# Novel anomaly detection and classification algorithms for IP and mobile networks

Agathe Blaise

**THÈSE DE DOCTORAT DE**
**SORBONNE UNIVERSITÉ**

Spécialité

**Informatique**

École doctorale Informatique, Télécommunications et Électronique de Paris
Laboratoire d'informatique de Paris VI

Présentée par

**Agathe BLAISE**

Pour obtenir le grade de

**DOCTEUR de SORBONNE UNIVERSITE**

Sujet de la thèse :

# Novel anomaly detection and classification algorithms for IP and mobile networks

soutenue le 14/12/2020

devant le jury composé de :

| | |
|---|---|
| Marco Fiore, IMDEA Networks | Rapporteur |
| Razvan Stanica, INSA Lyon, Inria | Rapporteur |
| Clémence Magnien, CNRS, Sorbonne Université | Examinateur |
| Sahar Hoteit, Univ. Paris Saclay, Centrale-Supélec | Examinateur |
| Aline Carneiro Viana, Inria Saclay | Examinateur |
| Thi-Mai-Trang Nguyen, LIP6, Sorbonne Université | Membre invité |
| Sandra Scott-Hayward, Queen University Belfast | Membre invité |
| Stefano Secci, Conservatoire National des Arts et Métiers | Directeur de thèse |
| Vania Conan, Thales | Co-encadrant |
| Mathieu Bouet, Thales | Co-encadrant |

**THÈSE DE DOCTORAT DE**
**SORBONNE UNIVERSITÉ**

Spécialité

**Informatique**

École doctorale Informatique, Télécommunications et Électronique de Paris
Laboratoire d'informatique de Paris VI

Présentée par


## Agathe BLAISE


Pour obtenir le grade de

**DOCTEUR de SORBONNE UNIVERSITE**

Sujet de la thèse :

# Nouveaux algorithmes de détection d'anomalies et de classification pour les réseaux IP et mobile

soutenue le 14/12/2020

devant le jury composé de :

| | |
|---|---|
| Marco Fiore, IMDEA Networks | Rapporteur |
| Razvan Stanica, INSA Lyon, Inria | Rapporteur |
| Clémence Magnien, CNRS, Sorbonne Université | Examinateur |
| Sahar Hoteit, Univ. Paris Saclay, Centrale-Supélec | Examinateur |
| Aline Carneiro Viana, Inria Saclay | Examinateur |
| Thi-Mai-Trang Nguyen, LIP6, Sorbonne Université | Membre invité |
| Sandra Scott-Hayward, Queen University Belfast | Membre invité |
| Stefano Secci, Conservatoire National des Arts et Métiers | Directeur de thèse |
| Vania Conan, Thales | Co-encadrant |
| Mathieu Bouet, Thales | Co-encadrant |

# Contents

# Abstract


## English version

Last years have witnessed an increase in the diversity and frequency of network attacks, that appear more sophisticated than ever and devised to be undetectable. At the same time, customized techniques have been designed to detect them and to take rapid countermeasures. The recent surge in statistical and machine learning techniques largely contributed to provide novel and sophisticated techniques to allow the detection of such attacks. These techniques have multiple applications to enable automation in various fields. Within the networking area, they can serve traffic routing, traffic classification, and network security, to name a few. This thesis presents novel anomaly detection and classification techniques in IP and mobile networks. At IP level, it presents our solution *Split-and-Merge* which detects botnets slowly spreading on the Internet exploiting emerging vulnerabilities. This technique monitors the long-term evolutions of the usages of application ports. Then, our thesis tackles the detection of botnet's infected hosts, this time at the host-level, using classification techniques, in our solution *BotFP*. Finally, it presents our *ASTECH* (for Anomaly SpatioTEmporal Convex Hull) methodology for group anomaly detection in mobile networks based on mobile app usages.

## French version

Ces dernières années ont été marquées par une nette augmentation de la fréquence et de la diversité des attaques réseau, qui apparaissent toujours plus sophistiquées et conçues pour être indétectables. En parallèle, des techniques sont développées pour les détecter et prendre des contre-mesures rapidement. Récemment, l'essor des techniques statistiques et d'apprentissage machine ("machine learning") ont permis un développement rapide de techniques innovantes visant à détecter de telles attaques. Ces techniques ont des applications dans de nombreux domaines qui gagneraient à être davantage automatisés. Dans le domaine des réseaux, elles s'appliquent par exemple au routage et à la classifcation de trafic et à la sécurité des réseaux. Cette thèse propose de nouveaux algorithmes de détection d'anomalies et de classification appliqués aux réseaux IP et mobiles. Au niveau IP, celle-ci présente une solution *Split-and-Merge* qui détecte des botnets qui se propagent lentement sur Internet en exploitant des vulnérabilités émergentes. Cette méthode analyse l'évolution à long-terme de l'usage des ports applicatifs. Ensuite, celle-ci aborde la détection d'hôtes infectés par un botnet, cette fois en utilisant des techniques de classification au niveau de l'hôte, dans une solution nommée *BotFP*. Enfin, cette thèse présente notre algorithme *ASTECH* qui permet la détection d'anomalies brutes dans les séries temporelles dans les réseaux mobiles, les regroupe en enveloppes convexes spatio-temporelles, et finalement induit plusieurs classes d'événements.

# Chapter 1

# Introduction

The diversity and frequency of network attacks have boomed in recent years, and such attacks appear more sophisticated than ever and devised to be undetectable. At the same time, adaptive techniques have been designed to detect them as soon as possible and to take rapid countermeasures. The recent surge in statistical and ML techniques largely contributed to provide novel and sophisticated techniques to allow the detection of such attacks. These techniques have multiple possible implications in any system that would require automation, in many fields. Within the networking area, they can serve traffic routing, traffic classification, network security, to name a few. We note that attackers may also leverage data analysis and ML techniques to finely craft their attacks and mimic normal end-user behaviors, which makes their detection even more complex. In this introduction, we first discuss the needs for appropriate data analysis techniques for cyber-security (Section 1.1). We then present the surge in statistical and Machine Learning (ML) techniques (Section 1.2), and their possible applications to address current challenges on network security (Section 1.3). We finally introduce our different contributions to the field (Section 1.4).

## 1.1 Context and motivation

The nature of anomalies detected in network traffic data is quite diverse [1]. Anomalies range from outages (including equipment malfunctions and outages from cloud and mobile network operators) and operational events (including updates and ingress shifts), to unusual end-users behaviors (including flash crowds and point to multi-point communications) and malicious ones (including denial of service attacks and malicious scans). Therefore, we rather look at different granularity levels and range of features to take into account each anomaly type's peculiarities. For example, Denial-of-Service (DoS) events may be detected by looking at per-flow volume anomalies, rather than to per-packet attributes. Network and port scanning may be detected at the flow-level (or even at the port-level), as each new port or combination of port and target IP generates a new flow. Finally, botnet detection may be performed at the flow-level and preferably at the host-level.

Furthermore, even if we focus on the detection of malicious behaviors, we also notice a wide variety of attacks that require specific detection techniques. The diversity in attackers' operating modes renders the appropriate detection more difficult. Attacks are also constantly more sophisticated, as illustrated by the Mirai botnet [2] that launched a massive attack towards DNS servers of major Internet providers in 2016, cutting access to high-profile websites for several hours. Beforehand, it reunited nearby 50,000 devices in its bot army, but has not been detected until too late. Mirai acted like a revolutionary IoT-based malware since the release of

its source code [3] that led to a huge increase in other botnets' development. As a matter of fact, malware that targets Internet-of-Things (IoT) devices is responsible for many Distributed Denial-of-Service (DDoS) attacks. It exploits the lack of security of connected objects to create botnets, spreading extremely fast. We expect to see an increase in such IoT attacks, along with the explosion of IoT devices that could grow up to 125 billion by 2030 [4]. Recently, DDoS attacks significantly increased in terms of number and duration; indeed, the first half of 2018 saw seven times more large attacks (higher than 300 Gbps) compared to the same period in 2017, as noted in a Kaspersky report [5]. Furthermore, these botnets slightly propagate and affect whole networks without even being noticed, until they reach their real target. Most botnets today are designed to serve economic ends, as illustrated by Botnet as a Service (BaaS) [6] services that sold instances of botnets to third parties.

Designed to ensure cyber-security in networks, Intrusion Detection Systems (IDSs) aim to identify malicious activities and related threats. However, as a matter of fact, some botnets are not detected during their spreading, but only at the time of the final attack. We specifically study the case of the Mirai botnet and we invoke several reasons why it has not been detected soon enough. *(i)* Current IDSs traditionally work with traffic granularity such as the flow, host, or packet-level. They do not monitor application ports and thus may miss global changes on the ports involved during the propagation of botnets. Ports can be scanned to fingerprint the target machine, to exploit known vulnerabilities, or to communicate with a Command-and-Control (C&C) server [7]. The sole common denominator for a botnet coming from very distinct sources and targeting lots of hosts is the port it scans. However, an IDS working on IP addresses would be unable to notice the anomalous port. *(ii)* Moreover, most IDSs work on small variations of traffic, generally using sliding windows of several seconds. Therefore, they cannot build long-term profiles per port and detect major changes in their usage. *(iii)* IDSs are usually deployed at a single point in the network, while ISP-scale attacks are only visible by looking at a holistic view of a wide area network. For these reasons, several botnets like Mirai have not been detected until too late. It is thus possible to develop dedicated algorithms to detect botnets of these kinds, but by design, they are not made to adapt to other anomalies types and thus appear deficient in detecting them. In fact, we have to find a trade-off between the detection accuracy (which is high if the algorithm is crafted to detect a specific kind of attack) and its scope (which is high if the algorithm can adapt to variants of such attacks or even other attack types).

This introduces a number of challenges for network security: attackers employ complex techniques to hide and the attacks become more and more sophisticated. There is thus an urgent need to detect this kind of threat as soon as possible. Dedicated techniques are designed to prevent systems and networks from being corrupted and to limit harms in the case of an attack. Cybersecurity researchers and attackers are the two players in this cat-and-mouse fight. The first ones seek to understand the modus operandi of attackers, sometimes very complex, and to design robust Intrusion detection systems (IDSs) adapted to constantly evolving attacks. In return, attackers employ innovative techniques to slip by unnoticed and go through the radar.

## 1.2 Statistical and ML techniques

In 1959, Arthur Samuel, a pioneer in the field of Machine Learning, introduced it as the field of study that gives computers the ability to learn without being explicitly programmed: "A computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program" [8]. Such techniques are designed to solve complex problems and enable automation in different areas. However, it was initially eschewed due to its large computational requirements and the limitations of computing power present at that time.

Due to the recent improvements in computing capacities and ML techniques and in big data storing and processing, last years have witnessed a large surge of statistical and ML techniques. Originally, statistical models have also been used to solve problems and enable automation. Like ML models, such techniques serve anomaly/outlier detection, for example through changepoint detection algorithms (e.g., based on the z-score metric) or time series decomposition. ML models are based on statistical learning theories and are classified into four learning paradigms: classification, regression, clustering, and rule extraction, each of those including several algorithms.

Therefore, both statistical and ML models contribute to the field of data analysis, but with slightly different purposes [9]. Statistical models require a good understanding of the data and are designed for inference about the relationships between variables, whereas ML models are designed to make the most accurate predictions possible. They also work well in conjunction with each other. In [10], the authors compare the performances of statistical and ML methods, for multiple forecasting horizons. According to their results, ML methods need to become more accurate, requiring less computer time, and be less of a black box. They also demonstrate that traditional statistical methods are more accurate than ML ones, as they point out the need to discover the reasons involved and devise ways to reverse the situation. Nevertheless, the authors specify that their findings are only valid for the specific dataset being used. For more details about both techniques, Chapter 2 provides extensive background and related work that are fundamentally related to data analysis applied to network security.

## 1.3 Data analysis applications to networking

Data analysis, composed of statistical and ML techniques, has a myriad of possible applications in the networking field. They include traffic prediction, traffic classification, traffic routing, congestion control, resource management, fault management, QoS and QoE management, and network security [11]. In our dissertation, we cover specifically the traffic classification and network security fields, seeking to provide novel algorithms designed to strengthen network cybersecurity. Traffic classification aims to accurately characterize and categorize network traffic into a number of classes of interest, according to various features. It enables network operators to perform a wide range of network operation and management activities, like capacity planning, differentiation, performance monitoring, or resource provisioning. Generally, network traffic classification methodologies can be decomposed into four broad categories that leverage port number, packet payload, host behavior, or flow features, that we study later in Section 2. Network security intends to protect the network against cyber-threats that may compromise the network's availability, or yield unauthorized access or misuse of network-accessible resources. Therefore, network security is quintessential for network operation and management. In addition, current IDSs must take into account real-time constraints and manage to process large and fast-changing datasets.

In addition to network security, data analysis techniques generally enable one to provide a better characterization of traffic in mobile or IP networks and of the end-users behavior. These built profiles can help the design of methodologies to automatically detect unusual phenomena and attacks, but not only. Gaining better knowledge about end-users behaviors can be useful for resource provisioning or pervasive computing applications.

## 1.4 Contributions and thesis outline

The dissertation discusses several novel anomaly detection techniques in relation to important fields of networking in association with emerging technologies in it. We thus present such anomaly detection and classification techniques in three different contexts: the detection of vulnerabilities' exploitation on the Internet, intrusion detection in IP networks (at enterprise-level), and anomaly detection cellular networks. On the same occasion, we develop methods that were not exploited before, by exploring novel points of view. The next three chapters of this manuscript correspond to each main technique.

First of all, **Chapter 2** presents the background and related work of this thesis, first introducing the state-of-the-art on statistical and machine learning techniques, and then covering the related work specific to each contribution.

**Chapter 3** proposes a technique for the early detection of emerging botnets and newly exploited vulnerabilities on the Internet, targeting botnets slightly spreading on the Internet not detected nor mitigated during their spreading. The Mirai botnet attack on September 2016, or more recently the memcached attack on March 2018, this time with no botnet required, are but two examples. Such attacks are generally preceded by several stages, including the infection of hosts or device fingerprinting; being able to capture this activity would allow their early detection. Our technique, named *Split-and-Merge*, consists in *(i)* splitting the detection process over different network segments, *(ii)* monitoring at the port-level, with a simple yet efficient change-detection algorithm based on a modified Z-score measure, *(iii)* aggregating local anomalies at a central correlation module to retain only the distributed ones. We argue how our technique can ensure the detection of large-scale attacks and drastically reduce false positives.

In **Chapter 4**, we explore another view angle for botnet detection, this time not at the Internet-level, but in IP networks and more especially in enterprise networks. Recent approaches supplant flow-based detection techniques and exploit graph-based features, incurring however in scalability issues, with high time and space complexity. Bots exhibit specific communication patterns: they use particular protocols, contact specific domains, hence can be identified by analyzing their communication with the outside. A way we follow to simplify the communication graph and avoid scalability issues is by looking at frequency distributions of protocol attributes capturing the specificity of botnets' behavior. We propose a bot detection technique named *BotFP*, for *BotFingerPrinting*, which acts by *(i)* characterizing hosts behavior with attribute frequency distribution signatures, *(ii)* learning benign hosts and bots behaviors through either clustering or supervised Machine Learning (ML), and *(iii)* classifying new hosts either as bots or benign ones, using distances to labeled clusters or relying on an ML algorithm.

In **Chapter 5**, we leverage machine learning techniques in cellular networks to analyze mobile app communications and unleash significant information about the current social and infrastructure states. A wide variety of events can engender unusual mobile communication patterns that may be studied for pervasive computing applications, e.g., in smart cities. Among them, local events (like concerts), national events (like natural disasters), and network outages can produce anomalies in the mobile access network load. We propose our ASTECH (for Anomaly SpatioTEmporal Convex Hull) detection methodology that first decomposes cellular data usage features time series, then detects raw anomalies in the residual components derived from the decomposition. Our method then aggregates raw anomalies into snapshots first, and groups the most abnormal ones to form spatiotemporal clusters. We can so unveil details about the mobile events timeline, their spatiotemporal spreading, and their impacted mobile apps, by clustering them into broad categories.

In addition, we provide in appendix A the description of research issued from the master

thesis, which covers anomaly detection in service chains of virtual network function. Then, appendix B provides additional experiments related to our *BotFP* detection methodology presented in Chapter 4, and in particular visual comparisons between different scanning processes and details about the feature selection process that we applied.

# Chapter 2

# Related work

This chapter introduces the concepts, background, and related work that are fundamentally related to data analysis applied to network security. It reviews the whole process of statistical and machine learning techniques, passing through their learning paradigms, performance metrics, and major applications. Then, the next three sections of this chapter follow the structure of the thesis, each of the sections corresponding to a chapter.

## 2.1 Statistical and machine learning techniques

Statistical and machine learning (ML) techniques can be leveraged for complex problems arising in network operation and management [11]. Last years have witnessed a surge of such techniques, thanks to significant improvements in computing capacities and recent advances in data storing and processing. Applied to network security, data analysis has been widely explored to develop novel automation and detection techniques. We first describe the theory behind statistical learning models and their relevance to the networking problem. Then we further review in detail the process for data analysis, which consists in various steps: *(i)* learning paradigms and ML techniques, *(ii)* data collection, *(iii)* features design, *(iv)* model evaluation, and *(v)* ML applications.

### 2.1.1 Statistical learning

Originally, ML detection tools rely on statistical learning theories to build their model. There also exist unsupervised detection tools that use plain statistical approaches; their central assumption is that the phenomena the most rarely observed are the most likely to be anomalous. In statistical approaches, the fine analysis of the built statistical profiles of traffic allows one to understand how the detected anomalous instances differ from the usual behavior; however, they work on a mono feature basis, thus do not correlate the different features by design. Moreover, statistical approaches work well combined with other algorithms as they are usually unable to provide additional information, such as the IP addresses of attackers or the attack root causes; in addition, they are of practically no computational complexity and easy to implement, which makes them a wise approach when the detection should operate with limited computational resources.

#### Hidden Markov models

Hidden Markov Models (HMMs) are based on augmented Markov chains [12]. HMMs consist of statistical Markov models where the probability functions modeling transitions between the

states are determined in the training phase, contrary to Markov chains where they are set a priori. HMMs are widely used in pattern recognition, and now tend to be extensively applied to intrusion detection as well. They generally show excellent performance, although they are not yet adapted to fit real-world network constraints. Indeed, they require a large amount of time to model normal behaviors and the false positive rate stands relatively high. In [13], authors propose an IDS where the payload is represented as a sequence of bytes, and the analysis is based on an ensemble of HMMs.

#### Changepoint detection-based techniques

The implicit assumption behind changepoint detection-based techniques is that anomalies induce significant changes in the probability distribution of feature values. Therefore, these approaches are quite fit to detect coarse anomalies, which have a significant effect on traffic, as DoS and DDoS would do. A pioneer work devising an online anomaly detection technique in computer network traffic using changepoint detection methods is [14]. The algorithm is based on the multi-cyclic Shiryaev–Roberts detection procedure, which is computationally inexpensive and performs better than other detection schemes.

For changepoint detection algorithms, the *z-score* is a well-known and simple statistical metric commonly used to automatically detect sudden changes in time series. More precisely, it is the measure of how many standard deviations below or above the mean a data point is. Basically, a z-score equal to zero means that the data point is equal to the mean, and the larger the z-score, the more unusual the value. An anomaly is detected if the absolute value of the modified z-score exceeds a given threshold. There also exist algorithms based on variants of this metric. The *modified z-score* uses the median and the median absolute deviation (MAD) from the median, instead of the classical mean and standard deviation respectively, which makes it outlier-resistant [15]. In addition, the *smoothed z-score* considers the influence of outliers, i.e., the weights of the past samples on the mean and standard deviation, with respect to the current sample.

#### Other approaches

Histograms are used to count or visualize the frequency of data (i.e., the number of occurrences) over bins, which consist of units of discrete intervals. Historically, they have been widely used in the data and image processing fields. Histogram-based algorithms, also named frequency-based or counting-based algorithms, rely on histograms containing the bins associated with the values of their attributes. [16] proposes an alternative approach to feature-based anomaly detection tools that builds detailed histogram models of the features and identifies deviations from these models. In addition, building comprehensive histograms is less computationally expensive than using coarse distribution or graph-based features.

### 2.1.2 ML techniques: paradigms and addressed problems

Now that we reviewed statistical learning-based approaches, we focus on ML techniques, investigating the whole ML design pipeline.

#### Learning paradigms

First, ML techniques can be classified into four learning paradigms:

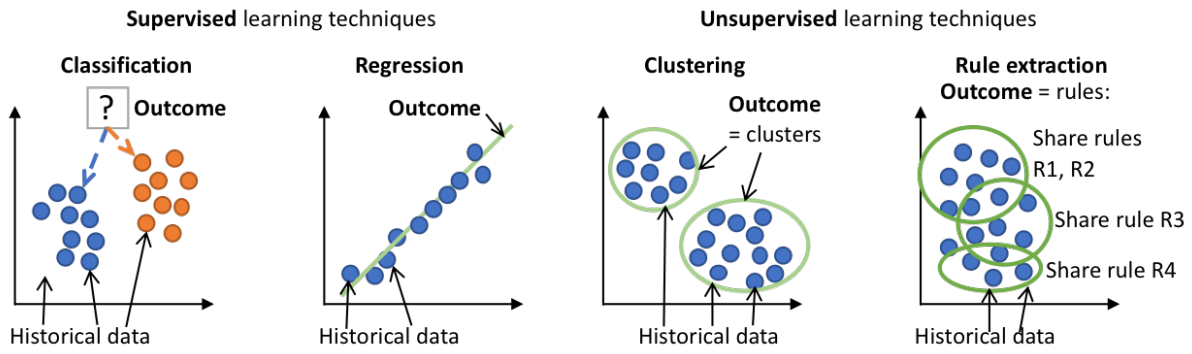



Figure 2.1: Learning paradigms that benefit from machine learning: classification and regression for supervised learning, and clustering and rule extraction for unsupervised learning.

