Network Traffic Anomaly Detection Using Shallow Packet Inspection and Parallel K-means Data Clustering

Radu VELEA[1], Casian CIOBANU[2], Laurențiu MĂRGĂRIT[3], Ion BICA[4]*

[1, 2, 3, 4] Military Technical Academy,

81-83 George Coșbuc Avenue, Bucharest, 050141, Romania,

radu.velea@mta.ro

ciobanu.casian94@gmail.com

laurentiu.margarit@mta.ro

ion.bica@mta.ro (**Corresponding author*)

Abstract: IT infrastructures around the world are targeted by malicious entities that want to steal data or compromise services. Protection measures for complex computer networks are expensive to deploy and maintain, and often do not offer protection against zero-day exploits. In-depth analysis of incoming and outgoing traffic can be problematic from legal and technical perspectives. The current work explores the possibility of implementing reliable security measures using machine learning algorithms to perform traffic classification. The new framework is mapped on existing parallel hardware and aims to provide a versatile solution for the detection of anomalous behaviour in network traffic through k-means clustering and without performing deep packet inspection. Trace analysis metadata is obtained by exploiting the features available in the *pcapng* file format. K-means clustering is implemented using multiple parallel APIs and a comparative analysis is presented together with performance considerations.

Keywords: K-Means clustering, Shallow packet inspection, Parallelization, Tracing.

1. Introduction

The security of computer networks can be compromised in multiple ways: individual hosts can become infected with malware, networking equipment can be disabled, security policies can be bypassed, etc. The attack surface is often as large as the network, with possible threats coming from inside (disgruntled employee) or outside hacker. There is significant emphasis on identifying security incidents in a timely and reliable manner and limiting the amount of damage that can be inflicted on the network and its users. The proposed framework workflow discusses aspects such as network tracing, data acquisition, feature extraction, traffic modelling, parallelization and visualization.

1.1 Problem Statement

Deployment of security solutions to safeguard a network against attackers is a complex task. Security analysts have to take into consideration the performance impact of the preventive measures and their reliability: known coverage of existing exploits, expected number of false positives, power consumption, scalability and load tolerance. More often than not, a compromise has to be reached between all the above factors. Applications can offer protection in real time or perform forensics operations on network traffic logs.

Depending on the position where they are deployed, antivirus-software (AV) and network intrusion detection and prevention systems (NIDPS) have access to all the traffic and resources managed by the system. Analysing all this data raises some important problems: the data may be encrypted, the volume can overload the capabilities of existing hardware or the data may belong to a third party entity and be protected by law - as is the case for most cloud infrastructures. All these factors make deep packet inspection prohibitive.

Security researchers have experimented with novel machine learning algorithms to remove the need for in-depth packet analysis. The idea behind this approach is to use metadata extracted from intercepted traffic and use it to build classification models that help identify vulnerabilities inside the network. This way the compute-intensive tasks inherent in deep packet inspection are circumnavigated and user privacy is protected to some degree. Two other advantages are speed and the theoretical ability to detect zero-day exploits by outlining anomalous behaviour. Machine learning algorithms have the advantage of scaling up to parallel and distributed architectures [4]. The drawback of this method is the inherent loss in detection accuracy, as the analysing tool has no way of being 100% sure if the intercepted traffic is malign or benign. Human intervention may still be required to make sense of some scenarios.

https://doi.org/10.24846/v26i4y201702 ICI Bucharest © Copyright 2012-2017. All rights reserved

1.2 Motivation

To solve the problems stated above, we've designed a framework that processes network traffic and host information to allow network analysts to create an overview of the events on the network based on a series of features of their choosing. Visualizing the output has the effect of identifying anomalies inside the network. These anomalies could in turn be used to track down security vulnerabilities or adjust network security policies. Our goal is to provide a versatile structure that scales well with the size and type of information available. Our implementation targets conventional hardware and is designed to run in parallel environments. The use of multi-core CPUs and GPUs is intended to provide significant speedups and power saving to make this solution viable in real-time scenarios. Information retrieval consists of the extraction of metadata from network traces and live traffic. Our framework pre-processes raw data into user-defined features and attaches the information to the existing data structures, without changing their base format. This information is then fed into clustering algorithms that create a visual model of the network situation. The intention is that a security analyst or network administrator could use the end results of this computation to identify various problems inside the network (ex: performance problems, security issues, malfunctioning hardware, etc.).

