Hindawi
Security and Communication Networks
Volume 2019, Article ID 8954914, 18 pages
https://doi.org/10.1155/2019/8954914

Research Article

Efficient Extraction of Network Event Types from NetFlows

Gustav Sourek and Filip Zelezny

Department of Computer Science, Czech Technical University, Czech Republic

Correspondence should be addressed to Gustav Sourek; souregus@fel.cvut.cz

Received 25 September 2018; Accepted 31 December 2018; Published 6 February 2019

Guest Editor: Jörg Keller

To perform sophisticated traffic analysis, such as intrusion detection, network monitoring tools firstly need to extract higher-level information from lower-level data by reconstructing events and activities from as primitive information as individual network packets or traffic flows. Aggregating communication data into meaningful entities is an open problem and existing, typically clustering-based, solutions are often highly suboptimal, producing results that may misinterpret the extracted information and consequently miss many network events. We propose a novel method for the extraction of various predefined types of network events from raw network flow data. The new method is based on analysis of computational properties of the event types as prescribed by their attributes in a given descriptive language. The corresponding events are then extracted with a supreme recall as compared to a respective event extraction part of an in-production intrusion detection system Camnep.

1. Introduction

Network traffic management analytics tools strive to provide users with high level abstracted view of network communication data. Apart from simple monitoring services, deeper analysis is often required to provide necessary insight into complex network behavior. A prominent example of a tool relying on deep traffic analysis is intrusion detection systems (IDS), monitoring network traffic data with the goal of revealing malicious activities and incidents. In this paper we refer to an existing solution for intrusion detection, a multistage collective network behavior analysis system called Camnep [1]. The purpose of Camnep is to monitor high volume traffic networks for incidents, based on statistical information collected from publicly accessible parts of lower layer packets, i.e., IP and data link layer with no deep packet inspection, aggregated into connections, also called network flows or *NetFlows* (IPFIX) (Section 5.1). In the lower stages of processing the network traffic, before assessing maliciousness to incidents, the system needs to extract higher-level information from lower-level data, mainly by constructing events and activities from individual connections. The reason behind extracting events is that classification of individual flows is not sufficient as we often need context of other flows involved to decide on their maliciousness [2]. At the same time, creating events also reduces the information payload for prospective analysis and detection layers. Clustering connections to meaningful entities is an open problem and existing solutions rely mostly on simple clustering techniques [3] or are driven by handmade rules designed by domain experts [1, 4].

In the task of incident extraction, assuming the view of network traffic as a sequence of NetFlows, we intent to group those NetFlows that logically belong together, forming the so-called *events*. An example of an event type might be a regular data transfer between two hosts, ssh traffic, or malicious port scan, ssh cracking, distributed denial of service attack, etc. Intuitively, it should be possible to cluster NetFlows according to their source or destination IPs, or according to other information present in the NetFlow record with respect to the observed NetFlow ordering. Although it is a step forward from the individual flow-level analysis, in practice this kind of straightforward clustering, e.g., k-means based on numeric vector NetFlow representation, leads to limited correspondence to known event types occurring in the network and suboptimal results in the final reconstruction of events.

We propose an automated method for event extraction, assembling various known types of network events from raw NetFlow data with a guidance of existing event-type descriptions for known classes of events[1]. To evaluate the method, we focus on the use case of event extraction for intrusion detection and compare the detection performance over predefined types of events against the respective part of an existing IDS Camnep. The new method is based on computational analysis of the existing event-type definitions as described by formulae in a simple descriptive language. We first show theoretically how the new approach leads to maximization of recall of these event types and further analyze the complexity and advantages of the method. Finally, we report experiments proving superior recall as compared to Camnep.

2. Related Work

The scope of this work lies on a border of general network traffic analysis and intrusion detection, while a special respect in processing the traffic is taken to allow for extraction of known types of events representing malicious behavior.

The related problem of identifying traffic events, i.e., mapping flows to known application classes, has attracted attention in the network management and security research community, since simple methods relying mostly on pure port number dictionaries were proved insufficient and misleading [5], due to the increasing volumes of HTTP tunneling, customized protocols, encryption, and applications camouflage, which makes the problem much more challenging. For that reason, more sophisticated statistical techniques began to emerge, utilizing machine learning for Internet traffic identification [6]. Most of these works considered the traffic on individual flow level [7]; others clustered flows based on their characteristics, utilizing statistical clustering [8] or machine learning techniques [9]. These clustering approaches were based on basic flow attributes, usually packet sizes and duration, typically grouping flows into a predetermined number of clusters [10] using generic techniques such as *k-means* [3]. A more related clustering approach utilizing structural properties of flow clusters with the concept of *graphlets* is presented in Blinc [2].

Considering the particular problem of aggregating NetFlows into malicious events as a part of intrusion detection, the literature is quite scarce. Although most of the existing IDS probably deal with the problem of higher level information extraction from individual traffic flows in some form, the authors are unaware of more publicly available works addressing generally the NetFlow aggregation part of the intrusion detection problem, with the event layer (Section 3) in Camnep [1] being the only exception.

Network security and management tools depend heavily on automated IDS to extract sensible information out of malicious activity to be presented to the user [11]. The amount of network generated data suggests employing machine learning tools to analyze the traffic data and the respective community of research has been very active in proposing various forms of models for the incident identification tasks. The approaches vary from the very early attempts using association rules [12], to learning neural networks for incident labeling [13]. Other approaches utilize decision trees to learn explicit knowledge [14] and unsupervised methods using clustering as an inherent part of network traffic analysis [15]. In more recent works, distributed, robust approaches utilizing strategies from trust modeling and game theory were introduced [16].

However, in real world practice, there are numerous issues affiliated with an online use of sophisticated machine learning methods on actual networks, and majority of the IDS currently in use are still either rule or expert system based [4]. The main advantage of these approaches is the human-readable form of the knowledge included in the system, and by those means the overall traceability and decomposability of the system behavior for given incident types. Moreover, the rule based models are not limited to design by hand as they can also be extracted by the means of machine learning.

While the need for interpretability of the rules can often form a limitation on the detection performance, they can be further combined with other machine learning-based detection techniques. An example of such a hybrid approach is presented in Camnep [1], an agent based intrusion detection system designed for deployment on high speed back-bone networks. Camnep combines expert rules forming explicit models of incidents for feature description and extraction in the lower stages, with classification, agent techniques, and trust modeling in the final stages [17].

A significant portion of the IDS works mentioned refer to a popular dataset created in 1999 named KDD'99 [18], published by Defense Advanced Research Projects Agency (DARPA). Although it was a step forward in comparing and evaluating IDS approaches, this work has received a lot of critique for including flaws in many statistical respects [19] and limitations that are inherited from the DARPA datasets [20]. This dataset is now almost 20 years old, which is a considerable time span in the network domain and makes such an evaluation not truly informative. The traffic has changed rapidly since then, causing the signatures and behavior of attacks in the dataset problematic for comparison with a real traffic today.

For the reasons mentioned, in this paper we refer to a real-use IDS Camnep [21], processing recent university traffic (rather than KDD'99) in a widely used Cisco NetFlow format. We take the respective part of event extraction in Camnep system as a baseline for the new approach, so that we can evaluate the method in real context of actual incidents and work with relevant feedback of a current system.

3. Camnep Event Layer

The respective part of Camnep responsible for event extraction consists of a multistage process based on a clustering strategy utilizing expert designed similarity metric. Since we propose a method for event extraction that can be seen as an improvement to the clustering-based approaches, we firstly describe the Camnep event extraction process (as detailed in a technical report on Camnep [22]) in this chapter.

The elementary communication data unit for Camnep is a network flow or *NetFlow*, which was introduced on Cisco routers to give the ability to collect IP network traffic as it enters or exits an interface. NetFlows are captured within a restricted time-frame used for data acquisition from a local network and stored in the observed order as captured by the network probes, from which they are further merged into a single data source. At first the flows are filtered to reduce their overall number, which decreases computational requirements of the event creation process in high throughput networks. Because of the fact that the system concentrates on the malicious behaviors in the network, this input filtering skips certain trustworthy (measured on individual flow-level) connections or connections with no chance of being a part of an event, yet such a functionality can be omitted for general analysis purposes in this paper.

The purpose of the respective event extraction process is to group all individual flows from the incoming network traffic into sets of flows (events) with respect to to their features. An event in Camnep then corresponds to some bounded, interpretable, specific network behavior, where the specification is based on features of the sources, destinations, and size characteristics of the transferred traffic.

To create an event, Camnep treats individual flows as feature-vectors in a predefined multidimensional space in which it considers the events as clusters, subject to a given similarity metric. At minimum, the feature vectors contain information on source and destination IP addresses, ports, and the used protocol. Additionally, they may also contain information from the transfer itself, such as the number of packets and bytes. The sequential clustering process then takes place in the corresponding feature space endowing the flows.

The clustering process works in two stages. Firstly, smaller elementary clusters are formed to represent partial network behaviors, and these elementary clusters are then further aggregated to create the actual events.

Each preprocessed flow is transformed into the feature vector which directly represents a so-called flow-cluster (ϕ_i^f), residing in the same feature space for clustering. These are to be further aggregated into so-called elementary clusters (ϕ_j^e) through the first stage of the clustering process.

The logic of the similarity metric is also divided into two subparts. In the first, structural properties of the flows in the clusters are checked against each other. In the second, similarity in the number of transferred bytes is being measured. Both parts have to be satisfied for the similarity condition to hold.

A flow-cluster ϕ_i^f can only be considered structurally similar to an elementary-cluster ϕ_j^e if both clusters share the same protocol and also one of the following combinations of IP addresses and ports.

(1) source IP address and source port (~response)

(2) source IP address and destination port

(3) destination IP address and source port

(4) destination IP address and destination port (~request)

Further, ϕ_i^f and ϕ_j^e can only be considered similar in the size of the bytes being transferred if the following condition holds:

$$\left| bytes\left(\phi_i^f\right) - avgBytes\left(\phi_j^e\right) \right| \leq \min\left\{ bytes\left(\phi_i^f\right), avgBytes\left(\phi_j^e\right) \right\} \tag{1}$$

where $bytes(\phi_i^f)$ denotes effectively the number of bytes in the single flow i and $avgBytes(\phi_j^e)$ stands for the average number of bytes across all the flows in ϕ_j^e. The rationale behind this formula is to allow for relatively greater discrepancies in size within greater clusters.