1. *Supervised* learning techniques learn from a labeled dataset what constitutes either normal traffic or attacks – there exist different techniques such as SVM-based classifiers, rule-based classifiers, and ensemble-learning detectors [17].

2. *Unsupervised* approaches learn by themselves what is normal or abnormal – among them, MAWILab [18] finds anomalies by combining detectors that operate at different traffic granularities (the results against the MAWI dataset are in [18]); numerous works compare themselves to MAWILab, as for instance change-detection techniques [14, 19] (defining an anomaly as a sudden change compared to a model), and ORUNADA [20] (relying on a discrete time-sliding window to continuously update the feature space and cluster events).

3. *Hybrid* or *semi-supervised* approaches benefit from only a small part of labeled traffic, meant to be enough to learn from, as proposed in [21].

4. *Reinforcement learning* (RL) is an iterative process with agents that take actions in order to maximize the notion of cumulative reward. In the purpose of decision making, the learning is traditionally based on exemplars from training datasets. The training data in RL constitutes a set of state-action pairs and rewards (or penalties).

### Problem categories

Four broad categories of problems can leverage ML, namely *classification*, *regression*, *clustering*, and *rule extraction*, as illustrated in Fig. 2.1. First of all, *classification* and *regression* are two supervised learning approaches; their objective is to map an input to an output based on example input-output pairs from labeled data. *Regression* approaches predicts continuous values output, whereas *classification* predicts discrete values, consisting in the different labels. Then, *clustering* and *rule extraction* are unsupervised learning techniques: *clustering* is the task of partitioning the dataset into groups, called clusters - the goal is to determine grouping among unlabeled data, while increasing the gap between the groups; *rule extraction* techniques are designed to identify statistical relationships in data, by discovering rules that describe large portions of the dataset.

Note that the choice of the learning paradigm strongly depends on the training data. For example, if the dataset is not labeled, supervised learning cannot be employed and other learning paradigms must be considered.

**Concrete ML techniques applications.**

To illustrate the diversity in ML techniques applications, we provide concrete use cases for each of the aforementioned learning paradigms:

- *Classification* techniques are traditionally used in problems that contain labeled datasets, with two or more distinct classes. Botnet detection is but one example of such cases, where we can distinguish between malicious and benign flows; this way, the ML algorithm implicitly learns the inherent characteristics of a bot, and those of a benign host.

- *Regression techniques* are traditionally used for time series forecasting [22, 23]. The objective is to construct a regression model able to induce future traffic volume based on previous instances of traffic. Regression techniques are also employed to assess the impact of the global network condition on the QoS or QoE [24]. Finally, monitoring Key Performance Indicators (KPI) in large-scale networks enables the quick detection of network outages and attacks.

- *Clustering* techniques are usually employed for outlier detection purposes. In network cyber-security, many intrusion detection schemes [20] rely on data clustering to highlight significant deviations compared to usual end-user behaviors.

- Finally, *rule extraction* techniques, also named *association rule mining*, are commonly employed for personalized recommendations. Market Basket Analysis [25] is one of the key techniques used by large retailers to discover correlations within sets of items. These techniques are also used by recommendation engines as for Netflix (for personalized movies recommendation) and Amazon (for suggestions of other articles related to the purchased one).

### 2.1.3 Data collection

The process of collecting data to apply and validate a given ML technique is an important step, but nonetheless difficult. Finding representative data, possibly without bias and labeled is a non-trivial task. Datasets also vary from one problem to another and from one time period to the next one.

Data monitoring techniques are classified into *active*, *passive*, and *hybrid* techniques [26]. *Active* monitoring uses traffic measurement in order to collect relevant data and examine the state of networks. Such approaches commonly consist in a set of distributed vantage points hosting measurement tools like `ping` and `traceroute`; among them, RIPE Atlas [27] is a global network of over 10,000 probes that measure Internet connectivity and reachability, used for instance in [28] where authors identify data-center collocation facilities in traceroute data from RIPE Atlas built-in measurements, then monitors delay and routing patterns between facilities. In [29], given an arbitrary set of traceroutes, the authors first spot routing paths changing similarly over time, then aggregate them into inferred events and collect details to identify its cause.

In contrast, *passive* monitoring collects existing traffic and infer the state of networks from it. Compared to active monitoring, it ensures that the inferred statistics correspond to real traffic and it does not introduce additional overhead due to bandwidth consumption from injected traffic. Passive monitoring data can be obtained from various repositories, given it is relevant to the networking problem being studied. Such traces include CDN traces [30], darknets [31], and network telescope datasets [32, 33]. The latter consists of a globally routed, but lightly utilized

network prefix - a /8 for the University of California San Diego (UCSD) Network Telescope Aggregated Flow Dataset [32] and /20 for the Network Telescope dataset from LORIA [33]. Inbound traffic to non-existent machines is unsolicited and results from a wide range of events, including misconfiguration, scanning of address space by attackers or malware looking for vulnerable targets, and backscatter from randomly spoofed DoS attacks. Other examples of passive data repositories include the Measurement and Analysis on the WIDE Internet (MAWI) Working Group Traffic Archive [34] and the CTU-13 dataset [35]. We later review these datasets in detail, respectively in Sections 3.3 and 4.2.

### 2.1.4 Feature design

Before applying an ML algorithm to the dataset, the collected raw data must be formatted to cover an adequate set of features. The first phase named *Feature Extraction* consists in cleaning the dataset that may contain missing values or noise. In addition, the collected raw dataset may be too voluminous to be handled. The need for dimensionality reduction is justified by multiple reasons. First, a large number of features may induce a high computational overhead. Also, a phenomenon called the *curse of dimensionality* refers to the sparsity in data increasing with the number of dimensions, which makes the dataset no more consistent. We first depict the features traditionally selected for anomaly detection depending on the aggregation level. We then review the main strategies employed for feature extraction.

#### Common feature choice

In reality, the feature choice directly depends on the problem formulation (i.e., the detection target) and thus on the granularity level. The taxonomy of aggregation levels for network anomaly detection includes *payload-based*, *host behavior-based*, and *flow feature-based* techniques.

*Payload-based* anomaly detection systems parse the packet payload looking for known application signatures. However, this incurs a high computational overhead and requires manual interventions from humans to monitor the alerts and regularly update the signatures database. In addition, the payload tends to be systematically encrypted due to privacy concerns. Payload-based systems usually employ features such as the payload size, but also more complex ones like specific byte sequences or particular key-words present in the payload that would execute malicious actions.

*Host behavior-based* anomaly detection systems compute per-host traffic features to model behavioral characteristics of hosts. Contrary to payload-based systems, it examines the inherent characteristics of hosts but also assesses graph-based features by considering hosts as nodes in a graph, to measure for example the centrality of nodes or the amount and frequency of traffic exchanged between the nodes. IDSes implemented at the host-level are named Host-based Intrusion Detection Systems (HIDSes), whereas those at the network-level are named Network-based Intrusion Detection Systems (NIDSes). Common features used by such systems include packet counts exchanged between nodes [36], service proximity, activity profiles, session duration, periodicity [37], and byte encoding [38, 39] or statistical characterization of bytes [40], for each packet or each flow coming from a host.

*Flow feature-based* anomaly detection systems aggregate communications on a per-flow basis, which consists of a 5-tuple made from the protocol, the source and destination IP addresses, and the source and destination port numbers. It is then a unidirectional exchange of consecutive packets on the network from a port at an IP address to another port at another IP address using a particular application protocol, including all packets pertaining to session setup and tear-down, and data exchange. A feature is an attribute representing unique characteristics

of a flow, such as the number of packets in a flow, mean packet length, packet inter-arrival time, flow duration, entropy, to name a few. Entropy basically represents the traffic distribution predictability and enables one to detect volume-based anomalies such as DoS and DDoS. Flow feature-based techniques use flow features as discriminators to map flows to classes of interest.

#### Feature extraction

Two main processes are usually employed to adequately select features. The process of *Feature Selection* consists in removing the features that are not relevant or redundant in order to keep only a limited set of features. The filtering strategy (e.g. information gain), the wrapper strategy (e.g. search guided by accuracy), and the embedded strategy (selected features are added or removed while building the model based on prediction errors) are three techniques for feature selection.

The second process, named *Feature projection*, projects the data from a high-dimensional space to a space of fewer dimensions. The variance between each class is accentuated in the resulting space, removing redundancy in data. Both linear and nonlinear dimensionality reduction techniques exist. The main linear technique is named Principal Component Analysis (PCA), which finds the directions of maximum variance [41]. The fraction of variance explained by a principal component is the ratio between the variance of that principal component and the total variance. The objective is to reduce the dimensionality while keeping a good amount of information, so that the cumulative explained variance ratio is close to 100%. PCA may also be used for traditional outlier detection [42, 43]. Using real traffic traces, [43] demonstrates that normal traffic data can reside in a low-dimensional linear subspace and form a low-rank tensor. The anomalies (outliers) should stay outside this subspace. Therefore, tensor-based approaches try to recover the normal data by separating the low-rank normal data and outlier data from the noisy traffic data captured, and then detect anomalies by using the outlier data separated.

### 2.1.5 Performance metrics and model validation

In the case of supervised learning, we are able to compute some metrics to assess the performance of our classification model. A confusion matrix is a table often used to evaluate the performance of a classification model [11]. The basic terms are the following (expressed as whole numbers and not rates): True Positive ($TP$) is the number of bots correctly classified; True Negative ($TN$) is the number of benign hosts correctly classified; False Positive ($FP$) is the number of benign hosts incorrectly classified; False Negative ($FN$) is the number of bots incorrectly classified. This is a list of rates that are often computed from the confusion matrix for a binary classifier:

- *Accuracy*, computed as $ACC = \frac{TP+TN}{TP+TN+FP+FN}$, shows the percentage of true detection over the total number of instances. High accuracy is required. However, a bias may be introduced if the dataset is too unbalanced, then we need to consider other metrics.

- *True Positive Rate*, defined as $TPR = \frac{TP}{TP+FN}$, also known as recall, shows the percentage of predicted malicious instances versus all malicious instances. A high $TPR$ value is desirable.

- *False Positive Rate*, computed as $FPR = \frac{FP}{FP+TN}$, also known as false alarm rate, refers to the ratio of incorrectly classified benign instances versus all the benign instances. A low $FPR$ value is desirable. If the dataset is too unbalanced, consider using the precision and recall instead of the TPR and FPR.

- *Precision*, computed as $P = \frac{TP}{TP+FP}$, refers to the ratio of incorrectly classified benign instances versus all the benign instances. A high $P$ value is desirable.
- *Recall*, computed as $R = \frac{TP}{TP+FN}$, also known as false alarm rate, refers to the ratio of incorrectly classified benign instances versus all the benign instances. A high $R$ value is desirable.
- *F1-score*, computed as $F1 = 2 \cdot \frac{P \cdot R}{P+R}$, is defined as the harmonic mean of the precision and recall. A high $F1$ value is desirable.

When no ground-truth is available (e.g., in unsupervised learning), we cannot directly assess the quality of a model. Then we must create ground-truth labels ourselves, for example by comparing several datasets and mixing various sources.

## 2.2 Intrusion detection

In this section, we present intrusion detection methodologies and review related work related to Chapter 3, including the different families for intrusion detection systems (IDSs) and large-scale IDSs, and those centered around botnet detection. Fig. 2.2 shows the general classification of intrusion detection systems. We introduce granularity levels in Section 2.1.4, intrusion detection methodologies in Section 2.2.1, and the different architectures in Section 2.2.2.

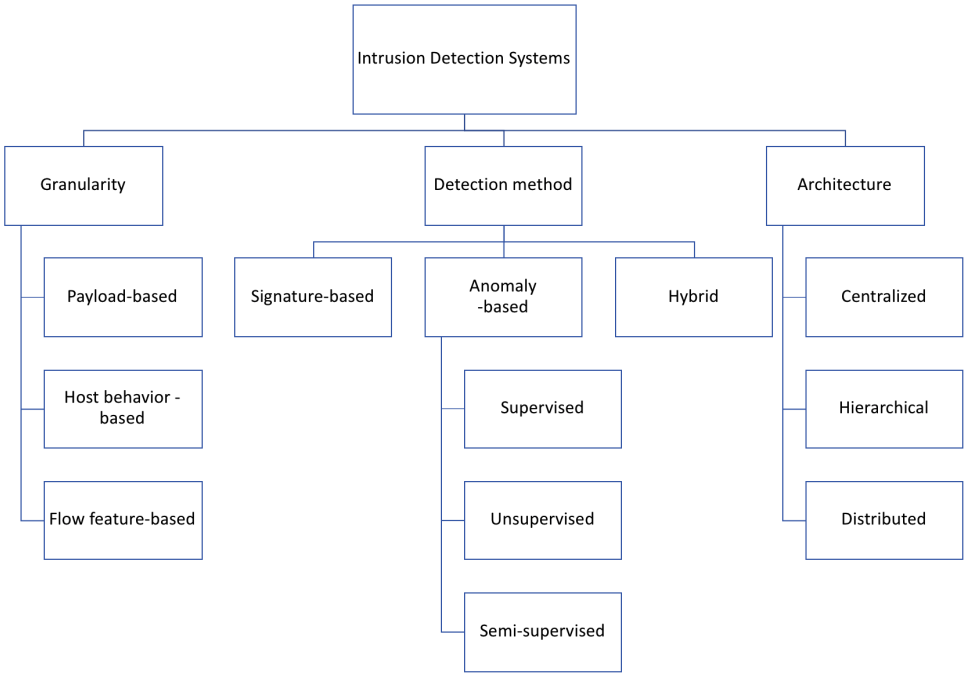



Figure 2.2: General classification of Intrusion Detection Systems.

### 2.2.1 Intrusion detection methodologies

Many algorithms are proposed in the literature for network intrusion detection [44]. We can classify them into three main families: signature-based, anomaly-based, and hybrid techniques.

*Signature-based*, also referred to as knowledge-based or misuse-based, solutions such as Snort [45], Zeek (formerly Bro) [46], or Suricata [47], rely on a signature database to find attacks that match given patterns, such as malicious byte sequences or known malware signatures. The first step consists in building a set of rules based on signatures; large rules databases can be purchased online and one can also create custom rules. Then, a rule is defined as: the action to apply if there is a match (e.g., `alert`, `log`, `pass`), the protocol to filter, the source and destination IP addresses, the port numbers, the traffic direction, and the options (TCP flags, payload size, etc.). The network administrator then collects the logs produced by the IDS and manually visualizes the set of produced alerts. Up to now, most companies rely on signature-based IDSs as they are expressive and understandable by network administrators. Nevertheless, they are not able to detect zero-day attacks, i.e., attacks exploiting unknown vulnerabilities, for which no patch is available [48], and the signatures database must be updated regularly.

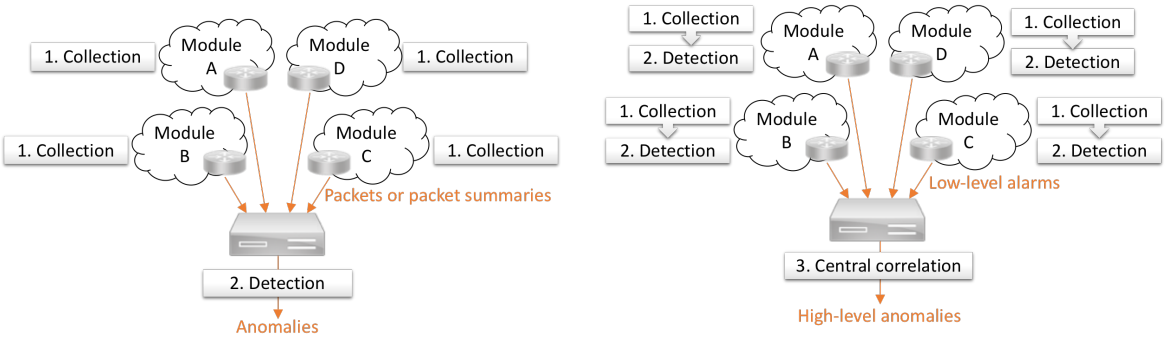
*Anomaly-based* approaches attempt to detect zero-day attacks, in addition to known ones. Compared to signature-based approaches, they require prior learning on data. They model the normal network traffic and qualify an anomaly as a significant deviation from it, with statistical or machine learning techniques. In such a case, we talk about anomalies rather than attacks. As addressed in Section 2.1.2, the methodology to detect anomalies can leverage statistical techniques, or supervised, unsupervised or semi-supervised ML techniques. BotSniffer [49] utilizes statistical methods to detect Command-and-Control channels (C&C) botnets, in which bot-infected hosts listen for attack commands from the attacker via this channel. To detect them, the authors seek for coincident behaviors among hosts, like messages to servers, network scans, or spam. The authors in [50] observe changes in feature distributions to identify anomalies. Entropy and/or volume are such metrics used for this purpose.

Various IDSs systems in the literature propose to combine signature-based or supervised techniques with unsupervised ones, that we call *hybrid systems*. They present the advantage of improving the detection rate and minimizing the false positive rate, inheriting the advantages of both methods. ADAM [51], which stands for Audit Data Analysis and Mining, is one of the most popular hybrid IDSs. ADAM has two stages of detection: *(i)* it builds a set of recurrent benign instances from attack-free datasets; then *(ii)* it finds frequent itemsets in connections and classify them compared to the previous database, into known or unknown attack types or false alarms. Another approach, [52], proposes a hybrid intrusion detection method, composed of a misuse detection model based on the C4.5 decision tree algorithm and multiple one-class SVM models to model the normal behavior. Finally, [53] uses a Self-Organizing Map (SOM) structure to model normal behavior, and J.48 decision trees for the misuse module.

### 2.2.2 Large-scale intrusion detection

Coordinated attacks arise in multiple networks simultaneously and include large-scale stealthy scans, worm outbreaks, and DDoS attacks [54]. Traditional IDSs tend to fail at detecting these attacks as they commonly monitor only a limited portion of the network. Large-scale IDSs, instead, have a global view over the network, and can better scale by distributing the computational load between several detection agents. Two large-scale IDS approaches can be identified.

The first IDS approach consists in distributing flow collectors in different subnetworks and in running a central detection engine against aggregated data, as shown in Figure 2.3a. Raw packets are transmitted from the flow collectors to the detection engine [55]. Solutions exist to avoid the collection traffic overhead, as done by Jaal [56], which creates and sends concise packets summaries to the detector - with Jaal, one reaches a 35 % bandwidth overhead to get an acceptable true positive rate, which is still important.




(a) Local collection and central detection.

(b) Centralized CIDS: local collection and detection, and central correlation.

Figure 2.3: Two possible approaches for large-scale IDS.

The second IDS approach consists in *Collaborative Intrusion Detection System* (CIDS), which is a two-level anomaly detection system where monitors are physically split in the network to perform local detection. They generate low-level alerts then aggregated to produce a high-level intrusion report. Three types of CIDSes exist depending on communication architecture:

1. *Centralized* CIDSes are composed of several monitors that transmit the alerts to a central correlation engine, as illustrated in Figure 2.3b.

2. *Hierarchical* CIDSes use a multistage structure of monitors to achieve an increasingly higher alert aggregation until the alerts reach the top correlation engine.

3. *Distributed* CIDSes share the detection and correlation tasks between all monitors. This approach can be set up by a peer-to-peer network.

For instance, [57] presents a centralized CIDS framework composed of IDS clusters implementing both the detection and the correlation; Snort signatures are therein used to detect known attacks, while an unsupervised learning algorithm detects unknown attacks. [58] proposes a sort of distributed CIDS, composed of Intrusion Prevention Systems forming rings around the hosts to protect, in order to collaborate and forward the traffic adaptively depending on their findings.

Inherent in the CIDSs, alert correlation algorithms can be divided into three categories [59]: *(i) similarity-based* algorithms, which compute the similarity between an alert and a cluster of alerts, and based on the result either merge it with the cluster or create a new one; *(ii) knowledge-based* algorithms, which rely on a database of attacks definitions; *(iii) probabilistic* algorithms, which use similar statistical attributes to correlate attacks.

### 2.2.3 Application to botnet detection

#### Flow-based botnet detection

In the past years, several novel algorithms for botnet detection have been proposed, which can be classified into packet- or flow-based ones and graph-based ones. Among them, [60] compares the performances of four different approaches: Snort, BotHunter, and two data-mining based system ones, either based on the packet header/payload or on flows. They run their algorithm on public datasets, including the Conficker dataset from CAIDA, the ISOT-UVic dataset, and Zeus botnet datasets from Snort, NETRESEC, and NIMS. As a result, they get detection rates

approaching up to 100%. BotMark [61] exploits both statistical flow-based traffic features and graph-based features to build its detection model, then considers similarity and stability between flows as measurements in the detection. The authors test their algorithm by simulating five newly propagated botnets, including Mirai, Black energy, Zeus, Athena, and Ares, and achieve 99.94% in terms of detection accuracy. In [62], the authors create a complete characterization of the behavior of legitimate hosts that can be used to discover previously unseen botnet traffic. They employ the ISCX botnet dataset, a publicly available dataset composed of various IRC, P2P, and HTTP-based botnets. They find that their framework can detect bots in a network with 100% TPR and 8.2% FPR. It is worth noting that the aforementioned algorithms perform their analysis at the network-level, on traffic generated by botnets. Their objective is to distinguish between benign hosts and bots, to then draw a confusion matrix and evaluate their classifier. Then they are not designed to run at an Internet carrier link-level. We later expose in Chapter 3 our wish to analyzing the current trends in Internet traffic over several years, including trends in terms of botnets.

