2015 IEEE Symposium Series on Computational Intelligence

Conversion of Decision Tree Into Deterministic Finite Automaton for High Accuracy Online SYN Flood Detection

Marcin Luckner
Warsaw University of Technology
Faculty of Mathematics and Information Science
00–662 Warszawa, ul. Koszykowa 75, Poland
Email: mluckner@mini.pw.edu.pl

***Abstract*—While collecting data from network traffic, one can create classifiers that recognize threats, anomalies, or other events. The set of labelled NetFlow records collecting traffic statistics is a very useful source of decision rules that classify the records. These rules can be created automatically using machine learning techniques. However, the classifiers learned on such records may recognise only past events and cannot recognise current events, because not all data were collected. A deterministic finite automaton is a classifier that can recognise events online. However, the automaton is hard to project in case of complex issues. The paper proposes how to convert a decision tree into a deterministic finite automaton. The decision tree learns how to recognise threats using the collected data. Consequently, the set of decision rules is transformed into a finite automaton that can detect events before the full complement of data is collected. The method is limited to small trees, but can solve real problems. As an example, the detection of the TCP SYN flood attack is presented. For that example, the created automaton has the same high accuracy ratio as the decision tree, but can take decisions over three times faster.**

I. Introduction

One of the modern issues is a quick detection of events in huge flows of data such as network traffic. The events that we want to detect among network traffic are commonly connected with cyber–threats. Therefore, it is extremely important to detect the threats when they are present to allow the administrators to counteract.

A typical approach in machine learning assumes that the classifier is trained and tested on the static data sets. Such a classifier can effectively handle events with a stable measure of observed features, but when connected to a dynamical stream, it can misclassify arising events.

One of possible solutions is a classifier dedicated to a stream analysis, such as a finite automaton. This classifier changes its state on the base of observed features. When the automaton reaches the final state, it alarms about the detected even. A common approach to design the finite automaton uses expert knowledge.

This papers proposes how to convert a decision tree into a deterministic finite automaton automatically. This allows us to replace expert knowledge by machine learning techniques. This approach also changes the method of a network traffic analysis from offline classification into an online detection.

To be more specific, the method converts a decision tree – trained on static data set – into a deterministic finite automaton (DFA). The automaton can work on data stream and detect the events classified by the tree. That allows the model to detect the ensuing threats. Moreover, when the decision tree faces events not included in the set of recognised events, it must misclassify the event. On the other hand, the automaton recognises only one event, but other events – different than neutral traffic – are not misclassified. They are ignored instead.

The proposed method uses the decision tree to generate a set of rules that detects the recognised event. Next, the rules are used to create the transition function for the nondeterministic finite automaton. Finally, the automaton is transformed into the deterministic finite automaton.

The proposed method solves real problems. In this paper, we present the problem of the detection of a TCP SYN flood attack. The SYN flood attack is a common cyber–threat when attackers flood a victim with synchronisation requests. From the decision tree that recognises threats with a high accuracy – less than 1 percent of false positives and zero false negatives – the method creates the deterministic finite automaton that repeats the tree decisions but works directly on data stream. That allows the automaton to take decisions over three times faster than the decision tree.

The rest of the paper has the following structure. After a short presentation of previous work in Section II, Section III describes the presented method. Section IV presents the application of the method for the TCP SYN floods detection. Finally, Section V summarises the results and describes future work.

II. Previous work

Several propositions how to detect TCP SYN flood attacks using machine learning techniques can be found in [1], [2], [3], [4], [5], [6], [7]. The methods used to detect TCP SYN flood attacks include: Kononen self–organising maps, deterministic finite automata, time automata, and decision trees.

The decision tree is one of common and well working techniques among all the examined approaches. Moreover, the

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.21

decision tree gives a clear set of decision rules. Therefore, we focused on this technique in our work.

A very useful form of a learning set for the decision tree that recognises cyber–threats is a labelled set of NetFlow records. The NetFlow record contains statistics that describe features typical for the recognised threat. The very wide set of features was proposed in [8]. However, we decided to limit a set of observed features to the standard template offered by nProbe [9], which is an open source NetFlow probe that can works with gigabit networks.

The decision tree trained on real data that describe both neutral network traffic and attacks can recognise threats with high accuracy. However, the reaction of the system that uses the tree learned on NetFlow records will depend on the period that was used to collect statistics. Therefore, if the records describe the whole attacks then the tree will recognise only attacks that have already been finished.

