware platform. Sidhu and Prasanna in 2001 provided an efficient method for intrusion detection using FPGAs and regular expressions. They showed that performing matches using regular expressions can be more efficiently done on FPGA implementations than on standard PC implementations. They ran their experiment using both the Intel Pentium III processor and a Virtex FPGA. Their study found that using regular expression matching will result in situations where not enough memory is available on the PC. In this case, an FPGA can be more suitable. Baker and Prasanna in 2004, proposed a methodology for building an efficient IDS using an FPGA as the hardware platform and optimizing the design of the system. They applied optimization processes in order to have efficient parallel multi-byte comparisons and partial matches for high performance calculations using FPGA. They showed that this methodology results in faster computing times. As a down side, their approach also increased the amount of false-positive alerts issued by the system.

Snort is a popular IDS and many researchers try to improve its performance using machine learning algorithms or by improving the supporting architecture to speed up processing. According to Antonatos et al. (2004), almost 70% of Snort's execution time is for string matching. Therefore, there have been several studies that focused on speeding up string matching. Aldwairi et al. (2005), for instance, tried to achieve better performance by increasing the throughput of the system. They suggested that using hardware accelerators offered up to 14Gbps in performance. They studied Snort rules and argued that 87% of the rules involve content matching. This suggests that there is a need for hardware acceleration of content matching. They proposed a hardware-based string matching technique implementing finite state machines.

Das et al. (2008) proposed an FPGA-based architecture for anomaly detection. At first, a feature extractor module (FEM) was developed. By using an FPGA, significant performance was achieved compared to software solutions like Snort. It was shown in that work that around 22 Gbps throughput can be reached. Their proposed IDS also used Principal Component Analysis (PCA) to reduce data dimensionality. The results of their analysis produced good

accuracy scores on the KDD Cup dataset. The FPGA used in their work was the Xilinx Vertex Family. In 2013 Yoon et al. suggested a multicore-based IDS to improve performance. This approach used multiple cores in parallel to improve processing speeds. They introduced the SecureCode framework to detect malicious behavior using statistical analysis. Although their study addressed parallelism and statistical analysis, they still used processors with standard architectures. In a sense, their approach may still suffer from the bottlenecks faced by software based implementations. Instead, a different type of processor may be needed to perform the statistical analysis. In particular, a processor designed for parallelism and with architecture optimized to run machine learning algorithms. This study proposes such an implementation and will report the results of the analysis.

Methodology

The methodology proposed in this study consisted of collecting and pre-processing network data, designing and building a hardware-based machine learning IDS prototype, and performing the analysis. To collect the data, a small packet sniffer was used based on Sniffex (Grabell, 2008). The data was pre-processed and normalized. Finally, using the normalized train and test sets, the IDS prototype was tested and an analysis of the results was performed.

IDS Architecture

A packet sniffer was used to collect the data. This packet sniffer ran on a Raspberry PI B+ with an embedded Linux distribution. The sniffer handles the data collection to create the data sets. To perform the classification, an IDS platform was developed. The platform was built around the

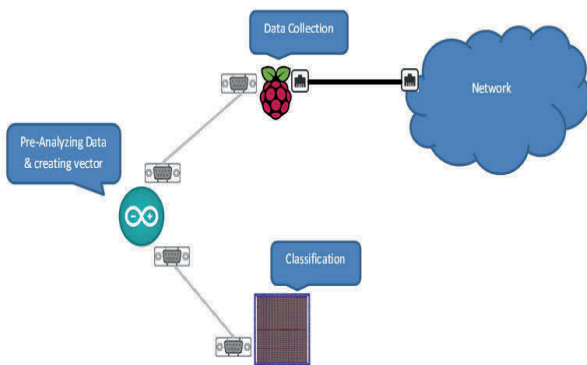


Figure 1 IDS architecture using the CM1K

CM1K-PGA69 machine learning co-processor produced by Cognimem Technologies. The CM1K is a low powered co-processor specifically designed for machine learning. It can store up to 1024 training samples or neurons. It includes 2 different classification algorithms, and is based on vector space type distance calculations. In essence, it compares a test sample to learned training samples and calculates the distance between the test sample and the model's learned neurons. The CM1K co-processor was connected to an Arduino Due board which provides the power and communication capabilities for the CM1K. The data is stored on a micro-SD card and is sent by the Arduino Due board to the CM1K processor. Figure 1 shows a diagram of the architecture and how the data flows between the devices.

A picture of the prototype embedded system device can be seen in Figure 2. The Arduino Due board is connected to the CM1K via serial I²C communication protocol.