Finally, if the similarity does not hold for any elementary-cluster, the flow-cluster ϕ_i^f transforms into a new ϕ_j^e on its own.

The point of the second stage of the clustering process is to reveal more compound network behavior. The rationale behind this stems from the observation that the elementary clusters often form only a part of some larger services. An example of such network behavior can be generic data transfers from web services, where one can expect high variance in the number of bytes transferred across individual events, yet all such events generally represent the same network behavior from an IDS point of view. Another example could be peer-to-peer traffic.

For this second stage of clustering, the similarity metric is slightly updated. Now the clusters need to share the same protocol and also either of (i) average value of number of bytes or (ii) the same source or destination port. Further, they need to be similar in the size of the bytes transferred as measured by the following condition parameterized by a given number $k > 1$.

$$\left| avgBytes\left(\phi_i^f\right) - avgBytes\left(\phi_j^e\right) \right| \leq k * \min\left\{ bytes\left(\phi_i^f\right), avgBytes\left(\phi_j^e\right) \right\} \tag{2}$$

The incoming sequence of flows is thus iteratively processed in the two sequential steps, so that each new coming flow or elementary cluster is checked against the existing clusters, one by one, to be merged with on a first match or create a standalone cluster otherwise. The final event clusters, representing second-stage clustering output, are then classified by matching against existing rule-based models of incident types described in Section 3.1. The output validated clusters can then be seen as individual network events.

We can see that the similarity metric is designed to capture common structural [2] and statistical [8] properties of events based on shared endpoint and flow size similarity, in a similar fashion to previous works [6]. Consequently, the clustering process in Camnep is very generic and should be thus generally robust and unbiased to various event types.

3.1. Incident Types.

3.1. Incident Types. The incident types are classes of events, defined on the domain of flow clusters (sets), that can be observed within a network and are in the current scope of IDS. Although the events can generally represent any

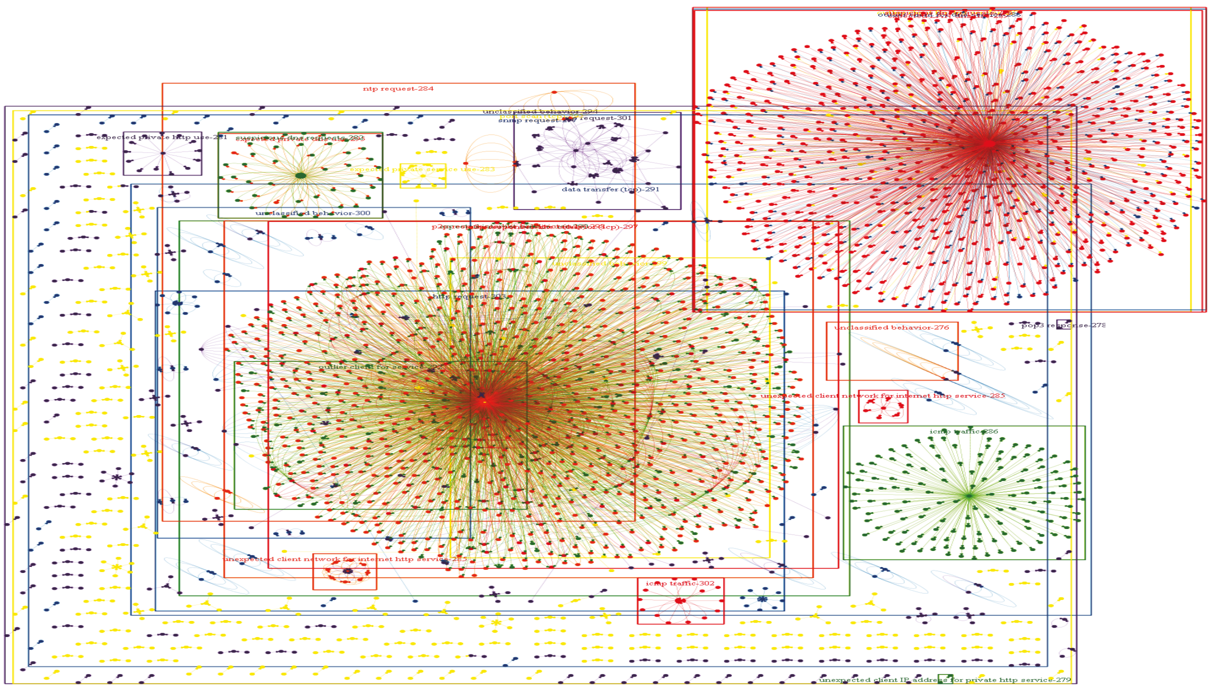
ntp request-284
unclassified behavior-294
snmp request-297
data transfer (tcp)-291
unclassified behavior-300
unclassified behavior-276
unexpected client network for internet http service-285
icmp traffic-286
icmp traffic-302
unexpected client IP address for private http service-279

Figure 1: A snapshot of a 5-minute window of traffic within the university network. Flows are displayed as directed arrows between two endpoints displayed as dots. Individual flows are grouped into events of particular types as distinguished by colors.

form of network behavior, the overall focus of Camnep is on intrusion detection, and so the scope of incident types in Camnep includes mainly forms of network attack and malicious behavior. An example of event-types in use is various classes of vertical and horizontal port scanning, ssh cracking, syn-flood and distributed denial of service attacks, etc. The important aspect of these events is that they are typically formed by plurality of network flows, whereas each of these network flows does not necessarily present any sort of malicious behavior when considered on its own, and thus the maliciousness of an event cannot be assumed on the individual NetFlow level. The complexity arising from the aggregation of flow traffic forms the interesting and intriguing part of this problem (Figure 1).

Example 1. As a simple example [23], consider a sequence of flows aiming at a particular endpoint port 22. If each of the flows is to be analyzed separately, it can easily be considered as a regular ssh communication request. Yet when we group those flows, according to their properties such as the same endpoint and size, we can explore potential exploit or malicious ssh-cracking behavior from the plurality of small flows checking the same ssh endpoint during a very short time-scope, possibly transferring a considerable amount of data back afterwards.

For the definition of these event-types a unified description language (Section 3.2) is being used within the framework of the system, capturing various forms of restrictions on how a cluster of flows, forming an event of a particular type, should look like. The complexity of event types, as captured by the description framework, varies from simple, clearly defined classes used more for administration purposes, such as *icmp traffic* or *udp data-transfer*, to types of events exhibiting more complex network behavior such as *p2p event indicators*, and malicious events, e.g., *port scanning* or *ssh cracking* attacks.

Example 2. For example, an event definition for *TCP port scan behavior* posed as a flow cluster description in the descriptive and natural language looks as follows:

packets = 100..∞ (more than 100 packets transferred)

protocol = TCP (the 4th layer protocol of flows is TCP)

uniqSrcIPs = 1 (there is only one (common) source IP adress)

uniqDstPrt = 2..∞ (there are at least two destination ports)

uniqDstIPs = 2..∞ (there are at least two destination IPs)

bytesPerPacket = 40..60 (flows carry between 40..60 bytes/packet)

fuzzyBytesPerPacket = 40,48,50,60 (actual bytes/packet $\in$ $\{40, 48, 50, 60\}$)

bytesPerFlowFuzzyDivision = 4 (the ratio is app. divisible by 4)

flows = 5..∞ (total number of flows is greater than 5) maxPacketsPerDstIP = 0..100 (less than 100 packets per destination IP)

maxFlowsPerDstIP = 0..11 (less than 11 flows per each destination IP)

bytesSimilarity = 0..2 (coefficient of variance of flow sizes less than 2)

The more complex the network incident is, the more difficult it is to define a clear description of the exhibited behavior. Namely, defining complex network events with rich underlying graph structure, such as the p2p and distributed attack behavior, and revealing corresponding cooperating or infected hosts is rather too difficult problem to be captured by a single description, and higher layer modules utilizing more sophisticated methods are being used for the task [24]. The purpose of the description language used for the event extraction is thus not to clearly identify any malicious behavior, but to set sort of necessary (must) bounds on the event types, with which all relevant information from flows can be extracted into events for the prospective analysis. By those means the main goal of the event extraction process is to maximize recall of incidents. The possible false alarms caused by the overgenerality of the descriptions of event types are thus further handled by subsequent dedicated detectors, where the information is iteratively refined and fused to decrease the false positive rate, before the final assessment of maliciousness is performed by the system.

3.2. Description Language. Analysis of the actual description language for event types is the core of the current process improvement and main contribution of this work. The language is used to indicate that a set of NetFlows can be seen as a potential manifestation of an event of a particular type (the actual event might be detected in the set in subsequent stages). The definition of an event type is a conjunction of formulae putting restrictions on the descriptives of the set of flows. These formulae can be either designed by an expert or learned by the means of machine learning, where the important aspect is interpretability of the learned formulae so that they are intelligible enough to be understood by an expert. These formulae apply sort of value and range restrictions to flow-set properties, such as flow addresses, sizes, protocols, etc. The formulae used may be existentially but are usually generally quantified.

Analyzing the features of the set of flows that these formulae put the restrictions on, we can notice they might be categorized into *flow-specific*, relational, and *aggregative* formulae, depending on what sort of information from the flow-set they evaluate. A deeper computational rationale behind these categories is explained later in Section 4.2.

Example 3. To introduce the categorization idea, what follows are examples of the formulae categories posed in natural language.