2. State of the Art

2.1 Network Intrusion & Machine Learning

Deep packet inspection (DPI) is a way of supplementing the normal capabilities of firewalls and NIDPSs. The technique involves looking beyond the transport layer headers and even analysing the contents of a packet during routing or filtering. The extra information obtained through DPI is useful in accurately classifying network traffic. The spread of encrypted protocols like HTTPS and concerns regarding user privacy and net neutrality hinder the reliability of DPI. DPI is also highly compute-intensive - whole packet analysis involves lots of pattern-matching operations on strings. Performance improvements of DPI usually focus on developing efficient string matching algorithms that reduce the amount of data that has to be analysed more thoroughly [10] [12]. For example, favouring fast partial matches or hashing incoming bytes in order to skip large amounts of harmless content. Researchers have observed that relying on partial information and other metadata such as packet arrival times or protocol header fields is sufficient to perform traffic classification that matches DPI.

Shallow packet inspection works under the assumption that network attacks behave in a fundamental way differently than normal data flows. Given enough information, security applications can build elaborate profiles that can be used to identify specific network flows, connections or even individual processes on the network [15]. This procedure is often hard to use in a real life scenario, where even normal applications might behave unpredictably in some circumstances. Machine learning steps in and promises to solve these problems by reducing the margin of error proportionally to the amount of input data used. Research in this field revolves around concepts such as [23]:

- Classification learning: training a model based on a set of samples in order to be able to recognize and classify new inputs
- Cluster algorithms: grouping existing items according to a set of features
- Association learning: finding similarities and discrepancies between items

Classification of network traffic requires large amounts of training data to be accurate [17]. Training samples have to be diverse enough to achieve a good bias-variance trade-off. Failure to create an accurate model would result in new vulnerabilities being misclassified or legitimate traffic being flagged as malicious [24]. Although the algorithms promise to deliver high detection rations and good performance [2], they are seldom deployed on their own outside academic circles. The current work focuses instead on cluster algorithms: more specifically k-means clustering [14]. We believe that machine learning implementations have yet to reach the desired maturity to be able to deliver 100% accurate results in industry-competitive environments, but can act as a good tool to provide hints and insight that can help conventional security applications and network analysts reach a decision faster. K-means is ideal from this point of view because it converges relatively fast towards a local optimum. This type of analysis can highlight anomalies present in network traffic [9]. When

combined with other techniques, k-means has been known to detect novel intrusions [7] and reduce false alarms [25].

2.2 Parallelism

Shallow packet inspection itself is faster than DPI. The algorithms involved in processing the metadata can be complex and require continuous iterations in order to keep up with new patterns and rules. An efficient solution to reduce the runtime is to make use of the single instruction multiple data (SIMD) paradigm and harvest the inherent parallelism available in most current hardware architectures. NIDPSs applications like Suricata already employ a multi-threaded design for their pattern-matching engine [8]. GPGPU computing can speed up mathematical computations and string matching operations. Researchers have experimented with offloading CPU-intensive tasks of regular NIDS to the GPU with promising results [19] [21].

Hardware vendors are investing in technologies like OpenCL and CUDA, which promise to make GPU programing more suitable for general-purpose applications. There is also a drive to design hybrid systems that can leverage the computing power of the CPU and GPU for the purpose of accelerating network intrusion detection [1] [11]. Parallel implementations of clustering algorithms have been used before to detect malicious behaviour on the network and in antivirus software [13]. The framework presented in this paper is designed in a versatile manner and can benefit from the extra computing power available in a common GPU.