For that reason, a method that changes the state with each new part of data can be preferred to detect the threats. Such a method can alarm the operator during the attack. In this group, there are the methods based on finite automata [10], [11], [12] or the Hidden Markov Model [13]. Both can be easily implemented on hardware devices to create fast detectors that work directly on data steam. However, in this solution it is often assumed that the events are already described by the regular expressions.

Our method builds a bridge between those two approaches. One can create the decision tree that works with a very high accuracy and then transform the decision rules to the deterministic finite automaton that can apply the rules to create online detection system.

III. Method

This section describes the conversion from a decision tree through a nondeterministic finite automaton into a deterministic finite automaton. First, the notation is proposed. Next details on the conversion are given.

A. Decision tree

A binary decision tree $G = (V, E)$ is described by the set of vertices V and the set of edges E. The tree contains two types of vertices. Leaves describe classes, while other vertices contain decision rules R.

The rule $\rho \in R$ is defined as the function

$$\rho : F \times T_F \to V \tag{1}$$

Where F is a set of features and T_F describes the set of thresholds created for the given feature.

When the values of the features are given by real numbers or by an ordered set, the rules chose the next vertices in the tree using the following formula:

$$\rho_j(f_i, t_i) = \begin{cases} v_s & \text{if } \breve{f}_i < t_i^j \\ v_t & \text{if } \breve{f}_i \geq t_i^j \end{cases} \tag{2}$$

Where $f_i \in F$ is a feature, $\breve{f}_i$ is an observed value of the feature, $t_i \in T$ is a set of threshold for the feature f_i and t_i^j is the threshold for the feature f_i used in the rule ρ_j. Additionally, $v_s \in V$ is the vertex selected when the rule is fulfilled and the vertex $v_t \in V$ is selected in the other case.

When an observation is classified a string of rules $\rho_1, \rho_2, \ldots, \rho_n$ is examined. The final rule ρ_n reaches one of the leaves connected with classes. The reached class is returned as the classification result.

B. Finite automaton

The proposed method uses two types of finite automata. A non–deterministic finite automaton is created from the decision three. Next, the non–deterministic finite automaton is transformed into a deterministic finite automaton. The classifier in this form can be easily implemented as software or hardware. A detailed description of the finite automata is given in [14].

1) Deterministic finite automaton: A deterministic finite automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F) \tag{3}$$

has the following components:

Q	a finite set of states
Σ	a finite input alphabet
q_0	the start state $q_0 \in Q$
F	the set of final states $F \subset Q$
δ	the transition function.

The transition function is defined as

$$\delta : Q \times \Sigma \to Q \tag{4}$$

The deterministic finite automaton starts in the start state q_0. Next, the automaton changes the states on the base of the state and the input symbols. If one of the final states is reached the input word is accepted.

2) Nondeterministic finite automaton: A nondeterministic finite automaton (NFA)

$$M = (Q, \Sigma, \delta, q_0, F) \tag{5}$$

has the identical components as DFA except the transition function, which is defined as

$$\delta : Q \times \Sigma \to 2^Q \tag{6}$$

The nondeterministic finite automaton works in the same way as the deterministic finite automaton, but in the same state and with the same input symbols the automaton can reach various states.

A nondeterministic finite automaton can be converted into a deterministic finite automaton. We assumed that the nondeterministic finite automaton has not ϵ-movements (the movements without reading data).

For each state q_i we examine the results of the transition function. If the result for each symbol $s \in \Sigma$ is the empty set or the single state q_j then the transition function for the state q_i stays the same. If the result for any symbol s is a set of two or more states then we create a new state labelled by the states reached by the transition function.

For example, the nondeterministic transition from the state q_i may reach the states $\{q_j, q_k\}$, when the symbol s is read

$$\delta(q_i, s) = \{q_j, q_k\}. \tag{7}$$

The transition is transformed into a deterministic one defined as

$$\delta(q_i, s) = [q_j, q_k] \tag{8}$$

where $[q_j, q_k] \in Q$ is a new created state.

For the new state, the transition function must be defined. The function is defined for each symbol s from the alphabet Σ separately. For each state that was used to create the state label, we look for all sets that can be reached directly with the symbol s. The union of the states creates a label for the new state.