(1) *flow-specific formulae*

- (i) protocol of all flows must be TCP
- (ii) all flows must be smaller than 20 bytes
- (iii) number of bytes per packet of each flow must be divisible by 4
- (iv) all flows must lead to destination port 80

(2) relational formulae

- (i) for each flow to IP1 there must be subsequent flow from IP1
- (ii) flows to port 22 must come from 3 or more predefined addresses

(3) *aggregative formulae*

- (i) number of unique destination IPs must be less than 3
- (ii) average number of bytes should be between 20 and 60
- (iii) percentage of unique ports should be higher than 50
- (iv) entropy of destination ports and IPs should be closer than 3

The *flow-specific* formulae express constraints on various properties of a flow, such as defining the protocol, flags, or limiting the number of bytes and packets. The flow-specific formulae represent majority of all event-type definition formulae designed by the experts. The main common attribute of formulae in this category is that they can be evaluated for each flow separately and (as generally quantified) hold true for all the flows in a particular event instantiation. They refer to flow attributes of Protocol, Start, Duration, Flags, Packets, Bytes, Addresses, Ports, and derived features, such as Bytes per Packet. The restrictions used here are formed by putting upper and lower bounds, or enumeration of plausible values of the attributes mentioned.

The category of relational features is experimental and it is not used in the actual system. The features in the relational category had reflected various structural and sequential properties of flows. As such they referred to time and address/port attributes and the formulae using them were checking existence of a given relational pattern in the whole set of flows or separately for each flow. We do not refer to them in the later sections as most of these features were simplified and passed into the aggregative category.

Aggregative category of formulae puts restrictions on features expressing attributes of the whole sets of flows. For that they utilize aggregation functions such as count, maximum, average, entropy, etc. The restrictions used are again intervals or enumerations of admissible values. Part of aggregation features may also be seen in an intersection with the relational category, for instance, restricting the timespan between the first and last flow. The aggregative formulae always hold true for the whole set of flows and cannot be evaluated separately which makes them more complex.

A depiction of how these formulae are generally assembled in the system can be seen in Figure 2. We can see that the descriptive language, utilizing conjunctions of various types of formulae mentioned, provides rich capabilities for creating complex definitions of various event types, such as the one

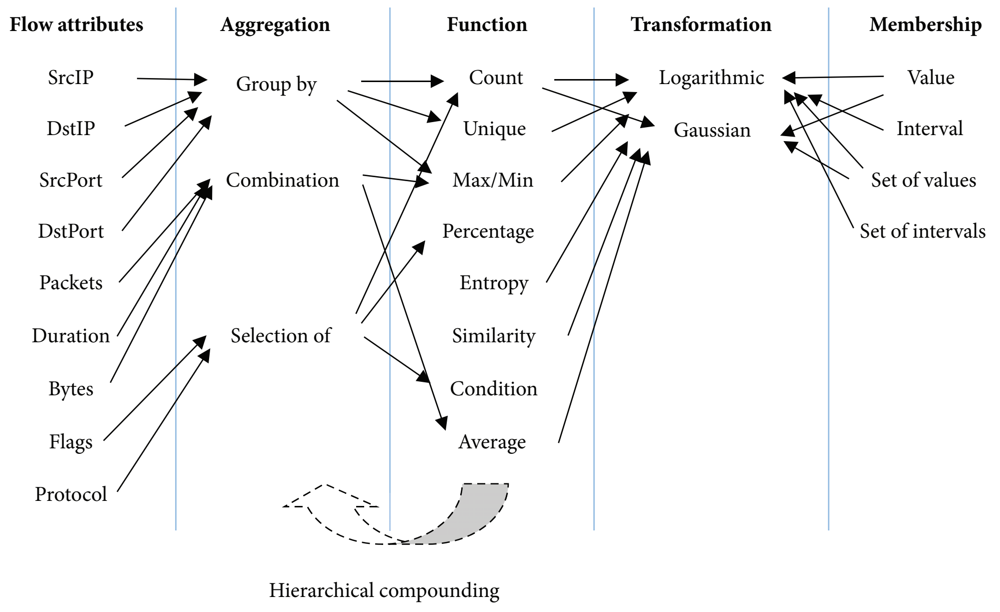

FIGURE 2: Language of formulae checking values of features assembled from sets of flows. The individual flows can be grouped by some of their attributes while their other attributes are further aggregated using some of the listed aggregation functions. This aggregation process might be repeated in a recursive manner. A final derived feature is subsequently checked against a membership function of the enumeration of values or intervals to find whether the respective formula holds true. Arrows represent aggregated combinations in use by formulae of actual event types in use.

from Example 2, and that by the means of the introduced language structure we can capture multitude of common malicious behaviors.

4. New Event Extraction

In Camnep, the incoming flows are continuously grouped, following the order in which they are collected by the network probes, with accordance to the flow similarity metric in the two stage clustering process, to be finally matched against existing event-type classification models.

The subject of this paper is a novel method of event construction from NetFlow traffic data that merges together the stage of flow aggregation and event description matching to globally optimize the set of identified events in order to maximize their recall. The primary objective of the method is to secure that no potential event (attack) comes through the process of event extraction unnoticed. At the same time the method should be able to input the flows sequentially, as they come, and must be fast enough to be applied on huge amounts of traffic data in real IDS.

4.1. Motivation. The existing solution described in Section 3 is better than naive clustering as it pays respect to the general nature of network events, but it is still suboptimal and produces often misleading results. One of the reasons is that defining an optimal clustering metric, forming a perfect similarity measure of flows (clusters) across various event types, is just not possible. Also the greedy nature of the clustering algorithm may lead to reconstruction errors in more complex types of events. Another reason is the unpredictability of NetFlow ordering, as generally NetFlows that logically belong together (i.e., are a manifestation of well recognizable and well defined cause, such as certain user or software action) may not necessarily be observed in a continuous sequence. The sequential greedy clustering algorithm may thus miss many real events once the clusters are matched against the event-type descriptions in scope.

Example 4. Let us consider a trivial incoming sequence of flows (f_1, f_2, f_3) with the endpoints $f_1(source_A, destination_B, \ldots,)$, $f_2(source_C, destination_B, \ldots,)$, and $f_3(source_C, destination_D, \ldots,)$. Following a greedy clustering process utilizing endpoint similarity (e.g. the one in Camnep from Section 3), such an observed sequence of flows would result into cluster $c_1(source_X, destination_B, \ldots,)$. Yet changing the processing order of the incoming flows to (f_2, f_3, f_1) would create cluster $c_2(source_C, destination_X, \ldots,)$, as depicted in Figure 3.

Even if we choose more robust clustering strategies such as spectral clustering [25], instead of the greedy one presented in Section 3, the algorithm will by definition always have to choose one resulting cluster over another for every set of flows. For event types $type_1$ and $type_2$ with "competing" descriptions $desc_1$ and $desc_2$ (e.g., *request* and *response* event types), i.e., where one form of clustering the same set of flows into $\{f_1, f_2\}, \{f_3\}$ in $desc_1$ might be preferred over another form $\{f_1\}, \{f_2, f_3\}$ in $desc_2$, will lead the clustering process to miss some of the potential events, analogously to the situation from Figure 3, and introduce false negatives into the output of IDS.

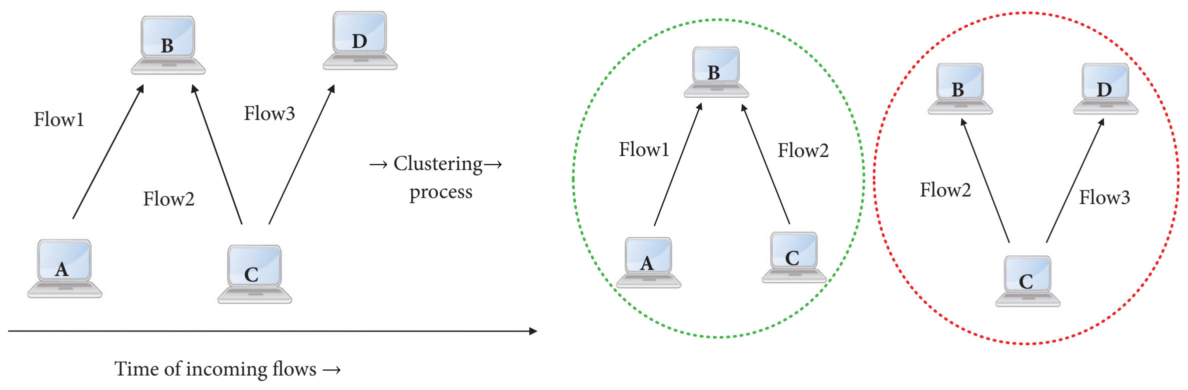

FIGURE 3: Using the greedy endpoint-based clustering process on the incoming sequence of flows (f_1, f_2, f_3) ordered by the time of collection (left) will result (right) into specific clustering (green) while missing another possible valid cluster (red).

A more profound reason why clustering possibly leads to suboptimal event reconstruction results lies deep in using a similarity metric function in the form of $\lambda(object_i, object_j)$ to determine whether two NetFlows f_i, f_j (see (1)) or two clusters ϕ_i, ϕ_j (see (2)) belong together into an event. While the criteria used in the current (rule-based) similarity metric are reflecting some desired properties of some event types, i.e., endpoint and flow-size similarity, there is no way to capture all properties of all events at once. Finally, using the categorization of event-type properties from Section 3.2, we note that a pairwise application of the similarity metric to flows (clusters), is naturally restricted to flow-specific features, and generally cannot capture desired aggregative properties of events.

Example 5. As a consequent example, let us consider a situation with event-type $type_1$ of description $desc_1$ containing aggregative formula "average of flow-sizes must be from 1 to 10 bytes." Assume again a flow sequence (f_1, f_2, f_3), with the respective properties $f_1(13bytes, \ldots)$, $f_2(6bytes, \ldots)$, $f_3(7bytes, \ldots)$. The resulting aggregative property of the average of presented flow set $(13 + 6 + 7)/3$ makes it naturally come out as a $type_1$ event. Following the clustering procedure based on a similarity metric in the form $\lambda(flow_i, flow_j)$, e.g., the one from (1), to group couples of flows together, we might end up grouping $\{f_2, f_3\}$ or $\{f_1, f_3\}$ while grouping of $\{f_1, f_2\}$ would not be possible, although it also constitutes a valid event of $type_1$ that might still be desired by the same measures if just f_3 was presented first. So while some couples of flows are consistent and others are not, as captured by the similarity $\lambda(flow_i, flow_j)$, the aggregative property should stay agnostic of the individual comparisons, yet this is not the case in the clustering process as, in the analogy of switching order, (f_1, f_2, f_3) ends up with a different result $\{f_1\}, \{f_2, f_3\}$, while (f_1, f_3, f_2) leads to the original $\{f_1, f_2, f_3\}$.