#### Port-based detection techniques

A few works specifically focus on port-based detection but they do not apply to CIDS. In [7], the authors propose a survey of the current methods to detect port scans. [63] aims to show the correlation between port scans and attacks. [64] examines the period during the release of a zero-day attack and its patching. Also, [63, 64] analyze port-usage but they do not use destination ports as the primary key. Actually, this last setting generates a high number of false positives, which can be mitigated by CIDS as we are doing. In the literature, the numbers of unique source IP addresses and unique active /24 blocks are used to detect Internet outages [65] and large-scale spoofing [66].

## 2.3 Botnet Detection

The last section covered the related work on intrusion detection systems, we now specifically address the field of botnet detection related to Chapter 4. The word "botnet" comes from the combination of "robot" and "network". In this display, the attackers infect and control thousands of machines, then send them malicious commands to execute, like infecting, attacking, or scanning other hosts. This large zombie network is then a major vector of large-scale attacks such as phishing DDoS, Trojans, spams, etc. To communicate with bots, cybercriminals use Command-and-Control (C&C) channels implemented in different ways (the most popular ones are IRC, HTTP, P2P, and Telnet [67]).

Considering the importance of the matter, an extensive number of works exist in this field. While traditional approaches rely on statistical and machine learning approaches over per-flow features, recently studied graph-based approaches analyze the relations between several hosts of a network.

### 2.3.1 Flow-based techniques

Flow-based techniques work by removing the packet payload and inspecting the packet header only [68]. Let us classify them as follows.

### Statistical methods

BotHunter [69] aims to recognize the infection and coordination dialog that occurs during a successful malware infection. A similar approach, BotSniffer [49], focuses on the detection of C&C channels which are essential to a botnet. Therefore it exploits the underlying spatiotemporal correlation and similarity property of botnet C&C (horizontal correlation). The C&C server uses to contact every bot at the same time, then each of them uses to undertake some malicious actions following the C&C commands; these behaviors can be observed simultaneously in a network to spot a C&C channel, thus an underlying botnet.

BotHunter and BotSniffer perform their evaluation on their own honeynet or on traces authors built by executing malware binaries. However, these traces are not publicly available and [70] highlighted the lack of suitable comparisons for botnet detection algorithms due to the lack of public botnet datasets. Hence the authors a labeled botnet dataset named the *CTU-13* dataset [35] (later introduced in Section 4.2), including botnet, normal, and background traffic. In addition, the authors present two methods to identify botnets in these traces. The first one named BClus is a botnet detection approach. It creates models of known botnet behavior by computing features per source IP address, then it uses them to detect similar traffic on the network. The second one named CAMNEP is a network behavior analysis system that combines various state-of-the-art anomaly detection methods, such as MINDS, Xu, Lakhina volume, and Lakhina entropy [71].

### Machine learning methods

Machine learning methods include artificial neural networks, support vector machines (SVM), $k$-nearest neighbor ($k$-NN), decision trees and clustering. As seen in Section 2.1.2, ML methods include supervised, unsupervised, and hybrid learning. Supervised learning techniques encompass different methods such as SVM-based classifiers, rule-based classifiers, and ensemble-learning detectors [17]. Due to its excellent generalization performance, Support Vector Machines (SVM) are used in many security applications [72, 73]. The unsupervised learning technique [74] proposes an unsupervised learning based ML solution to identifying known and unknown anomalies in IoT, more especially with auto-encoders; [75] also proposes an unsupervised approach, identifying the most dissimilar graphs. Finally, *hybrid* approaches benefit from only a small part of labeled traffic, meant to be enough to learn from, as proposed in [21].

### Other methods

Other methods use various entropy measurements. For instance, [58] proposes a technique to detect large-scale anomalies in the network traffic, by measuring the deviation between the profiles of normal traffic and incoming flow records. [70] proposes a behavioral botnet detection method using Markov Chains to model the different states in the C&C channel. The proposed method is trained and evaluated using the CTU-13 dataset and gives a 92% F1-measure and a 0.05% false positive rate. The authors in [76] focus on detecting bot-infected machines at the enterprise-level, by considering the complete DNS activity of a host per hour. They used an extensive set of features computed over campus DNS network traffic, and as a result, identified suspicious DNS connections to detect infected machines.

However, flow-based techniques may miss some communication patterns between hosts that are quite specific to a botnet. Furthermore, working on a per-flow or per-host basis may incur a high computational overhead.

### 2.3.2 Graph-based techniques

Graph-based approaches [77] aim to model the relations between several hosts of a network. They are studied for various situations, for example, to detect P2P bots [78, 79] or to recognize DNS traffic from malicious domains [80]. In [81], the authors distinguish between several kinds of traffic and make groups of flows from: *(i)* the most frequent 11 destination port numbers used by TCP and UDP, *(ii)* all other TCP/UDP destination port numbers, and *(iii)* ICMP flows. They use plain and derived features for each of these categories, then they train three unsupervised learning algorithms on normal traffic with these features. As a result, with $k$-NN they achieve over 91% detection rate with around 5% false positive rate. *BotGM* [75] proposes an unsupervised graph mining technique to identify abnormal communication patterns and label them as botnets. The authors first construct a graph sequence of ports for each pair of source and destination IP addresses, then they compare each graph between them using the Graph-Edit Distance (GED). As a result, they reach a very good accuracy between 78% and 95%. However, this technique is very costly as the GED is computed once for each pair of graphs and its computation is known to be NP-complete. The authors in [82] model network communications as graphs, where hosts are edges and communications between hosts vertices. They compute graph-based features such as In-Degree and Out-Degree and diverse centrality measures. They use a hybrid learning method and test various ML techniques to achieve a good detection rate. However, this technique incurs a high computational overhead as features are computed over a large communication graph, e.g., used by shortest paths algorithms computed for centrality measures. Other graph-based detection methods [83, 84] seem promising, but their complexity is often high, NP-complete as for [84] and [75], or cubic for [82] (see Section 4.6).

## 2.4 Spatiotemporal anomaly detection in cellular networks

In this section, we now review the related work on spatiotemporal anomaly detection, per-app mobile traffic analysis, and group anomaly detection, related to Chapter 5.

### 2.4.1 Detection of spatiotemporal anomalies

The survey [85] reviews large-scale mobile traffic analysis with respect to social, mobility, and network aspects. From a social perspective, authors show how the relationships between mobile traffic and a wide set of social features are addressed in the state of the art. Demographic, economical, or environmental factors do influence the way users consume mobile apps indeed. These factors are classified into four broad categories: users' interactions, demographics, environment, and epidemics. The possible relationships between the environment, in terms of both geographical and temporal features, and the communication structure are also described. Among them, the authors focus on the *detection of special events*, ranging from political happenings (e.g., elections or manifestations) to entertainment occasions (e.g., concerts, sports games) and accidents (e.g., power outages or exception road congestion).

Authors in [86, 87] propose threshold-based algorithms to detect special events. In [88], authors use the information on residual communication to determine how different geographical areas are affected by a same unusual event. Using a time series decomposition, they: *(i)* first exploit the Seasonal Communication Series (SCS) to segment the city into distinct clusters by noticing similar patterns of socio-economic activity, and *(ii)* compare the Residual Communication Series (RCS) of similar areas to detect local events. Another approach in [89] proposes a dedicated framework to detect general outlying behaviors, based on the hourly geographical

variations of mobile traffic, able to detect national holidays, political happenings, and sports events. First, it builds snapshots, i.e., representations of the load generated by mobile users at a given instant. Then, it forms snapshot graphs $\mathcal{G}(\mathcal{T}, \mathcal{E})$ where $\mathcal{T}$ contains the snapshots from the training set; it computes traffic volume and traffic distribution similarities to perform snapshot aggregation. Finally, it uses a hierarchical clustering method into a dendrogram structure and builds network usage profile categories. Authors so show it can coarsely classify snapshots: resulting clusters are mapped to network usage profile categories.

Finally, attention is also paid to events that are not the result of social behaviors, but of natural or human-caused disasters. E.g., in [90], authors focus on emergency situations, using a dataset containing a bombing, a plane crash, a mild earthquake, and a power outage in the target region.

### 2.4.2 Per-app mobile traffic analysis

Up to our knowledge, in the literature, the detection of anomalies such as special events is not tackled at the app level yet. We believe that analyzing per-app usages can give valuable details about the nature of events, thus can help finely characterizing them. In the literature, attention is paid to the app usage for other purposes than special events detection [91, 92, 93, 94]. In [91], authors provide an analysis of spatiotemporal heterogeneity in nationwide app usage - they notice a large bias between apps (even within the same category, like Chat or Download) that makes the time series clustering inconclusive, and some heterogeneity even when looking to activity peaks of individual apps. Authors in [92] investigate the similarities and differences across different apps using nine features; as a result, they identify several well-differentiated clusters for each category of apps. In [93], authors design a system able to identify key patterns of cellular tower traffic by clustering custom pattern identifiers in traffic into five categories: resident, transport, office, entertainment, and comprehensive, area. They study time and frequency-domain representations for traffic modeling by analyzing interrelationships between traffic patterns and using Fourier transform. [94] provides a complete comparative evaluation of the techniques for signature classification, including Weekday-Weekend, typical week, median week, etc. Results unveil the diversity of baseline communication activities across countries, but also evidence the existence of a number of mobile traffic signatures that are common to all studied areas and specific to particular land uses.

### 2.4.3 Group anomaly detection

While anomaly detection typically regards data point anomalies, group anomaly detection seeks to detect anomalous collections of points. Traditionally, Seeded Region Growing (SRG) [95] has been used in image processing to form regions into which the image is segmented, by grouping seeds (i.e., either individual pixels or regions). The Mixture of Gaussian Mixture Model (MGMM) uses topic modeling for group anomaly detection. Adaptive topics are useful in recognizing point-level anomalies, but cannot be used to detect anomalous behavior at the group level. More recently, [96] studies the group anomaly detection problem by discovering anomalous aggregated behaviors of groups of points. The authors propose the Flexible Genre Model (FGM), which is able to characterize groups' behaviors at multiple levels, contrary to traditional topic models. This detailed characterization enables the detection of various types of group anomalies. [97] performs group anomaly detection with an emphasis on irregular group distributions (e.g., irregular mixtures of image pixels). The authors formulate two deep generative models for group anomaly detection.

Other approaches specifically focus on spatiotemporal outlier detection. In [98], the authors review outlier detection for spatiotemporal data, among other things. Considering the temporal and spatial neighborhood for detecting outliers, they define a spatiotemporal outlier (ST-Outlier) as a spatiotemporal object whose behavioral attributes are significantly different from those of the other objects in its spatial and temporal neighborhoods. They propose a typical spatiotemporal-outlier detection pipeline, taking as input spatiotemporal data composed of: *(i)* processing spatiotemporal data, *(ii)* finding spatial objects, *(iii)* find spatial outliers, *(iv)* verify/track temporal outliers, and *(v)* producing spatiotemporal outliers, as existing approaches commonly find spatial outliers and then verify their temporal neighborhood. Many approaches leverage clustering to compute spatial outliers [99, 100]. Others use distance-based outlier detection and Voronoi diagrams to establish spatial clusters [101].

## 2.5 Summary

We draw in this chapter in a synthetic yet wide-enough way the composite field of machine learning, its recurrent relationships with statistics, with a particular focus on the metrics and methods we adopt in this thesis. We provide as well a background in the three main areas of application of the exposed principles and methods to our research. In the next chapters, we further precise our three contributions in the area, positioning our work with respect to this state of the art.

# Chapter 3

# Detection of zero-day attacks

Last years have witnessed more and more DDoS attacks towards high-profile websites, like the Mirai botnet attack on September 2016, or more recently the memcached attack on March 2018, this time with no botnet required. These two outbreaks were not detected nor mitigated during their spreading, but only at the time they happened. Such attacks are generally preceded by several stages, including infection of hosts or device fingerprinting; being able to capture this activity would allow their early detection. In this chapter, we propose a technique for the early detection of emerging botnets and newly exploited vulnerabilities, which consists in *(i)* splitting the detection process over different network segments and retaining only distributed anomalies, *(ii)* monitoring at the port-level, with a simple yet efficient change-detection algorithm based on a modified Z-score measure. We argue how our technique, named *Split-and-Merge*[1], can ensure the detection of large-scale attacks and drastically reduce false positives. We apply the method on two datasets: the MAWI dataset, which provides daily traffic traces of a transpacific backbone link, and the UCSD Network Telescope dataset which contains unsolicited traffic mainly coming from botnet scans. The assumption of a normal distribution – for which the Z-score computation makes sense – is verified through empirical measures. We also show how the solution generates very few alerts; an extensive evaluation on the last three years allows identifying major attacks (including Mirai and memcached) that current Intrusion Detection Systems (IDSs) have not seen. Finally, we classify detected known and unknown anomalies to give additional insights about them.

## 3.1 Introduction

Back in September 2016, the Mirai botnet [2] struck the internet with a massive distributed denial of service (DDoS) attack. During several months, it spread slowly and reunited nearby 50,000 bots distributed over various parts of the internet, without being noticed. More recently, a record-breaking DDoS attack hit Github in February 2018 with a new amplification attack vector: UDP-based memcached traffic [105]. The caching system is supposed to be used internally, but sometimes runs on servers exposed without any authentication protection; several days later, most memcached servers have been patched, making the attack not efficient anymore [106]. Actually, malware targeting Internet-of-Things (IoT) devices and misconfigured servers are responsible for many Distributed Denial-of-Service (DDoS) attacks [107]. Detecting these

[1]This work was first presented at *IFIP/IEEE IM 2019* [102], then extended for an article published in *Elsevier Computer Networks* [103]. In addition, we propose in the book chapter [104] a model to simplify the detection of large-scale network attacks combining data plane programming with control level collaboration, inspired from *Split-and-Merge*.

botnets and exploited vulnerabilities during their spreading could avoid many harms. There is thus an urgent need to detect this kind of threat as soon as possible, and current anomaly detection tools appear deficient in this respect.

Ensuring cyber-security in networks, Intrusion Detection Systems (IDSs) monitor network traffic for malicious activities and related threats. However, as a matter of fact, most botnets go under the radars for three reasons: *(i)* Current IDSs work at different traffic granularities, e.g., flow, host, or packet. However, they miss global changes on application ports that are involved during the propagation of botnets. Ports can be scanned to fingerprint the target machine, to exploit known vulnerabilities, or to communicate with a Command-and-Control (C&C) server [7]. The sole common denominator for a botnet coming from very distinct sources and targeting lots of hosts is the port it scans. However, an IDS working on IP addresses would be unable to notice the anomalous port. *(ii)* Most IDSs work on small variations of traffic, generally using time-sliding windows of several seconds. Therefore, they cannot build long-term profiles per port and detect major changes in their usage. *(iii)* IDSs are usually deployed at a single point in the network, while ISP-scale attacks are only visible by looking at a holistic view of a wide area network.

In this chapter, we propose an anomaly detection technique that spots main changes in the usage of a single port to identify botnets. Intuitively, the most obvious way to identify it is to observe a sudden rise in traffic towards a port. However, this may not be sufficient as it can be a well-known vulnerable port, already massively scanned. For example, before the Mirai attack, many TCP SYN scans targeted the Telnet port whose vulnerabilities were already known and exploited. Then, when the Mirai attack was actually hitting, one could not observe an increase in the number of scans targeting this port. Our goal is to detect early stealthy changes in the behavior of the scans, as an increase in the number of distinct attackers (i.e., source IP addresses) or an increase in port spoofing, to then spot them as unknown botnets or newly exploited vulnerabilities, even on ports already scanned before.

In our method, we use features representing particular port usages; large packets batches picked at a frequency of several days enable us to profile the evolution of features over time, then statistical measures can spot anomalies in the features time series. A port-based approach may generate a large number of alarms, as for instance, each ephemeral port used in an arbitrary manner would produce an anomaly. Therefore, we adopt a collaborative scheme to ensure that changes in one port are distributed and are not due to random or localized traffic variations. In our approach, called *Split-and-Merge*, local detection modules, geographically split in the network, collect traffic and send anomalies to a central controller in charge of aggregating them, like a Collaborative IDS (CIDS) [57] would do (Fig. 2.3b). The number of false positives can so be significantly reduced as only anomalies detected in several places are taken into consideration. Our contributions differ from existing botnet's detection approaches given the following reasons. First, it targets long-term anomaly detection enabling to detect major changes in the use of ports, and thus underlying botnets. As a matter of fact, current approaches for botnets' detection [49, 20, 55, 61, 62, 60, 17, 56, 57, 58, 14, 19, 64, 63] focus on real-time intrusions and may miss stealthy changes visible at a several days scale. Second, it focuses on destination ports, compared to other approaches [49, 20, 55, 61, 62, 60, 17, 56, 57, 58, 14, 19] aggregating packets per-flow or IP address, which thus are not able to detect scans coming from very distinct source IP addresses and targeting a large variety of destination IP addresses. Third, it leverages several detection modules geographically split in the network, in order to reduce the number of false positives. While most IDSs are localized at a single vantage point [49, 20, 18, 17, 61, 60, 19, 14, 64, 63], we explore a collaborative IDS approach only marginally adopted at the state of the art [55, 56, 57, 58, 62]. Finally, its features are computed over diversity indices, packet size, and TCP flags, using a

change-point detection system, which is not done in [49, 18, 17, 20, 55, 61, 62, 60, 56, 57, 58, 19, 63, 64]. We refer the reader to the specific state-of-the art addressed in Section 2.2.

For our evaluation, we use the MAWI dataset [34] that provides daily traces of a transpacific backbone link. The dataset is restricted to a single Internet Service Provider (ISP), hence corresponds to what could be used at the ISP-level. Differently than the common approach that uses real traces to generate background traffic, we use the MAWI traces as they are, with the aim at detecting real attacks from it, providing a better knowledge of the dataset at the same time. We also use the UCSD Network Telescope dataset that consists of a globally routed, but lightly utilized /8 network prefix. Inbound traffic to non-existent machines is unsolicited and results from a wide range of events, including misconfiguration, scanning of address space by attackers or malware looking for vulnerable targets, and backscatter from randomly spoofed denial-of-service attacks. This way, we are able to compare the anomalies found in both datasets. We present the intrusion detection results against known attacks arisen the last three years, not detected by the MAWILab detection algorithm [108], and we show that we can detect some unknown anomalies as well; in order to classify anomalies, we observe the simultaneous evolutions of features. We experimentally show that our algorithm greatly reduces the number of false positives compared to a single IDS running on the whole dataset. For the sake of reproducibility and further research, our source code is publicly available at [109].

The remainder of this chapter is organized as follows: Section 2.2 surveys the related work. Section 3.2 presents our solution detecting distributed changes in port usages, along with the analysis of its complexity. Section 3.3 introduces the two datasets we leverage for our analysis. In Section 3.4, we present results from the numerical evaluation, highlighting the benefits *Split-and-Merge* can grant in terms of false detection rate and detection accuracy, and also proposing a classification of the noticed anomalies. Finally, Section 3.6 concludes this chapter.

## 3.2 Split-and-Merge Port-centric Network Anomaly Detection

We present our anomaly detection proposal, detailing the reference CIDS architecture and the design of the features.

### 3.2.1 Rationale

We already anticipated some of our key modeling choices: we aggregate traces based on destination ports, in a distributed CIDS setting, and target to design features minimizing the degree of arbitrarity in their choice and interpretation. Our objective is to model the usage of each port, by computing features each time the same day at the same daytime slot. The features characterize the port usage, e.g., if it is mainly targeted by port scan or not, if the hosts are numerous or not, etc. We work on a limited time window over a day, which we assume to represent port usages this day.

In our reference distributed CIDS setting, several *detection module* agents run on different subnetworks so that they can capture each subnetworks' peculiarities and cover the CIDS network context completely. Based on the time evolution of the features of a port, the detection modules detect anomalies and report them to a *correlation module*. Hereafter we detail the different steps of our detection module logic, as well as the anomaly aggregation logic of the correlation module. At each daytime slot, every detection module performs several tasks in a row (each task is then further detailed in the following subsections).

#### Data collection

First, the detection module collects packets in its scope in a single group of $N_{batch}$ elements, and stores packet attributes in lightweight Collection Tables (CTs). For each incoming packet, it identifies the destination port and updates four key-value CTs and a counter for the given port: CT1: unique source IP addresses; CT2: unique destination IP addresses; CT3: unique source ports; CT4: unique size of packets; and Counter: number of SYN packets. Each entry in one CT (e.g., a source IP address in CT1) is associated with a counter of occurrences.

#### Features computation

After data collection, a filter is applied on CTs so that only the ports with at least $N_{min}$ packets stored are kept to be analyzed. For every remaining destination port, the detection module computes some features based on CTs and updates the Features Table (FT) with new values. The FT constantly contains $N_{days}$ entries (we use one day per week in our tests) so that for every new capture, the former value is deleted and the new one added.