2.3 Related Work

Similar works have explored the high degree of parallelism of the k-means algorithm and proposed a series of optimizations on FPGA hardware [3]. Machine learning represents a current trend in network security and is applied to counter new types of threats. Security experts that develop applications performing some form of intrusion detection or malware scans are starting to look into new technologies to improve the performance and reliability of their software. Detection probability increases with the amount of data analysed. Solutions based on shallow packet inspection and machine learning models are expected to provide low overheads and reasonable amounts of accuracy. In practice this means that a second line of defence is needed to increase the security of a network or system. A consequence of this hybrid approach is that we can trade the accuracy of the first line of defence for speed and the workload of conventional security solutions performing DPI will be reduced. Thus the overall protection levels remain the same and the cost of DPI is amortized. Academic research in this field is looking for solutions to complement existing detection methods by classifying existing traffic patterns and flagging anomalous behaviour that deviates from accepted baselines [26]. Metadata is extracted from large traffic databases and live streams. Data acquisition techniques are often proprietary and customized to fit the needs of the environment they are deployed in. To differentiate from this practice, the current paper looks at the raw format exported by open source traffic interception tools and proposes a framework that does not require the adoption and any proprietary format or technology for extracting and storing packet information. Instead we rely on the particularities of the pcapng file format and try to encapsulate traffic metadata inside the pcapng traces in order to make them available to other network analysers and forensics tools.

3. Data Acquisition and Feature Extraction

3.1 Test Data Format

The first part of the framework described in this paper addresses data acquisition and feature extraction. The framework uses as input data trace files outputted by traffic interception software such as Wireshark [22] or tcpdump [18]. These tools capture all the traffic contents, including physical level headers, and encapsulate it using a specific format. The oldest and most common format is pcap. Pcap files contain a header with capture information like timestamp accuracy, maximum length of a packet and data link information. This simple format has become de facto standard and is recognized by most networking applications. Network contents are immediately available upon parsing the header and can be used to simulate the speed of packet processing tools if live traffic is not available. The format is useful for storage and presentation of network data but lacks some of the advanced features that may be useful for security analysts. For instance, any pre-processing information extracted from a pcap file would need to be stored in either a separate format or in a separate location, otherwise the original trace file

ICI Bucharest © Copyright 2012-2017. All rights reserved

might become unusable for other tools. To answer some of these needs a more flexible format was proposed: pcapng [6]. The great advantage of this format is that it allows applications to insert custom blocks of information into a trace file. Some custom blocks have been standardized to include information about network interfaces, statistics or name resolution. Applications are allowed to define new blocks and insert them into the trace file without breaking it for other tools (which will simply ignore any unrecognized block). Pcap files can be upgraded to pcapng format and pcapng traces can be downgraded to pcap, albeit with a loss of any non-standard data. The fact that it can be customized makes pcapng ideal for applications that want to extract and embedded metadata into network trace files.

NetFlow [5] is a commercial standard developed by Cisco for extracting metrics from network traffic. There is an entire industry built around the analysis of NetFlow output files; the metrics generated are used for a wide range of purposes including data mining, threat mitigation or marketing purposes. Real time acquisition requires dedicated hardware, but sample files are available for testing purposes online. The format consists of a series of templated blocks that can be manipulated by our framework in a similar manner to the pcapng custom blocks. Most information that can be extracted from pcapng files can be readily found in NetFlow files (the opposite is not true as NetFlow does not store packet payloads). An argument for selecting pcapng rather than NetFlow is that pcapng does not require any dedicated hardware or proprietary software in order to capture traces. The availability of packet contents allows the traces to be used by deep packet inspection tools if metadata analysis is deemed insufficient.

3.2 Pre-processing and Feature Extraction

In order to manipulate pcap and pcapng files we've created software pack (consisting of a library and a set of tools) that can parse trace files and compute relevant information based on a set of user defined metrics (for example: average packet size, total connection time, etc.). The library APIs can be used to store the data for easy retrieval inside the original trace. The pre-processing module offers further support for:

- Packet filtering
- Flow reconstruction
- Host metrics (CPU load, amount of memory used, etc.)

Implementation details for the library and its helpers can be found here [20]. The most important task performed at this stage is the generation of key-value pairs that can be used as input features for the next stage (clusterization).