Generally speaking, the problem with clustering is in the quality measure it optimizes, derived from the algorithm and similarity metric used, which is generally different from the quality criteria desired for event extraction, although it intuitively might seem to solve the task. A high quality clustering is only confirmed indirectly when, in the end, all the real events are explored and correctly classified. By this indirect reasoning, we can see that clustering and the subsequent classification stages are tied together, since suboptimal clustering will harm the classification stage and vice versa, and so we can conclude that a perfect clustering should be equal to event identification based on the classification models, defined in the established language framework (Section 3.2).

Example 6. Should one want to group a population of individuals to reveal the underlying set of families, using a clustering approach would seem reasonable, while it might likely end up creating clusters of men and women instead. Since we know what the general properties of families are, it follows as a more reasonable strategy to utilize those properties and search for those directly.

4.2. Strategy. Analyzing the weaknesses of the clustering process, the key idea of the new method is to replace NetFlows clustering phase directly with an exhaustive exploration of events. While the clustering process is sensitive to many settings (Section 4.1) and different measures of similarity, preferring different clusters in different network contexts, the new method is robust and context invariant, as it removes the similarity metric function, constituting just an approximation of desired properties of clusters, and replaces it directly with event-type descriptions formulae, which have been either learned or designed by the experts. In other words, instead of first clustering the flows and then checking whether the extracted clusters match the specifications, we explicitly try to create the clusters in a way to match the specifications whenever possible. This naturally leads to maximization of recall. Thus to reconstruct an event it means to search for all valid data instantiations of a given event-type definition in the data. The event exploration process then corresponds to search for a model of a logical theory, given by the set of event-type formulae, in the NetFlow data. Such a search falls naturally into the exponential category, as it would generally require to check all subsets (power-set) of flows, which would be inadmissible.

The key factor to make the search admissible is to provide search bounds. We base these bounds on the constraints

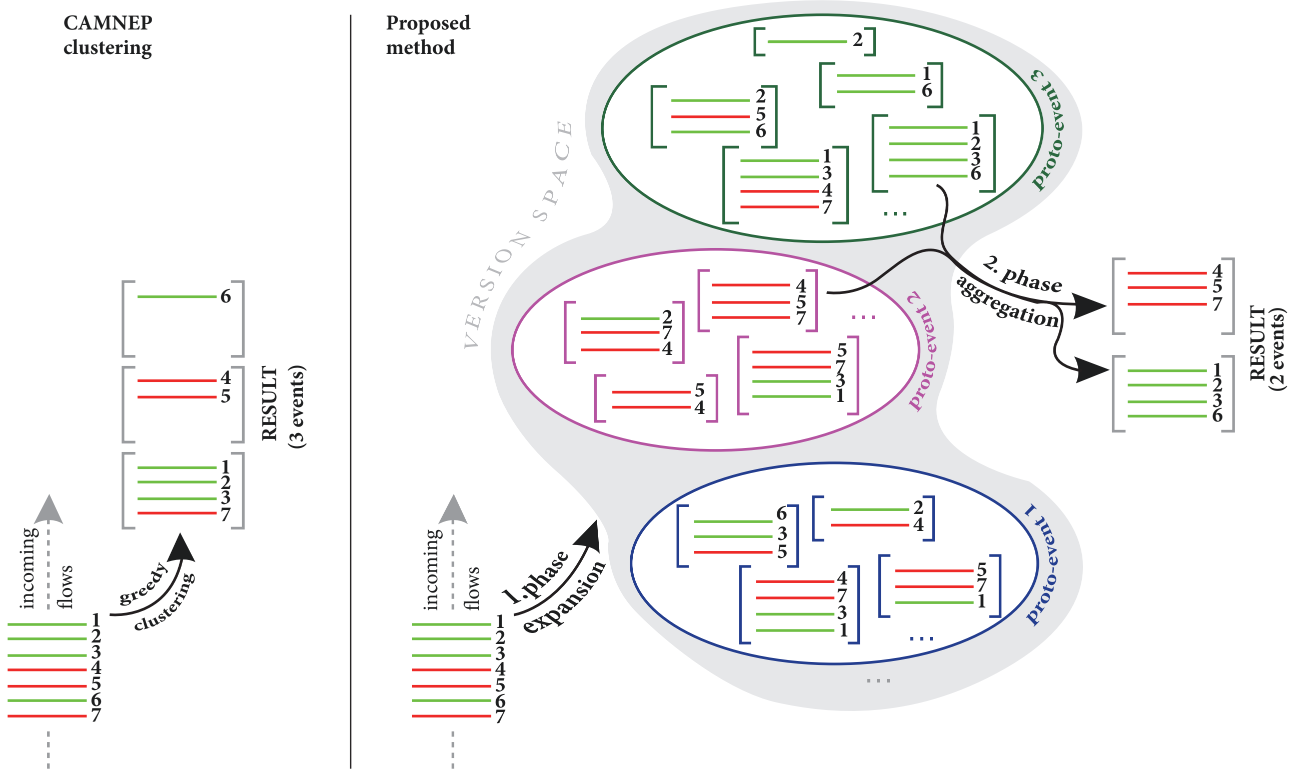

Figure 4: Depiction of comparison of the new two staged event extraction method (right) with the actual one in Camnep (left). In the first stage, also called expansion of version space, the flow-specific formulae are being processed to create proto-events. In the second stage, referred to as the aggregation phase, the proto-events are processed through the aggregative formulae to produce output events.

defined in the language used for event-type definitions (Section 3.2). Following this framework, we can see that the complexity of search raises from the descriptions, in contrast to the clustering approach where the complexity raises purely from the NetFlow data, following the intuition that it is much easier to look for simple event types, e.g., TCP data transfer, than for those with complex structure and aggregated properties, such as distributed forms of attacks. To formalize the event types complexity, we have actually divided the formulae constituting the event definitions into categories, introduced in Section 3.2, according to the complexity of searching for their model in data.

The first category we refer to as *flow-specific* formulae represents simple rules that we can find a model for in linear time in the number of flows. The main common attribute of formulae in this category is that they can be evaluated for each flow separately and hold true for all the flows in a particular event instantiation. These features of flow-specific formulae result in their special treatment in the algorithm (Section 4.3).

The second category consists of so-called *aggregative* formulae, representing various aggregation functions. The search for their model falls generally into the NP-complete category, and rather than describing particular flows they express attributes of relations between them and attributes of whole sets of flows. The aggregation formulae thus hold true for the whole event and cannot be evaluated separately, which makes the search for their model more computationally complex.

In the stage of event extraction from set of flows, the primary objective is to maximize recall, while leaving the burden of identifying suspicious events on subsequent classifiers trained to do so. On the other hand, we also cannot extract all subsets of flows as events because there would be an astronomical number of them. Therefore the event extraction process is driven directly by the definitions of the relevant event types in scope of IDS so as to make sure it produces all and only the valid relevant events.

4.3. Algorithm. The goal of the algorithm is to process the input flows in the order they naturally come to produce all possible future events in scope. The idea of the algorithm is based on decomposition (Section 4.2) of the event-type formulae (Section 3.2). Using this decomposition we can order the formulae from computationally easy to hard and utilize that within the search method. The idea of the search method is depicted in Figure 4, and we describe it more closely in this section.

At first we utilize the features of flow-specific formulae, allowing us to consider them as being shared by all the flows within a particular event, on which we base a procedure that maps every pair of flow and flow-specific part of j-th event-type description $(flow_i, flowSpec_l)$ onto a specific *hash-key* $hash_k$. The hash-key reflects the fact that having an instantiation $event_k$ of an event-type $type_l$ based solely on its flow-specific $flowSpec_l$ part of description $desc_l$, all the flows $\{flow_i, flow_j, \ldots, flow_n\}$ within

```
1: function CreateKey($flow; flowSpec$)
2:     $keys \longleftarrow \emptyset$
3:     for all $formula \in flowSpec$ do
4:         $attValue \longleftarrow (formula \lhd flow)$          ▷flow-formula evaluation
5:         $keys \longleftarrow key \cup attValue$
6:     end for
7:     $hashKey \longleftarrow$ hash($keys$)                      ▷arbitrary hash function
8:     return $hashKey$
9: end function
```

Algorithm 1: Hashing of the flows.

```
1: function DistributeFlows($Flows$)
2:     $minTime \longleftarrow$ time of the first $flow \in Flows$
3:     $maxTime \longleftarrow$ time of the last $flow \in Flows$
4:     $intervalCount \longleftarrow (maxTime - minTime)/winSize$
5:     $flowWindows \longleftarrow \emptyset$
6:     for all $flow \in Flows$ do
7:         $idx = (flow.time - minTime)/winSize$
8:         $flowWindows_{idx} = flowWindows_{idx} \cup flow$          ▷flow distribution
9:     end for
10:    return $flowWindows$
11: end function
```

Algorithm 2: Distribution of flows into time windows.

this instantiation must have yielded the same evaluations $\{attValue_a, attValue_b, \ldots, attValue_m\}$ with flow-specific formulae $flowSpec_l$. The evaluation of a formula $spec_a$ on a flow $flow_i$ (marked as $spec_a \lhd flow_i$) yields a value $attValue_a$, which might be, e.g., a destination IP address of the flow $flow_i$ if the formula specifies that there is a unique destination IP address in the event-type description $desc_l$, or simply $true$ if the formula evaluates as such, e.g., when $spec_b$ specifies that all flows must be smaller than x-bytes, which flow $flow_i$ satisfies.

