2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)

Real-time Hybrid Intrusion Detection System using Apache Storm

Goutam Mylavarapu
Computer Science Department
Oklahoma State University
Stillwater, Oklahoma
Email: goutam.mylavarapu@okstate.edu

Dr. Johnson Thomas
Computer Science Department
Oklahoma State University
Stillwater, Oklahoma
Email: jpt@cs.okstate.edu

Ashwin Kumar TK
Computer Science Department
Oklahoma State University
Stillwater, Oklahoma
Email: tashwin@okstate.edu

***Abstract*—Networks are prone to intrusions and detecting intruders on the Internet, is a major problem. Many Intrusion Detection Systems have been proposed to detect these intrusions. However, as the Internet grows day by day, there is a huge amount of data (big data) that needs to be processed to detect intruders. For this reason, intrusion detection has to be done in real-time before intruders can inflict damage, and previous detection systems do not satisfy this need for big data.**

Using Apache Storm, a Real time Hybrid Intrusion Detection System has been developed in our work. Apache Storm serves as a distributed, fault tolerant, real time big data stream processor. The hybrid detection system consists of two neural networks. The CC4 instantaneous neural network acts as an anomaly-based detection for unknown attacks and the Multi Layer Perceptron neural network acts as a misuse-based detection for known attacks. Based on the outputs from these two neural networks, the incoming data will be classified as "attack" or "normal." We found the average accuracy of hybrid detection system is 89% with a 4.32% false positive rate. This model is appropriate for real time detection since Apache Storm acts as a real time streaming processor, which can also handle big data.

***Keywords*—Hybrid intrusion detection, real time streaming, Apache Storm, CC4 neural network, multi-layered perceptron, ISCX 2012 Intrusion dataset.**

I. Introduction

Intruder attacks can be detected and restricted using several mechanisms like Cryptography, Authentication, etc., and the Intrusion Detection System is one of these. Intrusion detection is an attempt to detect the intruders in a computer system or a network. As the use of sophisticated computing tools in almost every aspect of life increases, risks and malicious intrusions are also increasing. We view network traffic data as big data that has the properties of high volume, high velocity and high variety. Therefore, network intrusion detection is a real-time big data problem.

Nathan Marz developed Apache Storm in December 2010. The idea behind this development is that there is a need for a real-time stream processing system that can be presented in a single program since all the other Big data processors like Apache Hadoop handles only batch processing. Apache Storm can process unbounded streams of data very fast. A benchmark clocked it at over a million tuples processed per second per node.

In this paper, the problem of handling big data in real-time was solved by Apache Storm. CC4 and MLP combined made the detection of known and unknown intrusions possible.

In the following section, we describe related work in this area. In section III, the tools used in this work are presented. Our approach to detecting Intrusions is described in section IV. Finally, we demonstrate the efficiency of this hybrid detection model in Apache Storm with experimental results using ISCX 2012 intrusion dataset.

II. Related Work

In 2003, Ravindra Balupari et al [1], proposed a real-time network based anomaly intrusion detection system, which analyzes data streams generated by Real-time Tcptrace. The real-time Tcptrace periodically reports statistics on all the open TCP/IP connections in the network. Then, using the Abnormality Factor method, statistical profiles are built for the normal behavior of the network services. Abnormality activity is then flagged as an intrusion. This approach has the advantage of being able to monitor any service without the prior knowledge of modelling its behavior. Although it is efficient, it fails in detecting flows from different network streams.

In 2013, Devikrishna K.S et al [2], proposed an Intrusion detection system that is able to perform live data capturing, so that the system can act as real-time IDS. For testing purpose they used KDD '99 benchmark dataset, which is indeed captured by the model for detection of intrusions. The processing of each packet flow is performed in sequence. The technique for detection used in this model is signature based. However, since the processing of streams is done sequentially which is not suitable for obtaining instantaneous or real-time results. In addition, this model fails to handle big data because of no parallel processing.