The keys are associated with user-defined metrics. Each of them has a corresponding compute kernel that processes an atom trace. We refer to an atom trace as a subset of packets that were filtered during the previous stage of our application and may only be relevant for a very specific scenario. An example of an atom trace would be all the packets captured within a 1 ms window, regardless of upper level protocol or interface. This kind of specialized analysis might be useful in identifying botnet behaviour by singling out command and control traffic [16]. The compute kernel will output a scalar or vector result. The current framework implements this pre-processing stage on the CPU. If multiple files are used the process becomes I/O intensive and is unsuitable for GPUs. In our study, the kernels used for gathering the metadata do not analyse a packet's payload. The feature extraction stage can be performed in parallel on the CPU because there are no data dependencies between atom traces.

Once the features have been extracted, they are stored in memory or on the disk and a map is created connecting them to the original trace or flow. The end result is a matrix of floating point numbers that will serve as input for the machine learning algorithms.

4. Parallel Clustering

4.1 K-means

The floating point matrix is composed of values scaled to [0, 1] interval. Each of them represents a normalized feature computed during the previous stage. A parallel k-means algorithm is then applied to the dataset. The parallel frameworks used for the clusterization are CUDA, OpenCL and OpenMP. The results section will contain a performance summary for all 3 variants and a serial version. For the graphics APIs, the memory is transferred from the CPU to the GPU and the centroids are randomly initialized. Each GPU core iterates through all the centroids and attaches itself to the nearest, thus forming a new cluster. After

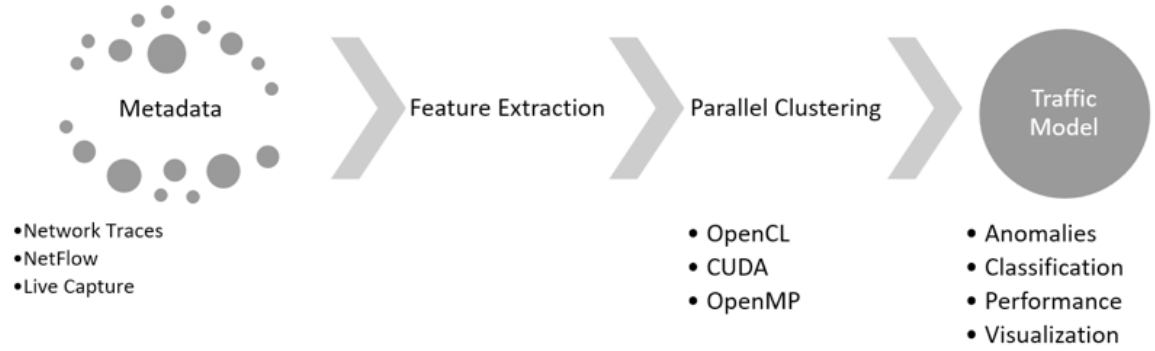

Figure 1. Framework workflow from data acquisition to detection or visualization

this operation, the data is copied back to the CPU and new centroids are computed. These actions are repeated until convergence is achieved. During the loop, centroid values and indices are updated between CPU and GPU. A global error value is computed for the final configuration. Based on the error value we decide whether to repeat the process using new random centroids or a different number of clusters. The implementation supports Euclidian distance as well as other metrics in order to be able to create outputs with different properties (for example an analyst may decide that the network source address outweighs the size of the packets in a trace).

The CPU-parallel version follows the same logic; the difference is we don't have to perform any memory transfers and the number of threads is much smaller (for an 8-core CPU, no performance improvements were recorded for 16 threads or more).

4.2 Result Visualization

After the algorithm has converged for the current configuration (given by the features and cluster count), we offer the possibility of visually displaying the data via Octave or Matlab. Network analysts can create scripts to single out flows or rely on visual inspection to identify anomalous behaviour. Our framework allows users to alter or create new features from the metadata and restart the computation. Multiple iterations with different features are intended to cover a wider range of scenarios. If any suspect behaviour is identified the feature values can be used to create filtering rules for future traffic. The results of the analysis can be stored alongside the original trace so that they are easily accessible for future iterations or to other tools. For example as new traffic is being captured, the old configuration and features could be used for a faster and more accurate classification. If the results are not conclusive, the features used for the shallow analysis can be recycled to provide hints for a more in depth inspection. The whole workflow from data acquisition to visualization can be seen in Figure 1.