This does not mean that any flow, possibly belonging into some event of type $type_l$, will yield the same hash-key $hash_k$ through evaluation against its description $flowSpec_l$, since there can be more than one instantiation $event_k, \ldots, event_o$ of that event-type $type_l$, but exactly all the flows $\{flow_i, flow_j, \ldots, flow_n\}$ belonging to that particular instantiation $event_k$ of event-type $type_l$ will yield the same hash-key $hash_k$. The flow-specific hash-key generation is described by the procedure in Algorithm 1.

At the initialization, the traffic is divided into (5-minute) time intervals, as described in Algorithm 2, which, although not necessary for the new method (Section 4.4), serves for comparison against Camnep, and is sort of a standard in traffic analysis [26], as it is generally expected that the events occur within short limited time-frames [23].

In the next preprocessing step, described in Algorithm 3, we divide all descriptions $Descriptions \longleftarrow \{desc_1, desc_2, \ldots, desc_n\}$ of the event types in scope $\{type_1, type_2, \ldots, type_n\}$ into *flow-specific* $\{flowSpec_1, flowSpec_2, \ldots, flowSpec_n\}$ and *aggregation* formulae $\{agg_1, agg_2, \ldots, agg_n\}$ (Section 3.2). This division is processed by a parser based on a predefined formulae dictionary (3).

$$Dictionary \longleftarrow \{(\{att\}, func, \{restr\}) \longmapsto \{Aggregative, FlowSpec\}\} \quad (3)$$

The dictionary maps all possible formulae onto the two formula types based on their structure, i.e., the used flow attribute(s), (aggregation) function, and restriction(s) on the corresponding outcome.

The main search method then works in two steps. The first step processes the flow-specific formulae and loads a stream of incoming flows into a hashmap-based memory structure referred to as a *version space*. In the version space, every possible instantiation of all event types exists as an event prototype, or simply *proto-event*, holding a specific *hash-key* as defined earlier (Algorithm 1). These instantiations are generally overly inclusive sets of flows based solely on the *flow-specific* formulae and thus, using the hash-key, there is no need of comparing new incoming flow with any of the previous (Section 4.4). This first phase of the algorithm, we refer to as the expansion of the version space (expansion phase in Figure 4), is described in Algorithm 4.

The next step after the version space creation is the extraction of events from proto-events. In this step we apply the aggregation formulae (aggregation phase in Figure 4) on the proto-events from version space. The idea here is that, although remaining in the NP-complete category, the search for a model of an aggregation formulae is now performed on much smaller sets of flows, already divided into exclusive

```
function ParseDescriptions($Descriptions, Dictionary$)
    $flowSpec \longleftarrow \emptyset$
    $aggregative \longleftarrow \emptyset$
    $parser \longleftarrow$ new Parser($Dictionary$)                ▷predefined dictionary parser
    for all $desc \in Descriptions$ do
        $i \longleftarrow i + 1$
        $flowSpec_i \longleftarrow parser$.parseFlowSpec(desc)
        $aggregative_i \longleftarrow parser$.parseAggreg(desc)
    end for
    $descriptions \longleftarrow \{\{flowSpec\}, \{aggregative\}\}$
    return $formulae$
end function
```

Algorithm 3: Parsing event descriptions into flow-specific and aggregative.

```
$flowWindows \longleftarrow$ DistributeFlows($Flows$)
$descriptions \longleftarrow$ ParseDescriptions($Descriptions, Dictionary$)
$versionSpace \longleftarrow \{(Keys \longleftarrow \emptyset) \longmapsto (ProtoEvents \longleftarrow \emptyset)\}$

procedure ExpandVersionSpace                                        ▷flow processing
    for all $flowWindow \in flowWindows$ do
        for all $flow \in flowWindow$ do
            for all $flowSpec \in descriptions.flowSpec$ do
                for all $formula \in flowSpec$ do
                    if $flow \nvDash formula$ then
                        break
                    end if
                end for
                $hashKey \longleftarrow$ CreateKey($flow, flowSpec$)              ▷hashing the flow
                if $hashKey \in versionSpace.Keys$ then
                    $protoEvent \longleftarrow (versionSpace \longleftarrow hashKey)$
                    $protoEvent.Flows \longleftarrow protoEvent.Flows \cup flow$         ▷add flow
                else
                    $protoEvent \longleftarrow \emptyset$                         ▷create new proto-event
                    $protoEvent \longleftarrow protoEvent.Flows \cup flow$
                    $versionSpace \longleftarrow versionSpace \cup (hashKey \longmapsto protoEvent)$
                end if
            end for
        end for
    end for
end procedure
```

Algorithm 4: Expansion of flows into version space.

subsets by flow-specific formulae. To find the model of $desc_i$ and the resulting event of $type_i$ means to search for a valid subset of flows from the proto-event satisfying the given aggregation formulae agg_i. This process is described in Algorithm 5.

While searching for a valid subset satisfying a given aggregation formula we would theoretically still have to proceed through a power set of thousands of flows in some cases. For that purpose, heuristics are defined to speed up this subset search. They are designed specifically for the aggregation function types used commonly by the experts, e.g., the average, entropy difference, count, unique count, etc.

Example 7. An example of a subset search driven by a very simple heuristic for the aggregation formula "average of bytes per flow must be within a given range" is described in Algorithm 6. After sorting the flows with respect to their size the search iteratively removes the flows with size outside of the specified range, in the direction of the desired average value change. This heuristic subset search thus works in $O(n * \log(n) + n)$ time, or in $O(n^2)$ without presorting.

Example 8. An example of a more complex heuristic driving the search for a model of formula "entropy of destination

```
1: function ExtractEvents(versionSpace)
2:    $events \longleftarrow \emptyset$
3:    $descriptions \longleftarrow$ ParseDescriptions(Descriptions, Dictionary)
4:    for all $protoEvent \in versionSpace$ do
5:       $flows \longleftarrow protoEvent.flows$
6:       $desc \longleftarrow (descriptions \longleftarrow protoEvent.type)$        ▷description for event
7:       $repaired = true$
8:       while $repaired = true$ do
9:          $repaired = false$
10:         for all $formula \in desc.Aggregative$ do
11:            if $flows \nvDash formula$ then
12:               $flows \longleftarrow$ subsetSearch(formula, flows)
13:               $repaired = true$        ▷set was changed $\Longrightarrow$ redo
14:            end if
15:         end for
16:      end while
17:      if $protoEvent.flows \not\equiv \emptyset$ then        ▷valid subset was found
18:         $events \longleftarrow events \cup protoEvent$
19:      end if
20:   end for
21:   return events
22: end function
```

Algorithm 5: Extracting events from proto-events.

```
1: function subsetSearch($formula_{avg}$, Flows)
2:    $flows \longleftarrow sort(flows, flows.Bytes)$        ▷optional pre-sorting
3:    while $flows \not\equiv \emptyset$ do
4:       if $avg(flows.Bytes) < formula_{avg}.range$ then
5:          $flow \longleftarrow \min(flows, flows.Bytes)$
6:          $flows \longleftarrow flows \setminus flow$
7:       else if $avg(flows.Bytes) > formula_{avg}.range$ then
8:          $flow \longleftarrow \max(flows, flows.Bytes)$
9:          $flows \longleftarrow flows \setminus flow$
10:      else
11:         break
12:      end if
13:   end while
14:   return flows
15: end function
```

Algorithm 6: Heuristic driven flow subset with limited average search.

IPs must be close to entropy of destination ports within a given margin" is presented in Algorithm 7. At first we arrange buckets b_j for all IPs and ports and distribute all n flows into the buckets $\{b_1, \ldots, b_q\}$ accordingly. From the buckets distribution, the entropies can be easily calculated in linear time in the number of buckets l as

$$H = -\sum_{k=1}^{q} \frac{b_k}{n} \ln\left(\frac{b_k}{n}\right) \tag{4}$$

Assuming that the difference of entropies is not satisfactory, for each $flow$ we calculate a measure ED_{flow} representing the decrease in entropy difference after removal of that $flow$. This calculation, based on the existing buckets $\{b_1, \ldots, b_q\}$ as performed by the $entropyDiffAfterRemovalOf$ function, can be, interestingly, carried out in constant $O(1)$ time for each $flow$. This follows from decomposition of the formula for calculating Shannon entropy iteratively while adding or removing flows as

$$-H_i = \sum_k \frac{b_k}{i} \ln\left(\frac{b_k}{i}\right) = \frac{1}{i}\sum_k b_k \ln(b_k) - \frac{\ln(i)}{i}\sum_k b_k \tag{5}$$

$$= \frac{1}{i}\sum_k b_k \ln(b_k) - \ln(i) \tag{6}$$

After adding (i+1)th flow, the entropy changes to

$$-H_{i+1} = \sum_l \frac{b_l}{i}\frac{i}{i+1} \ln\left(\frac{b_l}{i}\frac{i}{i+1}\right)$$

$$= \frac{1}{i+1}\sum_l b_l \ln(b_l) - \frac{\ln(i+1)}{i+1}\sum_l b_l \tag{7}$$

$$= \frac{1}{i+1}\sum_l b_l \ln(b_l) - \ln(i+1) \tag{8}$$

where

(a) if the added flow falls into existing j-th bucket ($b_j \longleftarrow b_j + 1$)

$$\sum_l b_l \ln(b_l) = \sum_k b_k \ln(b_k) - b_j \ln(b_j) + b_{j+1} \ln(b_{j+1}) \tag{9}$$

(b) if the added flow creates a new bucket m ($b_m \longleftarrow 1$)

$$\sum_l b_l \ln(b_l) = \sum_k b_k \ln(b_k) - b_j \ln(b_j) + 1 \ln(1) \tag{10}$$

The flows are then sorted into a heap with respect to this new measure ED, and the $flow$ with the maximal ED_{flow}, i.e., the one causing the biggest portion of $IP - port$ entropy difference, is being iteratively removed until the entropy difference falls within the restricted range. In each iteration, it is necessary to update the buckets, entropies, and individual ED_{flow_i} measures for all flows sharing a bucket (IP or Port) with the actually removed one. We note that all these updates are performed in constant $O(1)$ time by the same means as calculating the original ED measure, resulting from the entropy formula decomposition (see (8)). It follows that the complexity of this heuristic falls into $O(n^2)$, yet on average it is much faster since the sizes of buckets, which are iterated after each removal, are generally smaller than the cardinality of the whole flow set.