In 2011, Marcelo D. Holtz et al [3], proposed an architecture for distributed Network Intrusion Detection Systems where comprehensive data analysis is executed in a cloud computing environment. Network traffic, operating system logs and general application data are collected from various sensors in different places in the network, comprising networking equipment, servers and user workstations. The data collected from different sources are aggregated, processed and compared using the Map-Reduce framework, analyzing event correlations, which may indicate intrusion attempts and malicious activities. The detection mechanism proposed is hybrid and above all it can easily handle Big Data streams as it uses Apache Hadoop. Nonetheless, Apache Hadoop is a batch-processing tool, so,

978-1-4799-8937-9/15 $31.00 © 2015 IEEE
DOI 10.1109/HPCC-CSS-ICESS.2015.241

this model cannot perform real-time intrusion detection.

Almost all the architectures discussed above are tested using KDD '99 DARPA dataset which is not suitable in the current network environment [4], since it does not have any recent attacks and the current network routers can easily handle the attacks in KDD '99 dataset.

III. Neural Networks and Real-time Environment

A. CC4 Neural Network

The CC4 Neural Network is an instantaneously trained neural network proposed by Kak [5]. Such a memory requires quick learning. CC4 uses a feedforward network architecture consisting of three layers of binary neurons. The number of input neurons is equal to the length of the input vector plus one, the additional neuron being the bias neuron having a constant input of 1. The input layer requires the input to be in unary format. The number of hidden neurons is equal to the number of training samples, where each hidden neuron corresponds to one training sample. The hidden layer is connected to all the neurons in the input layer and similarly the ouput layer is fully connected to the hidden neurons.

The activation function is the binary step function, whose output equals 0 if the sum of all weighted inputs does not exceed its threshold, and equals 1 otherwise. The thresholds for all hidden and output neurons are taken as 0 for simplicity.

The weights on hidden neurons are assigned by using the training sample itself. For each training vector presented to the network, if an input neuron receives a 1, its weight to the hidden neuron corresponding to the training vector is set to 1. Otherwise it is set to -1. The bias neuron is treated differently. If s is the number of 1s in the training vector, excluding the bias input, and the desired radius of generalization is r, then the weight between the bias neuron and hidden neuron corresponding to this training vector is r - s + 1. Thus, for any training vector x_i of length n including bias, the input layer weights are assigned according to following equation:

$$y = \begin{cases} 1 & \text{if } x[n] > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Similarly, if the training vector produces a 1 at an output neuron, the weight from its hidden neuron to that output neuron is set to 1. Otherwise, it is set to -1.

The CC4 algorithm can be formally stated as follows [6]:

1) Radius of Generalization: Generalization vector is the user-defined value with which there will be changes in the output [7]. Therefore the value should be chosen carefully as it helps in classification of the input vectors based on the class of stored vectors. If the hamming distance between the new input vector and any of the stored vectors is less than or equal to the user defined radius, the outputs of all such stored vectors is considered for generating output of the input vector.

B. Two Layered Feed Forward Neural Network

The real-time intrusion detection system uses a two-layered feed forward neural network. This network is multi-layered perceptron and we call it as MLP. There are many activation functions and error minimizing mechanisms for MLP. We use sigmoid activation function, which works best for back propagation algorithm and gradient descent mechanism for error minimizing.

Algorithm 1 CC4 training algorithm

for each training vector x_i[n] **do**
  $s_i \leftarrow no.\,of\ 1s\ in\ x_i[1 : n-1]$
  for j = 1 to n -1 **do**
    if x_i[j] = 1 **then**
      $w_i[j] \leftarrow 1$
    else
      $w_i[j] \leftarrow -1$
    end if
  end for
  $w_i[n] \leftarrow r - s_i + 1$
  for k = 1 to m **do**
    if y_i[k] = 1 **then**
      $u_i[k] \leftarrow 1$
    else
      $u_i[k] \leftarrow -1$
    end if
  end for
end for

C. Apache Storm

The job inside a storm cluster is distributed to different components that are each responsible for performing a specific task. The input stream of the cluster is handled by a component called "spout". Spout sends a stream of tuples (named list of values). The transformation of the data in a bolt can typically occur in two ways, i.e. either a bolt produces the result or a bolt passes the transformed data to another bolt [8].