#### Anomaly detection

Lastly, the local detection module analyzes the port-specific features time series over $N_{days}$ in order to detect an anomaly with a change-detection algorithm. When an anomaly is spotted, based on a warning threshold $T_i$ on a given feature $i$, an alert is created and transmitted to the central correlation module. The collection and detection parameters resumed in Table 3.1 are to be customized. At the end of the detection process, the correlation module aggregates the alerts received from all detection modules. It is then able to deduce and qualify an attack by noticing the distributed alerts.

| Notation | Definition |
|---|---|
| $N_{batch}$ | Number of packets collected per day |
| $N_{min}$ | Minimum number of packets per port |
| $N_{days}$ | Number of days in the sliding window |
| $T_i$ | Threshold to spot an anomaly for feature $i$ |

Table 3.1: Parameter notations.

### 3.2.2 Features design

To observe an anomaly on a port, looking at the number of packets over time is not sufficient. Indeed, subtle changes in the nature of packets can happen on a port already massively scanned. Therefore, we need to design significant features.

Our features choice is resumed in Table 3.2. *nbPackets* represents the number of packets stored for this port and enables one to see if a port is suddenly massively used. *srcDivIndex* and *destDivIndex* highlight significant variations in the proportion of unique source and destination IP addresses. An increase in *srcDivIndex* may be an attack perpetrated by bots, while its decrease can indicate an attack led by only a few actors. A rise in *destDivIndex* may represent a large number of victims, as a botnet scanning random IP addresses or the whole IPv4 range would cause. *portDivIndex* reflects the diversity in source ports, its diminution may represent the usage of a spoofed port. A variation in the *meanSize* feature suggests a change in packets nature, like crafted packets sent by bots. A variation in the *stdSize* feature can be caused by a change

in packets nature as well, and in addition, is not easy to fool for an attacker: if it increases, the diversity among packets is higher, so probably there are suddenly both crafted and regular packets; if it decreases, the diversity among packets is lower, hence the traffic more specific. This can be caused by malicious software that kills other processes bound to the same port. Finally, a variation in *perSYN* implies an increase or a decrease in the port scan. Therefore each port $p$ at a given day is characterized by the set of features computed from CTs, shown in Table 3.2.

| Feature | Computed from | Description |
|---|---|---|
| *srcDivIndex* | CT1 | % of unique source IP addresses |
| *destDivIndex* | CT2 | % of unique destination IP addresses |
| *portDivIndex* | CT3 | % of unique source ports |
| *meanSize* | CT4 | Mean packets size |
| *stdSize* | CT4 | Standard deviation of packets sizes |
| *perSYN* | Counter | % of SYN packets |
| *nbPackets* | Any CT | Number of packets |

Table 3.2: Features definition.

We denote the time series of feature $i$ containing $N$ days (i.e., $N_{days}$) for port $p$ as $f^p_{i,N} = (x^p_{i,1}, ..., x^p_{i,j}, ..., x^p_{i,N})$, with $x^p_{i,j}$ being the value of feature $i$ for port $p$ on day $j$. Features are computed at a given frequency, set to once every week in the following simulations (in particular the same day at the same daytime slot, in order not to be influenced by weekly or daily variations).

Algorithm 1 below shows how to update the $FT\{ports * features * N_{days}\}$ by computing features by port from packet attributes found in CTs.

**Algorithm 1:** updateFT(CTs, FT, $N_{min}$)

```
Delete first column of FT and shift others
for all port p ∈ ports do
    if length(CT1[p]) > N_min then          ▷ Check condition on the number of packets
        for all att ∈ attributes do
            feature f ← relativeMetric(att)     ▷ 1 or 2 features per attribute (e.g., mean and std for packet size)
            FT[p][f][currentDay] = CTf[p].apply(f)
return FT
```

### 3.2.3 Local anomaly detection

Assuming a feature is more or less likely to vary (standard deviation) depending on its type, and usually around the same (mean) value, the normal distribution logically quite fits as its distribution. The validity of this assumption is assessed later in Section 3.4.1. We model the time series $f^p_{i,N} = (x^p_{i,1}, ..., x^p_{i,N})$ over $N$ days as a normal distribution $\mathcal{N}(\mu^p, {\sigma^p}^2)$ of mean $\mu^p$ and standard deviation $\sigma^p$ such that:

$$\mu^p = \sum_{j=1}^{N} x^p_{i,j} \text{ and } \sigma^p = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(x^p_{i,j} - \mu^p)^2}. \tag{3.1}$$

As previously introduced in Section 2.1.1, the *Z-score* is a well-known simple statistical metric, commonly used to automatically detect a change in the time series. It is the measure of how

many standard deviations below or above the mean a data point is, and the larger the Z-score, the more unusual the value. For the given time series $f^p_{i,N} = (x^p_{i,1}, ..., x^p_{i,N})$ approximated by a normal distribution $\mathcal{N}(\mu^p, \sigma^{p^2})$, the Z-score of the new value $x^p_{i,N+1}$ of feature $i$ at time $N+1$ is computed as follows:

$$Z^p_{i,N+1} = \frac{x^p_{i,N+1} - \mu^p}{\sigma^p}. \tag{3.2}$$

However, the Z-score is computed from the mean, a metric influenced by outliers and especially extreme values. Alternatively, the *modified Z-score* uses the median and the median absolute deviation (MAD) from the median, instead of the classical mean and standard deviation respectively, which makes it outlier-resistant [15].

Given the time series median $\widetilde{f}^p_{i,N}$, the modified Z-score $M^p_{i,N+1}$ of the new value $x^p_{i,N+1}$ of feature $i$ at time $N+1$ is computed as:

$$M^p_{i,N+1} = \frac{0.6745 \cdot (x^p_{i,N+1} - \widetilde{f}^p_{i,N})}{median(|x^p_{i,N+1} - \widetilde{f}^p_{i,N}|)} \tag{3.3}$$

An anomaly is detected if the absolute value of the modified Z-score exceeds a threshold $T_i$. For all $i$, we adopt a threshold value of 3.5 as recommended in [15]. Algorithm 2 presents the anomaly detection process taking place in each local detection module, to detect anomalies from features time series found in FT.

**Algorithm 2:** runDetection(FT, $N_{days}$)

```
median ← 0
mad ← 0                                   ▷ median absolute deviation
mZ ← 0                                    ▷ modified Z-score
list anomalies
for all port p ∈ FT.ports do
    for all feature f ∈ FT.features do
        series ← FT[p][f]
        orderedSeries ← quickSort(series)
        median ← orderedSeries[(N_days+1)/2]
        sum ← 0
        for all value ∈ series do
            sum ← sum + |value - median|
        mad ← sum / N_days
        mZ ← 0.6745 · (series[currentDay] − median) / mad
        if mZ > 3.5 then anomalies.add({p, f})
return anomalies
```

The modified Z-score is used to identify anomalies on all features, except for *nbPackets*: the latter is only used to spot emerging ports, i.e., ports that were not in use before. That is, an anomaly is spotted if at least a given number of packets $N_{min}$ is collected on one port for the first time in $N_{days}$, so that $x^p_{i,N+1} \geq N_{min}$ and $x^p_{i,j} < N_{min}$ for each $j \epsilon [1, N]$.

Once all features of all ports have been analyzed, the detection module sends the content of the anomalies to the correlation module as alerts. For each alert, the module specifies its ID $m$, the anomalous port $p$, the involved feature $i$, the time series $f^p_{i,N}$, and the new anomalous value

$x^p_{i,N+1}$. An alert is so defined by a 5-tuple $\{p, m, i, f^p_{i,N}, x^p_{i,N+1}\}$. For example, in Fig. 3.1, the detection module B notices an anomaly on port 89 for feature *srcDivIndex*. It also provides the time series of feature $f^p_{i,N}$ and $x^p_{i,N+1}$, though not written on the Figure.

Note that in our implementation, the time series are composed of one value picked each week at the same hour. This enables one to avoid variations in the time series due to seasonality, without introducing any additional mechanism that would induce an increased complexity. As a future work, we could also refine our methodology so that it also takes into consideration trend variations.

### 3.2.4 Central correlation

The correlation module receives low-level alerts from all detection modules. The distinction between localized (noticed in one subnetwork) and distributed (noticed in several subnetworks) alerts is made here. As we are searching for distributed attacks, the correlation module groups the low-level alerts to keep only the ones reported by at least $k$ subnetworks; we set $k = 2$ in this work. In the example of Fig. 3.1, several detection modules send alerts to the correlation module; among them, two subnetworks report a change in the *portDivIndex* feature on port 23. Hence the correlation module induces an anomaly on this port. It is even better if similar anomalies have been noticed on the same port for several features.

We define the *Anomaly Score* ($AS$) as the number of anomalies noticed for one port by all monitors and for all features; e.g., if for one port, a monitor detects anomalies on two features and another on six features, the $AS$ is 8. The correlation module is able to compute the $AS$ after having received alerts from all monitors during the same time slot. When it identifies top-level anomalies, it warns all detection modules about the anomalous ports. Thus they are able to analyze these ports as a priority next time. Ad-hoc actions can also be taken, as a function of the programmability of the local network, such as port blocking, mirroring, deep-packet-inspection, for the sake of reporting in a possible further detailed analysis.

## 3.3 Network traffic datasets

The WIDE project provides researchers with daily traces of a transpacific link, named the MAWI archive [34]. Traces are collected between their network and the upstream ISP. Each file contains 15 minutes of traffic flows, captured between 14:00:00 and 14:15:00 local time. This represents usually between 4 and 10 GB of traffic for one file. Before being released, traces are anonymized so that no personal information can be extracted. Specifically, the application data is removed and IP addresses are scrambled with a modified version of *tcpdpriv* following two principles: 1) it is collision-free so that there is a one-to-one mapping between IP addresses before and after anonymization; 2) it is prefix-preserving so that if two IP addresses share $k$ bits before anonymization, the two anonymized IP addresses will also share $k$ bits. This enables one to retrieve the subnetworks after anonymization.

In addition, the Center for Applied Internet Data Analysis (CAIDA) provides the UCSD Network Telescope dataset [32]. It consists of a globally routed, but lightly utilized /8 network prefix, that is, 1/256th of the whole IPv4 address space. It contains a few legitimate hosts; inbound traffic to non-existent machines - so-called Internet Background Radiation (IBR) - is unsolicited and results from a wide range of events, including misconfiguration, scanning of address space by attackers or malware looking for vulnerable targets, backscatter from randomly spoofed denial-of-service attacks, and the automated spread of malware. CAIDA continuously captures this anomalous traffic discarding the legitimate traffic packets destined to the few reachable IP

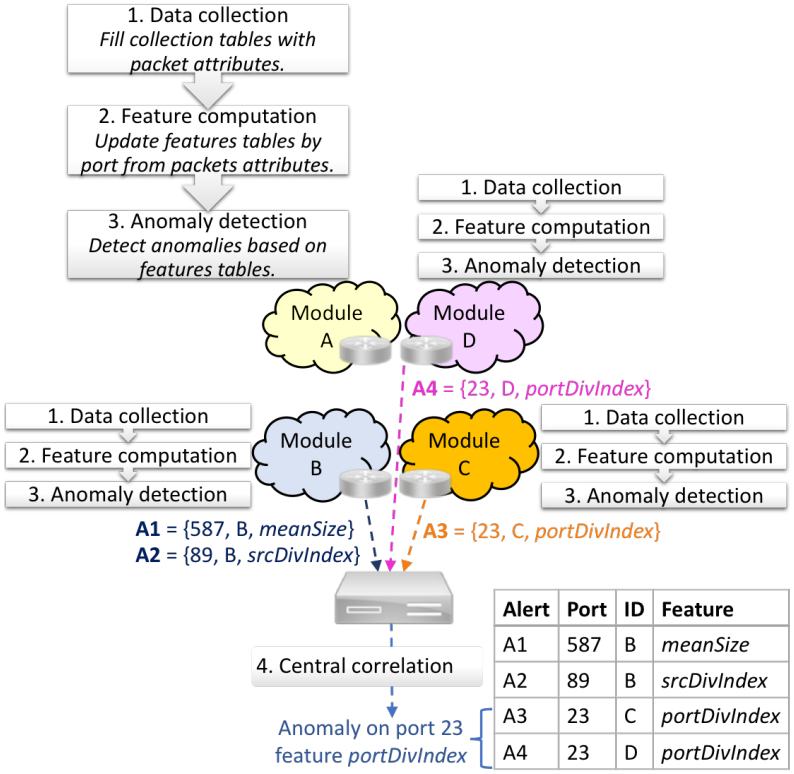


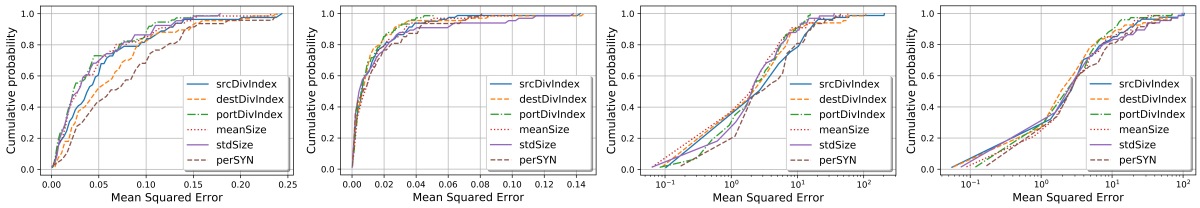

Figure 3.1: Architecture example. Local modules run at different points in the network and send alerts to the central correlation module. The controller will then be able to keep only distributed ones. Here it spots an anomaly on port 23 for feature *portDivIndex*, coming from two different places.

addresses in this prefix. They provide two types of files: raw data stored into pcap files for the ongoing month, and hourly FlowTuple files for the last 13 years – that we used. Note that this dataset does not contain several subnetworks as the MAWI one does. Thus the number of false positives (i.e., detecting an attack targeting only this subnetwork but not the whole Internet) may be higher. The objective of using this second dataset is to evaluate the efficiency of our method as of detected botnets and, most of all, to compare the anomalies found in each of the datasets and to study the common ones.

Note that by default, we refer to the MAWI dataset when we do not specify which one we use. As the UCSD dataset only contains one vantage point, we do not use it for all experiments, instead we use it to cross-check the anomalies found in the MAWI dataset since there is no ground-truth.

## 3.4 Evaluation

In this section, we evaluate the performance of the *Split-and-Merge* detection process using real traffic traces. First, we aim to validate our assumption that the feature data is normally distributed around its median. We also analyze the results to adequately determine the features and parameters. Finally, we look at the anomalies found during the last three years and aim to classify them. The source code used for the detection and evaluation is available in [109].




(a) Normal distribution based on mean.

(b) Normal distribution based on median.

(c) Log-normal distribution based on mean.

(d) Log-normal distribution based on median.

Figure 3.2: Empirical CDF of the MSE between the true distribution and the regression.

### 3.4.1 Normal distribution fitting

It is important to empirically assess the validity of a key *Split-and-Merge* assumption, which is that the features data is expected to be normally distributed around the median. Indeed, we use the modified Z-score to support the detection logic.

It is well known from the state of the art that Internet traffic exhibits a power-law behavior [110] for the packet counts. Among *Split-and-Merge* features, we consider the *nbPackets* feature only to characterize ports' behavior (for the ports with sufficient traffic). Moreover, other features represent diversity indices, attributes means, and standard deviations.

To assess the assumption that normal distribution is a well fit for the *nbPackets* feature, and that it is better than the power-law distribution, we compute the mean square error (MSE) between the measured and synthetically generated histogram [111], for each tuple of port and feature so that:

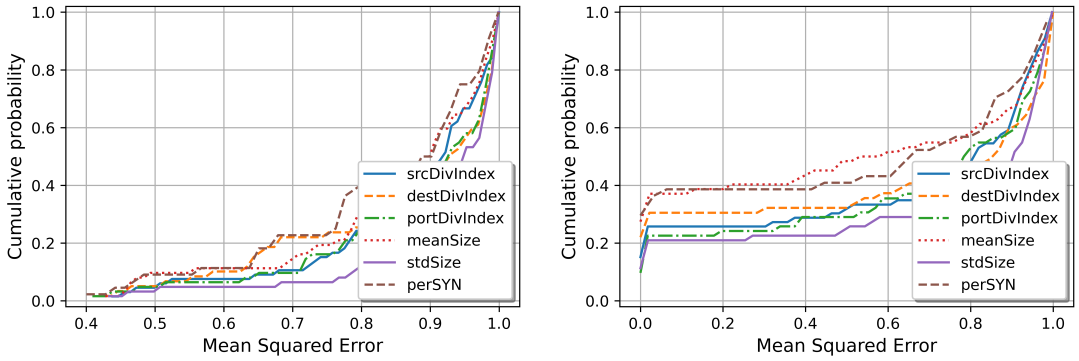
$$MSE = \frac{1}{N_{bins}} \sum_{b=1}^{N_{bins}} [H^p_{i,N}(b) - \widehat{H^p_{i,N}}(b)]^2 \tag{3.4}$$

where $H^p_{i,N}$ denotes the normalized histogram of the $N$ days time series $f^p_{i,N}$ of feature $i$ and port $p$, $\widehat{H^p_{i,N}}$ the histogram with matching mean and standard deviation, and $N_{bins}$ the number of bins in the histograms. The latter is chosen according to Sturges' rule stating that the number of bins $K$ should be equal to $K = 1 + 3.322(\log_{10}(N))$ with $N$ the number of samples. Using this method, we used 4 bins for 10 samples.

Given the several thousands of ports to analyze each day for each feature, the Cumulative Distribution Function (CDF) represents the cumulative probability for one feature to reach a given MSE by taking into account all ports. We plot the empirical CDF of the MSE by considering four different regressions: a normal distribution with matching mean and standard deviation in Fig. 3.2a, a normal distribution with matching median and median absolute deviation in Fig. 3.2b, a log-normal distribution with matching mean and standard deviation in Fig. 3.2c, and a log-normal distribution with matching median and median absolute deviation in Fig. 3.2d. The reported results are those for 2016 traffic (we observe similar results in 2017 and 2018).

We observe that: *(i)* the regression using the log-normal distribution gives far worse results than the normal distribution; *(ii)* for the normal distribution, the regression using the median and the median absolute deviation gives a better approximation than the one with the mean and the standard deviation; *(iii)* for the normal distribution, all features produce more or less the same MSE.

By using a normal distribution, we found out that the MSE is very low for all features, which is an empirical validation of this assumption.




(a) Normal distribution based on mean.

(b) Normal distribution based on median.

Figure 3.3: Empirical CDF of the chi-squared test between the true distribution and the regression.

In addition to this approach comparing the MSE of both distributions, the chi-squared test is commonly used to test association of variables in two-way tables where the assumed model of independence is evaluated against the observed data. It is used to test whether observed data differ significantly from theoretical expectations. Fig. **??** shows hereafter the p-value for the chi-squared distribution: *(i)* between the real distribution and the normal distribution based on mean and standard deviation (Fig. **??**), and *(ii)* between the real distribution and the normal distribution based on median and median standard deviation (Fig. **??**).

From these figures we can draw the same conclusions than for those using the MSE, that is: *(i)* the normal distribution based on median is fitter than the one based on mean, and *(ii)* all features have more or less the same p-value.

### 3.4.2 Local anomaly detection

This section gives the outcome of several local detection modules running simultaneously, each of them being situated in a MAWI subnetwork. We pick each Thursday from March 31 to Oct. 20, 2016. Thresholds $T_i$ for an anomaly are all set to 3.5. The minimum number of packets $N_{min}$ is set to 20. The number of days we chose is $N_{days} = 10$. We will tune these two last values later in Section 3.4.6.

Fig. 3.4 gives an example of the modified Z-score evolution for the *srcDivIndex* feature on port TCP/3389. On Sept. 29, the absolute value of the modified Z-score is over the threshold for four detection modules situated in different subnetworks, resulting in an anomaly. The subnetwork F contains only a few points because most of the time, there is little (fewer packets than $N_{min}$) or no traffic on port 3389 in this subnetwork. The same explanation applies to subnetworks that do not appear at all in the legend.

### 3.4.3 Comparison between aggregated and split views

In this experiment, we compute the number of alarms for each feature considering two approaches: *(i)* an aggregated view where an anomaly is observed considering the traffic from all subnetworks aggregated, *(ii)* a split view where only distributed anomalies (i.e., seen in at least two subnetworks) are conserved. The results are presented in Table 3.3 for 2016, while similar findings have been observed in 2017 and 2018. We observe that the number of anomalies – thus the number of false positives – is significantly lower with the split view.

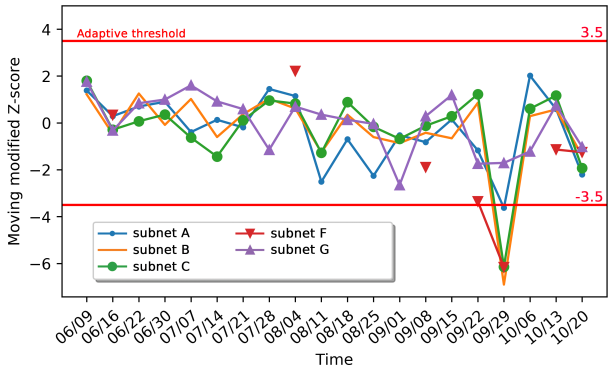



Figure 3.4: Evolution of the modified Z-score in 5 subnetworks for feature *srcDivIndex* on port 3389 over time (2016).

| Feature | Aggregated view | Split view |
|---|---|---|
| *srcDivIndex* | 11,376 | 101 |
| *destDivIndex* | 11,409 | 96 |
| *portDivIndex* | 11,375 | 102 |
| *meanSize* | 10,978 | 91 |
| *stdSize* | 10,549 | 67 |
| *perSYN* | 851 | 98 |

Table 3.3: Number of anomalies for both approaches (2016).