Once deployed, the model can be subject to improvement. A high number of false-positives could be an indication that the original dataset used to build the model is not relevant for the current environment. Our framework allows network analysts to perform a quick investigation and determine if the how the model could be improved based on the new information.

If the detected anomaly is the manifestation of a previously unknown benign behaviour the k-means iterations can be restarted. The whole process is designed to be streamlined as to necessitate a minimum of effort from the human side.

5. Case Studies and Results

A constant challenge encountered during the development and testing process was the availability of network traces. In order to test our framework we selected a set of *pcap* and *pcapng* files from publicly available sources[123]. The objective was to analyse these traces and identify anomalous behaviour. To add more variation into our input data we created a "virtual packet/flow" component which was integrated at the "Trace/Log" stage mentioned in the figure above. This component could perform the following tasks on the extracted metadata:

- Multiply a flow (simulate similar behaviour by either copying the exact features or adding some minor variations)
- Create obvious anomalies (given the minimum, average and maximum values for a feature in a set of flows, it could generate new entries that are out of the "normal" range)

ICI Bucharest © Copyright 2012-2017. All rights reserved

These new entries would augment the original dataset and allow us to tune the algorithm during the training phase.

Attacks that can be singled out through flow clustering are distributed denial of service (DDoS). These attacks use remote terminals to send multiple service requests to their target, overwhelming its ability to respond to legitimate clients. The results experienced by the victim could range from increased response times to total shut down. The features our implementation will look in the attacker's traffic are origin of incoming traffic, increased packet frequency, unusual amount of data transferred, malformed requests, application characteristics (port numbers, protocol flags, etc.). These features will be computed and stored inside the trace files during the pre-processing stage. Feature toggling is done via a configuration file that is used by the clustering application. This allows the user to experiment with different combinations of features until the clustering provides adequate results. The features, stored as type-length-value (TLV) inside *pcapng* custom options are loaded by the parallel cluster application. In the case of GPU parallelism they are sent to the device together with an empty vector which will contain the desired output (cluster id for each element). The k-means loops are repeated until a clear pattern is identified and the clustering error reaches an acceptable minimum.

The input set consisted of metadata extracted from over 1400000 flows corresponding to approximately 50 TB worth of traces. In order to determine the best configuration for the current data, the k-means algorithm was run iteratively for clusters ranging from 1 to 32. The intentions was to determine the "drop-off" point where performance degradation no longer justifies an increase in accuracy.

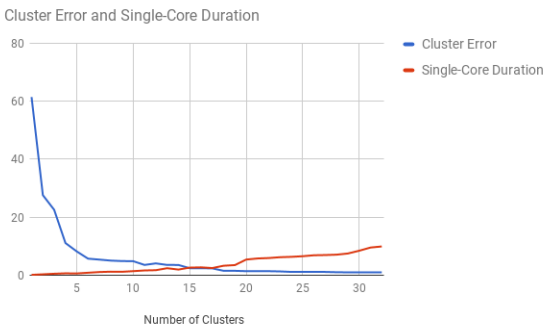

Figure 2. Evolution of clusterization error vs. computing time as the number of clusters increases

The above chart shows there is a significant drop in the error function after 6 or more clusters and increasing the number of clusters above 16 causes significant performance penalties for our specific dataset. For our experimental measurements we selected a configuration that uses 8 clusters and 8 features to map the captured traffic. The list of features is as follows:

1. Total amount of data transferred
2. Duration of the flow
3. Average time interval between two consecutive packets
4. Relation between source and destination address (if they are in the same network or not)
5. Type of protocol (TCP, UDP, IGMP, ICMP or other)
6. Number of packets in a flow
7. Transport layer protocol (boolean feature)
8. Use of encryption (boolean feature)

The performance was measured on a desktop with Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz quad-core CPU with hyperthreading and an GeForce GTX 970 graphics card. The operating system used was Linux (Ubuntu 16.04). The CUDA and OpenCL implementations were provided by proprietary NVidia SDK and drivers. The total computing time for the selected dataset was less than 1 seconds for all configurations.