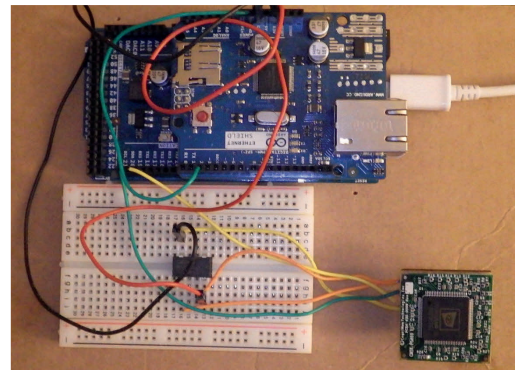


Figure 2 Arduino Due connected to the CM1K

Data Collection

Two data sets were used to perform the analysis. A small dataset was collected and compared against the well-known NSL-KDD intrusion detection corpus (Tavallae et al. 2009). The NSL-KDD corpus is derived from the KDD cup dataset. The NSL-KDD corpus consists of 41 features extracted from network traffic captures.

The smaller data set consisted of a total of 2048 normal samples and 2048 abnormal samples. The data was collected using a modified version of the Sniffex packet sniffer which was installed on a Raspberry PI board. All the packet header information gathered by this tool was saved in a CSV file. A total of 10 features were used for this study. The features are: src_ip, dst_ip, tos, len, id, off, prt, src_p and dst_p. The source IP (src_ip) represents the host that sends packets to the packet sniffer. The sniffer that was used in this project stores this IP address as a digit. The destination IP (dst_ip) address in most cases is the host which the sniffer resides on. The type of service (tos) is a 6-bit differentiated service code. The length (len)

feature stores the length of the packet. The id feature (id) represents the unique identification of a packet. The off feature stores the fragmentation offset which is used for assembling packets. The ttl feature is the time-to-live information stored in each packet. This 8-bit field is used to limit the packet life span to a certain number of routers or hops. The prt field stores the protocol being used such as icmp, tcp, udp, etc. Finally, source (src_p) and destination (dst_p) ports were also used as features.

In order to collect the required network traffic samples, a small isolated LAN was set up. The LAN consisted of a network switch and 3 PCs plus a raspberry pi. Normal packets like ping trace route and other TCP streams were generated in this network. To collect attack traffic, some network attacks were performed. These included: ARP poisoning, ICMP Redirect attack, SYN Flooding attack, TCP RST attack over SSH, and telnet and Video streaming session hijacking. Table 1 shows an example of the collected data set.

Table 1: Dataset collected by sniffer program

src_ip	dst_ip	tos	len	id	off	ttl	prt	src_p	dst_p
1828716554	3705422800	0	16384	45235	64	64	17	44176	53
1828716554	3705422800	0	16384	45491	64	64	17	44176	53
16777226	1828716554	192	23552	55309	0	64	1	0	0
16777226	1828716554	192	23552	55565	0	64	1	0	0

According to the CM1K datasheet, each neuron can only handle 1 vector with up to 256 features. Additionally, each feature in the vector can only be represented with 1 byte of memory. As a result, feature values can only range from 0 to 255. Therefore, the data needs to be normalized to fit within this range. Another constraint of the chip is that it only accepts integer values (no float values) which means that the values must also be rounded.

Classification approach

The CM1K co-processor that was used in this study supports the K-Nearest Neighbor (KNN) classifier and the Restricted Coulomb Energy (RCE) classifier. The KNN algorithm is a famous technique used in machine learning. It does not require optimization and is easy to implement in both software and hardware. The KNN algorithm stores all training samples in its knowledgebase. It then compares a new test sample to all learned samples or neurons in its knowledgebase. It uses a vector space model approach and calculates a distance metric between the test sample and all learned neurons. In contrast to the KNN algorithm, the RCE algorithm (Schofield et. al, 1987; Reilly et al., 1982) does not need to store all training samples. Instead, this method selects only samples that best help to represent the model. The advantage of this approach is that it is not limited by the number of neurons available on the chip. For

the case of the CM1K, only 1024 neurons are available per chip. As a result, the KNN algorithm can only hold 1024 training samples on 1 chip. To add more training samples, more chips need to be added under KNN. The RCE method, on the other hand, could in theory build a model with less than 1024 neurons even if the training set is much larger than 1024 samples.

To perform classification, each RCE neuron j calculates an L_1 Manhattan distance denoted as d_j between an input vector x and the neuron's learned vector u_j . This distance can be denoted as follows:

$$d_j = \|x - \mu_j\|_1 = \sum_{i=1}^n |x_i - \mu_{ji}| \quad (1)$$

where n represents the number of features per vector. For each RCE neuron, the calculated distance d_j is compared to the neuron's learned threshold. Let us denote this threshold as z_j . The calculated distance d_j is compared to the threshold z_j . The neuron is said to fire if d_j is less than z_j . Given D to be the set of distance values from the firing neurons, the previous rule can be stated as:

$$D = \{d_j \mid d_j < z_j\} \quad (2)$$

The predicted class is then defined as the category "y" associated with the minimum distance neuron from the set D . The predicted category hypothesis can be defined as follows:

$$h(x) = y \arg \min D \quad (3)$$

This predicted category represents the predicted class for the given test sample.

