Behavioral Clustering of Non-Stationary IP Flow Record Data

Christian Hammerschmidt*, Samuel Marchal†, Radu State*, and Sicco Verwer‡
*SNT, University of Luxembourg
†Aalto University
‡TU Delft
Email: {christian.hammerschmidt,radu.state}@uni.lu; samuel.marchal@aalto.fi, s.e.verwer@tudelft.nl

***Abstract*—Automated network traffic analysis using machine learning techniques plays an important role in managing networks and IT infrastructure. A key challenge to the correct and effective application of machine learning is dealing with non-stationary learning data sources and concept drift. Traffic evolves overtime due to new technology, software, services being used, changes in user behavior but also due to changes in network graphs like dynamic IP address assignment. In this paper, we present an automatic online method to detect change-points in network traffic based on IP flow record analysis. This technique is used to segment an observed behavior into smaller consecutive behaviors differing one from another. The segmented traffic is used to learn small communication profile characterizing accurately the activities present between two observed change-points. We validate our method using synthetic data and outline a real-world application to botnet hosts behavior modeling.**

I. Introduction

Network management and intrusion detection systems heavily rely on automatically identifying communication behaviors [1], [2]. To keep up with new malware and changing usage patterns, models and detection rules must be updated regularly. The network traffic generated by a machine evolves overtime depending on its current activity e.g. different attacks (denial of service, malware spreading, scan, command-and-control, etc.) performed by a malicious host. Applying machine learning to the whole network traffic observed leads to build one model representing several activities and does not give a fine-grained representation of the communication behavior. A key assumption in machine learning algorithms is the stationarity of the data source [3]. In practice, concept drift, i.e. the relationship between input and desired output, is changing over time and needs careful treatment. However, identifying a change, e.g. to relearn models, is not straightforward and is often done every predefined time period without considering changes in input data.

To learn fine-grained communication profiles representing single activities and to cope with concept drift, we propose a technique to detect changes in communication behavior. It relies on a change-point detection algorithm that can identify when a drift affected the input data and aid the decision to learn a new model [4]. An example application is for monitoring incoming honeypot traffic and detect the different attacks performed. We can easily isolate the traffic corresponding to one given attack in order to extract an accurate signature that can further be applied for the detection of this specific attack. Learning signatures from specific, well-defined malicious activity reduces the chance for false alarm when applied in intrusion detection.

Several conceptual approaches have been identified for concept drift and change-point detection [4]. The methods, often coming from statistics, have been applied to various problems in networking individually, e.g. detection of SYN-flood attacks [5], DoS attack [6] or network traffic behaviors [7] using the CUSUM algorithm. These methods often consider a single-feature time series and need to be aggregated for multiple features. Anomaly detection in networking is often seen as a change-point identification problem, see e.g. [8]. Within finite state machine learning, [9] presents a change-point detection method leveraging a similarity threshold on the probability distributions of the automaton transitions. In contrast with previous work, our approach clusters the continuous traffic flow of a given host in different segments corresponding to its different activities over time, working on aggregated features, rather than interpreting change-points in individual features as indicators of a specific activity itself. We show that it is particularly relevant, but not limited, to automata-based algorithms. This class of algorithm has been successfully applied in traffic analysis and malware detection [10], [11], [12] and is used as the basis to apply our clustering technique on IP flow records.

To summarize the contributions of this paper:

- We introduce the concept of *freshness*, an indicator to identify new observations in data and use it to detect change-points in network traffic.
- We present how to use *freshness* computation on unlabeled IP flow records to cluster the traffic generated by a host according to its different activities overtime.
- We validate this clustering technique on synthetic and real-world data to model the different behaviors of botnet hosts.

The remaining of the paper is organized as follows: Section II provides background knowledge about finite states machine learning. Section III introduces our solution for detecting change-points in communication profiles built from finite state machines. Section IV and V present experiments and their evaluation. We conclude in Section VI.

978-3-901882-85-2

II. Background

A. Probabilistic Deterministic Finite Automata (PDFA)

Finite state automata are a type of automaton model often used to describe computation and processes in a formal way. We use finite state automata with probabilities, called probabilistic deterministic finite automata (PDFA). Introductions to the field of automaton theory can be found in [13] and [14]. A *Probabilistic Deterministic Finite Automaton (PDFA)* is quintuple $A = \langle Q, T, \Sigma, q_0, P \rangle$ where Q is a finite set of states, $T : (Q, \Sigma) \to Q$ are labeled transitions with labels drawn from an *alphabet* Σ, $q_0 \in Q$ is the start state. The probability matrix P gives the probability of observing event $a \in \Sigma$ in state q by $p_{a,q}$. A PDFA starts in the start state q_0 and generates strings by traversing transitions and drawing events using P. For example, the probability of generating abc is given by $p_{a,q_0} p_{b,q_1} p_{c,q_2}$ where $q_1 = T(q_0, a)$ and $q_2 = T(q_1, b)$. Figure 1 (right) shows the graphical depiction of an automaton with 4 states represented as nodes and 5 transitions represented as directed edges. Edges labels l describe the transition symbol $l \in \Sigma$. Transition probabilities are omitted in the figure.