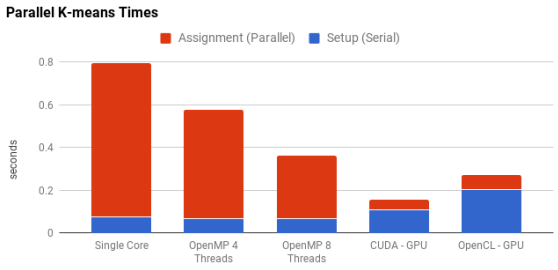

Figure 3. Performance of parallel k-means implementations

The results show an overwhelming advantage of GPGPU over parallel CPU implementation of k-means clustering: increasing the training speed by a factor of 5 and the compute-intensive centroid assignment by a factor of 15.

After the training phase we deployed the model and performed classification on 500000 new flows. The algorithm mapped the new entries to the previously computed centroids. The speedups for deployment phase were consistent with the previous training results.

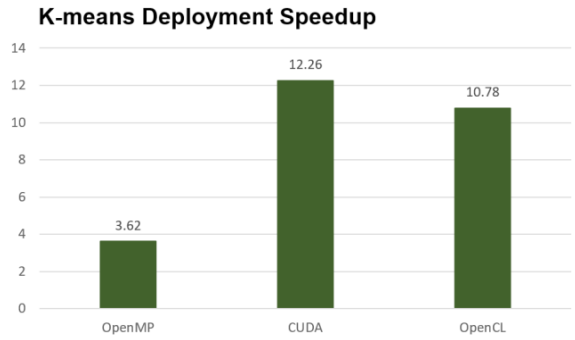

Figure 4. Mapping new traffic to existing model performed better on the GPU

The parallel kernel used to compute the results in Figure 4 can be seen in the following listing.

K-means nearest cluster GPU kernel

```
Input:
__global const double *data
const int nr_obj
const int nr_coord
__global const double *centroids
const int nr_centroids
Output:
__global int *membership
```

```
{
  int object_id = get_global_id(0);
  int index = 0;
  double min_dist = 1000000.0;
  double tmp;
  int i, j;
  __global double *current =

&data[object_id * nr_coord];

  for (i = 0; i < nr_centroids; ++i)
  {
    double dist = 0;
    __global double *cluster =

&centroids[i * nr_coord];
    for (j = 0; j < nr_coord; ++j)
    {
      tmp = current[j] - cluster[j];
      dist += tmp * tmp;
    }
    if (dist < min_dist)
    {
      min_dist = dist;
      index = i;
    }
  }
  membership[object_id] = index;
}
```

The visual representation of network flows can be viewed in figure 4 (features 3, 4 and 5 have been omitted for better visual representation).

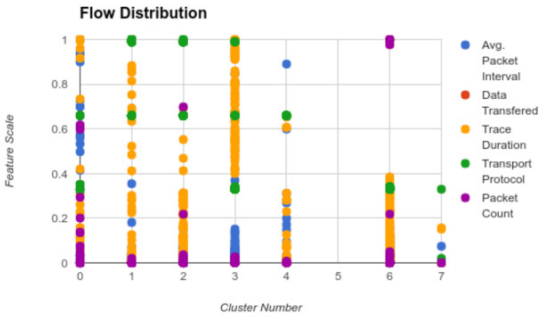

Figure 5. Flow visualization after k-means training stage. The dots in the chart represent the characteristics of the items being part of that cluster

Cluster centroids can be used to determine if future traffic matches an existing pattern. If the framework is trained with legitimate traffic, large clustering errors could be the sign of unusual activity on the network. Upon further inspection, a network analyst could determine if the unusual activity is part of a normal change in behaviour or a threat. The framework could be trained with samples from known attacks to recognize them amongst the data in the future.