For example if

$$\delta(q_j, s) = \{q_s, q_t\} \wedge \delta(q_k, s) = \{q_s, q_r\} \tag{9}$$

where $s \in \Sigma$ is a symbol from the alphabet, and $q_j, q_k, q_s, q_t, q_r \in Q$ are states then

$$\delta([q_j, q_k], s) = [q_s, q_t, q_r] \tag{10}$$

where $[q_s, q_t, q_r] \in Q$ is a new created state.

For each new state the procedure must be repeated. More detailed information about the transformation from nondeterministic finite automata into deterministic finite automata can be found in [14].

C. Conversion of decision tree into finite automaton

The transformation proposed in this section – from a decision tree to a deterministic finite automaton – converts an offline classifier into an online classifier. However, the process has several limitations.

First, the created finite automaton detects only one class. For this class we must find all strings of rules that reached the leaves with the detected events.

A second limitation is the size of the transformed tree. The details of this problem are given in the next section.

The algorithm has three steps. In the first step, the strings of rules are generated from the tree. Next, the strings of rules are used to create a nondeterministic finite automaton. In the last step, the nondeterministic finite automaton is transformed into a deterministic finite automaton.

1) Generation of strings of rules: The algorithm observes each path that goes from the root to a leaf labelled with the detected event. Each node on the path was achieved because one case of the formula (2) had been fulfilled. When we write down all fulfilled rules we will obtain the string of rules $\rho_1, \rho_2, \ldots, \rho_n$.

The order in the string depends on the detection tree structure. However, the finite automaton can collect data observed by the rules in various orders. That shows the next limitation of the method because all string must be permuted.

As the result of the permutation process, we have $n!$ strings of rules instead of one. Therefore, the method can be not effective for the complex decision tree. However, when the observed feature is described by a monotonic function, the number of permutation can be limited by the following steps.

Let us define a family of rules as

$$\Lambda(\rho_j(f_i, t_i)) = \{\rho_l(f_k, t_k) : k = i\} \tag{11}$$

where $\rho_i, \rho_l \in R$ are rules defined for the same feature $f_i \in F$. The rules, defined as (2) must base on various values of the thresholds $t_i^j, t_i^l \in T_F$.

From the structure of the rules (2) it is clear that in the same string of rules all rules from the family $\Lambda(\rho_j(f_i, t_i))$ must be ordered by the thresholds. When for the thresholds t_i^k, t_i^l, t_i^m we have the relations $t_i^k < t_i^l < t_i^m$ then the rules in the string can be only localised as $\rho_k(f_i, t_i), \rho_l(f_i, t_i), \rho_m(f_i, t_i)$ or $\rho_m(f_i, t_i), \rho_l(f_i, t_i), \rho_k(f_i, t_i)$. Moreover, an observation of the feature f_i influences all rules from the family $\Lambda(\rho_j(f_i, t_i))$. Therefore, the rules from the same family should not be separated by rules from other families.

The introduction of a family of rules allows us to reduce the permutation problem. It is enough to create a permutation of families instead of a permutation of rules.

2) Creation of finite automaton: To create a finite automaton we must define the alphabet Σ, the states Q, and the transition function δ. The alphabet Σ is created from the set of rules R that should be fulfilled on the decision path. Each symbol r_i from the alphabet represents one rule ρ_i. The special states: the initial state q_0 and the final state $q_A \in F$ are added to the Q.

Next, for each path of the rules $r1, \ldots, r_{k+1}$ the line of states is created $q_0, q_1, q_2 \ldots, q_k, q_A$. The number of states is equal to the number of rules plus one. The following transitions are added to the automaton $\rho(q_0, r_1) = q_1, \rho(q_1, r_2) = q_2 \ldots, \rho(q_k, r_{k+1}) = q_A$.

The states from the next paths have no connection with the created path except the states q_0 and q_A. Different paths can start with the same rule. Therefore, the created automaton is nondeterministic and in the next step it is converted into a deterministic automaton using the algorithm described before.

IV. SYN FLOODING ATTACKS DETECTION

To illustrate that the proposed conversion can be used to solve real problems we created a detector of TCP SYN flooding attacks.