B. State-Merging Algorithms

Inferring a PDFA from a given set of observations is the task of finding a PDFA accepting the words representing the observed behavior. State-merging algorithms [15] outperformed other approaches in learning competitions [16]. For a set S_+ of observed input behaviors, encoded as words over an alphabet Σ called the *input sample*, the goal is to find a (non-unique) *smallest* PDFA A that is *consistent* with S_+. To be consistent with S_+, a PDFA needs to satisfy a type of Markov property: for every prefix $s \in S_+$ that reaches the same state q in A, the sample probabilities of future suffixes $P(s' \mid s) = count(ss')/count(s)$ of the states are not significantly different. State-merging algorithms typically start from a a tree-shaped PDFA A constructed from the input sample S_+, called augmented prefix tree acceptor (APTA). Figure 1 (left) shows a prefix tree for a small input sample. It contains all samples from S_+ in a directed graph, using the symbols of the samples in S_+ as labels for the edges. Two samples share a path if they share a prefix. A state merging algorithm reduces the size of the automaton iteratively by reducing the tree through heuristically merging a pair of states in A. This reduces the size of the automaton (number of states), and introduces loops which generalize the model beyond the input training sample. Too much generalization introduces false positives, too little leads to many false negatives. Figure 1 (right) depicts the automaton after a state-merging operation.

III. Behavior Change Detection and Clustering

A. Building Communication Profiles with PDFA

Our goal is to extract key behaviors from IP flow records using features listed in Table I that are extracted from each IP flow. We use the notion of *communication profile*, previously presented in [17], to reduce IP flow data into a compact snapshot describing an observed behavior over a time period. Formally speaking, a *communication profile* is a PDFA learned from sequences of IP flow records as described in Section II.

Features	Description	Values
$protocol$	transport protocol of the flow	categorical: tcp, udp, etc.
$time$	time since previous flow started	timestamp
$duration$	duration of the flow	time in ms
$pakets$	Count of packets exchanged	numerical
$data_{exc}$	Amount of data exchanged	numerical, in KB
$data_{rec}$	Amount of data received	numerical, in KB

TABLE I: Features of IP flow records. Hosts of interest are filtered by IP address and remaining fields are input for learning. $Time$ is used to aggregate sliding windows.

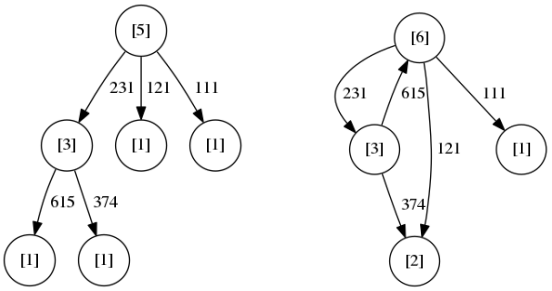


Fig. 1: Left: a prefix tree for a dataset containing the words $\{231, 231.615, 231.374, 111, 121\}$. States contain occurrence counters. Transitions are labeled with the symbol firing them. Right: an automaton obtained by merging the transitions 615 with the root and 374 with the state lead to by 121.

To obtain input words for PDFAs from IP flow records, we follow the approach presented in [17] to convert each IP flow record into a discrete symbol. Each numeric feature of a record, as given in Table I, is put into a discrete bin and represented by the bin number. We use percentiles as bin boundaries, e.g. using 25-percentile ranks we create four bins (labeled $[0, 1, 2, 3]$) and calculate feature values such that 25%, 50%, 75% and 100% of the data fall below. For categorical values (*protocol*), we assign each feature value a unique number. The symbolic representation of an IP flow record is the concatenation of the values for all its five features (excluding *time*) and represents a letter e.g. *01323*. All flow representations starting within a short, fixed time are concatenated by sliding a window over the recorded flows, incrementing the start of the window one flow at a time. The input for a communication profile then is a sequence of symbols forming a window.