Various other heuristics might be used and found in literature [27] and by those means allowing us to convert the flow aggregative formulae subset search into some of the existing state-of-the-art search methods for combinatorial problems. Most of the heuristics we used are usually simple and greedy, and by those means generally suboptimal (see Section 4.4). On the other hand they remain intuitive and work in linear or low polynomial time. Finally they still provide some very good results, as discussed in Section 5.

4.4. Analysis. The brute-force solution complexity of this problem would fall into $O(k * (2^N))$, where k is the number of event types and n is the number of flows, ignoring the complexity of individual formulae evaluation. While this remains as the worst case scenario complexity in the approach introduced, exploiting the typical structure of event types, the new method will perform much faster in a typical scenario, where many flow-specific criteria are posed within the event type.

As stated in Section 3.2, the event-type descriptions are specified by a conjunction of formulae, or can be actually easily decomposed into a set of such, using de Morgan's laws, from any other rule-based description. In the algorithm we utilize this to treat any description as a single *sequence* of formulae and further exploit the order *invariability* of the conjunction to reorder the formulae according to the categorization into the flow-specific and aggregative types. By these means we first find a model for flow-specific formulae; i.e., we are looking for the biggest subsets of incoming flows satisfying flow-specific criteria, which we further refine to meet the aggregative criteria. The reason to search for the biggest subset rather than just any subset is to naturally maximize the recall of all flows (and endpoints) involved in the incidents.

Finding a model for the flow-specific criteria through version space creation, constantly adding valid flows into proto-events, is monotonic process and thus linear in the number of flows. The reason this phase is so fast is that it actually requires no formula checking and comparisons with the already processed objects (flows, clusters), in contrast to the clustering approaches (Camnep). This setting is similar in spirit to stream data mining [28] as the processing speed is invariant to the size of the processed traffic. This feature also allows refraining from using the (5-minute) time-windows to reduce the traffic load, possibly introducing errors when an event is undesirably split into two consecutive frames. Should there be desirable features of the time-windows kept, e.g. restrictions on the duration of an event type keeping the version space small, it could be implemented by just recording separately the time of the first flow for every proto-event and invalidating it after the time period expires.

The fast version space creation capability is implemented through the flow hashing (Algorithm 1). Assuming constant amortized time of existential queries in the hashmap, creation of version space takes only linear time in the number of incoming flows n. Since this is done for all k of the event types, this phase runs in $O(k*n)$. The factor k can be further reduced (into app. $\log(k)$), considering the (sometimes significant) overlap of flow-specific formulae across the classes of event types, which can be by those means arranged into a tree of common formulae sets.

The version space creation not only is faster but serves as a more accurate model of the real network events scenario. By continuously constructing not just a single sequence but several parallel temporary proto-events from the incoming flows, it results in that some of the flows get duplicated among various proto-events of different types, which comes naturally from nondistinctive definitions of event types and allows for their complete fit on the data, which was not possible in the disjoint clustering approach.

It is favorable to realize that decomposing (with repetitions) the original flow sequence into the version space divides the problem into smaller subproblems and breaks down every possible relevant information connection between the flows into independent subsets defined by the proto-events. This fact allows considering every proto-event as completely independent of each other and by those means parallelize all further proto-event processing (e.g., the aggregation phase), leading to additional speedup.

The second (aggregation) phase of the algorithm, i.e., processing individual proto-events with the aggregative formulae model search, is not monotone and cannot be reduced in

```
1: function SUBSETSEARCH($formula_{entropy}$, $Flows$)
2:    $portBuckets \longleftarrow$ groupBy($Flows$, $unique(Flows.Ports)$)
3:    $ipBuckets \longleftarrow$ groupBy($Flows$, $unique(Flows.Ips)$)
4:    $entropies \longleftarrow \{entropy(portBuckets), entropy(ipBuckets)\}$
5:    for all $flow \in Flows$ do                                    ▷each calculation $\in O(1)$
6:        $ED_{flow} \longleftarrow$ entropyDiffAfterRemovalOf($portBuckets$, $ipBuckets$, $flow$)
7:    end for
8:    $heapFlows \longleftarrow$ heap($Flows$, $ED$)                           ▷flows sorted w.r.t. ED
9:    while $|entropy_{portBuckets} - entropy_{ipBuckets}| > formula.Diff$ do
10:       $mFl \longleftarrow heapFlows$.removeMax()
11:       $\{portBuckets, ipBuckets\} \longleftarrow$ remove($portBuckets$, $ipBuckets$, $mFl$)
12:       $entropies \longleftarrow update(entropies, mFl)$                        ▷constant $O(1)$
13:       for all $flow \in ((portBuckets \cap mFl.port) \cup (ipBuckets \cap mFl.ip))$ do
14:           $ED_{flow} \longleftarrow entropyDiffAfterRemovalOf(portBuckets, ipBuckets, flow)$
15:           $heapFlows.heapify(flow, ED_{flow})$
16:       end for
17:   end while
18:   return $flows$
19: end function
```

ALGORITHM 7: Heuristic subset search for ports-IPs entropy difference limit.

a similar manner. It means to search again within a proto-event for a maximal subset of flows satisfying given restriction on the result of an aggregation function applied over them. For most formulae with aggregation function (e.g., Average, Entropy) this falls into the NP-complete category as it can be shown to be *subset-sum problem* reducible [29]. Also for some formulae it might be desirable that one set (proto-event) may yield more than one subset (event) of flows. As such it could be posed as a *CSP* optimization problem and solved explicitly by means of, e.g., dynamic programming [30]. Although the search in this step is now performed on typically much smaller subsets (proto-events) of the original set, for the sake of speed we suffice with approximate solutions and heuristics (e.g., Algorithms 6 and 7).

It is important to notice that the application of these heuristics is the only place in the whole method where any false negative error may arise and possibly an actual event be missed. Leaving the rest of the method exact and complete, this is thus the place to incorporate more extensive search should a greater recall be required. On the other hand, the heuristics provide good explanation for missing a particular flow-event classification (explained by a failure message of the respective formula), they take part in only some of the event types, and they have been experimentally proved to be very robust (Section 5).

5. Experiments

In the experiments we evaluate the new event extraction method against the clustering based extraction approach in Camnep over the scope of various event types. Since the models of these event types are given (Section 3.1), there is no correct or wrong classification considered, and we are interested only in the number of identified events at this stage, representing the recall of all events from the network satisfying these models. The decrease in recall of the events at this stage signifies false negative errors to be made by the system. These errors may arise due to the suboptimal clustering results in Camnep, or due to the greedy heuristics used for aggregative model search in the new method. The purpose of experiments is to evaluate the extent of these errors between the two methods by proportion of the recall over the selected event types.

5.1. NetFlow Data.

The traffic data we are working with were collected from a university network during one week. Statistical descriptives of the traffic over selected classes of events, as processed by Camnep, can be seen in Table 1. For the IDS context, the reported classes are limited to a potentially malicious subset, as assessed by a severity measure (1 – 9) used in Camnep; however the method is equally applicable to nonmalicious classes (severity < 4), too.

In its raw form the data consist of elementary information aggregated from network packets in the NetFlow format, i.e., the unidirectional component of TCP (UDP, ICMP equivalent), identified by shared source and destination endpoints, together with the aggregated attributes. Particularly, these are tuples of

$$\begin{gathered}(\mathit{Start\text{-}time}, \mathit{Duration}, \mathit{Protocol}, \mathit{src\text{-}IP}, \mathit{src\text{-}Port},\\ \mathit{dest\text{-}IP}, \mathit{dest\text{-}Port}, \mathit{Flags}, \#\mathit{Packets}, \#\mathit{Bytes})\end{gathered} \tag{11}$$

corresponding to the start time of the first packet in the flow, the duration and 4th layer protocol, source-destination ports and addresses, all flags aggregated from packets during the connection and overall number of packets and bytes transferred. This representation is widely adopted across computer networks analysis community and is a standard for security event logging as well as a number of other affiliated applications [31].

Table 1: Distribution of communication data captured within a university network over selected, potentially malicious, classes.

Event type	Severity	#IPs	#events	#flows
ssh cracking	9	149	3412	291538
ssh cracking response	9	143	3406	287686
port scan (in/out, tcp)	7	996	17463	21732
port scan (tcp)	7	539	1486	125799
port scan (vertical, tcp)	7	4	6	1485
dns tunnel-like behavior	6	6	14	14
dns tunnel-responses-like behavior	6	4	6	6
p2p-like behavior (tcp)	6	238	1009	215333
p2p-like behavior (udp)	6	135	9156	886345
p2p-responses-like behavior (tcp)	6	32	742	250956
p2p-responses-like behavior (udp)	6	147	5424	985336
data transfer (tcp)	4	244	4639	108729
data transfer (udp)	4	94	793	26026
icmp traffic	4	1127	6370	93315
scan-like behavior (horizontal, tcp)	4	940	1165	11358
scan-like behavior (horizontal, udp)	4	1	10	2228
scan-like behavior (vertical, tcp)	4	6	37	29969
scan-like behavior (vertical, udp)	4	3	17	5700
scan-responses-like behavior (horizontal, tcp)	4	300	3742	5489
scan-responses-like behavior (horizontal, udp)	4	0	0	0
scan-responses-like behavior (vertical, tcp)	4	7	48	32785
scan-responses-like behavior (vertical, udp)	4	3	26	8495

In cooperation with Cisco research, we used full-day NetFlow traffic records collected from inline university network probes, where the regular amount of traffic counts up to more than 6 million flows a day, corresponding to more than 30 thousand events with thousands of IPs involved.

5.2. Results. We evaluated both methods on the same full-day traffic data and compared the results for the numbers of identified events from flows over selected event types. Although both methods used the same input flows and event-type models, the results naturally differed due to a number of causes in the identification of flows as a part of some event. In analysis of these causes, four different types of event from flows identification situations (Figure 5) occurred.