The combination of spouts and bolts in a network is called a topology. There can be links between any spout and any first layer bolts and any first layer bolt to any next layer bolt depending upon the need for the job. A simple storm topology is shown in figure 1.

D. Intrusion Dataset

We use the dataset for Intrusion Detection System created by Information Security centre of Excellence(ISCX) [9] in the year 2012. It is the benchmark dataset collected for seven days under practical and systematic conditions [9]. It is a labeled dataset consisting of normal and attack flows. The total number of flows is 2,450,324, among which the normal flows constitutes of 2,381,532 and the attack flows are 68,792. The total size of the dataset is 85 Giga Bytes. The attack scenarios of the dataset are infiltrating the network from inside, HTTP denial of service, Distributed denial of service using an IRC botnet and Brute force SSH.

IV. Hybrid Intrusion Detection System

A. Intrusion Dataset Pre-Processing

The total number of features available for each packet of ISCX Intrusion dataset is 19 [9]. In order to extract the essential subset of attributes out of these 19, we use the State

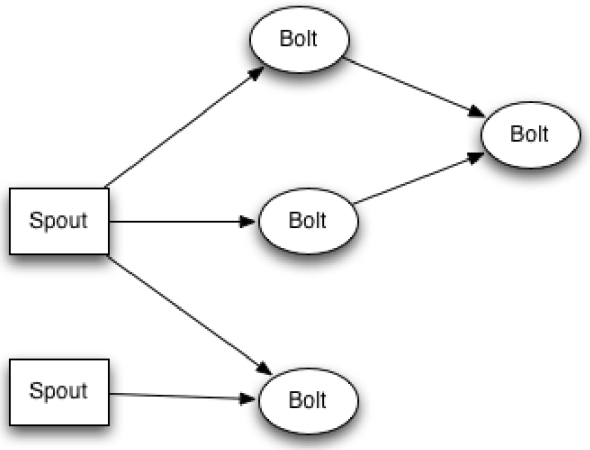

Fig. 1. A Simple Storm Topology

Space Search feature extraction method built in the Weka Data mining software [10]. In our work, best subset typically means the subset that gives highest accuracy. This feature extraction method suggested 11 important features. However, in our work we did an essential change by creating a new feature called ”duration”, which contains the same result, which is obtained from combining two features ”startDateTime” and ”stopDateTime”. The list of features used for the simulation are ”appName”, ”totalDestinationPackets”, ”totalSourcePackets”, ”direction”, ”source”, ”protocolName”, ”destination”, ”duration”, ”tag.” The proportions of normal and attack flows we use are 76% and 24% respectively and the proportions used for training and testing are 51% and 49% respectively. The packet distribution for training and testing from each day of the dataset collection can be observed in table I.

Date	Training		Testing	
	Normal	Attack	Normal	Attack
12th	20673	2086	0	0
13th	25692	1386	0	0
14th	16318	1978	0	0
15th	0	0	30809	37241
16th	33747	11	0	0
17th	0	0	22588	5219
Total	**101891**		**95857**	

TABLE I. DISTRIBUTION OF TRAINING AND TESTING DATA.

1) Conversion into decimal format: Initially the dataset is converted into decimal format. Each unique feature value is assigned a decimal number starting from 0, which will be its Unique Identification Number. This list is different for each feature in the dataset, resulting in the format where a packet contains the values for each feature indicating its unique identification number. In our implementation, among the two neural networks i.e. CC4 and MLP, the MLP neural network can work and indeed works faster with the decimal format. However, the CC4 neural network accepts only the unary format as input. So, the dataset therefore needs to be further processed for the CC4 neural network.