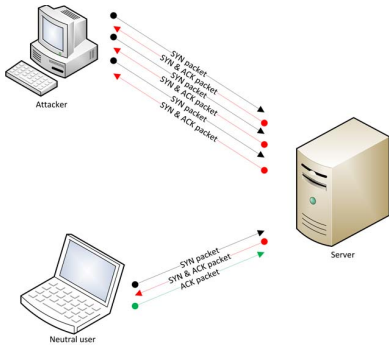

Fig. 1: TCP SYN flood attack compared with neutral session

The TCP SYN flooding attacks use the handshake mechanism of Transmission Control Protocol (TCP) and its limitation in maintaining connections. The traffic between a sender and a receiver is labelled with special flags according to the protocol. The attack uses the flags to confuse the receiving server.

A conversation based on the TCP starts when a sender sends a SYN request. As a response, the receiver sends a SYN/ACK packet. In normal cases, the sender responses with an ACK packet. However, the attacker sends a next SYN request instead. Until the SYN/ACK packet is acknowledged by the sender, the connection remains in a half–open state for a period of up to a TCP connection time out limit. The receiver tries to manage all half–open connections that can finish with exhaustion of resources. Eventually, all later connection requests will be dropped.

Figure 1 presents the differences between a conversation with an attacker and a neutral user.

A. Data

The described detectors were created using data from the WiSNet laboratory [15]. The data contain both neutral traffic and TCP SYN flood attacks. They are labelled according to the types.

Three different hosts preformed the attacks. Each host launched attacks at various rates including three low–rate (0.1, 1, 10 pkts/sec) and two high-rate (100, 1000 pkts/sec) instances. Each instance covered a period of two minutes. Neutral traffic was collected from the edge router of a medium–sized Internet Service Provider for about 10 minutes [15].

The data were described by NetFlow [16] records. Such a record contains the following information: the source IP address, the destination IP address, the IP protocol, the source port, the destination port, and the IP type of service.

The NetFlow version 9 can collect additional features. To detect the SYN flooding attacks we collected the following features: the incoming flow bytes (IN_BYTES), the incoming flow packets (IN_PKTS), the IP protocol ($PROTOCOL$), the cumulative value of all flow TCP flags (TCP_FLAGS), the longest packet of the flow ($LONGEST\_FLOW\_PKT$), and the shortest packet of the flow ($SHORTEST\_FLOW\_PKT$). The features are defined by nProbe templates [9].

The NetFlow records were limited to the six given features for two reasons. First, information about the TCP flags and the number of packets seems to be essential in the SYN flood detection. Second, to add additional features that are not listed in the nProbe templates it is necessary to create nProbe plug-ins, which is not an easy task.

The NetFlow records collected by nProbe were split into the learning set and the testing set. The learning set contains 312342 NetFlow records that describe attacks and 208639 records of neutral traffic. The testing set contains 624684 records of attacks and 190067 of with neutral traffic.

B. Decision tree

Discrimination rules that separate TCP SYN flood attacks from the rest of the traffic were created for the learning sets. This task was done using a C&RT tree [17], which creates clear decision rules.

Figure 2 presents the obtained tree. Four leaves contain neutral traffic. Only one leaf contains TCP SYN flood records.

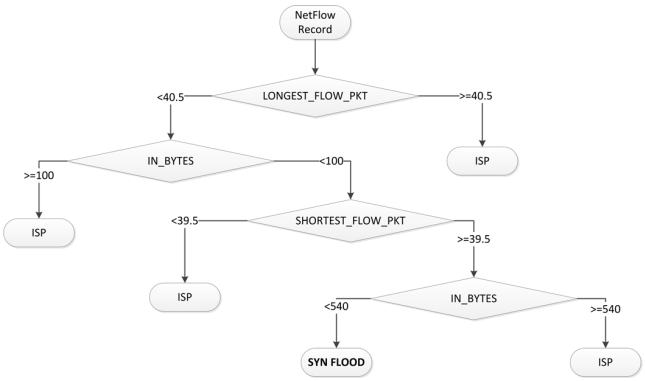

Fig. 2: TCP SYN flood detection tree

Only three features were selected to create the tree: $SHORTEST\_FLOW\_PKT$, IN_BYTES, and $LONGEST\_FLOW\_PKT$. The IN_BYTES feature was used twice in the decision results.

The created tree recognised all attacks. Among the neutral traffic 287 and 59 NetFlow records were recognised as attacks in the learning set and the testing set respectively. In both cases, that is less than one percent of the NetFlow records.

However, the attacks were labelled when the whole NetFlow record had already been calculated. A finite automaton can detect attacks during the calculation of the features.