(a) The new method finds some extra flows belonging to an event

(b) The new method finds a completely new event

(c) The new method misses some flows while incorporating majority of others

(d) Both methods identify exactly the same event

Firstly, the results showed that the new method dominated the clustering approach of Camnep in the numbers of identified flows. The only situation where it was not clearly dominating was the situation *c*, where some flow-event classifications were missed. In these situations, however, there was always a majority of flow-event classifications for the same event that were identified as opposed to Camnep. This results from the aggregative formulae model search (Algorithm 5), which searches for the maximal subset of valid flows, securing that any subset found either will completely encompass the subset resulting from clustering (situation *a*) or will find the same subset that happened to be maximal already (situation *d*), or, in the case they intersect only partially or not at all, the new method's subset will always be larger. Also we can notice that for the case of completely new event exploration by the new method (situation *b*) there is no complementary situation for the clustering method, which is rooted in the version space expansion phase of flow-specific formulae that secures the completeness of proto-events. In other words, there is no way to create a valid cluster that would be missed by the new approach as a part of some proto-event in the expansion phase, and it can only choose to prefer a different cluster to that one for being larger in the aggregation phase.

Of course this quality of the new method might be theoretically broken by the use of inadmissible aggregation heuristics in the maximal subset search (e.g., Algorithm 6). In practice, however, our experiments confirm exactly the mentioned scenarios from Figure 5. Moreover, the faulty situation *c* was observed for only 100 out of 6.5 million flows tested, corresponding to app. 0.000015% relative (to Camnep) error in flow-event identification recall. Also for every such a missed flow-event $flow_i \longmapsto event_x$ classification there is a corresponding explanation for excluding that particular flow $flow_i$ from event $event_x$ of a type $type_v$, based on the heuristic removing it while solving particular aggregative formula $agg_k \in desc_v$ of $type_v$, e.g., for the fact that the flow $flow_i$ is

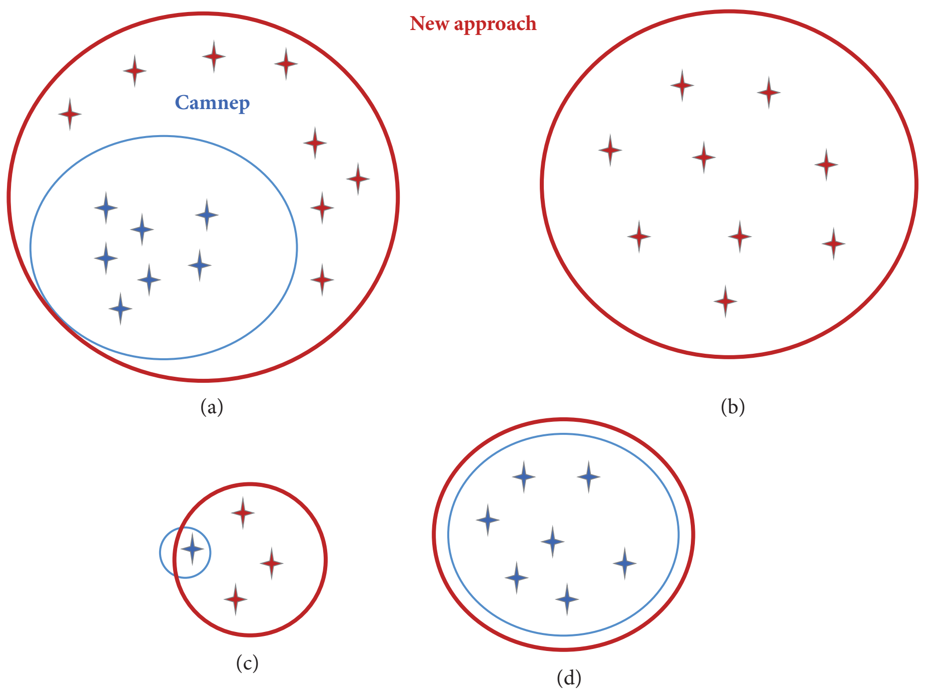

Figure 5: Depiction of the four different flow (star) to event (circle) conformations arising from comparison of the new method (red) with Camnep (blue). The diagrams are roughly proportional to the actual occurence of the cases.

distinctly bigger than a required average in the formula agg_k. This condition, i.e., situation *c* based on average aggregation heuristic search from Algorithm 6, was actually the case for a number of the 100 flows, which were included in a *p2p-like-behavior* event by the clustering approach, yet removed as outliers and replaced with more suitable flows (with respect to average-restricting formula $agg_k \in desc_v$ of *p2p-like-behavior*) by the new method, and so even the small relative error remains disputable to be an actual error.

The resulting comparison of the recall of events across selected event types is presented in Figure 6. As apparent from the results, within the generally smaller proto-events, there is not that much space for errors caused by the (inadmissible) heuristics, as compared to the clustering approach that is sensitive to many factors (Section 4.1). The number of involved factors and the space for errors in recall grow with the complexity of event types. For that reason we can see that the difference between performances of both methods varies across the event types, being generally higher and more favorable for the new method in complex types. On the contrary the performances tend to be more similar for event types with simple descriptions, such as in the case of *icmp traffic* (very simple definition). Nevertheless, the new method performs consistently no worse than Camnep event extraction in all types of events and although the results were filtered for potentially malicious classes only, the new method fares significantly better on the nonmalicious event-type classes, too.

The second result we present in Figure 7 shows the distribution of newly identified events from previously unclassified flows, i.e., the flows that, processed by the clustering method in Camnep, ended up in clusters that did not match any of the models in scope. These newly identified events correspond to false negatives of the clustering approach and can thus be seen as a pure contribution to the previous approach corresponding to the situation *b* from Figure 5. Importantly, representatives of false negative errors of clustering in Camnep explored by the new method over various event types were consequently studied in flow-level detail and confirmed with the experts to contain real incidents of the respective event types.

Finally, we tested the speed of the new method, since fast computation is a crucial requirement for online NetFlow traffic processing and any method with too high complexity, although provably more accurate, would be rendered useless. The first version space expansion phase of the method is very fast and can be deployed online with negligible overhead. The second aggregation part is much more complex; however, introducing the mentioned improvements and heuristics for speedup (Section 4.4), we were able to decrease the complexity to very reasonable levels. The new method, implemented in Java, was verified to process 24 hours of university network traffic with app. 7 million flows within $10-15$ minutes of computational time on a single-threaded (for fair comparison with Camnep) personal pc, which is comparable to time complexity of the Camnep clustering approach with $7-10$ minutes of computation in the same setting. Thus, for a reasonable set of event classes, such as those reported in the performed experiments, the method is, on average, able to process traffic of a university-scale network in real time using just a single thread. Moreover, it can be easily parallelized for further scaling.

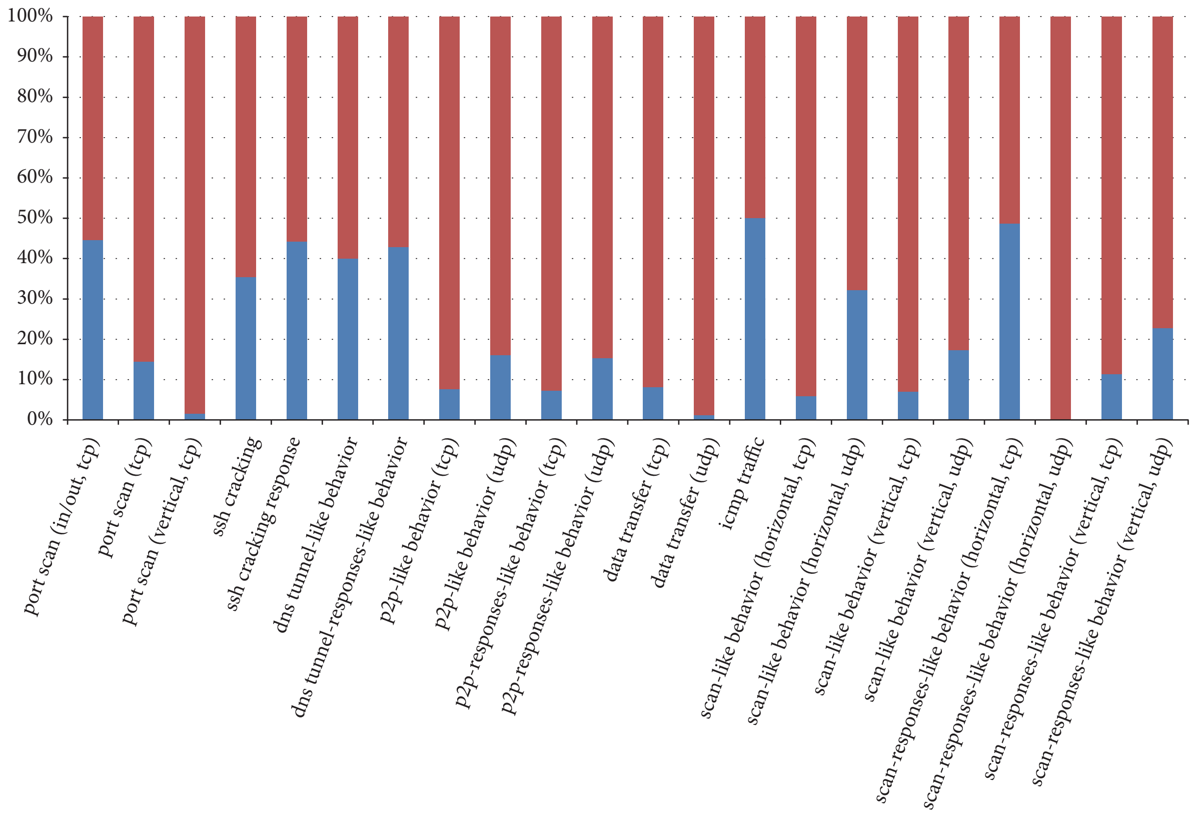

Figure 6: Statistical comparison of the new method (red) with Camnep (blue) over selected malicious classes for the number of identified events. Due to the immense difference of frequency of occurence between some of the event types, the total number of events (Camnep + new method) is normalized to 100% for every class.