B. Measuring Information and Detecting Change Points

To handle data sources that evolve overtime, it is necessary to detect changes in their observed behavior. In our case, the data is IP flow records. The detection can either happen before encoding as in Section III-A, or after. We chose the latter, as it allows to observe all relevant features at the same time and ensures that observed changes in behavior are relevant to the learning algorithm. Formally speaking, we observe the joint distribution over all features, rather than the individual distributions.

The prefix tree is created from the dataset by inserting a state between each symbol of a word, and introducing a transition for each symbol in each sample to connect the states. Words with common prefixes share states and transitions. To measure the amount of new data in a sample with respect to a given prefix tree, we calculate how many new transitions are created versus the number of transitions present in the APTA. The number of new transitions created measures the longest suffix of the sample, not yet contained in the ATPA, and tells us how new the information in the sample is. Formally, for a given APTA A and a word w, the ratio $\Delta = \frac{[\![w]\!]}{|w|}$ is called the *freshness* of w with respect to A. $[\![w]\!]$ is the number of transitions newly created in APTA A when adding the sample w and $|w|$ is the length of w (total number of transitions) in APTA A. $[\![\cdot]\!]$ denotes the length of the word w minus the length of its longest prefix in A. When w is a set, we define $[\![w]\!] = \sum_{w_i \in s} [\![w_i]\!]$ as the sum of states created from the samples in the set, and $|w|$ as $\Sigma_{v \in w}|v|$. The freshness ranges from 0 to 1. Low values indicate that the input sample already has many duplicates or long prefixes in the APTA.

The freshness of the set $\{121, 111, 231, 231.615, 231.374\}$ with respect to an empty prefix tree is calculated as follows. There are 7 transitions in total. Only the words 231, 231.615 and 231.374 share one transition, so there are 5 unique transitions, and the single shared prefix is 231. The freshness therefore is $\frac{5}{7}$. The freshness with respect to the prefix tree in Figure 1 (left) is 0, as all data of the set is already in the tree.

C. Clustering using Freshness

The freshness is used to divide the input dataset into multiple segments along the identified change-points: It measures redundancy in observed data. The freshness is not a monotone and falling function, but for redundant and repetitive data, it decreases. It can be used as an indicator for clustering. A period of monotone falling freshness with low values indicates redundant observed data, indicating the lack of new behavior. Subsequent increases in freshness, and extreme values indicate new information contained in the input data. By identifying these changes we can determine when the behavior of a host changes by introducing new, different behaviors.

IV. Experiments

A. Datasets

We use data from two different sources to validate our approach. To validate the concept of *freshness*, we analyze synthetic IP flow records generated using the `fs` flow-level traffic generator [18]. It allows to generate homogeneous network traffic presenting a given characteristic. Table II lists the four *activities* with distinct traffic patterns created for the experiments (*Synthetic_X*). A second dataset is composed of publicly available real-word botnet traffic and is represented as two scenarios [19]. *Scenario_10* contains 106,316 malicious flows from botnets and 18,565 benign flows from clean hosts among 5,180,852 flows in total collected over 4.75 hours for ten infected hosts. *Scenario_11* contains 8,161 botnet flows among 40,836 flows in total collected over 0.26 hours for three infected hosts.

Name	Type	#flows	duration
Synthetic_1	normally distributed	19,241	15min
Synthetic_2	short TCP flows	41,620	15min
Synthetic_3	variable rate UDP flows	23,771	15min
Synthetic_4	SYN flood	87,121	15min
Scenario_10	UDP flooding / ICMP spam	5,180,852	4h15min
Scenario_11	botnet traffic	40,836	15min

TABLE II: Flow captures. *Synthetic_X* represents traffic synthetically generated. *Scenario_X* represents botnet hosts traffic.

B. Data Preparation

We extracted words from the IP flow traces using the strategy presented in Section III-A. Bin boundaries were calculated on an initial segment of the data using percentiles in 25%-steps (4 bins). This initial segment of data was discarded from any further experiment to prevent any knowledge transfer. All flows irrespective of their duration, starting within a fixed time period (time window) are aggregated. Windows are advanced on a per-flow level i.e. a new word is computed for every new starting flow.