For that, the path to the tcpflood leaf must be defined. The following rules lead to the tcpflood leaf:

$$\neg\rho_1 \quad : \quad LONGEST\_FLOW\_PKT >= 40.5, \tag{12}$$
$$\neg\rho_2 \quad : \quad IN\_BYTES >= 100, \tag{13}$$
$$\rho_3 \quad : \quad IN\_BYTES < 540, \tag{14}$$
$$\neg\rho_4 \quad : \quad SHORTEST\_FLOW\_PKT < 39.5. \tag{15}$$

C. Nondeterministic finite automaton

There is only one path in the decision tree that goes to the tcpflood leaf. To create a finite automaton we need to create all permutations of the decision rules from the tree. However, the rules ρ_2 and ρ_3 are connected by the feature IN_BYTES. Therefore, we do not have to change their order. The created automaton is presented in Figure 5.

Formally, the automaton according to the formula (5) is defined as

$$M_{NFA} = (\bigcup_{i=0}^{18} q_i \cup \{q_A\}, \{\neg 1, \neg 2, 3, \neg 4\}, \delta_1, q_0, \{q_A\}). \quad (16)$$

Table I describes the transition function δ_1.

The symbols $\{\neg 1, \neg 2, 3, \neg 4\}$ describe the rules from the decision tree. The symbol i means that the rule ρ_i was fulfilled and the symbol $\neq i$ means that the negation of the rule ρ_i was fulfilled. The construction of the automaton causes that the automaton accepts only words that describe the sequence of rules leading to the tcpflood leaf.

TABLE I: Nondeterministic finite automaton

δ_1	$\neg 1$	$\neg 2$	3	$\neg 4$
q_0	$\{q_1, q_2\}$	$\{q_3, q_4\}$	-	$\{q_5, q_6\}$
q_1	-	-	-	q_7
q_2	-	q_8	-	-
q_3	-	-	q_9	-
q_4	-	-	q_{10}	-
q_5	-	q_{11}	-	-
q_6	q_{12}	-	-	-
q_7	-	q_{13}	-	-
q_8	-	-	q_{14}	-
q_9	q_{15}	-	-	-
q_{10}	-	-	-	q_{16}
q_{11}	-	-	q_{17}	-
q_{12}	-	q_{18}	-	-
q_{13}	-	-	q_A	-
q_{14}	-	-	-	q_A
q_{15}		-	-	q_A
q_{16}	q_A	-	-	-
q_{17}	q_A	-	-	-
q_{18}	-	-	q_A	-
$\overrightarrow{q_A}$	-	-	-	-

The first row in Table I shows us that the automaton is nondeterministic. From the initial state q_0 we can go to more than one state using the same symbol. In the next step, a deterministic automaton will be created.

D. Deterministic finite automaton

The created nondeterministic finite automaton was transferred into the deterministic finite automaton. For that, each set of the states was grouped as a single state. This approach creates the smallest automaton.

Figure 6 presents the deterministic finite automaton. The automaton – according to the formula (3) – is defined as

$$M_{DEF} = (Q, \{\neg 1, \neg 2, 3, \neg 4\}, \delta_2, q_0, \{q_A\}) \quad (17)$$

where,

$$Q = \bigcup_{i=11}^{18} q_i \cup \{q_0, [q_1, q_2], [q_3, q_4], [q_5, q_6], q_7, q_8, [q_9, q_{10}]\} \cup \{q_A\}. \quad (18)$$

Table II presents the transition function δ_2 for the automaton M_{DEF}.

TABLE II: Deterministic finite automaton

δ_2	$\neg 1$	$\neg 2$	3	$\neg 4$
q_0	$[q_1, q_2]$	$[q_3, q_4]$	-	$[q_5, q_6]$
$[q_1, q_2]$	-	q_8	-	q_7
$[q_3, q_4]$	-	-	$[q_9, q_{10}]$	-
$[q_5, q_6]$	q_{12}	q_{11}	-	-
q_7	-	q_{13}	-	-
q_8	-	-	q_{14}	-
$[q_9, q_{10}]$	q_{15}	-	-	q_{16}
q_{11}	-	-	q_{17}	-
q_{12}	-	q_{18}	-	-
q_{13}	-	-	q_A	-
q_{14}	-	-	-	q_A
q_{15}		-	-	q_A
q_{16}	q_A	-	-	-
q_{17}	q_A	-	-	-
q_{18}	-	-	q_A	-
$\overrightarrow{q_A}$	-	-	-	-

The automaton detects an attack when the state q_A is reached. When another state is reached then traffic cannot be classified as an attack. The automaton detects all attacks form the WiSNet data set.