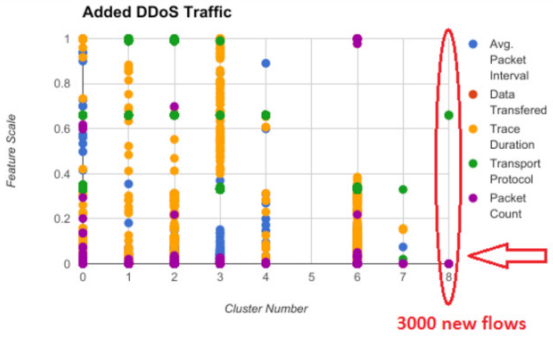

Figure 6. We are able to detect new traffic patterns by deploying the model computed in Figure 5

New elements can be appended to the training dataset and the cycle can be repeated with ease. To test this scenario, we added a set of DNS-based attack samples (DNS flood attack) to our model and increased the number of clusters. The new flows stood out from the original traffic and were easily identifiable from the output. Once singled-out, the anomalous traffic can be analyzed using dedicated tools to determine its true nature.

The parallel nature of the framework makes it ideal for deployment in distributed environments. Since the information processed is based on extracted metadata, the impact of data transfers in the network should be minimal. Besides faster

ICI Bucharest © Copyright 2012-2017. All rights reserved

execution, GPU-offloading allows the CPU to enter a low-power state earlier and reduce system load.

6. Conclusion and Future Work

Using our framework, the amount of information extracted for the proof of concept accounted for 0.002% of the original trace size: from a flow that averaged 3.2 MB in total bytes transferred we extracted 8 features as double precision floating point values. The accuracy of the model depends greatly on the amount of data used for training and feature selection. The framework described in this work allows users to add filters and customize feature selection and k-means arguments. We exploit the *pcapng* format to encapsulate the extracted information inside the original traces for future use and provide the means for rapid retrieval. The preprocessed data is organized in a way that makes it friendly for high performance applications to manipulate. Machine learning techniques such as clustering algorithms provide a fast way to classify the data and perform well in parallel environments. The experiments described in the current paper demonstrate the effectiveness of GPU-programming in accelerating the k-means algorithm.

The measurements favor CUDA over OpenCL on the test setup. The OpenCL implementation has a GPU runtime comparable with CUDA, but suffers a large overhead during the build-up stage. This penalty was attributed to the performance of the OpenCL system library.

Each of the components described here can be decoupled and act independently. This modularity makes it a useful tool in developing new network-analysis applications. The technologies used are publically available and run on most conventional devices. A careful selection of features can result in deployments that can quickly sort out large amounts of data. This could be useful in eliminating uninteresting traffic early on. Future efforts will aim to implement a real-time filtering mechanism based on the current experience.

In conclusion we advocate that the new *pcapng* trace format can be used to extract and store metadata from network traffic, which in turn can be used to perform shallow packet inspection. The case study presented offers an example of detecting DDoS attacks with a parallel implementation of a classic clustering algorithm. The results suggest that offloading this type of computation to the GPU can increase the performance of a software application serving as a network intrusion detection system.

Endnotes

[1]http://www.unb.ca/cic/research/datasets/nsl.html
[2]http://www.netresec.com/?page=PcapFiles
[3]https://www.simpleweb.org/wiki/index.php/Traces

REFERENCES

1. Aaron, S. S. & Balasubramanian, R. (2015). A Comprehensive Survey of Technologies for Building a Hybrid High Performance Intrusion Detection System, *International Journal of Computer Applications*, *113*(15).

2. Al-Jarrah, O. & Arafat, A. (2014, April). Network Intrusion Detection System using attack behavior classification. In *2014 5th International Conference on Information and Communication Systems (ICICS)* (pp. 1-6). IEEE.

3. Amaricai, A. (2017). Design Trade-offs in Configurable FPGA Architectures for K-Means Clustering, *Studies in Informatics and Control*, *26*(1), 43-48.

4. Bekkerman, R., Bilenko, M. & Langford, J. (Eds.). (2011). *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press.

5. Claise, B. (2004). *Cisco systems netflow services export version 9*.

6. ”Development/PcapNg” (n.d.). Retrieved June 17, 2017, from <https://wiki.wireshark.org/Development/PcapNg>.

7. Elbasiony, R. M., Sallam, E. A., Eltobely, T. E. & Fahmy, M. M. (2013). A hybrid network intrusion detection framework based on random forests and weighted k-means, *Ain Shams Engineering Journal*, *4*(4), 753-762.