C. Change-points and Cluster Identification

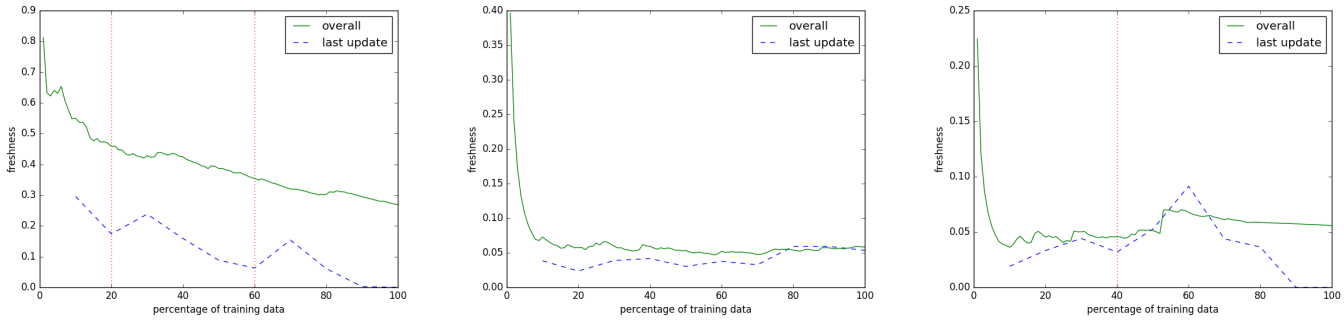
We observe two indicator values based on freshness. First, the freshness of the *overall* dataset, indicating the redundancy globally. Second, the freshness of the *last update* containing 10% of the dataset with respect to the APTA created up to the update. This indicates how much new information was in the last update. Between two minima in the freshness, we learn a communication profile using the `dfasat` software package [20]. The obtained communication profiles indicate whether a given sequences of IP flow records agrees with its associated segment of the data or not by accepting or rejecting a sequence of IP flow records aggregated into a window.

V. Results

A. Behavioral Clustering

The freshness computation is used to identify different network activities in a single IP flow capture as presented in Section IV-C.

1) Synthetic Datasets: The synthetic traces from Table II are appended to produce single traces of different activities. Table III presents the combinations used for experiments e.g. *Synthetic_1-2* means one capture containing *Synthetic_1* flows followed by *Synthetic_2* flows. Figure 2(a) shows the freshness evolution (overall and last update) in *Synthetic_3-1-4* traffic. We observe two minimas in *last update* freshness at 20% and 60%, indicating two changes in the underlying network activity. In contrast, we can observe that the *overall* freshness is not a good indicator of activity change with no abrupt variation in this plot. Table III presents as well the count of network activities in the different synthetic captures and the count of activities we identified using *last update* freshness computation, which shows a 100% identification rate.



(a) (b) (c)

Fig. 2: Freshness evolution for *Synthetic_3-1-4* (a), *Scenario_11* (b) and *Scenario_10* (c). The plain line shows the freshness of the first $x\%$ of the dataset with respect to an empty APTA. The dashed line shows the freshness of the last update containing 10% of the data with respect to the APTA at point x. Figure (a) presents three different activities, (b) one and (c) two, separated by the minima observed in *last update* freshness (red vertical lines).

Experiment	Activities present	Activities identified
Synthetic_1-2	2	2
Synthetic_1-3	2	2
Synthetic_3-1-4	3	3
Scenario_10	2	2
Scenario_11	1	1

TABLE III: Count of labelled different activities in the traces and activities identified by clustering using *freshness*.

2) Botnet Dataset: Figures 2(b)(c) show the freshness evolution for the botnet dataset. Figure 2(b) shows a single behavior in *Scenario_11* with fluctuating global and local freshness, but no clear extreme values. Figure 2(c) shows one minimum around 40%. According to the dataset description, the botnet first started UDP flooding followed by ICMP spam with increasing bitrates. The change in protocol together with the bandwidth increase causes to extract longer windows with different symbols, leading to the spike in *last update* freshness at around 60%.

The global conclusion of this experiment is that the freshness is efficient at distinguishing different activities in network traffic for both synthetic and real-world data. However, we must consider the freshness computed with respect to the APTA under ongoing generation i.e. *last update*.

B. Cluster Communication Profiles

By using the freshness as a change-point indicator, we split the dataset of malicious botnet flows into clusters. On each cluster we learn a PDFA as a communication profile. Table IV shows the performance obtained by applying this method to data of *Scenario_10*. The dataset is split in clusters 1 and 2 according to Figure 2(c), and profiles 1 and 2 are learned to identify data from the respective cluster. We use benign traffic, i.e. traffic not from botnets, as a control. It should be reject, as a profile learned on botnet data characterizes only botnet behaviors. Because of behaviors present in both clusters, e.g. command and control flows, each PDFA also accepts, some flows from the other clusters, i.e. some behaviors are assigned to both clusters. Likewise, because profile 2 was only trained on cluster 2, it rejects behaviors only present in cluster 1. In this table we can see that profile 1 accepts all windows from cluster 1 while most of them are rejected by profile 2. Almost all benign activity is rejected by both profile learned from malicious traffic. This proves that our method is able to differentiate malicious behaviors in network traffic and discriminates malicious activities from benign activities with high accuracy.