2) Conversion into floating point: In order to convert the resultant dataset into unary format, we consider each feature value and its contribution towards any kind of attack in the entire dataset. Based on the impact of each feature value on an attack, their contribution is calculated and the dataset is converted accordingly with the list of contribution values for each feature. It ranges from 0.0 to 0.9. Further, based on the floating values we created a quantization mapping table which indicates a unary value for each range of floating values. Hence the entire dataset is converted into unary format based on the mapping table II.

Range	Mapping
0.0	0000000000
0.00000001 - 0.0000001	0000000001
0.0000001 - 0.000001	0000000011
0.000001 - 0.00001	0000000111
0.00001 - 0.0001	0000001111
0.0001 - 0.001	0000011111
0.001 - 0.01	0000111111
0.01 - 0.1	0001111111
0.1 - 0.3	0011111111
0.3 - 0.55	0111111111
0.55 - 0.9	1111111111

TABLE II. QUANTIZATION MAPPING OF FLOATING POINT DATA TO UNARY DATA.

B. Proposed Storm Topology

The proposed Storm topology (see figure 2) consists of following components:

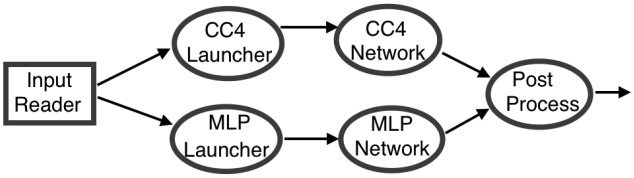

Fig. 2. Proposed Storm topology

Input Reader: This is a spout, which does most of the real-time processing. It sends each tuple to two bolts (CC4 Launcher and MLP Launcher) in decimal format by performing ”Shuffle Grouping”.

CC4 Launcher: This bolt converts the decimal format input to floating, then to unary, using quantization mapping table and sends the unary format of input to a bolt that executes the CC4 algorithm.

MLP Launcher: This bolt obtains the weights of best matched packet from the signatures and sends the data to a bolt that performs MLP intrusion detection.

CC4 Network: CC4 neural network algorithm will be executed by this bolt, and output is sent to post-process unit.

MLP Network: This bolt implements Multi layer perceptron algorithm and sends the output to post-process unit.

Post-process unit: This is a bolt that acts as final phase of the system. Based on the outputs from CC4 and MLP networks, it

gives us the classification i.e. normal, infiltrating the network from inside attack, HTTP denial of service attack, distributed denial of service attack, brute force SSH attack, unknown attack.

C. Implementation of the CC4 Neural network

The number of bits in the input to the CC4 bolt are 80 bits (8 features x 10 unary bits). The CC4 neural network size increases based on each unique input vector. Hence the size of network increases significantly if the input is given as 80 bits. This decreases performance. Hence, we use a divide and conquer strategy and implement the CC4 network individually for each input bit. This improves the performance since the unique input is minimal. Each CC4 network gives a one bit output i.e. either a 0 or 1. The total input (80 bits) gives 80 bits as output.

During the CC4 network training process, all the possible unique training packets are processed and the resultant of each packet is stored as a CC4 signature for testing. That is, during the testing phase, after we obtain the 80 bit output we compare those bits with the signatures we have. We calculate the best percentage match for the output, given the different signatures and if the best percentage match is above the threshold it outputs as 0 which indicates the normal and known attack flows. Whereas, if the percentage match is below the threshold, the CC4 outputs the result as 1, which indicates the unknown attack. After multiple simulations with the ISCX dataset we observed the threshold value to be best at 85%. The best percentage match graph for the CC4 network can be seen in figure 3. After obtaining the final output based on the threshold function the output is sent to the post-processing bolt.