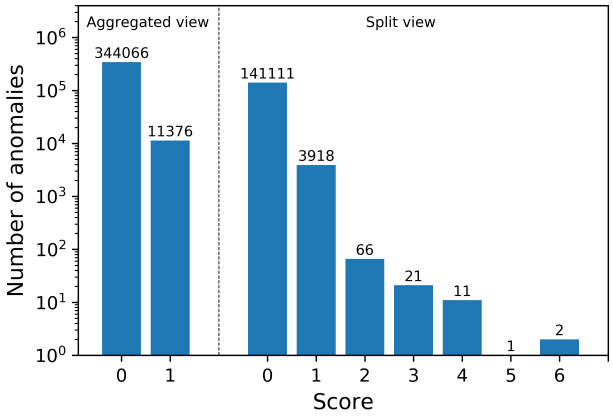



Figure 3.5: Number of anomalies for feature *srcDivIndex*. In aggregated view, the score is 1 if there is an anomaly on the whole traffic, else 0. In split view, it is the number of anomalous subnetworks.

The example of feature *srcDivIndex* is shown in Fig. 3.5, with the number of anomalies expressed in logarithmic scale. With a split view, we observe that considering only distributed alerts considerably diminishes the number of anomalies to deal with. Indeed, the number of anomalies for a single variation (score of 1) is 100 times higher than for a distributed variation (score of 2), decreasing the number of alerts from 3,918 to 66.

### 3.4.4 Last years panorama

In this subsection, we launch the anomaly detection process on a large period to examine the type of anomalies we can detect. We describe hereafter the main anomalies arisen these last three years, in the MAWI dataset and the UCSD dataset.

#### In the MAWI dataset

Fig. 3.6a (2016), 3.6b (2017), and 3.6c (2018) show the number of ports with a given anomaly score each day, highlighting the main anomalies arisen these last three years. In all cases, we observe very few alarms each day, which is quite convenient for the network administrator, as too numerous alerts is considered as one reason why IDS are underused. Furthermore, none of these anomalies have been detected by MAWILab. Also, we tag events observed in both datasets with a red frame in Fig. 3.6. Note also that lots of ports score 5 anomalies, as a significant variation on one port in one subnetwork generates simultaneous alerts for all features (except for the feature *perSYN* that gives poor results as shown later in Section 3.4.6), i.e., 5 alerts. We therefore describe the main anomalies, whose anomaly score is the highest, and we also indicate if we retrieved these anomalies in the UCSD dataset.

**2016 period.** Eight noticeable scores appear in Fig. 3.6a depicting this first period.

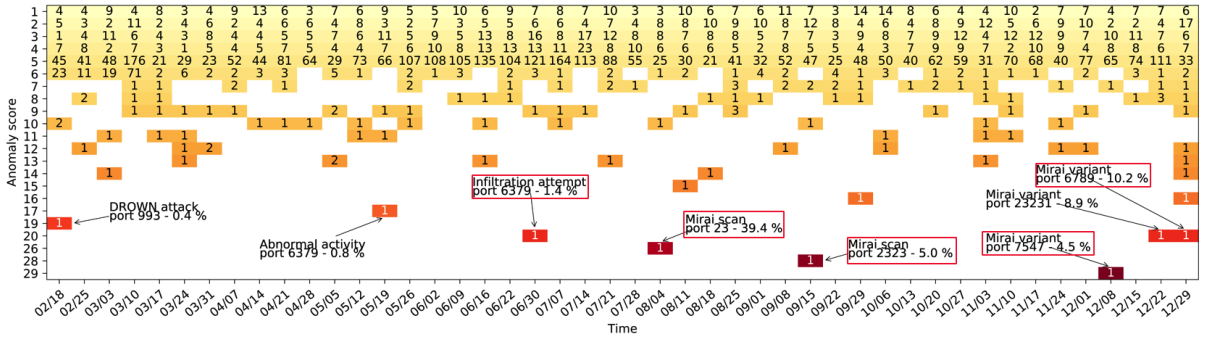
*i)* The 19-score on Feb. 19 is a scan prior to the DROWN attack [112], exploiting a vulnerability in Secure Sockets Layer version 2.0 (SSLv2) (CVE-2016-0800).

*ii)* The 17-score on May 19 corresponds to an exploit on port 6379 Redis, an in-memory key-value store used as a database or a cache. This day, numerous IP addresses with different source port numbers targeted this port. It could be a botnet or numerous different hosts scanning for vulnerable devices. Indeed, Redis servers do not require authentication by default and therefore are easy victims of this type of scan. Also, this happens only a few days after buffer overflow vulnerabilities were discovered, leading to arbitrary code execution (CVE-2016-8339, CVE-2016-10517).

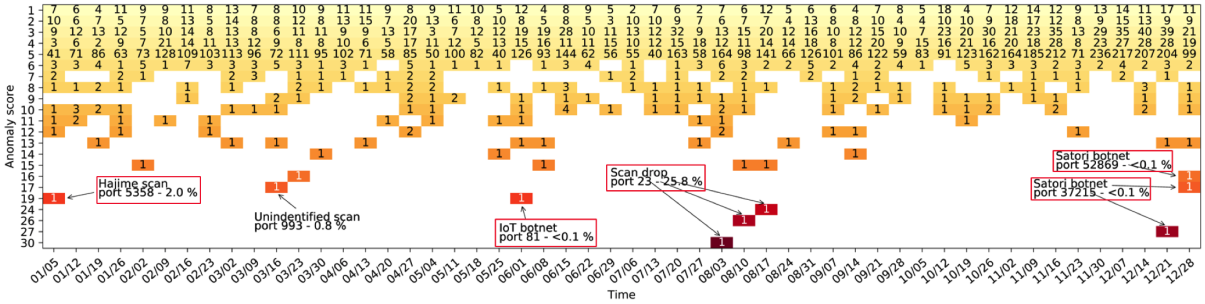
*iii)* Once again, the 20-score on June 30 corresponds to an exploit on port 6379. This day, a large SYN scan is observed coming from the same source IP address and targeting numerous hosts in several ASes of MAWI. This is either a large scan targeting the whole IPv4 space, through a tool like ZMap [113] that performs Internet-wide network scans in under 45 minutes, or someone trying to penetrate the MAWI network. This anomaly has been detected with a score of 4 in the UCSD dataset.

*iv)* The IoT Mirai botnet [2] is a major attack arisen in 2016. First, Mirai infected hosts send TCP SYN packets to random IP addresses on Telnet ports 23 and 2323, except those on a blacklist. Hosts whose Telnet port is open send back an SYN/ACK packet. Then, infected hosts try to establish a Telnet connection to them using a hard-coded list of credentials, and send the credentials to another server if it is successful. From there, a separate program executes architecture-specific malware. The victim is now infected and listens for attack commands from the C&C server, then starts scanning to infect other hosts. This is how Mirai spread into connected objects and form a worldwide army of bots. Indeed, the 26-score on Aug. 4 corresponds to the Mirai scan on port 23 and the 28-score on Sept. 15 relates to port 2323. Numerous Mirai variants exploit vulnerabilities on other ports later in 2016, as evidenced by the 29-score on Dec. 8 on port 7547, the 20-score on Dec. 22 on port 23231, and the 20-score on Dec. 29 on port 6789. The attack has been detected in the UCSD dataset with a score of 4 for port 2222, of 5 for ports 23 and 7547, and of 6 for ports 2323 and 6789.

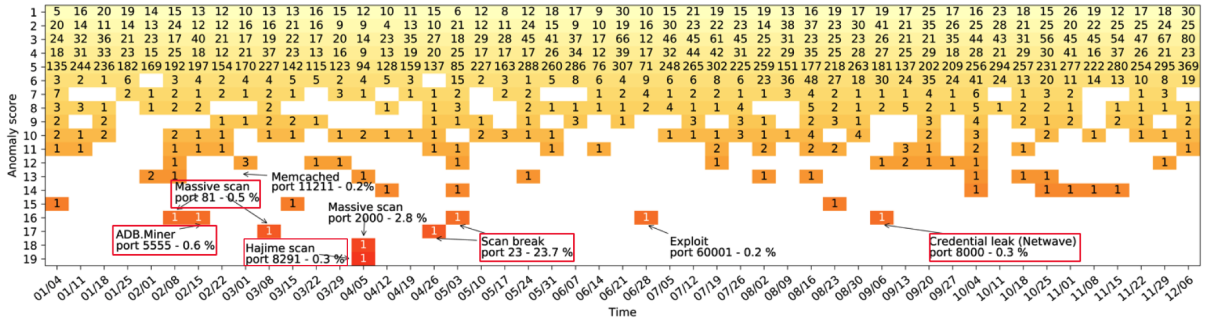
**2017 period.** Nine noticeable scores appear in Fig. 3.6b showing anomaly detection results in 2017.




(a) In 2016.




(b) In 2017.




(c) In 2018.

Figure 3.6: Anomaly scores for the MAWI dataset (number of anomalies for one port, taking into account all features and all monitors). In the coloured squares are given the numbers of ports with this anomaly score this day. Events detected in both datasets are tagged with a red frame.

*i)* A 19-score on Jan. 5 highlights a scan on port 8291, carried out by Hajime bots. Hajime [114] is an IoT worm revealed only a few days after the release of the source code for Mirai. The botnet is continuously evolving, taking advantage of newly released vulnerabilities. At the beginning of 2017, the efficient SYN scanner implementation scans for open ports 5358 (WS-DAPI) [115]. In the same way Mirai did on port 23, the extension module tries to exploit the victim using brute-force shell login. This anomaly has been detected with a score of 6 in the UCSD dataset.

*ii)* The 17-score on March 16 corresponds to a massive scan on port 993 which deals with secure IMAP (IMAPS). This day, a massive scan coming from the same IP address from port 993 as well has been observed. It may be a ZMap scan, or an attacker trying to infiltrate the MAWI network specifically. Indeed, IMAPS belongs to the list of ports scanned by default by

Nmap [116] and is permanently stuck with the vulnerabilities in SSL 3.0.

*iii)* The June 1 a 19-score is observed, corresponding to a new IoT botnet spreading and exploiting a vulnerability in security cameras [117], several days after the researcher Pierre Kim released a vulnerability analysis report on GoAhead and other OEM cameras. This anomaly has been detected with a score of 6 in the UCSD dataset.

*iv)* The 30-score, 26-score, and 24-score, respectively on Aug. 3, Aug. 10, and Aug. 17 correspond to a sensible drop in the scan perpetrated by Mirai on port 23. This may be due to the Internet of Things (IoT) Cybersecurity Improvement Act of 2017 [118], adopted on Aug. 1, 2017. The latter seeks to improve the security of internet-connected devices, so that devices do not contain any known security vulnerabilities and are conceived using standard protocols. It also claims that the eventual patches should be applied even retroactively, which can explain the scan drops. These changes have also been noticed in the UCSD dataset, producing anomalies with scores of 4 and 5.

*v)* The 27-score on Dec. 21, and 15-score and 16-score on Dec. 28 involve ports 37215 and 52869, that received numerous scans from the newest version of Satori (a Mirai variant) [119]. These anomalies have been detected in the UCSD dataset with scores of 4 and 5.

**2018 period.** Nine noticeable scores appear in Fig. 3.6c depicting the 2018 year. Compared to previous years, large scores are much rarer this time. Indeed, the maximum $AS$ is up to 19, compared to 29 and 30 respectively in 2016 and 2017. We can still identify several main anomalies.

*i)* On Feb. 8 and March 8, exploits on port 81 are noticed. These days, almost the same IP address launched TCP SYN scans from the same source port number, targeting numerous MAWI subnetworks. This may be once again an Internet-wide network scan (e.g., by ZMap) or an attacker that targets the MAWI dataset specifically.

*ii)* The 16-score on Feb. 15 is actually a scan on port 5555. It comes from the ADB.Miner botnet, which identifies Android devices with Android Debug Bridge turned on, to control them and make them execute commands [120]. Hence, this day, numerous IP addresses sent SYN packets to various hosts in the MAWI network using different source port numbers, as seen for the Mirai botnet in 2016. This anomaly has been detected in the UCSD dataset with a score of 4.

*iii)* The 12-score on March 1 corresponds to the memcached attack on port 11211. Prior to the huge DDoS attack towards Github, large TCP SYN scans across the world targeted port 11211 in order to identify memcached servers exposed without any authentication protection (CVE-2018-1000115). The anomaly observed this day is a mark of this large scan. These changes have also been noticed in the UCSD dataset, producing anomalies with scores of 4 and 5.

*iv)* The 18-score on Apr. 5 corresponds to a large scan on port 2000 coming from various source IP addresses with different source port numbers, and targeting many IP addresses from several ASes in the MAWI dataset. Cisco Skinny Call Control Protocol (SCCP) is often bound to this port, allowing terminal control for voice over IP. This scan is symptomatic of an IoT botnet, willing to exploit the few vulnerabilities disclosed last years for this protocol, and maybe IoTroop [121].

*v)* The 19-score on Apr. 5 highlights a scan on port 8291, carried out by Hajime bots. In May 2018, it exploits a vulnerability (CVE-2018-7445) published 13 days before. First, infected hosts scan random IP addresses on port 8291 to identify MikroTik devices. Once the bot has identified one device, it tries to infect it with a public exploit package sent via port 80 or an alternate port. If successful, the device infects new victims in turn under the same protocol. This day, as for the Mirai botnet, our program saw many IP addresses targeting the MAWI network on port 8291, using various source port numbers. This anomaly has been detected with a score

of 4 in the UCSD dataset.

*vi)* On Apr. 26 and March 3, two anomalies on port 23 are detected. We observe that these days, *meanSize* considerably rises while *srcDivIndex* and *destDivIndex* fall. The number of packets is also lower than usual. Thus it looks like there are fewer malicious scans towards this port these days. Actually, botnets tend to use alternate ports because vulnerabilities are progressively patched and devices are armed against possible exploits on port 23. These changes have also been noticed in the UCSD dataset, producing anomalies with scores of 4 and 5.

*vii)* On June 28, a 16-score is stored for port 60001, probably corresponding to a Trojan named Trinity. It first connects to one of 11 IRC servers on UnderNet. The Trojan then joins a chat room and waits for commands to attack individual agents on the channel. The noticed anomaly probably results from that trojan, that we identified coming from two different IP addresses.

*viii)* On Sept. 6, a 16-score anomaly is observed on port 8000. Several days before, the possibility for an unauthenticated attacker to exfiltrate sensitive information about the network configuration (network SSID and password) has been made possible by an information disclosure in Netwave IP camera (CVE-2018-11653 and CVE-2018-11654). This anomaly has been detected in the UCSD dataset with a score of 4.

#### In the UCSD Network Telescope dataset

Fig. 3.7a (2016), 3.7b (2017) and 3.7c (2018) show the number of ports with a given anomaly score each day, highlighting the main anomalies found these last years in the UCSD dataset. Once again, we frame common anomalies (found in both datasets) in red, and describe the anomalies detected uniquely in this one hereafter.

**2016 period.** Several noticeable scores appear in Fig. 3.7a showing anomaly detection results in 2016.

*i)* On Feb. 18, we observe many scans targeting destination ports from 36242 to 36560 (even numbers only), producing 6-score anomalies. The scans probably come from a botnet, because the source and destination IP addresses are very diverse. It is hard to determine the nature of these scans, because they target registered ports with no known vulnerabilities. It may be a stealthy scan technique to first determine if the host is up, to then focus on some ports showing vulnerabilities (as the Reaper botnet does, cf. [122]).

*ii)* On March. 17 and 24, a decrease in the scans targeting the range 36242-36560 is observed, producing 6-score anomalies.

*iii)* Once again, several scans targeting the range 19328-19454, producing 6-score anomalies have been noticed on Dec. 1.

*iv)* On Dec. 8, an anomaly is detected with a score of 5, highlighting a scan performed by Mirai on port TCP/2222, running Rockwell Automation ControlLogix whose several vulnerabilities are known [123]. The MAWI dataset contains traffic from a backbone link, and port 2222 may also be used as an alternative port for SSH, producing noise in data. This may explain why we did not detect the scan in the MAWI dataset.

**2017 period.** Several noticeable scores appear in Fig. 3.7b showing anomaly detection results in 2017.

*i)* A 6-score anomaly on Jan. 5 highlights a scan on port TCP/27017 that runs MongoDB. A few days later, it has been reported and confirmed that many unsecured MongoDB databases have been scanned and vandalized around the world [106]. This attack affects only those databases which maintain default configurations, which leaves the database open to external connections via the Internet. We observe also a scan of the same nature on Feb. 16.

*ii)* We observe on March. 30 several scans targeting destination ports 88, 443, 3389, 6666, and 54313, coming from very distinct source IP addresses and with no flags (a network scanning technique known as Inverse TCP Flag Scanning). All of these ports present known vulnerabilities. A vulnerability on port 54313 to exploit a Netis Router Backdoor has been detected and exploited by botnets back in Aug. 2016[124]. We did not find any information about a botnet targeting all these ports conjointly in the literature, thus we assume it is a small-scale attack.

*iii)* A 6-score on May. 18 highlights a scan on port TCP/445 that runs Server Message Block (SMB) known for its "EternalBlue" vulnerability. On May. 17, the recent WannaCry ransomware takes advantage of this vulnerability to compromise Windows machines [125].

*iv)*A 6-score on Oct. 12 highlights a scan on port 20480, probably from the Reaper botnet targeting a sequence of destination ports including 20480 [122]. However, instead of doing aggressive, asynchronous SYN scans for open Telnet ports, Reaper performs a more elaborate, conservative TCP SYN scan on a series of different ports, one IP at a time, targeting uncommon ports. Only after the first wave of scans on the victim, a second wave starts consisting of potential IoT web service ports: 80, 81, 82, 83, 84, 88, etc.

*v)* As in 2016, we observe a decrease in the scans targeting the range 19328-19454, producing 6-score anomalies on Dec. 7 and 14.

**2018 period.** Several noticeable scores appear in Fig. 3.7c showing anomaly detection results in 2018.

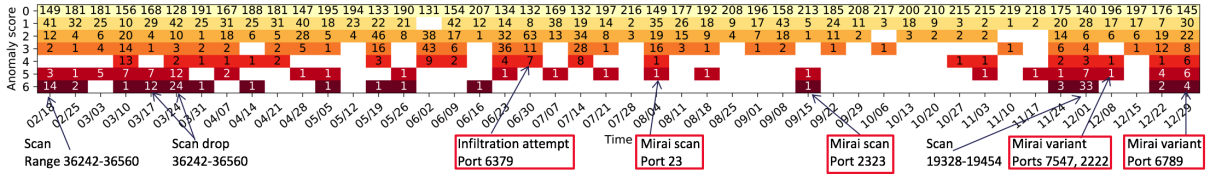
*i)* A 6-score on May. 17 highlights a scan on port TCP/8000, performed by the Satori botnet exploiting a buffer overflow vulnerability, tracked as CVE-2018-10088 [126]. The exploit could be used by remote attackers to execute arbitrary code by sending a malformed package via ports 80 or 8000. The upsurge of malicious scanning activity has been observed on June 15 but we could think that we detected a preliminary scan campaign from Satori developers. Moreover, this scan was probably not targeting the whole IPv4 range because we did not detect it in the MAWI dataset.

*ii)* A 6-score on May. 17 highlights a scan on port TCP/445, a port known for its "EternalBlue" vulnerability exploited by Wannacry. We noticed at this time an important increase in the botnet scan activities.

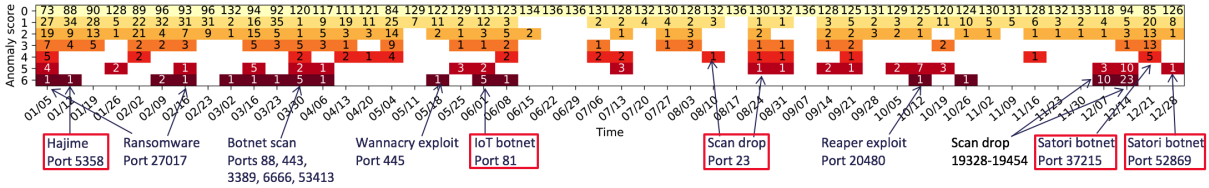
*iii)* A 6-score on Aug. 2 highlights a scan on port TCP/37215, a port used by Huawei HG532 routers. At the end of July, an IoT hacker identifying himself as "Anarchy" claimed to have hacked about 18000+ Huawei routers [127]. It works by exploiting an already known vulnerability which CVE is 2017-17215, used in Satori.