	Cluster 1	Cluster 2	Benign
Profile 1	100%	58%	6%
Profile 2	38%	60%	0%

TABLE IV: Classification accuracy for profile built from *Scenario_10*. Profiles 1 and 2 are built from cluster 1 and cluster 2 respectively. The reported values are percentages of flow windows accepted, i.e. the accuracy of the classification task. The acceptance for benign traffic is presented as well.

VI. Conclusion

This paper provides an approach for clustering behavioral observations and matching models from network level data in face of an unknown and possibly changing number of behaviors. Our analysis method leveraged the prefix tree, a central component in state-merging algorithms for PDFA models. The results showed that we can accurately detect changes in synthetic as well as real-world data without having to aggregate multiple classical change-point detectors. Our controlled experiments show that the freshness can detect changes backed-up by evidence. The approach is computationally efficient as we update counters for freshness in linear-time as flows are available, i.e. in real-time. For a given batch of data, the freshness is calculated in a single pass at the same time as the prefix tree is created.

While not yet an automatic solution, freshness can be successfully used as a manually inspected indicator in a similar fashion as the elbow test in clustering [21]. In the future, we plan to introduce a time-decay factor to forget parts of the prefix tree—if updates on these parts do not occur anymore overtime—and thus emphasize recent behaviors over past ones in an online-learning fashion, as well as working towards automatically deciding on the change-point.

REFERENCES

[1] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 567–581, 2013.

[2] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Profiling internet backbone traffic: Behavior models and applications," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2005, pp. 169–180.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[4] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014.

[5] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002, pp. 1530–1539.

[6] ——, "Change-point monitoring for the detection of dos attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 4, pp. 193–208, 2004.

[7] C. Levy-Leduc and F. Roueff, "Detection and localization of change-points in high-dimensional network traffic data," *The Annals of Applied Statistics*, vol. 3, no. 2, pp. 637–662, 2009.

[8] A. G. Tartakovsky, A. S. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 4–11, 2013.

[9] B. Balle, J. Castro, and R. Gavald, "Adaptively learning probabilistic deterministic automata from data streams," vol. 96, no. 1, pp. 99–127.

[10] C. Y. Cho, E. C. R. Shin, D. Song *et al.*, "Inference and analysis of formal models of botnet command and control protocols," in *ACM CCS*, 2010, pp. 426–439.

[11] D. Babić, D. Reynaud, and D. Song, "Recognizing malicious software behaviors with tree automata inference," *Form. Methods Syst. Des.*, vol. 41, no. 1, pp. 107–128, Aug. 2012.

[12] T. Krueger, H. Gascon, N. Krämer, and K. Rieck, "Learning stateful models for network honeypots," in *ACM AISEC*, 2012, pp. 37–48.

[13] J. E. Hopcroft and J. D. Ullman, *Introduction To Automata Theory, Languages, And Computation*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[14] T. A. Sudkamp, *Languages and Machines: an introduction to the theory of computer science*, 3rd ed. Addison-Wesley, 2006.

[15] J. Oncina and P. Garcia, "Inferring regular languages in polynomial update time," in *Pattern Recognition and Image Analysis*, ser. Series in Machine Perception and Artificial Intelligence. World Scientific, 1992, vol. 1, pp. 49–61.

[16] S. Verwer, R. Eyraud, and C. De La Higuera, "PAutomaC: a probabilistic automata and hidden Markov models learning competition," *Machine learning*, vol. 96, no. 1-2, pp. 129–154, 2014.

[17] C. Hammerschmidt, S. Marchal, G. Pellegrino, R. State, and S. Verwer, "Efficient learning of communication profiles from IP flow records," in *Proceedings of the 41st IEEE Conference on Local Computer Networks (LCN)*. IEEE, pp. 1–4.

[18] J. Sommers, R. Bowden, B. Eriksson, P. Barford, M. Roughan, and N. Duffield, "Efficient network-wide flow record generation," in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 2363–2371.

[19] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, Sep. 2014.

[20] N. Walkinshaw, K. Bogdanov, C. Damas, B. Lambeau, and P. Dupont, "A framework for the competitive evaluation of model inference techniques," in *Proceedings of the First International Workshop on Model Inference In Testing*, ser. MIIT '10. New York, NY, USA: ACM, 2010, pp. 1–9.

[21] D. J. Ketchen and C. L. Shook, "The application of cluster analysis in strategic management research: An analysis and critique," *Strategic Management Journal*, vol. 17, no. 6.