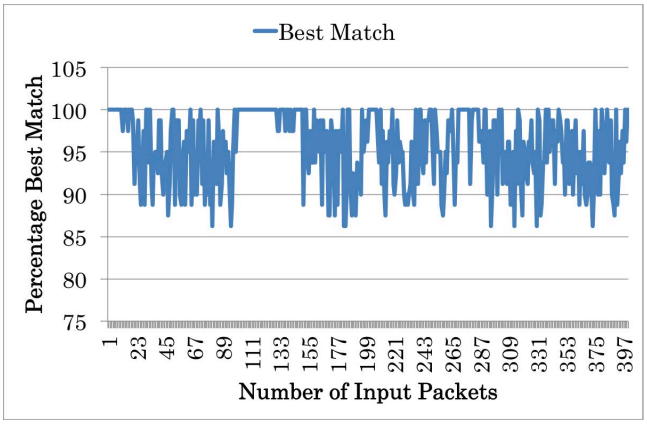

Fig. 3. Best Match Graph of CC4 Neural Network.

D. Implementation of the MLP Neural Network

We implemented the second neural network, multi layer perceptron, with 8 input neurons, 8 hidden neurons and one neuron for the output layer indicating 1 for attack and 0 for normal. 8 neurons were used because each neuron of the input layer carries each feature from the input vector.

During the testing phase, the input is searched with available signatures obtained from training. Even though we may or may not find the exact input we consider the most/best matched input among all the stored ones and then consider its weights for testing the network. If the output is 1 it indicates attack and normal if the output is 0. However, in our simulation we send this output to the post-processing phase/bolt for classification of the input packet.

E. Post Processing Unit

After receiving the output from both the bolts i.e. the MLP Network and the CC4 Network with respect to an input packet, the combination of outputs from the two neural networks creates the following possible cases in the final output.

Normal Packets: If the CC4 output indicates 0 i.e. the best percentage match graph is above a threshold level, and if the MLP outputs as 0, the post-processing unit declares the final output as a normal packet.

Known Attack Packets: If the CC4 output indicates 0 i.e. the best percentage match graph is above a threshold level, and if the MLP outputs as 1, the post-processing unit declares the final output as a known attack packet. It also indicates the class of attack obtained from the signatures.

Unknown Attack Packets: The detection of an unknown attack is difficult and in our simulation the CC4 neural network detects such attacks. The MLP is a purely training based neural network and so when there is a new attack for which the MLP is not trained, MLP will fail to detect it. However, according to [6], the CC4 neural network is prone to produce irregular percentage matches if the radius of generalization is not chosen properly. This radius varies from input to input i.e. it changes according to the dataset. So, in order to confirm the correctness of the CC4 result, we iterate the packet in CC4 for a certain number of times if the result is 1. Even after the iterations, if the result is still 1, we introduce a new MLP called MLP-2 and implement this as a separate bolt (see figure 4), which works offline. MLP-2 is trained with the new attack and signatures produced by MLP are updated so that the actual MLP will detect the new attack for which now it has the signatures. If after the iterations the CC4 produces the result as 0 by adjusting its errors, the system treats the packet as normal.

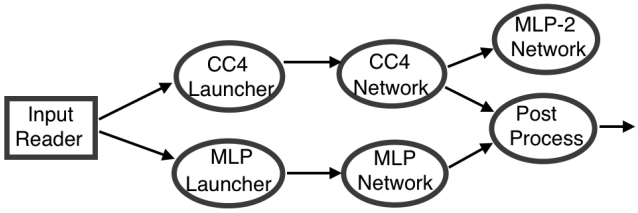

Fig. 4. Improved Storm topology

V. Experimental Results

This section includes the experimental results that are obtained during the simulation. All the simulations were run on a system with 4.00 GB of memory, 2.5 GHz Intel core i5 processor running with Macintosh OS X 10.9.5 operating system. The neural network coding and Apache Storm cluster development were written in Java.

A. Detection of New attacks using CC4 Neural Network

As explained in the previous sections the CC4 neural networks primary purpose in our work is to detect anomaly attacks. In order to prove the capability of the CC4 neural network for anomaly detection, we intentionally let the CC4 train for all attacks except the Distributed Denial of Service attacks, which are found on the 5th day of our intrusion dataset collection. Hence when the CC4 is given this as input, it should detect this as an unknown attack, since there is no signature available for this attack.