### 3.4.5 Anomaly score distribution

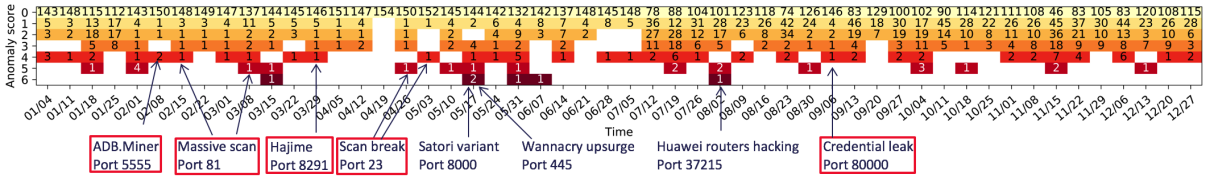
The last subsection shows that there are only a few emerging anomalies, and a lot more with small anomaly scores. We can so wonder which anomalies are greater enough to be analyzed. Fig. 3.8 shows the mean anomaly score occurrence per day in logarithmic scale for each year. For the MAWI dataset (Fig. 3.8a), for 6 features, we observe different levels of 6 items, e.g., $AS$ from 1 to 6 are close to 10, then from 7 to 12 close to 0.3, then the number of anomalies progressively declines. Once again, the occurrence $AS = 5$ is high because a given variation on one port in one subnetwork generates one anomaly per feature, except for *perSYN*. For next evaluations, we set the threshold to define an anomaly either to 6 or 12, depending on if we want to detect a large variety of anomalies or only significant ones. For the UCSD dataset (Fig. 3.8b), we observe a very low number of significant anomalies, i.e., with a score at least equal to 4. For the MAWI dataset, we observed a decrease starting from $AS = 7$ (thus at least two impacted subnetworks) and not 4, probably because it contains more false positives due to traffic generated by internal




(a) In 2016.

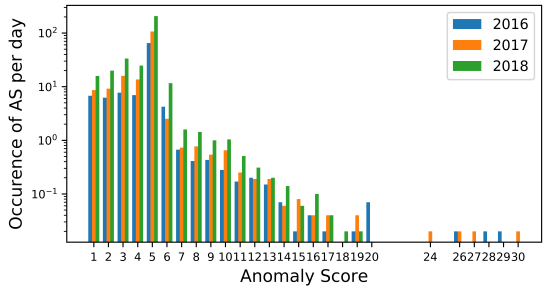



(b) In 2017.

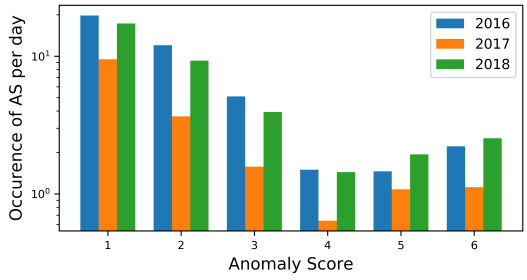



(c) In 2018.

Figure 3.7: Anomaly scores for the UCSD Network Telescope dataset (number of anomalies for one port, taking into account all features and all monitors). In the coloured squares are given the numbers of ports with this anomaly score this day. Events detected in both datasets are tagged with a red frame.




(a) For the MAWI dataset.




(b) For the UCSD dataset.

Figure 3.8: Occurrences of each anomaly score during the last three years, showing a low number of significant anomalies.

hosts (while there is no such traffic in the UCSD dataset).

### 3.4.6 Features and parameters choice

In this subsection, we aim to refine the detection accuracy by analyzing how the features and the parameters impact the results.

### Feature selection

We use two metrics to determine which features are the most useful in the anomaly detection process. The two following experiments are made on the whole traffic from 2016 to 2018. For both, a large number of anomalies, i.e., with an $AS > 6$ is first considered, then only major ones, i.e., with an $AS > 12$.

**F-Test based feature selection:** F-Test is a statistical test used to compare between models, here between the input (features) and the output (anomaly detection results). It is useful in feature selection as we get to know the significance of each feature in improving the model. First, the anomalies are generated by launching our solution, and only the ones whose $AS$ is higher than $T$ are kept. Then, we observe the impact of each variation of features on the detected anomalies. Table 3.4 gives the results of the F-Test applied to the produced anomalies, for each feature variation. We observe that the *perSYN* feature, which scores 0.114 / 0.009 and 0.152 / 0.039 respectively for an increase and a decrease, has the least impact on the results.

| **Feature** | **F-Test score ($T = 6$)** | **F-Test score ($T = 12$)** |
|---|---|---|
| *+srcDivIndex* | 0.269 | 0.136 |
| *-srcDivIndex* | 0.689 | 1.000 |
| *+destDivIndex* | 0.599 | 0.378 |
| *-destDivIndex* | 1.000 | 0.610 |
| *+portDivIndex* | 0.633 | 0.188 |
| *-portDivIndex* | 0.648 | 0.899 |
| *+meanSize* | 0.321 | 0.158 |
| *-meanSize* | 0.932 | 0.463 |
| *+stdSize* | 0.692 | 0.302 |
| *-stdSize* | 0.478 | 0.825 |
| *+perSYN* | 0.114 | 0.009 |
| *-perSYN* | 0.152 | 0.039 |

Table 3.4: F-Test scores per feature, among anomalies whose $AS$ is higher than $T$. The higher the F-Test score of a feature is, the greater impact the feature has on final results.

**Number of absolute variations**: we analyze for each anomaly whether the variations of features are distributed in several subnetworks, i.e., if the feature only rises or decreases in several subnetworks. This kind of variation is wanted as it means a distributed change, and not some random, differing variations. In Table 3.5 is given the number of absolute anomalies, i.e., where one given feature is only rising in several subnetworks, or decreasing in several subnetworks. We observe that the *perSYN* feature contains a majority of random variations, contrary to other features.

We can induce from these two experiments that the *perSYN* feature has the lowest impact on final results and does not often rise or decrease in the majority of subnetworks. This observation is interesting as this feature is widely used in TCP SYN scan detection algorithms.

### Parameters tuning

Our algorithm has four parameters (see Table 3.1). In this subsection, we evaluate and discuss the impact of the key ones: the window size ($N_{days}$) and the minimum number of packets on one port ($N_{min}$).

| Feature | # absolute variations ($T = 6$) | # absolute variations ($T = 12$) |
|---|---|---|
| *srcDivIndex* | 318 | 12 |
| *destDivIndex* | 296 | 17 |
| *portDivIndex* | 334 | 16 |
| *meanSize* | 260 | 14 |
| *stdSize* | 211 | 7 |
| *perSYN* | 154 | 6 |

Table 3.5: Number of absolute variations per feature (either rise in several subnetworks, or decrease in several subnetworks), among anomalies whose $AS$ is higher than T.

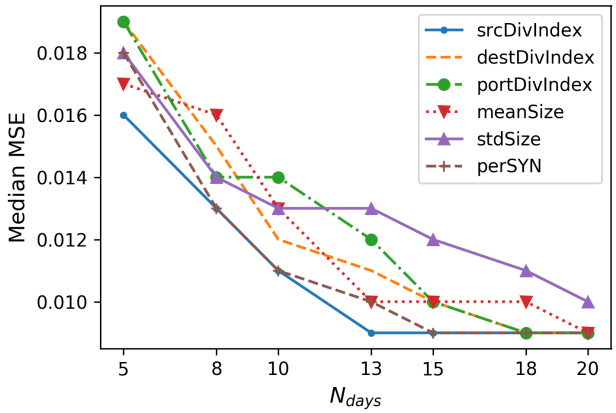
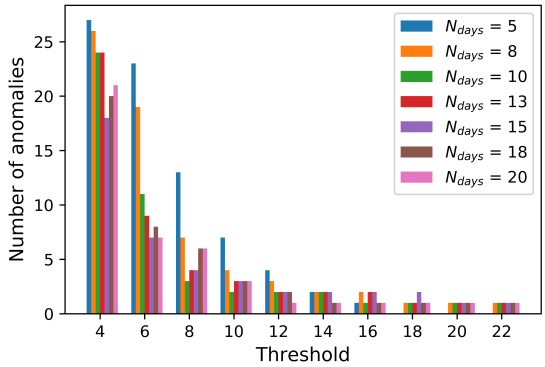
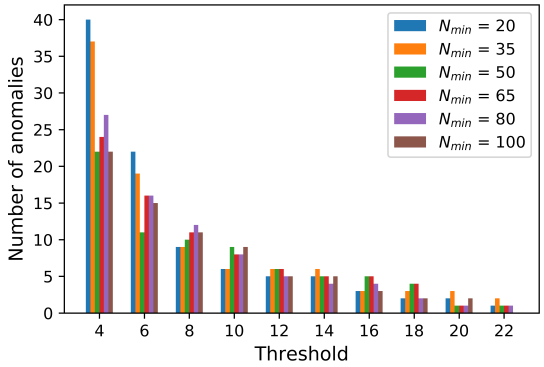



Figure 3.9: Median MSE (between the true and the normal distribution) per feature, with the window size ($N_{days}$) varying.




(a) Varying the number of days in the model ($N_{days}$).




(b) Varying the minimum number of packets on one port ($N_{min}$).

Figure 3.10: Number of anomalies depending on a parameter for diverse thresholds (2016).

**Number of days in the window ($N_{days}$)**: first, we wonder how the number of elements in the time-window impacts the validity of a normal distribution based on the median. For each feature, we compute the median MSE (for all ports) between the true distribution and the normal distribution based on the median, by making $N_{days}$ vary. The result is shown in Fig. 3.9. We observe that the more days in the model, the more normally distributed the set of data, making the model with 20 days the best approximation. However, it may be possible to find a trade-off between a good approximation and a minimized time execution and memory allocation.

Looking at the anomaly detection results can also give evidence about the appropriate window size. The objective is to find an optimal window size that is not too large, but still gives a satisfactory detection accuracy. In Fig. 3.10a is shown the number of anomalies in 2016 depending on the threshold (minimum $AS$ value) for diverse $N_{days}$ values. We observe that 10 days (in green) enable us to see all substantial anomalies, i.e., with an $AS$ higher than 12. We keep $N_{days} = 10$ for the following simulations to minimize the window size.

**Minimum number of packets on one port ($N_{min}$)**: this time, we show the number of anomalies in 2016 depending on the threshold (minimum $AS$ value) for diverse $N_{min}$ values in Fig. 3.10b. We observe that for a threshold higher than 12, the number of anomalies is the same no matter $N_{min}$. It means that anomalies happen most of the time on ports with a sufficient amount of traffic (more than 100 packets). Therefore we choose $N_{min} = 100$ to limit the number of ports to deal with, hence reducing the time execution and memory allocation.

To complete the study, Fig. 3.11 shows the number of ports to deal with in each subnetwork in 2016, by making $N_{min}$ vary. With $N_{min} = 100$, the maximum amount of ports to analyze is around 6,000 for subnetwork E, which is quite reasonable.

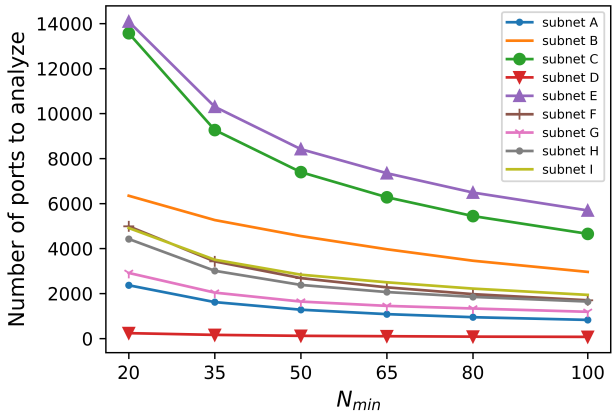



Figure 3.11: Number of ports to analyze in every subnetwork, with the minimum number of packets per port ($N_{min}$) varying.

### 3.4.7 Anomalies classification

We define a set of commonly-called 'expert rules' with some conditions about the evolution of the features. For each characteristic, the variation has to be noticed in at least two subnetworks, while the opposite variation must not be noticed in any subnetwork. For example, if the characteristic is *-srcDivIndex*, an anomaly is verified if *srcDivIndex* decreases in at least two subnetworks (*-srcDivIndex* $\geq 2$) and if it does not rise in any subnetwork (*+srcDivIndex* $= 0$). Furthermore, the feature *portDivIndex* is not self-speaking and thus not used in the classification. Nonetheless, this feature gives additional information about the likelihood for the source port to be spoofed or not. Therefore, if *portDivIndex* rises in at least two subnetworks while it does not decrease in any subnetwork, then the source port may be randomly chosen. If the exact opposite is observed, the port may be spoofed. In addition, the *perSYN* feature is not used for the classification either, because of its small impact on the results (see Section 3.4.6).

In Table 3.6 are given several classes of anomalies observed in the last few years, followed by their characteristics. They are ordered so that the most specific rules override the lower ones. Some anomalies referring to attacks are identified, such as "forged packets", "large scan" probably launched with ZMap, "DDoS attack", "targeted scan", "botnet scan" and "botnet

expansion" that kills other processes bound to a port. There are also anomalies caused by a drop in malicious activities, such as "normal packets" and "less botnet scan".

Only significant anomalies (whose $AS > 12$) are kept to be classified. The fact that some anomalies do not belong to any class is not worrying, as it means a port whose nature evolves rapidly, rather than a port targeted by one clearly identified attack. We observe a nature of traffic seemingly different between 2016 and 2018, with a majority of scans and DDoS attacks in 2016 and more normal activity in 2018.

| Classes and characteristics | 2016 | 2017 | 2018 |
|---|---|---|---|
| **More normal packets** *+meanSize,+stdSize* | 1 | 5 | 12 |
| **More forged packets** *-meanSize,-stdSize* | 0 | 1 | 1 |
| **Large scan** *-srcDivIndex,+destDivIndex,-meanSize* | 3 | 5 | 0 |
| **DDoS** *+srcDivIndex,-destDivIndex* | 6 | 1 | 3 |
| **Botnet scan** *+srcDivIndex,+destDivIndex,-meanSize* | 5 | 2 | 6 |
| **Botnet expansion** *+srcDivIndex,+destDivIndex,-stdSize* | 2 | 2 | 2 |
| **Targeted scan** *-srcDivIndex,-destDivIndex* | 1 | 2 | 4 |
| **Less botnet scan** *-srcDivIndex,-dstDivIndex,+meanSize,+stdSize* | 0 | 5 | 3 |
| **Total** | 18/21 | 23/32 | 31/40 |

Table 3.6: Definition of classes and their characteristics, with their occurrences each year.

### 3.4.8 Ground-truth

Since we are detecting anomalies on a long-term scale and based on destination port numbers, we are not able to compare our work to other intrusion detection systems on a fair basis. We are only comparing our results to the MAWILab database, showing that it did not detect the main anomalies that we identified, either in the MAWI dataset or the UCSD dataset. Moreover, as a matter of fact, we focus on detecting major botnets and attacks arisen these last years and we do not benefit from a labeled dataset of a several-year period.

Nevertheless, we attempted to provide a retrospective analysis of the major botnets and attacks arisen between 2016 and 2018. The literature in this field is not prolific so we combined different attack data sources [128, 129] to create the list; we then evaluated whether we detected them using different datasets. Table 3.7 hereafter references the IoT and malware botnets reported between 2016 and 2018, and the anomaly score of the event in case we detected it.

The false detection rate is computed as the number of benign events classified as malicious over the total number of benign events. However, we do not have labeled data in the available datasets and thus it is not feasible to compute the false positive rate. For instance, we cannot distinguish between a false positive and a small attack that targets only this network and that is not referenced as a botnet. However, we estimate hereafter the number of **unknown anomalies**,

| Botnet | Port | Year | MAWI dataset | UCSD dataset |
|---|---|---|---|---|
| **Mirai (IoT botnet)** | 23 | 2016 | ✓(26/54) | ✓(5/6) |
| **Mirai (IoT botnet)** | 2323 | 2016 | ✓(28/54) | ✓(6/6) |
| **Mirai (IoT botnet)** | 7547 | 2016 | ✓(29/54) | ✓(5/6) |
| **Mirai (IoT botnet)** | 6789 | 2016 | ✓(20/54) | ✓(6/6) |
| **Mirai (IoT botnet)** | 2222 | 2016 | X | ✓(5/6) |
| **Mirai (IoT botnet)** | 23231 | 2016 | ✓(20/54) | X |
| **Hajime (Malware botnet)** | 5358 | 2017 | ✓(19/54) | ✓(6/6) |
| **Reaper (IoT botnet)** | 20480 | 2017 | X | ✓(6/6) |
| **Satori (IoT botnet)** | 37215 | 2017 | ✓(27/54) | ✓(4/6) |
| **Satori (IoT botnet)** | 52869 | 2017 | ✓(17/54) | ✓(5/6) |
| **ADB.Miner (IoT botnet)** | 5555 | 2018 | ✓(19/54) | ✓(4/6) |
| **Memcached (Malware botnet)** | 11211 | 2018 | ✓(12/54) | X |
| **Hajime (Malware botnet)** | 8291 | 2018 | ✓(19/54) | ✓(4/6) |
| **Satori (IoT botnet)** | 8000 | 2018 | X | ✓(6/6) |
| **Total** | | | 11/14 detected ($TPR = 78.6\%$) | 12/14 detected ($TPR = 85.7\%$) |

Table 3.7: List of the most impactful botnets reported these last three years, and their $AS$ attributed by our *Split-and-Merge* system, in case we detected it, for two datasets.

defined as major detected anomalies that are not part of a known botnet. For the MAWI dataset, we consider as major anomalies those starting from an anomaly score ($AS$) of 12 (therefore the equivalent of 2 subnetworks impacted with 6 anomalous features); for the UCSD dataset, which only contains one vantage point, we consider as major anomalies those with an $AS$ of 6 (i.e., with all features anomalous). We count 71 (respectively 22, 25, and 24 in 2016, 2017, and 2018) unknown anomalies for the MAWI dataset and 26 (respectively 14, 9 and 3 in 2016, 2017, and 2018) for the UCSD dataset.

As a result, we detected a slightly higher number of anomalies in the UCSD dataset: this happens likely because it is less noisy and only contains unsolicited traffic. On the contrary, the MAWI dataset shows the advantage of containing proportionally less unknown anomalies. This is likely due to the several subnetworks in the MAWI dataset; this enables one to discard the local anomalies (i.e., seen in only one subnetwork), which are caused by occasional traffic peaks and may be considered as false alerts, and to keep only distributed anomalies.

## 3.5 Complexity and performances analysis

### 3.5.1 Complexity analysis

To evaluate the scalability of our algorithm, we provide the space and time complexity analysis of each step. Table 3.8 hereafter aims at simplifying the notations to express the complexity.

| Notation | Definition |
|---|---|
| $p$ | Number of packets collected in the subnetwor |
| $a$ | Number of attributes per collected packet |
| $p'$ | Number of ports after applying $N_{min}$ filter |
| $f$ | Number of features |
| $l$ | Number of anomalies during one time slot |
| $n_i$ | Number of entries in $CTi$ (for feature $i$) |

Table 3.8: Complexity notations.

**Data collection**

*Space complexity*: the program stores $a$ attributes per packet, hence the complexity is $O(a \cdot p)$.

*Time complexity*: packets are collected on the fly, and packets' attributes are instantly stored. Therefore the execution of the algorithm is near real-time.

**Features computation (see Algorithm 1)**

*Space complexity*: the memory space needed to compute features from attributes is O(1) for all features, i.e., the three diversity indices, the mean, the standard deviation, the percentage, and the number of packets. Therefore, the total space complexity is $O(f)$.

*Time complexity*: first, we need to parse ports and compute a set of features for each one. The feature computations, i.e., three diversity indices, a mean, a standard deviation, a percentage, and a counter, multiplied by the number of ports, generates a total complexity of $O(p' \cdot \sum_{i=1}^{f} n_i)$.

**Anomaly detection (see Algorithm 2)**

Note that steps 2 and 3 can be merged to avoid parsing twice each tuple of ports and features.

*Space complexity*: the median computation requires to order values using quick sort, whose memory space allocated is logarithmic: $O(log(N_{days}))$. Then, one needs to store values for the median, the median absolute deviation, the modified Z-score, and the anomalies. These values are updated for each feature and port, thus the space complexity is $O(3 + l)$.

*Time complexity*: The time complexity for the quick sort needed to compute the median is $O(N_{days} \cdot log(N_{days}))$.

Finally, as $a << p$, the total space complexity for all steps is equal to $O(p + log(N_{days}))$. The total time complexity is $O(p' \cdot \sum_{i=1}^{f} n_i + N_{days} \cdot log(N_{days}))$.

Both space and time complexity directly depend on $N_{min}$ (which determines the number of ports $p'$ to deal with) and on $N_{days}$. That is why, in the following evaluations, we better minimize these values, while keeping a good detection rate.

### 3.5.2 Execution performance

We performed our experiments on a 2017 MacBook Pro with 2.3 GHz Intel Core i5 Processor and 16GB RAM. The time spent to detect anomalies depends on the number of analyzed ports, thus on $N_{min}$, the minimum number packets on a port to consider it. We consider here the UCSD dataset, composed of 152 days picked from 2016 to 2018, each one containing one million packets. The learning phase (building per-port profiles) and the detection phase (computing the Z-score) are run simultaneously. In fact, we dispose of a sliding window, thus the model is updated each time and the detection is made on the updated model on the fly. In total, it took 6,390 seconds to run the full algorithm (i.e., the learning and detection phases) on 152 days, covering 17,152 ports and one subnetwork.

## 3.6 Conclusion

In this chapter, we presented *Split-and-Merge*, our method that leverages statistics to detect main changes in the usage of application ports. The port-based anomaly detection algorithm that we propose is able to detect known and unknown attacks targeting connected objects and

servers around the world. Early stages of the attacks, namely the exploited vulnerabilities, can be detected beforehand. The empirical results obtained applying our method to real traces are very promising, since our algorithm detected a number of world-wide attacks from 2016 to 2018. In contrast, current IDSs, among which the notorious MAWILab, have not detected them. We evaluated the scalability of our algorithm by computing its complexity in terms of space and time. Moreover, we showed that our algorithm produces a very low number of false positives. We demonstrated the validity of our statistical model assumptions, showing how leveraging on distribution analysis of features we can refine the detection accuracy by analyzing how the parameters and features impact the results. We provide a classification of the detected attacks given expert rules by analyzing jointly the evolution of the features.