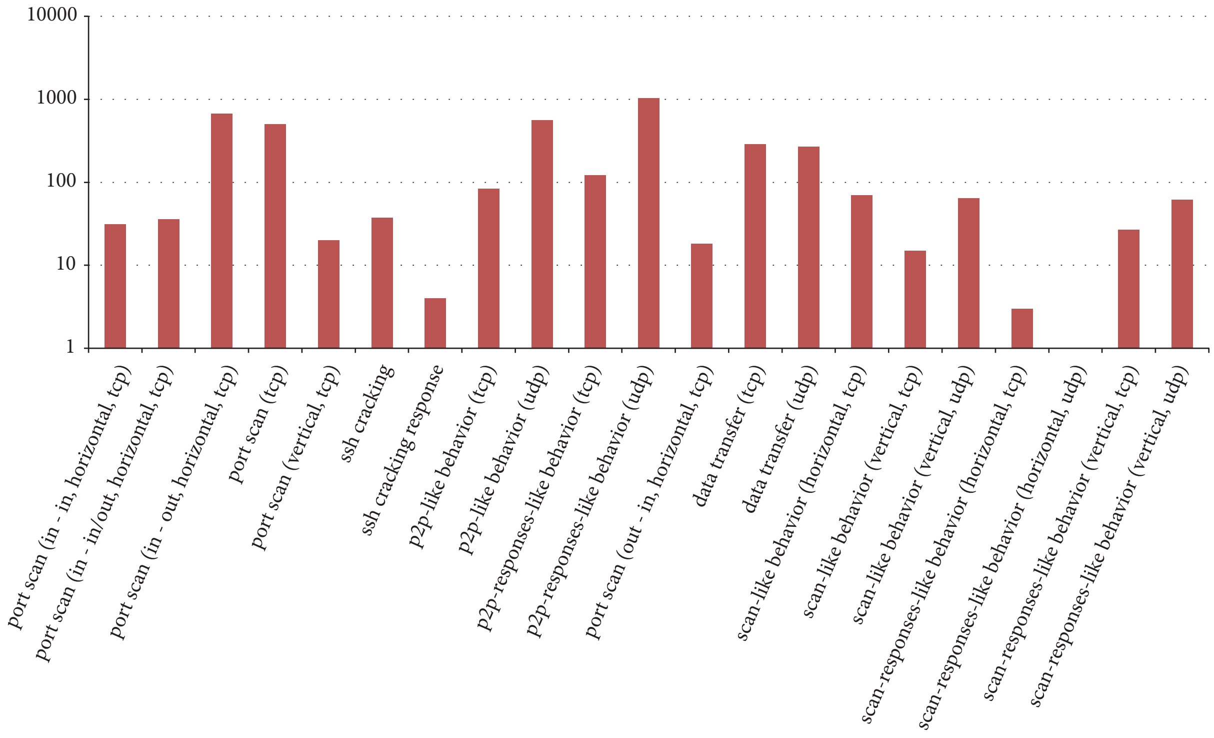

Figure 7: Unidentified behavior flows resulting from Camnep clustering, identified by the new method to belong into selected types of events. The displayed numbers of these events represent clear domination in recall over the clustering approach.

6. Conclusions

We introduced a method for efficient extraction of predefined network event types from raw NetFlow traffic data aimed at maximization of their recall. The core idea of the method is in explicit search for events based on analysis of expert descriptions of their corresponding types. We motivated and discussed the need for such an approach in the context of modern intrusion detection systems based on clustering and statistical models. Particularly, we compared with an in-production state-of-the-art intrusion detection system Camnep, developed by Cisco Research. Following this approach we decomposed the typically multistaged intrusion detection process, into stages that are, rather than on software concepts and intuition, based on computational properties of the sought-for event types.

We introduced corresponding efficient algorithms for event extraction and discussed their advantages. We also introduced a number of extending heuristics that enable scaling onto real life traffic volumes, at the expense of theoretical incompleteness of the event extraction. We analyzed properties of the method and showed experimentally that, in agreement with the analysis, the new method achieves supreme recall of known event types while keeping a very low computational overhead.

Data Availability

The university traffic data in the NetFlow format used to support the findings of this study were supplied by Cisco Cyber Threat Defense under license and so cannot be made freely available. Requests for access to these data should be made to Gustav Sourek, souregus@fel.cvut.cz.

Disclosure

Present address is Dept. of Computer Science, Karlovo namesti 13, Prague 2, 120 00, Czech Republic.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors are grateful to researchers from Cisco Cognitive Threat Analytics, namely, Karel Bartos, Petr Somol, and Tomas Pevny, for collaboration on the project. The work in this paper was supported by OP VVV project Research Center for Informatics no. CZ.02.1.01/0.0/0.0/16_019/0000765. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures."

Endnotes

1. The system for event extraction from NetFlows detailed in this paper is covered by US Patent US9374383B2 [32]

References

[1] M. Rehak, M. Pechoucek, P. Celeda, J. Novotny, and P. Minarik, "CAMNEP: Agent-based network intrusion detection system," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2008*, pp. 1813–1816, Portugal, May 2008.

[2] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," *Computer Communication Review*, vol. 35, no. 4, pp. 229–240, 2005.

[3] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *Computer Communication Review*, vol. 36, no. 2, pp. 23–26, 2006.

[4] M. Mizutani, K. Takeda, and J. Murai, "Behavior rule based intrusion detection," in *Proceedings of the CoNext Student Workshop '09*, pp. 57-58, Italy, December 2009.

[5] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proceedings of the 6th International Workshop on Passive and Active Network Measurement (PAM '05)*, pp. 41–54, April 2005.

[6] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[7] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang, "Lightweight application classification for network management," in *Proceedings of the 2007 SIGCOMM Workshop on Internet Network Management, INM '07*, pp. 299–304, Japan, August 2007.

[8] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the ACM SIGMETRICS International Conference On Measurement and Modeling of Computer Systems (SIGMETRICS '05)*, pp. 50–60, Alberta, Canada, June 2005.

[9] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 3015, pp. 205–214, 2004.

[10] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core," in *Proceedings of the 16th International World Wide Web Conference (WWW '07)*, pp. 883–892, Banff, Canada, May 2007.

[11] F. Mansmann, F. Fischer, D. A. Keim, and S. C. North, "Visual support for analyzing network traffic and intrusion detection events using TreeMap and graph representations," in *Proceedings of the 3rd ACM Symposium on Computer-Human Interaction for Management of Information Technology, CHIMIT 09*, 2009.

[12] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining in work flow environments: Experiments in intrusion detection," in *Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining*, 1999.

[13] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '02)*, 2002.

[14] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, “Learning intrusion detection: supervised or unsupervised?” in *International Conference on image analysis and processing, (ICAP)*, 2005.

[15] K. Leung and C. Leckie, “Unsupervised anomaly detection in network intrusion detection using clusters,” in *Proceedings of the Twenty-eighth Australasian conference on Computer Science*, vol. 38, pp. 333–342, Australian Computer Society, Inc., Newcastle, 2005.

[16] K. Bartos and M. Rehak, “Trust-based solution for robust self-configuration of distributed intrusion detection systems,” *Europian conference in Artificial Intelligence*, 2012.

[17] M. Rehák, M. Pê, M. Grill, J. Stiborek, K. Bartoŝ, and P. Ĉeleda, “Adaptive multiagent system for network traffic monitoring,” *IEEE Intelligent Systems*, vol. 24, no. 3, pp. 16–25, 2009.

[18] A. A. Olusola, O. S. Adeola, and O. A. Daramola, “Analysis of KDD’99 Intrusion Detection Dataset for Selection of Relevance Features,” in *Proceedings of the World Congress on Engineering and Computer Science*, 2010.

[19] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *Proceedings of the 2nd IEEE Symposium on Computational Intelligence for Security and Defence Applications*, pp. 1–6, 2010.

[20] J. McHugh, “Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory,” *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 262–294, 2000.

[21] M. Rehak, M. Pechoucek, K. Bartos, M. Grill, and P. Celeda, “CAMNEP: An intrusion detection system for highspeed networks,” *Progress in informatics, special issue: the future of software engineering for security and privacy*, 2008.

[22] M. Pechoucek, “Game theoretic, multi-agent approach to network traffic monitoring: final report,” *US Defense Technical Information Center*, 2012.

[23] G. Šourek, O. Kuželka, and F. Železný, “Learning to Detect Network Intrusion from a Few Labeled Events and Background Traffic,” in *Intelligent Mechanisms for Network Configuration and Security*, vol. 9122 of *Lecture Notes in Computer Science*, pp. 73–86, Springer International Publishing, Cham, 2015.

[24] J. Jusko and M. Rehak, “Revealing cooperating hosts by connection graph analysis,” *Security and Privacy in Communication Networks*, 2013.

[25] Y. N. Andrew, M. I. Jordan, and W Yair, “On Spectral Clustering: Analysis and an algorithm,” *Advances in Neural Information Processing Systems*, 2002.

[26] So-In. Chakchai, *A Survey of Network Traffic Monitoring and Analysis Tools. Cse 576m computer system analysis project*, Washington University, St. Louis, USA, 2009.

[27] C. R. Reeves, Ed., *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley & Sons, NY, USA, 1993.

[28] H. O. Nasereddin Hebah, “Stream data mining,” *International Journal of Web Applications*, 2009.

[29] J. C. Lagarias and A. M. Odlyzko, “Solving low-density subset sum problems,” *Journal of the ACM*, vol. 32, no. 1, pp. 229–246, 1985.

[30] S. Martello and P. Toth, “A mixture of dynamic programming and branch-and-bound for the subset-sum problem,” *Management Science*, vol. 30, no. 6, pp. 765–771, 1984.

[31] D. Rossi and S. Valenti, “Fine-grained traffic classification with Netflow data,” in *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference (IWCMC ’10)*, pp. 479–483, July 2010.

[32] G. Sourek, K. Bartos, F. Zelezny, T. Pevny, and P. Somol, “Events from network flows,” US Patent 9,374,383, 2016.