Figure 5 shows the best match graph of CC4 before it is trained

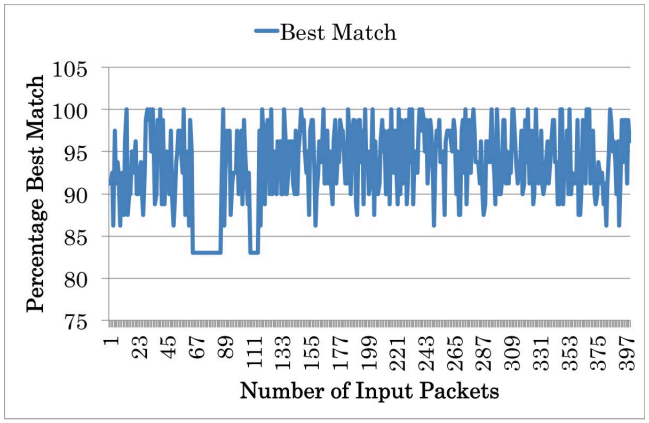

Fig. 5. Best Match Graph of CC4 Neural Network before training for Distributed Denial of Service attack.

for the DDoS attack. A few packets fall below the threshold i.e. 85% indicating that the signatures are not available. For this case the CC4 outputs the result 1.

When the CC4 is trained with Distributed Denial of Service attacks, the attack will be detected, since now it has the signatures stored for the DDoS attacks. After training for these attack packets, the CC4 is ran with the same dataset which was ran before but now the DDoS packets are detected since the best percentage match is above the 85% threshold, which can be observed in figure 6.

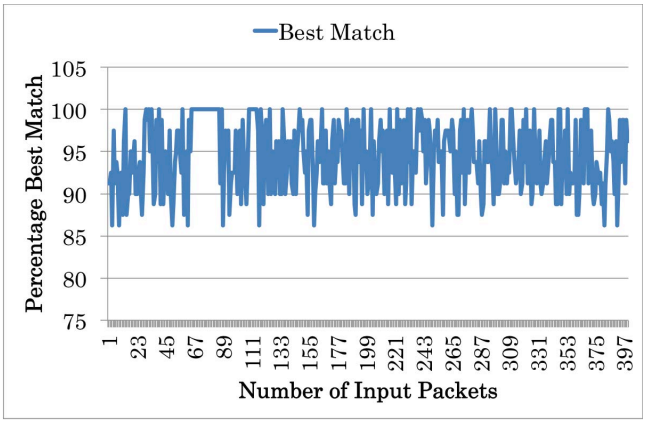

Fig. 6. Best Match Graph of CC4 Neural Network after training for Distributed Denial of Service attack.

B. Accuracy of CC4 based on Radius of Generalization

The CC4 neural network uses perspective learning unlike MLP, which means an input packet is sent only once through the network during the training phase, which means it has no back propagation. For this reason we use this neural network, since it is instantaneous. Unlike MLP, CC4 does not depend on the length of training for accuracy. However, it has a factor, radius of generalization, which distinguishes the trained packets to new packets and so this value cannot be static to any kind of input. After vigorous training and testing we found that, for ISCX intrusion dataset the CC4 has better accuracy in detecting unknown attacks for the radius of generalization, r = 1. It was different in the case of [6], since that work used KDD '99 DARPA dataset. The accuracy of the CC4 network can be observed in the figure 7.