We then studied the large trends in terms of botnets and exploitation of vulnerabilities these last years, by inspecting IP traffic. A complementary angle of view for botnet detection consists in detecting botnet's infected hosts within its own network, for example in enterprise networks. The next chapter presents the classification of end-hosts either as a bot or benign host, based on their behavioral characteristics and communications within hosts and with the outside.

# Chapter 4

# Botnet Fingerprinting

Efficient bot detection is a crucial security matter and widely explored in the past years. While in the previous chapter we focused on the macro detection of botnets exploiting novel vulnerabilities to expand themselves, we aim in this chapter to detect botnet-infected hosts within a given network that we would like to monitor, for instance in an enterprise network. For botnet detection, recent approaches supplant flow-based detection techniques and exploit graph-based features, incurring however in scalability issues, with high time and space complexity. Bots exhibit specific communication patterns: they use particular protocols, contact specific domains, hence can be identified by analyzing their communication with the outside. A way we follow to simplify the communication graph and avoid scalability issues is by looking at frequency distributions of protocol attributes capturing the specificity of botnets' behavior. We propose a bot detection technique named *BotFP*[1], for *BotFingerPrinting*, which acts by *(i)* characterizing hosts behavior with attribute frequency distribution signatures, *(ii)* learning benign hosts and bots behaviors through either clustering or supervised Machine Learning (ML), and *(iii)* classifying new hosts either as bots or benign ones, using distances to labeled clusters or relying on a ML algorithm. We validate *BotFP* on the CTU-13 dataset, which contains 13 scenarios of bot infections, connecting to a Command-and-Control (C&C) channel and launching malicious actions such as port scanning or Denial-of-Service (DDoS) attacks. Compared to state-of-the-art techniques, we show that *BotFP* is more lightweight, can handle large amounts of data, and shows better accuracy.

## 4.1 Introduction

Back in September 2019, the French cyber police freed over 850,000 computers from a botnet named Retadup [132]. The worm spread through malicious email attachments, then installed cryptomining software on infected machines. Over nearly a million infected hosts mined Minero cryptocurrency, reaping a huge amount of money – that is often the first reason for attackers to handle a botnet. Retadup is also suspected of being used in several ransomware attacks and data thefts. At the end of 2019, hackers also mass-scan for Docker vulnerability (Docker admin port TCP/2376) to mine Monero cryptocurrency [133]. 2019 saw an increase of up to 55% of IoT malware attacks like Retadup [134]. Hence quickly detecting botnets is a major concern.

Botnet early detection is crucial to limit harms as soon as possible. However, bots often mimic normal traffic and hide their payload characteristic by encryption. Recently they are also more

[1]This work was first presented at *IEEE/IFIP NOMS 2020* [130], then extended in a article published in *IEEE Transactions on Network and Service Management* [131].

likely to use HTTP rather than IRC to be confounded with classic web traffic. Also, HTTP being a widely-used protocol, firewalls seldom block it, contrary to IRC [135]. Furthermore, dynamic ports and run-time protocol changes enable botnets to bypass signature-based firewalls and intrusion detection systems (IDS). For robust detection systems, several flow-based botnet detection approaches [69, 49, 70, 70] were recently proposed, working without packet payload information. Differently than flow-based detection, other recently proposed botnet detection approaches consist in characterizing and analyzing relationships between hosts in the network, with techniques commonly referred to as graph-based anomaly detection [75, 81, 82]. However, these techniques suffer from high time and space complexity, as they need to compute complex features over very large graphs.

In this chapter, we propose a lightweight bot detection technique named *BotFP* that builds signatures modeling the behaviors of hosts in a network. These signatures reflect the communication pattern of each host, to highlight the differences between normal hosts and bots. In particular, we account for the fact that a botnet performs various kinds of actions; one can simultaneously infect and scan other hosts, perform click fraud, launch DDoS attacks, actions that can be qualified by finely analyzing IP addresses, TCP and UDP port numbers and ICMP types and codes. Then, we aim at accurately defining what constitutes bot and normal communications based on the signatures of labeled hosts; we propose a clustering variant (*BotFP-Clus*) - classifying new hosts based on their distances to labeled clusters - and a supervised machine learning variant (*BotFP-ML* ) - for which we evaluate three different supervised ML classification approaches.

For our evaluation, we use the CTU-13 bot traffic dataset [35], containing 13 scenarios of different botnet samples. In each scenario, a specific malware sample is executed, which used several protocols and performed different actions. We first learn from a training set what constitutes normal communications and malicious behaviors, based on the distribution of IP addresses and port numbers used by hosts. We compare our two approaches, *BotFP-Clus* and *BotFP-ML*, in terms of bot detection performances and complexity. We demonstrate that the former enables us to detect all bots with a better recall and a reduced complexity. Then, we show that, while having a comparable or lower time complexity than the state of the art bot detection techniques, we outperform them all with a recall from 84% to 100% and a precision from 75% to 93%, depending on the method. We also show that using an adaptive quantification based on the volume of traffic enhances the results. For the sake of reproducibility and further research, we made the source code publicly available at [136].

The remainder of this chapter is organized as follows: Section 4.2 introduces the dataset we used. We refer the reader to the specific related work section addressed in 2.3. Section 4.3 presents the data processing methodology, as well as our rationale about looking at per-host fingerprints to describe host communications. In Section 4.4, we introduce two signature-based algorithms to detect bots, namely *BotFP-Clus*, based on a clustering technique, and *BotFP-ML*, composed of various supervised machine learning techniques; while Section 4.3 describes to the computation of attribute distribution signatures, in Section 4.4 we introduce two different techniques based on the aforementioned signatures. In Section 4.5, we numerically evaluate the proposals, discussing the performance in terms of precision and recall. In Section 4.6, we qualify the space and time complexity of *BotFP*, and we compare it to other recent bot detection methods. Finally, Section 4.7 concludes this chapter.

## 4.2 Dataset

In this section, we describe the dataset we leverage for identifying characteristics inherent to botnets.

We used the publicly available CTU-13 dataset [70] containing 13 scenarios of bot infections. Different botnet malware samples are executed in a virtual network to mimic the behavior of an infection that is spreading. Each scenario contains between 294 and 508 hosts, including 1 to 10 bots, and there are 4923 hosts in total. Table 4.1 below draws the main characteristics for each scenario; it describes if hosts used IRC, P2P, or HTTP protocols and if they sent spam, did Click-Fraud (CF), port scanning (PS), or DDoS attacks. This dataset is widely used for the already discussed recent bot detection methods [75, 81, 82].

| Id | Duration (hrs) | # bots | Bot | Activity |
|---|---|---|---|---|
| 1* | 6.15 | 1 | Neris | IRC, SPAM, CF |
| 2* | 4.21 | 1 | Neris | IRC, SPAM, CF |
| 3 | 66.85 | 1 | Rbot | IRC, PS |
| 4 | 4.21 | 1 | Rbot | IRC, DDoS |
| 5 | 11.63 | 1 | Virut | SPAM, PS |
| 6* | 2.18 | 1 | Menti | PS |
| 7 | 0.38 | 1 | Sogou | HTTP |
| 8* | 19.5 | 1 | Murlo | PS |
| 9* | 5.18 | 10 | Neris | IRC, SPAM, CF, PS |
| 10 | 4.75 | 10 | Rbot | IRC, DDoS |
| 11 | 0.26 | 3 | Rbot | IRC, DDoS |
| 12 | 1.21 | 3 | NSIS.ay | IRC, P2P |
| 13 | 16.36 | 1 | Virut | HTTP, SPAM, PS |

Table 4.1: Characteristics of the botnet scenarios. The scenarios included in the test set are marked by the symbol *.

To evaluate the performances of our bot detection method, we used scenarios 1, 2, 6, 8, and 9 for the test set, and other scenarios for the training set, as recommended by the authors of the CTU-13 dataset [70]. Note that the bot species are different in the training and test sets, i.e., we do not test our algorithm on the same bot malware that we learned from.

The CTU-13 dataset is widely used for bot detection and contains malware samples from botnets still spreading, but we also found it interesting to explore the latest datasets from the stratosphere project [137]. In particular, the IoT-23 dataset [138] contains 23 samples of IoT network traffic, each one being either malicious or benign. In addition, the Malware Capture Facility Project [139] contains 342 long-term botnet captures, captured from 2015 until now. However, traces from these datasets are not labeled on a per-flow level, and in addition, they contain either captures from botnets or from benign hosts, but no mixed captures are needed to run the algorithm. Therefore, the application of our algorithm to IoT-23 is not pertinent, in particular its learning phase which normally trains on flows both from bots and benign hosts and labeled as such. Nevertheless, we report in the appendix (Chapter B) a visual comparison between three scanning events, one from each dataset, showing similarities and differences. In the following, we focus on the CTU-13 dataset because it has labeled flows and is used as a reference dataset by many existing methods at the state of the art we can so compare to.

## 4.3 Bots Fingerprints

In this section we first describe some typical behaviors of bots, then we introduce our bot detection technique *BotFP*, detailing the different processing steps, including the flow records collection, the bot fingerprints computation, and the signatures formatting.

### 4.3.1 Preliminary example

Fig. 4.1 gives an example of dissimilar histograms for a benign host and a bot, for three attributes namely $\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{UDP}}$ and $\text{Dip}_{\text{UDP}}$; in this example, histograms are made of 32 regular bins (each bin aggregating multiple attribute values) and are normalized with respect to the number of packets. $\text{Sport}_{\text{TCP}}$ for the benign host are in the range [49152, 61000] and [1025, 5000] for the bot, which indicates a first difference in the ephemeral ports thus the OS (all bots from the dataset display this characteristic). Looking to $\text{Dport}_{\text{UDP}}$ does not show any anomaly, as both hosts show a high usage of UDP/53 which runs DNS. We thus expect to find one single IP address corresponding to the DNS server in $\text{Dip}_{\text{UDP}}$, but we found a multitude of them for the bot. Thus, it is not a classic DNS usage, but in fact an attempt of port scanning. This example illustrates why it is important to not only compare attributes one to one but also to take into account correlations among attributes.

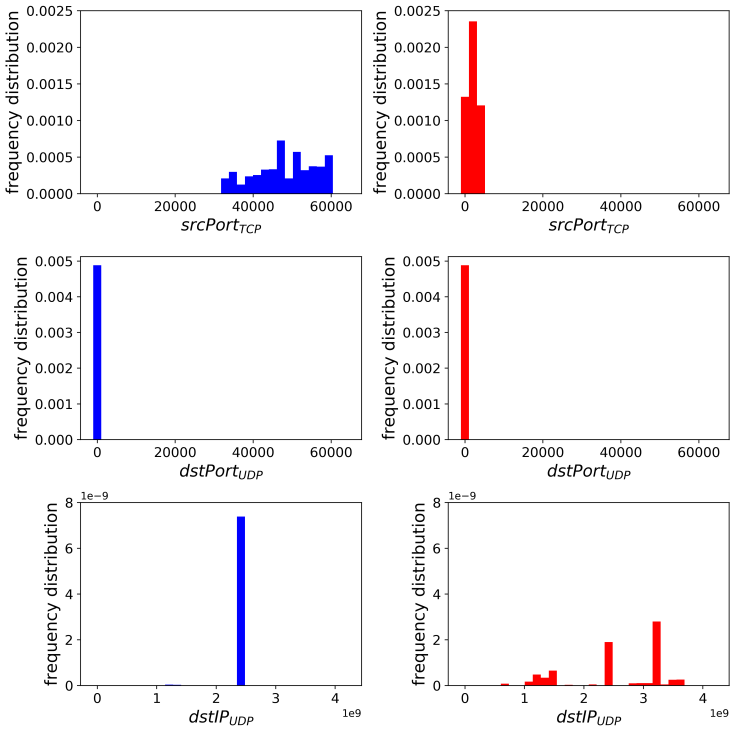



Figure 4.1: Histograms showing the frequency distributions of attributes ($\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{UDP}}$, and $\text{Dip}_{\text{UDP}}$ respectively) for a benign host in blue/left (`147.32.84.17` from scenario #1) and a bot in red/right (`147.32.84.165` from scenario #1).

### 4.3.2 Methodology

The primary goal of our bot anomaly detection algorithm, *BotFP*, is to label bots as such, avoiding false positives. Let *Sip*, *Dip*, *Sport*, and *Dport* represent respectively the source and

the destination IP addresses, the source, and the destination port numbers, of a layer-4 flow. Fig. 4.2 depicts the *BotFP* steps, through a trace example.

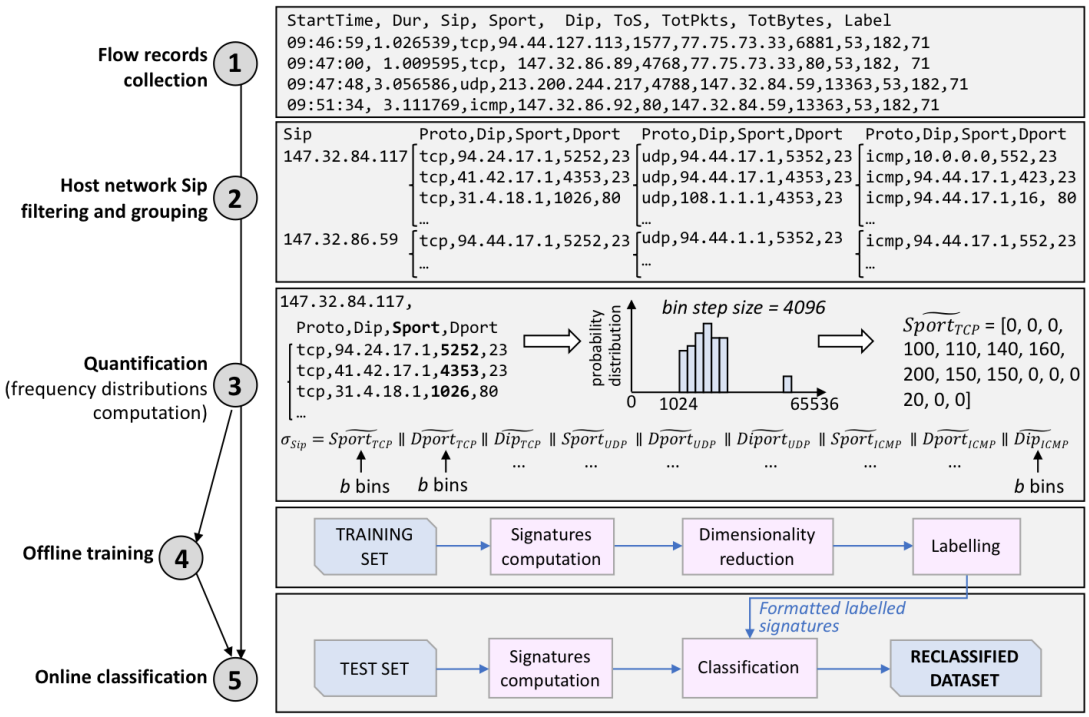



Figure 4.2: Description of the processing steps of our solution. We first select flow records (**step 1**) that are in the host network and group them by such addresses (**step 2**). Signatures $\sigma_{Sip}$ of each host are defined as the concatenation of the frequency distributions of each attribute (**step 3**). The training phase consists in learning what constitutes either malicious or benign signatures hosts (**step 4**). Finally, we classify hosts from the test set, with a supervised learning algorithm that considers their distance to signatures of previously labeled hosts (**step 5**).

1. **Flow records collection**: flow records are first collected to form a dataset. We split the dataset into two distinct sets: one for training and one for testing.

2. **Host network *Sip* filtering and grouping**: from flow records, we select the ones whose *Sip* is in the host network and group them by such addresses.

3. **Quantification** (attribute frequency distributions computation): signatures of each host, denoted $\sigma_{Sip}$, are defined as the concatenation of the normalized frequency distributions of each attribute. TCP, UDP, and ICMP flows are characterized separately to better take into account each protocol specificity.

4. **Offline training**: this phase consists of learning from the training set what constitutes either malicious or benign host signatures. Different methods can be used to do so, including clustering algorithms, supervised learning algorithms, or neural networks. We further describe two approaches we propose, namely *BotFP-Clus* and *BotFP-ML*, in Section 4.4. This step is optional and does not apply in the case of an unsupervised learning algorithm.

5. **Online classification** (distances computation): finally, we classify hosts from the test set either as benign or bot, through a learning algorithm consistent with the previous step. We compute the distance between one labeled host from the training set and one host to classify from the test set.

Table 4.2 defines the key parameters we use, as well as the notations for the variables of the algorithm.

| Notation | Definition |
|---|---|
| $m$ | Minimum number of packets per host for one protocol |
| $b$ | Number of intervals (bins) in the frequency distributions |
| $\epsilon$ | Density in the clustering algorithm |
| $a_i^j$ | Frequency distribution of attribute $i$ for host $j$ |
| $\sigma_j$ | Signature of host $j$ |

Table 4.2: Notations.

### 4.3.3 Flow records collection and formatting

Flow records are first collected to form a dataset (**step 1** in Fig. 4.2). We split the dataset into two distinct sets: one for training and one for testing. We name the training set as $\mathcal{T}$ and the test set as $\mathcal{E}$. We select only flows whose $Sip$ belongs to the host network prefix and group them by such addresses (**step 2** in Fig. 4.2). As we search for internal bots, we exclude source IP addresses belonging to external Internet networks.

### 4.3.4 Quantification (attribute frequency distributions)

To characterize the hosts' behavior, let $A$ be the set of attributes used to characterize a host. In this work, we consider 9 attributes in total, discriminating between TCP, UDP and ICMP packets, as follows (following CTU-13 notations): $\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{TCP}}$, $\text{Dip}_{\text{TCP}}$, $\text{Sport}_{\text{UDP}}$, $\text{Dport}_{\text{UDP}}$, $\text{Dip}_{\text{UDP}}$, $\text{Type}_{\text{ICMP}}$, $\text{Code}_{\text{ICMP}}$ and $\text{Dip}_{\text{ICMP}}$ [2].

Let $a_i^j$ denote the attribute vector for attribute $i$ and host $j$, representing the attribute frequency distribution, i.e., the ratio of packets received for attribute $i$ over its attribute range. More precisely, each attribute vector contains $b$ bins, where $a_i^j[k]$ is the value of the $k^{th}$ bin of attribute $i$ for host $j$. For each attribute, a bin aggregates the attribute occurrences over the possible attribute range (e.g., many successive port numbers grouped together in a bin) available for the specific attribute (e.g., TCP source port), in a way that depends on a bin aggregation policy as detailed hereafter.

In practice, to avoid statistically negligible attributes to influence the detection logic, it makes sense to set the attribute vectors with a too low number of TCP packets exiting a host $j$ to null values, i.e., $a_i^j[k] = 0\ \forall k$ and for all TCP-type attributes $i$. Let $m$ denote such a minimum number of packets threshold, that we later numerically assess.

Let $\sigma_j$ denote the signature of node $j$ – keeping in mind that a host is uniquely identified by its $Sip$ (we use $\sigma_{Sip}$ instead of $\sigma_j$ in the figures). It is built as the concatenation of all its

[2]Note that in the CTU-13 dataset, the used notation for the ICMP type is $\text{Sport}_{\text{ICMP}}$ and for the ICMP code is $\text{Dport}_{\text{ICMP}}$.

attribute vectors; it can then be expressed as:

$$\sigma_j = \Big\|_{i=1}^{|A|} a_i^j = a_1^j \parallel a_2^j \parallel \cdots \parallel a_{|A|}^j \tag{4.1}$$

where $\parallel$ represents the concatenation operator between vectors. The result of the concatenation is therefore one single array $\sigma_j$ of $|A| \times b$ entries.

### Quantification technique

Let us further clarify how the attribute frequency distributions can be aggregated in a set of bins. To compute the attribute vector, $b$ bins are used to cover the attribute range, say $[0, max]$; e.g., for source and destination port numbers, $max$ is equal to 65536, and for destination IP addresses, it is equal to $2^{32}$. It makes sense to set $b$ as a power of 2, as port numbers and IP addresses are typically organized into ranges of powers of 2 (e.g., reserved ports are in [0, 1023] and ephemeral ports in [49152, 65536], while IPv4 addresses are denoted by 4 Bytes).

We consider two different ways to aggregate bins.

**Regular bins**: attribute range intervals are uniformly distributed, of a fixed bin width set to $max/b$. Fig. 4.2 (**step 3**) shows an example of attribute frequency histogram for attribute $\text{Sport}_{\text{TCP}}$: the attribute range corresponds to the possible TCP source port numbers used by the *Sip* host. The example shows a regular partition of the attribute range; e.g., with a bin width set to 4096, the number of bins is 16.

**Adaptive bins**: intervals are chosen depending on the amount of traffic. Intuitively, the more density of information there is, the more sensitive (small) the step should be. Thus we aim to define the individual bin width so that we equalize the occurrences over the different bins, i.e., it is always the same for all the bins, each bin having potentially a different bin width. To do so, we first start with the highest attribute granularity (e.g., 65536 for the port number), and compute the frequency distribution for all hosts. Then, we sum up the obtained vectors across all hosts. At this point, we are able to define individual bin steps so that the occurrences are evenly distributed across bins; this operation can be done for instance taking the cumulative distribution function and evenly dividing the probability range from 0 to 1 in the number of desired bins. We repeat this process for all the attributes.