E. DFA Versus Tree

The automaton implements exactly the same decision rules as the decision tree. Therefore, the classification accuracy is the same about 100 percent. However, the automaton can change a state with each new packet and reject an input before the whole NetFlow record is calculated.

We used data from the WiSNet laboratory [15] to simulate network traffic observed by the automaton. The same data, which were used to train and test the decision tree, were examined packet by packet to detect when the automaton recognised the final class of the flow between two IP addresses.

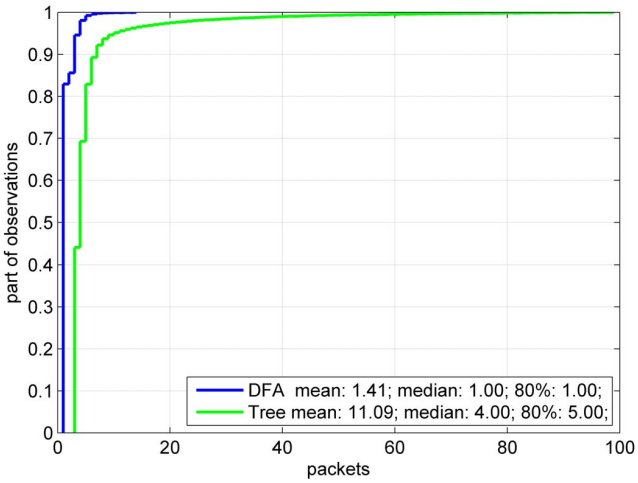

Fig. 3: The number of the packets used to classify traffic by the automaton and the tree. The detailed numbers are given for mean, median, and 80 percent of the records.

Figure 3 compares the number of packets necessary to take the final classification decision for the tree and the

deterministic finite automaton. The mean number of packets that have to be analysed to recognise the class of a flow is nearly ten times bigger for the decision tree than for the deterministic automaton. However, the mean for the tree is so big because of NetFlows with very big number of packets (over several thousand). A better measure to compare both methods is the median. The median for the decision tree is four times bigger than for the automaton.

Moreover, the automaton needs to analyse only the first packet to decide about the classification of the NetFlow in over 80 percent of cases.

Figure 4 presents a relative reduction of the number of analysed packets. We see that in most cases the automaton needs to analyse no more that 33 percent of all packets summarised in the NetFlow to recognise the class of the record.

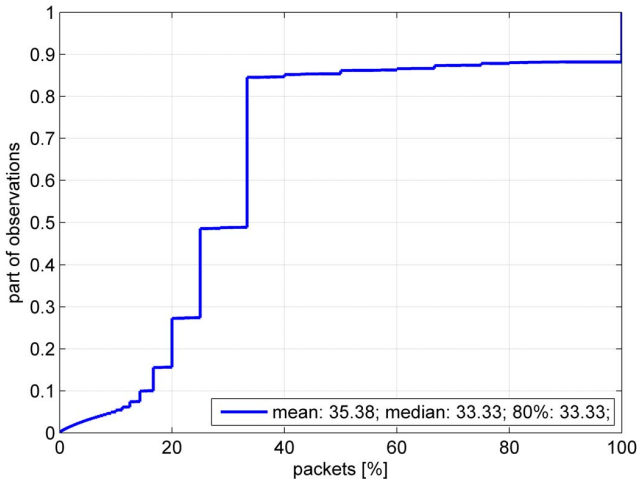

Fig. 4: The percent of the packets used to classify traffic. The number of packets necessary to recognise the class by the automaton to the number of all packets in the NetFlow. The detailed numbers are given for mean, median, and 80 percent of the records.

V. Conclusion

In the paper, we presented how to convert a decision tree into a deterministic finite automaton. In the result, an offline classifier is replaced by an online detector. The process works on one of the classes recognised by the tree. The other classes as well as other events that are not considered by the decision tree are ignored.

In the current form, the method has several limitations. First, only one class is detected by the automaton. On the other hand, the automaton can ignore the events that must be misclassified by the decision tree.