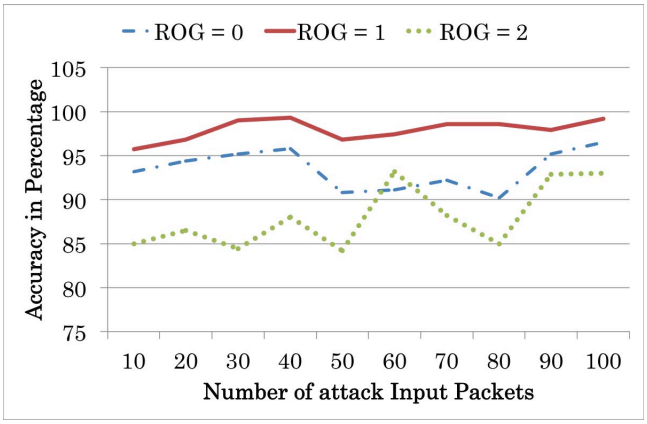

Fig. 7. Accuracy of CC4 Neural Network for different radii of generalization.

C. Use of Apache Storm in MLP Training

As discussed earlier, the MLP training phase is time consuming. This is because in order to improve the accuracy of the neural network, it needs to be trained over many iterations since MLP has back propagation to adjust weights. We are using Apache Storm in our implementation, not only to avail the advantage of real-time streaming but also to handle big data. Hence, we can expect faster processing in Apache Storm compared to normal training outside the Apache Storm cluster. As MLP training takes more time, we tested the training process in Apache Storm to take advantage of big data handling. The results in figure 8, shows that storm although shows normal execution time for small input (i.e. fewer number of packets) it reforms better and better with an increase in number of packets or data size. A data size of 100,000 packets shows that Apache Storm produces an improvement of 25.56% over regular training.

D. False Positives and False Negatives

Recognition systems, in most cases cannot have 100% accurate results, since they are trained using data, which may or may not match the nature of data that is used for testing. Hence, there are possibilities for false positives and false negatives of the recognition systems. In our work, we used

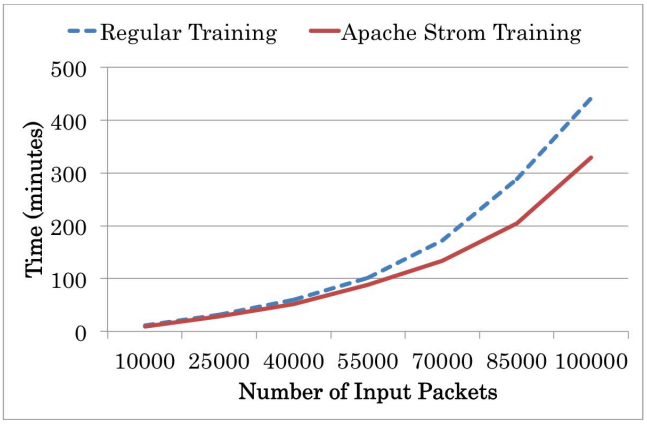

Fig. 8. Training of MLP inside and outside Apache Storm.

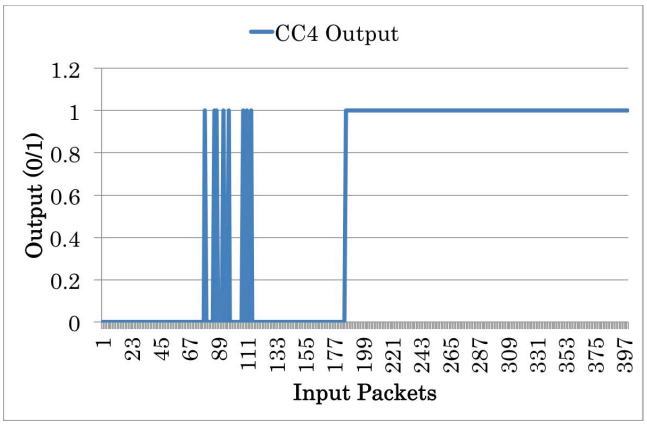

Fig. 10. False Positives and False Negatives of CC4 Neural Network.

two neural networks for the hybrid intrusion detection system. In order to test the false positives and false negatives of the CC4 and MLP neural networks we used 400 random packets from the dataset which constitutes of 185 normal packets and 215 attack packets. These are tested for false positives and false negatives sequentially with normal and attack packets. The MLP neural network shows 0% false positive and 0% false negative rate with respect to atacks, which can be seen in figure 9. However the CC4 neural network has 4.32% of false positive rate and 0% false negative rate with respect to attacks. Figure 10 shows the false positives and false negatives of the CC4 neural network.