Analysis and Results

The data sets were classified using both the KNN and RCE methods available on the CM1K co-processor. The analysis was performed using the 2 data sets which are the NSL-KDD corpus and the 10-feature data set collected for this study. For each data set, a series of 10 runs was performed to determine the accuracy rates for both classifiers. Table 2 shows the average accuracy score and variance for each method using the 10-feature data set. Table 3 shows the resulting accuracy score per individual run on the 10-feature data set. Each run in Table 3 and Table 5 was performed using 1024 training samples and 1024 test samples. These samples were chosen randomly for each run from the original 10-feature dataset (Table 3) or the NSL-KDD corpus (Table 5). As can be seen from Table 2, the average accuracy score for the RCE algorithm

was 73.23% while the accuracy score for the KNN algorithm was 70.98%.

Table 2: Statistical analysis on the 10-feature corpus

	RCE	KNN
Average	73.23%	70.98%
Variance	0.00940922	0.04730602
Standard Deviation	0.09700113	0.21749947

The RCE classifier appears to perform better than the KNN algorithm using the 10-feature dataset. It is also important to note that the variance for the RCE algorithm was lower than the variance for the KNN algorithm. Table 3 also shows the processing speeds per run and algorithm. From the results it can be seen that the KNN implementation appears to be slightly faster. The average testing time for the KNN implementation was 110342.5 milliseconds. The average testing time for the RCE implementation was 110443.9 milliseconds. This shows that the required time to test the samples is almost the same for both methods. Dividing the total testing time by the number of test samples, we can estimate the average processing time per 1 test sample which is around 108 milliseconds.

Table 3: RCE and KNN runs on 10-feature corpus

Run	RCE	RCE_N	RCE_TIME	KNN	KNN_N	KNN_TIME
1	76.76%	3	110249	71.68%	1024	110163
2	82.03%	3	110112	80.66%	1024	110261
3	83.59%	3	110207	85.35%	1024	110453
4	63.78%	5	110003	30.37%	1024	110446
5	85.44%	4	110075	87.21%	1024	110463
6	77.54%	3	110136	87.40%	1024	110240
7	61.82%	3	111890	87.79%	1024	110335
8	58.89%	3	111331	77.25%	1024	110322
9	66.31%	3	110177	32.91%	1024	110486
10	76.17%	3	110259	69.14%	1024	110256

Table 3 also shows the number of neurons that fired to perform classification under each method. As can be seen, RCE on average required no more than 3 fired neurons to perform the classification. The time values in Table 3 are expressed in milliseconds.

Finally, to get a better understanding of the methodologies, the experiment was repeated using the NSL-KDD intrusion detection corpus (Tavallae et al. 2009). The results are presented in Table 4 and Table 5.

Table 4: Statistical Analysis on NSL-KDD corpus

	RCE	KNN
Average	70.44%	83.07%
Variance	0.015135246	0.007278273
Standard Deviation	0.123025387	0.085312794

As can be seen in Table 4, on average the KNN method performed better than the RCE method. This result is different from the results obtained using the smaller 10-feature data set from Table 2 and Table 3. The variance of the RCE methods appears to be larger than the variance of the KNN method. As a result, it appears that the KNN method obtained more consistent results. Given that the NSL-KDD runs were randomly extracted from non-overlapping sections of a much larger corpus, it is possible that these results are more representative of the true state of the data. However, it can be argued that both methods performed well. Finally, the speed performance across both data sets appears to be consistent. This shows that using more features (within the constraints of the cognitive processor) did not affect performance speeds.

Table 5: RCE and KNN runs on NSL-KDD corpus

Sample #	RCE	RCE_N	RCE_TIME	KNN	KNN_N	KNN_TIME
1	79.39%	2	123728	87.01%	1024	123195
2	58.40%	3	123522	59.67%	1024	123500
3	79.59%	2	123853	87.40%	1024	123146
4	50.88%	7	123188	84.86%	1024	123430
5	57.91%	2	123662	86.72%	1024	123505
6	80.57%	2	123824	84.47%	1024	123338
7	79.88%	5	123362	88.77%	1024	123691
8	80.66%	3	123611	81.05%	1024	123448
9	58.30%	2	123678	83.40%	1024	123426
10	78.81%	2	123974	87.30%	1024	123286