8. Jiang, H., Zhang, G., Xie, G., Salamatian, K. & Mathy, L. (2013, October). Scalable high-performance parallel design for network intrusion detection systems on many-core processors. In *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems* (pp. 137-146). IEEE Press.

9. Jianliang, M., Haikun, S. & Ling, B. (2009, May). The application on intrusion detection based on k-means cluster algorithm. In *International Forum on Information Technology and Applications*, *2009 (IFITA'09)* (Vol. 1, pp. 150-152). IEEE.

10. Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P. & Turner, J. (2006, September). Algorithms to accelerate multiple regular expressions matching for deep packet inspection, *ACM SIGCOMM Computer Communication Review*, *36*(4), 339-350. ACM.

11. Lee, C. L., Lin, Y. S. & Chen, Y. C. (2015). A Hybrid CPU/GPU Pattern-Matching Algorithm for Deep Packet Inspection, *PloS one*, *10*(10): e0139301.

12. Lin, P. C., Lin, Y. D., Lai, Y. C. & Lee, T. H. (2008). Using string matching for deep packet inspection, *Computer*, *41*(4).

13. Lin, W. C., Ke, S. W., & Tsai, C. F. (2015). CANN: An intrusion detection system based on combining cluster centers and nearest neighbors, *Knowledge-based systems*, *78*, 13-21.

14. MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 1, No. 14, pp. 281-297).

15. McGregor, A., Hall, M., Lorier, P. & Brunskill, J. (2004, April). Flow clustering using machine learning techniques. In *International Workshop on Passive and Active Network Measurement* (pp. 205-214). Springer Berlin Heidelberg.

16. Strayer, W. T., Walsh, R., Livadas, C. & Lapsley, D. (2006, November). Detecting botnets with tight command and control. In *Proceedings 2006 31st IEEE Conference on Local Computer Networks* (pp. 195-202). IEEE.

17. Suthaharan, S. (2014). Big data classification: Problems and challenges in network intrusion prediction with machine learning, *ACM SIGMETRICS Performance Evaluation Review*, *41*(4), 70-73.

18. T. (2010, September 20). *Tcpdump/Libpcap public repository*. Retrieved June 17, 2017, from <http://www.tcpdump.org/>.

19. Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E. P. & Ioannidis, S. (2008, September). Gnort: High performance network intrusion detection using graphics processors. In *International Workshop on Recent Advances in Intrusion Detection* (pp. 116-134). Springer Berlin Heidelberg.

20. Velea, R., Apostol, I. & Patriciu, V. V. (2016, August). LightPcapNg: Implementing a library for general-purpose tracing based on PcapNg. In *2016 14th IEEE International Symposium on Intelligent Systems and Informatics (SISY)* (pp. 211-214). IEEE.

21. Vokorokos, L., Ennert, M., Čajkovský, M. & Radušovský, J. (2014). A Survey of parallel intrusion detection on graphical processors, *Open Computer Science*, *4*(4), 222-230.

22. *Wireshark* (n.d.). Retrieved June 17, 2017, from <https://www.wireshark.org/>.

23. Witten, I. H., Frank, E., Hall, M. A. & Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

24. Xiao, L., Chen, Y., & Chang, C. K. (2014, July). Bayesian model averaging of bayesian network classifiers for intrusion detection. In *2014 38th IEEE International Computer Software and Applications Conference Workshops (COMPSACW)* (pp. 128-133). IEEE.

25. Yassin, W., Udzir, N. I., Muda, Z. & Sulaiman, M. N. (2013, August). Anomaly-based intrusion detection through k-means clustering and naives bayes classification. In *Proc. 4th Int. Conf. Comput. Informatics* (No. 49, pp. 298-303). ICOCI .

26. Zhang, J., Jones, K., Song, T., Kang, H. & Brown, D. E. (2017, April). Comparing unsupervised learning approaches to detect network intrusion using NetFlow data. In *Systems and Information Engineering Design Symposium (SIEDS)* (pp. 122-127). IEEE.

ICI Bucharest © Copyright 2012-2017. All rights reserved