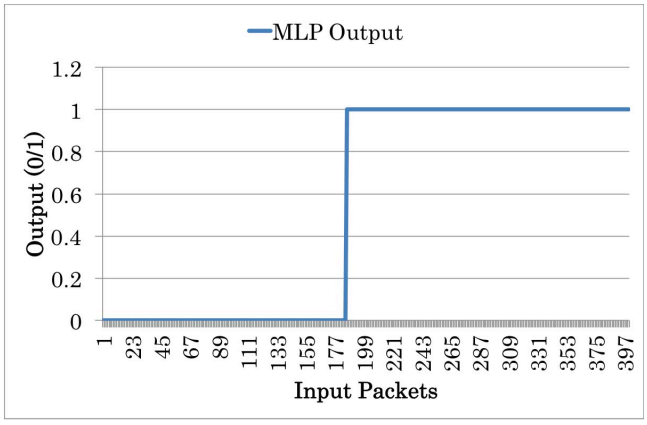

Fig. 9. False Positives and False Negatives of MLP Neural Network.

VI. Conclusion

A successful and efficient real-time hybrid intrusion detection system has been developed, that consists of an anomaly detection for new attacks using CC4 neural network and misuse based attack detection using multi-layer perceptron implementation. The entire simulation was performed inside Apache Storm cluster with ISCX 2012 Intrusion dataset.

For future work, in order to be able to work in any network environment and to store large amount of signatures, we can take advantage of batch-processing of big data using Apache Hadoop. Apache Storm can be combined with Hadoop to get better results, which support the integrated data processing, combining real-time streaming and Hadoop distributed file system. In order to test the accuracy of this model in the real world, a network can be used which is able to introduce natural real-time intrusions with numerous packets and diverse network scenarios. Since there is no limit to the number of neural networks, we can implement this model using different neural networks which require less pre-processing, unlike the CC4 neural network, which requires input data in unary format and conversion to unary format requires a number of steps.

References

[1] R. Balupari, B. Tjaden, S. Ostermann, M. Bykova, and A. Mitchell, "Real-time system security," B. Tjaden and L. R. Welch, Eds. Commack, NY, USA: Nova Science Publishers, Inc., 2003, ch. Real-time Network-based Anomaly Intrusion Detection, pp. 1–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=903866.903869

[2] D. K and R. B, "An Artificial Neural Network based Intrusion Detection System and Classification of Attacks," *International Journal of Engineering Research and Applications*, vol. 3, no. 4, 2013.

[3] M. Holtz, B. David, and R. Timoteo, "Building Scalable Distributed Intrusion Detection Systems Based on the MapReduce Framework," *Revista Telecommunications*, 2011.

[4] V. Engen, J. Vincent, and K. Phalp, "Exploring discrepancies in findings obtained with the kdd cup '99 data set," *Intell. Data Anal.*, vol. 15, no. 2, pp. 251–276, Apr. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1971751.1971760

[5] S. Kak, "New algorithms for training feedforward neural networks," *Pattern Recognition Letters*, vol. 15, no. 3, pp. 295 – 298, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167865594900620

[6] R. Pillay, "Instantaneous Intrusion Detection System," Master's thesis, Department of Computer Science, Oklahoma State University, 2010.

[7] S. Reddy, "Generalization and Efficient implementation of CC4 Neural Network," Master's thesis, Department of Computer Science, Oklahoma State University, 2008.

[8] "Apache storm," https://storm.incubator.apache.org/, last accessed 9 July 2014.

[9] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2011.12.012

[10] Z. Markov and I. Russell, "An introduction to the weka data mining system," *SIGCSE Bull.*, vol. 38, no. 3, pp. 367–368, Jun. 2006. [Online]. Available: http://doi.acm.org/10.1145/1140123.1140127