Conclusion

As a result of this study, it seems that the use of a hardware-based machine learning cognitive processor for network intrusion detection holds promise. Using the 10-feature corpus, the average accuracy score for the RCE method was 73.23% and 70.98% for the KNN method. The variance for RCE was 0.0094 and 0.0473 for KNN. For the 10-feature corpus, the RCE implementation performs slightly better than the KNN implementation and the results seem to be more consistent. In contrast, using the NSL-KDD corpus, KNN performed better than RCE. It is difficult to determine why the results were opposite. It is possible that they are a result of the intricacies of 2 different datasets. However, the NSL-KDD corpus is larger and uses more features (41), and as a result the samples drawn from it were more diverse. Additionally, it can be noted that with more features, the KNN method performed better. Although the RCE method performed worse than KNN using the NSL-KDD corpus, it is important to note that for a low powered, low cost IDS system, being able to use only 1 chip to build complex classification models is a great advantage. The KNN algorithm may require more neurons and hence more chips to perform the same job. Future work will focus on collecting additional features for the 10-feature data set and on collecting more samples.

Additionally, future work will focus on comparing the hardware-based KNN and RCE implementations of the CM1K to software-based implementations of the algorithms using standard PC processors. This analysis will be performed for both classification accuracy and speed performance.

References

- BCC Research. 2012. Embedded systems: Technologies and market. Retrieve from <http://www.bccresearch.com/market-research/information-technology/embedded-systems-technologies-markets-ift016d.html>
- Savitz. 2012. Figuring DDos Attack risks into IT Security budget. Retrieve from <http://www.forbes.com/sites/ciocentral/2012/05/08/figuring-ddos-attack-risks-into-it-security-budgets/>
- Scarfone, K., & Mell, P. 2007. Guide to intrusion detection and prevention systems (idps). NIST special publication, 800 (2007), 94.
- Barman, D. K., & Khataniar, G. 2012. Design of Intrusion Detection System Based On Artificial Neural Network and Application of Rough Set. *International Journal of Computer Science & Communication Networks*, 2(4), 548-552.
- Roesch, M. 1999. Snort: Lightweight Intrusion Detection for Networks. In *LISA* (Vol. 99, pp. 229-238).
- Axelsson, S. 2000. Intrusion detection systems: A survey and taxonomy (Vol. 99). Technical report.
- Kumar, S. 1995. Classification and detection of computer intrusions (Doctoral dissertation, Purdue University).
- Lee, W., Stolfo, S. J., & Mok, K. W. 1999. A data mining framework for building intrusion detection models. In *Security and Privacy*, 1999. Proceedings of the 1999 IEEE Symposium on (pp. 120-132).
- Sabhnani, M., & Serpen, G. 2003. Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context. In *MLMTA* (pp. 209-215).
- Mukkamala, S., Janoski, G., & Sung, A. 2002. Intrusion detection using neural networks and support vector machines. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on* (Vol. 2, pp. 1702-1707).
- Sidhu, R., & Prasanna, V. K. 2001, March. Fast regular expression matching using FPGAs. In *Field-Programmable Custom Computing Machines, 2001, FCCM'01. The 9th Annual IEEE Symposium on* (pp. 227-238).
- Baker, Z. K., & Prasanna, V. K. 2004. A methodology for synthesis of efficient intrusion detection systems on FPGAs. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004, 12th Annual IEEE Symposium on* (pp. 135-144).
- Antonatos, S., Anagnostakis, K. G., & Markatos, E. P. 2004. Generating realistic workloads for network intrusion detection systems. *ACM SIGSOFT Software Engineering Notes*, 29(1), 207-215.
- Aldwairi, M., Conte, T., & Franzon, P. 2005. Configurable string matching hardware for speeding up intrusion detection. *ACM SIGARCH Computer Architecture News*, 33(1), 99-107.
- Das, A., Nguyen, D., Zambreno, J., Memik, G., & Choudhary, A. 2008. An FPGA-based network intrusion detection architecture. *Information Forensics and Security, IEEE Transactions on*, 3(1), 118-132.
- Yoon, M. K., Mohan, S., Choi, J., Kim, J. E., & Sha, L. 2013. SecureCore: A multicore-based intrusion detection architecture for real-time embedded systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th* (pp. 21-32).
- Grabell, M. 2008. Bomb Sniffing Scam Exposed. *Wired*. Retrieved March 14, 2012.
- Reilly, D., Cooper, L., Elbaum, C. 1982. A neural model for category learning. *Biol. Cybern* vol.45, 1982, pp. 35-41.
- Scofield, C., Reilly, D., Elbaum, C., Cooper, L. 1987. Pattern class degeneracy in an unrestricted storage density memory. in *Neural Information Processing Systems*, Denver, CO, 1987, ed. D. Z. Anderson, American Institute of Physics, New York, NY, 1988, pp. 674-682.
- Tavallae, M., Bagheri, E., Lu, W., Ghorbani, A. 2009. A Detailed Analysis of the KDD CUP 99 Data Set. In *proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2009)*.