Second, the method uses permutations of the decision rules generated by the tree. That is the most costly part of the algorithm that may limit the method to small decision trees. However, the number of the permutations can be reduced when the observed features are monotonic functions.

Even with the described limitations, the method can solve real problems. In the paper we presented the online TCP SYN flood detector that can raise an alarm during an attack.

This work focused on the conversion algorithm that replaces a decision tree with a deterministic finite automaton. However, we presented the real application. The method can be used to detect the TCP SYN flood attacks. Moreover, the automaton analysed less data than the decision tree. As the result, decisions about classification of traffic can be taken over three times faster.

The presented results are the results of the simulation on data from the repository. The method must be verified online on real traffic. It has to be done in the future.

Additionally, we want to circumvent the limitation of the method and test the method against a wider spectrum of cyber–treats, that can be detected using a decision tree [18].

Acknowledgment

The research is supported by the National Science Center, grant No 2012/07/B/ST6/01501, decision no UMO–2012/07/B/ST6/01501.

References

[1] J. Haggerty, T. Berry, Q. Shi, and M. Merabti, "Diddem: a system for early detection of tcp syn flood attacks," in *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 4, Nov 2004, pp. 2037–2042 Vol.4.

[2] H.-R. Tang, R.-L. Sun, and W.-Q. Kong, "Wireless intrusion detection for defending against tcp syn flooding attack and man-in-the-middle attack," in *Machine Learning and Cybernetics, 2009 International Conference on*, vol. 3, July 2009, pp. 1464–1470.

[3] S. Haris, R. Ahmad, and M. Ghani, "Detecting tcp syn flood attack based on anomaly detection," in *Network Applications Protocols and Services (NETAPPS), 2010 Second International Conference on*, Sept 2010, pp. 240–244.

[4] S. Haris, R. Ahmad, M. Ghani, and G. Waleed, "Tcp syn flood detection based on payload analysis," in *Research and Development (SCOReD), 2010 IEEE Student Conference on*, Dec 2010, pp. 149–153.

[5] P. Sangmee, N. Thanon, and N. Elz, "Anomaly detection using new mib traffic parameters based on profile," in *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, vol. 2, April 2012, pp. 648–653.

[6] T. Arai and S.-y. Nishizaki, "Model checking approach to real-time aspects of denial-of-service attack," in *Communications and Information Processing*, ser. Communications in Computer and Information Science, M. Zhao and J. Sha, Eds. Springer Berlin Heidelberg, 2012, vol. 288, pp. 86–94.

[7] M. del Pino, P. Bez, P. Lpez, and C. Araujo, "Self-organizing maps for early detection of denial of service attacks," in *Recent Advances in Intelligent Engineering Systems*, ser. Studies in Computational Intelligence, J. Fodor, R. Klempous, and C. Surez Araujo, Eds. Springer Berlin Heidelberg, 2012, vol. 378, pp. 195–219.

[8] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 50–60, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1071690.1064220

[9] L. Deri, "nprobe: an open source netflow probe for gigabit networks," in *In Proc. of Terena TNC 2003*, 2003.

[10] T. Liu, Y. Sun, and L. Guo, "Fast and memory-efficient traffic classification with deep packet inspection in cmp architecture," in *Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on*, July 2010, pp. 208–217.

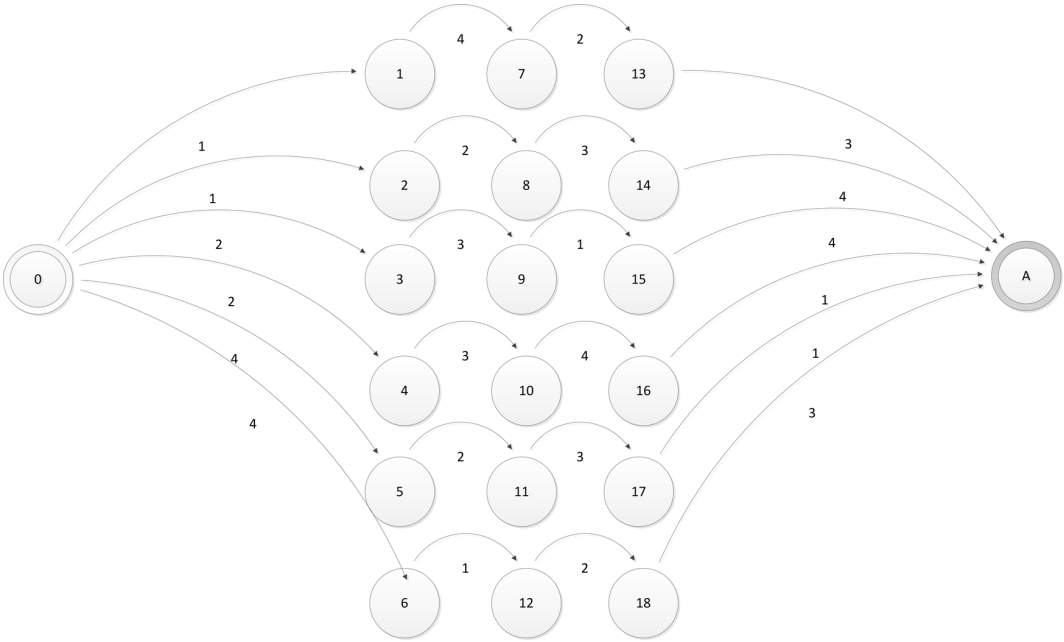

Fig. 5: Nondeterministic finite automaton

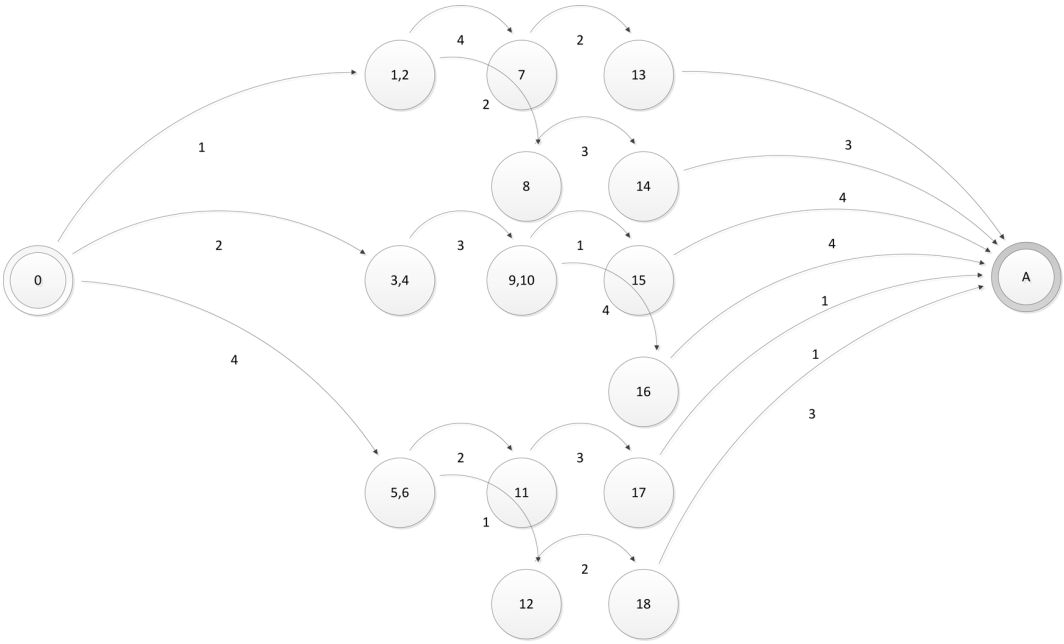

Fig. 6: Deterministic finite automaton

[11] Y. Liu, L. Guo, M. Guo, and P. Liu, "Accelerating dfa construction by hierarchical merging," in *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, May 2011, pp. 1–6.

[12] A. Khalid, R. Sen, and A. Chattopadhyay, "Si-dfa: Sub-expression integrated deterministic finite automata for deep packet inspection," in *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, July 2013, pp. 164–170.

[13] R. Rangadurai Karthick, V. Hattiwale, and B. Ravindran, "Adaptive network intrusion detection system using a hybrid approach," in *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, Jan 2012, pp. 1–7.

[14] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[15] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, ser. RAID'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 161–180.

[16] B. Claise, *Cisco systems NetFlow services export version 9*, 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[17] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[18] N. Wattanapongsakorn and C. Charnsripinyo, "Web-based monitoring approach for network-based intrusion detection and prevention," *Multimedia Tools and Applications*, pp. 1–21, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11042-014-2097-9