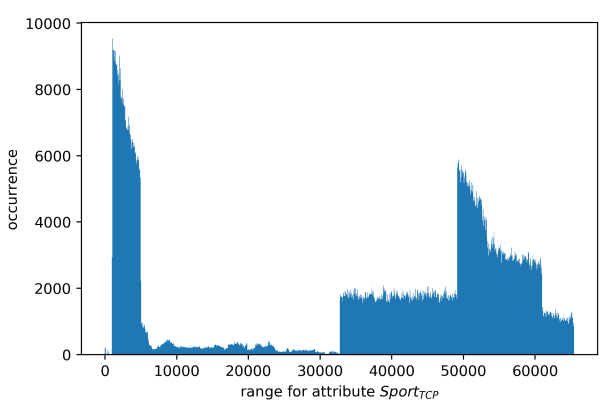
Fig. 4.3 illustrates the computation of adaptive bins for the $\text{Sport}_{\text{TCP}}$ attribute. We first compute the number of unique values for very small bins as shown in Fig. 4.3a, then we divide the cumulative sum by a fixed number of regular sampling intervals and compute adaptive bins so that each of them contains the same number of packets, as shown in Fig. 4.3b.

The number $b$ of bins and the bin aggregation strategy (regular vs. adaptive bins) are to be assessed experimentally.

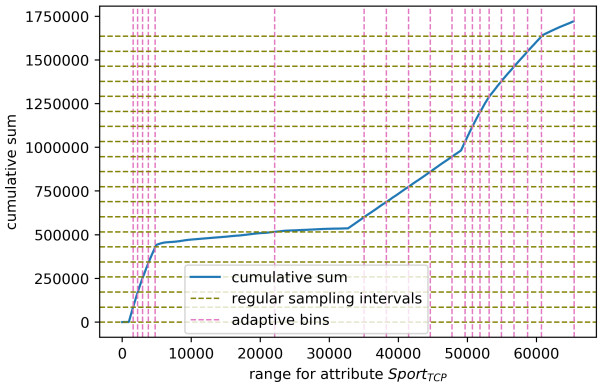
### Observable bot behavior and attributes

Let us report on the observable behaviors for TCP, UDP and ICMP attributes from traces we could have access to. To get a visual representation of such behaviors, we propose in Appendix B fingerprints of infected hosts highlighting their specific malicious activities. First, we observe uncommon behaviors specific to a botnet for **TCP** flows:

- *destination ports* ($\text{Dport}_{\text{TCP}}$) usually range between 0 and 1023. These service ports are associated with given services by the Internet Assigned Numbers Authority (IANA) [140], e.g., TCP/80 typically runs HTTP and TCP/443 HTTPS. However, bots show different




(a) Number of unique values using 1000 bins.




(b) Cumulative sum and definition of 20 adaptive bins.

Figure 4.3: Example partitioning in 20 adaptive bins based on the traffic load, for attribute $\text{Sport}_{\text{TCP}}$.

usage of destination ports: they are usually diverse and represent services often targeted by attackers such as TCP/25 (SMTP) or TCP/23 (Telnet), vulnerable to spam and attacks. We also observe some exotic destination port numbers used to access proxies that host the C&C server.

- *source ports* ($\text{Sport}_{\text{TCP}}$) are often ephemeral ports, allocated automatically from a predefined range by the IP stack software. The range recommended by IANA is 49152 to 65535, while many Linux kernels use the port range 32768 to 61000. FreeBSD has used the IANA port range since release 4.6. Previous versions, including the Berkeley Software Distribution, use ports 1025 to 5000 as ephemeral ports. Microsoft Windows Operating Systems (OS) until Windows XP used the range [1025, 5000] for ephemeral ports, while they use the IANA range now.

  We observe that bots rarely use the IANA recommended range, but rather the [1025, 5000] range. This obviously depends on the OS of the infected host. A report from Kaspersky Labs [5] shows that Linux and Windows botnets represent respectively 95.75% and 4.25% of all botnets, which is very different from the OS distribution for regular devices (not bots), probably because bots infect vulnerable devices including connected objects.

- *destination IP addresses* ($\text{Dip}_{\text{TCP}}$): not all subnetworks are covered, but only some specific ones are contacted by normal hosts. Among them, it is common to observe addresses in the same range of the source IP address, private networks including `192.168.0.0/16`, and cloud service subnetworks, mostly Google ones, often contacted for Google Analytics and similar collateral services. Destination IP addresses cover a larger space for bots than for normal nodes, in the case of spam for example. Looking to the AS details in the *whois* database [141] also gives additional information, such as the age of the domain or its originated country.

There are specific botnet behaviors also for **UDP** flows:

- UDP *destination ports* ($\text{Dport}_{\text{UDP}}$) are associated to particular services by IANA, as for TCP. In the case of UDP, we often observe a fixed destination port set to 53. It represents connections to the local DNS server as UDP/53 typically runs DNS.

- UDP *source ports* ($\text{Sport}_{\text{UDP}}$) are used for ephemeral ports as for TCP, their range depends on the OS implementation. We notice that the range for ephemeral ports used by bots is often different than for common hosts.
- there is usually a fixed *destination IP address* ($\text{Dip}_{\text{UDP}}$) that represents the DNS server IP address.

Finally, **ICMP** flows also show specific botnet behaviors:

- the ICMP *type* ($\text{Type}_{\text{ICMP}}$) indicates the type of ICMP message and gives a global information about the kind of message (e.g., 0 for Echo Reply and 3 for Destination Unreachable) as specified in RFC2780 [142]. We often observe only a small amount of ICMP packets. In the case of a botnet, we sometimes observe many ICMP messages with uncommon types and codes, consisting of a Ping Flood or an ICMP DoS attack.
- the ICMP *code* ($\text{Code}_{\text{ICMP}}$) represents the ICMP sub-type and gives additional context information for the message (e.g., if the type is 3, the code can be 0 if the destination network is unreachable or 1 if the destination host is unreachable, etc).
- the hosts frequently reply to *destination IP addresses* ($\text{Dip}_{\text{ICMP}}$) that targeted them, with messages like "port unreachable" if it was a port scanning. The number of such packets is low for benign hosts and larger for bots.

Looking to attributes individually enables us to retrieve some botnets' behaviors, but it is even better to analyze these attributes together. Actually, sometimes it is the combination of two attributes that makes a host behavior abnormal.

### 4.3.5 Signatures formatting

As shown in Fig. 4.2 (**step 4**), the training phase deals with flows from the training set through several modules. Data preparation is accomplished through pre-processing and dimensionality reduction. In the pre-processing phase, signatures of hosts belonging to the internal network are computed as described in the previous step. Finally, we reduce the dimensionality of the space by finding the directions of maximal variance, in order to reduce the spatial complexity. Thus we project the dataset into the new dimensional space.

#### Removing the less significant hosts

The threshold $m$ is the minimum number of packets per host and protocol to consider it. We analyze the distribution of the number of packets per host to better understand its impact. The distribution is a long-tailed one, with hosts with a very high number of packets (up to 200,000 packets per host). Therefore, Fig. 4.4 shows the Probability Distribution Function (PDF) of the number of TCP packets per host, only from 0 to 1500 packets per host, avoiding very large outliers. The plot is about TCP, while UDP and ICMP exhibit the same distribution. Eliminating hosts with less than $m$ packets has a minor impact on the results, as we notice that after removing hosts with less than 150 packets, the number of hosts goes from 4923 to 1933, which represents only 0.7% of removed traffic in terms of numbers of packets (from 13,342,675 to 13,295,640 packets). We could miss very stealthy bots using the filter, but we assume that bots have to be a minimum active to be efficient (including attacks, scans, and communications with the C&C server).

To choose its exact value, we evaluate the impact of parameter $m$ on the bot detection results in Section 4.5.1.

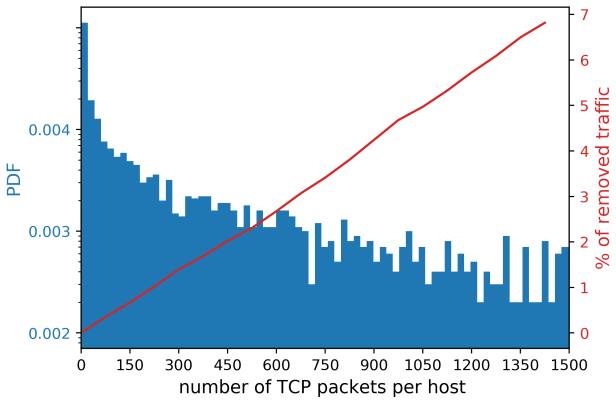



Figure 4.4: PDF of the number of TCP packets per host, and (in red, right axis) possible $m$ threshold values and corresponding traffic volume ratio.

#### Dimensionality Reduction

After the pre-processing step, the signatures contain $|A| \times b$ columns, with $b$ the number of bins and $|A| = 9$. Our purpose is to reduce the number of columns, by restricting the scope to the most meaningful ones. We reduce the dimensionality of the space working with the Principal Component Analysis (PCA) technique. PCA is applied on hosts from the training set $\mathcal{T}$, thus on a matrix of size $(|\mathcal{T}|, |A| \times b)$. PCA reduces the number of components of the already smallest dimension, then $|A| \times b$ must be smaller than $\mathcal{T}$ to reduce the number of features. Therefore, it is not applicable if $|A| \cdot b$ is larger than $|\mathcal{T}|$.

PCA finds the directions of maximum variance. The fraction of variance explained by a principal component is the ratio between the variance of that principal component and the total variance. Our goal is to reduce the dimensionality while keeping a good amount of information, so that the cumulative explained variance ratio is close to 100%. Fig. 4.5 hereafter shows the cumulative explained variance ratio vs. the number of factors, with $b = 128$. To get 99% of the variance ratio, we can reduce the factor number by ten, approximately, keeping around 150 and 200 factors respectively for regular and adaptive bins, over the 1152 original factors. These values vary with the number of bins: for each of them, we need to choose the right number of factors. For all values of $b$, we notice that we can reach 99% of the variance ratio by using 9 times fewer factors.

## 4.4 Bot Detection

In this section, we introduce two different bot detection techniques for *BotFP* that we designed for the training and the classification. *BotFP-Clus* relies on clustering as described in our previous work [130], and *BotFP-ML* relies on other supervised machine learning (ML) techniques.

### 4.4.1 *BotFP-Clus*

#### Training

Clustering algorithms are designed to group similar vectors into clusters and identify isolated ones as outliers. The similarity between two vectors is evaluated using a distance function like the Euclidean distance. Two vectors are defined as similar if they are close to each other, else dissimilar. We use as clustering algorithm DBSCAN (Density-Based Spatial Clustering of

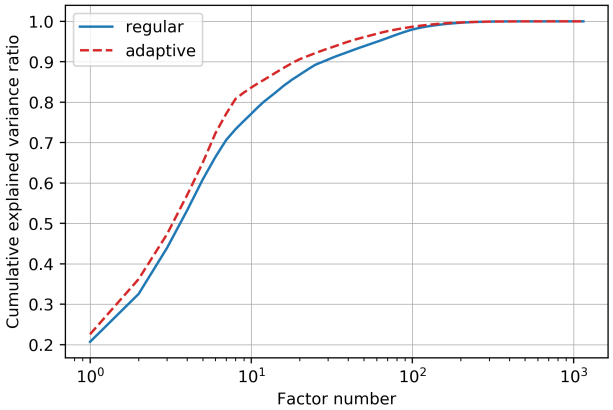



Figure 4.5: Explained variance vs. number of factors, for PCA applied with $b = 128$. The factor number equals to 1 if only one feature is used, while it is close to $10^3$ if all ($|A| \times b = 9 \times 128 = 1152$) features are used.

Applications with Noise) [143], because it presents the advantage of discovering clusters without knowing the number of clusters in advance, which fits our needs. In addition, it works well on our data because clusters have close densities. DBSCAN uses two parameters:

- $\epsilon$ specifying the radius of a neighborhood with respect to some point. Every point situated within a distance $\epsilon$ from a point $p$ is a neighbor of $p$; [3]
- $minPts$ which defines the minimum number of points in a radius $\epsilon$ to form a cluster.

DBSCAN defines a cluster as the maximal set of points where every pair of points $p$ and $q$ are within a distance $\epsilon$ from each other, and considers points that do not belong to any cluster as outliers. In our solution, DBSCAN is used in a slightly different manner as illustrated in Step 4 of Fig. 4.2. We set $minPts$ to 1 in order to consider singleton clusters as well. Then DBSCAN is applied on the vectors of $\sigma_j$ from the training set to build clusters of similar host signatures. Using clusters instead of singular hosts enables us to filter abnormal hosts and get more consistent data. This also reduces the number of coordinates to store.

Let $\mathcal{C}$ be the set of clusters obtained applying DBSCAN on the training set. Each cluster $c \in \mathcal{C}$ contains several attributes:

- a set $H_c$ of hosts belonging to the cluster;
- its position $P_c$ defined as the centroid of the set of signatures $\{\sigma_1, \sigma_2, ..., \sigma_N\}$ of hosts in $H_c$, computed as $P_c = (\sigma_1 + \sigma_2 + ... + \sigma_N)\ /\ N$;
- a label identifying the nature of the cluster $c$, i.e., malicious or benign, denoted $L_c$.

  The nodes that are bots are known from the ground truth of the training set. The cluster is identified as a bot cluster if it contains at least one bot, else it is benign.

$$L_c = \begin{cases} \text{'bot' if } H_c \text{ contains a bot node} \\ \text{'benign' otherwise} \end{cases} \tag{4.2}$$

[3] The metric that we use for the computation of the distance between two hosts in DBSCAN is the $\ell 1$-norm defined as $||\sigma_h||_1 = |\sigma_h[1]| + ... + |\sigma_h[n]|$: this distance is robust and does not vary with the number of bins, as the cumulative sum of all elements stays equal. We consider it better than the $\ell 2$-norm – defined as $||\sigma_h||_2 = \sqrt{|\sigma_h[1]|^2 + ... + |\sigma_h[n]|^2}$ which increases with the number of bins.

One may find the above condition to label bot clusters too strong. However, our tests showed that by appropriately tuning $\epsilon$, one can get a good clustering solution. A good setting we found is an $\epsilon$ set to 300 and 512 adaptive bins, giving that bot clusters always contain one bot at maximum, except one case with two bots, and hence they did not contain any normal host. Thus it appears sufficient to label a cluster as a bot if it contains at least one bot. About the cluster density, we notice that other clusters contain up to 2400 hosts (all of them benign), but most of the time only one or several ones.

### Classification

We classify hosts from the test set based on their distance to the set of labeled clusters $\mathcal{C}$. For a host $h \in \mathcal{E}$, if the closest cluster is labeled as bot, $h$ will be classified as a bot. If the closest cluster is benign, $h$ will be classified as benign too. Let $dist()$ be a function measuring the distance between a signature and a cluster, $c^*$ is the closest cluster such that $dist(\sigma_h, P_{c^*}) = \min_c[\text{dist}(\sigma_h, P_c)]$. Then hosts are classified based on the label of $c^*$.

$$L_h = \begin{cases} \text{'bot' if } L_{c^*} = \text{'bot'} \\ \text{'benign' otherwise} \end{cases} \tag{4.3}$$

Note that the classification is based on a nearest neighbor approach. One could think that a host with an atypical signature, distant from a benign cluster but also very different from other bots, could be classified as benign. However, we found that all hosts to classify are quite close to a labeled cluster, and that the dataset does not contain atypical hosts. Fig. 4.6 hereafter shows the distribution of the distances between the hosts to classify and the closest cluster. We found that the maximal of such distances is around 0.055, which is about two times the mean value. This means that there are no really hosts whose signature is atypical and that would be very far from all clusters.

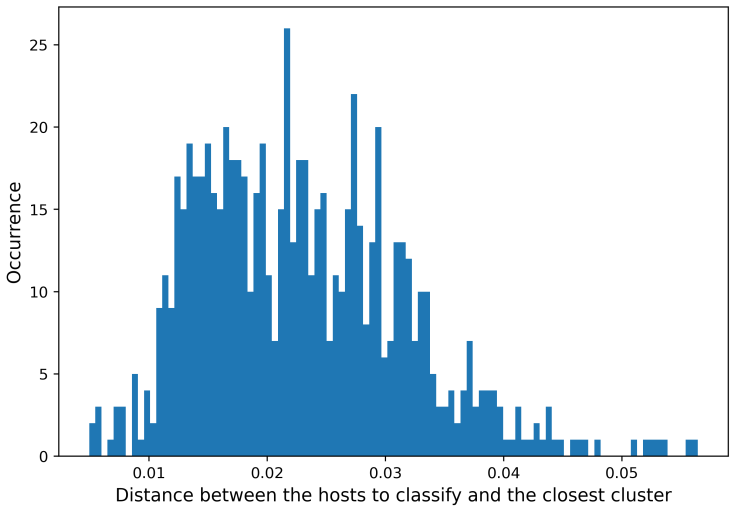



Figure 4.6: Histogram showing the distribution of the distances between the hosts to classify and the closest cluster.

In case of another dataset that would contain more sparse data and atypical hosts, one could refine our technique to classify as "anomaly" the hosts whose signature is quite far from all labeled clusters.

### 4.4.2 *BotFP-ML*

Several techniques can be used to learn from the training set what constitutes signatures either from benign hosts or from bots, then classifying hosts from the test set based on that knowledge. We evaluate four such techniques for *BotFP*: *(i)* Logistic Regression, used to predict the probability of a binary dependent variable, *(ii)* Support Vector Machines (SVM), which, given labeled training data, output an optimal hyperplane which categorizes new examples, *(iii)* Random Forest, which creates a forest with a number of decision trees, and (iv) Multilayer Perceptron (MLP) classifier, which can be thought as a deep artificial neural network, composed of an input and output layers, and an arbitrary number of hidden layers.

Supervised learning algorithms and neural networks take into account hyperparameters that must be tuned to obtain the best results of classification. Grid search [144] is used for model tuning, it builds and evaluates a model for every combination of hyperparameters specified, then selects the best one to improve a given evaluation metric. We used it by favoring the recall criterion. The hyperparameters are different according to the type of classifier. For instance, with SVM and logistic regression, the parameter $C$ controls the sparsity: the smaller $C$, the fewer features selected. Another parameter common for both these algorithms is the penalty, which is used to specify the norm used in the penalization (regularization or noise variance).

## 4.5 Evaluation

In this section, we evaluate the performance of *BotFP*. We first propose an evaluation of the method *BotFP-Clus*, tuning its parameters including DBSCAN $\epsilon$, the number of bins $b$, and the type of bins. We then analyze and compare *BotFP-Clus* and *BotFP-ML* performances and we select a set of best solutions. Finally, we compare them to other state-of-the-art detection techniques. Note that the source code for *BotFP* is available in [136]. The subnetwork address needs to be set up, then the reader can run the learning phase (learning the normal and malicious behaviors of the network and tuning the key parameters), and then launch the detection process.

### 4.5.1 *BotFP-Clus*

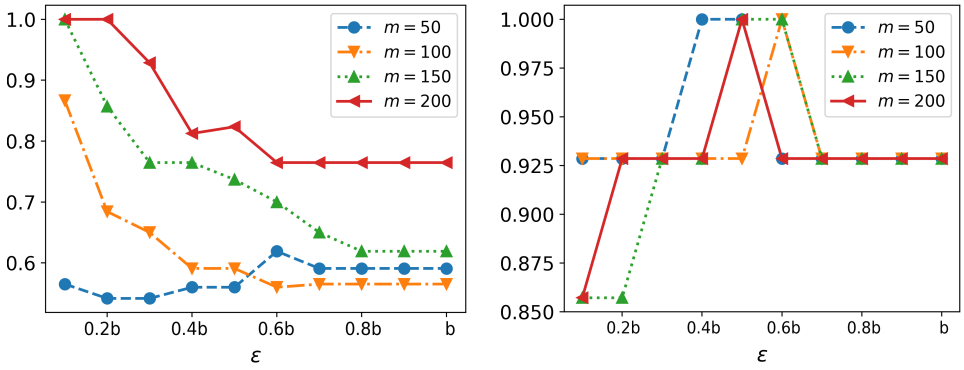
*BotFP-Clus* is our proposal relying on labeled clusters of similar hosts behaviors. We apply it on the CTU-13 dataset, following the methodology presented in Sections 4.3 and 4.4.

We analyze the results as a function of the three key parameters to be chosen for *BotFP-Clus*: the minimum number of packets threshold ($m$), the number of bins ($b$), and the $\epsilon$ DBSCAN parameter. We proceed as follows: first, we show precision and recall results as a function of $m$ to find a reasonable choice; second, we elaborate on the impact of $\epsilon$, and identify one good setting. We also show the benefit of using adaptive bins rather than regular ones.

Fig. 4.7a and 4.7b respectively show the precision and the recall of *BotFP-Clus*, for 512 adaptive bins, $\epsilon$ in $[0.1 \times b, 0.2 \times b, ..., b]$, and $m$ varying in [50, 100, 150, 200]. We observe that for all $m$ values, the recall is very high, reaching 100% in many settings. It is important to note that the precision is directly correlated with the minimum number of packets threshold: the higher $m$ and the higher the precision.

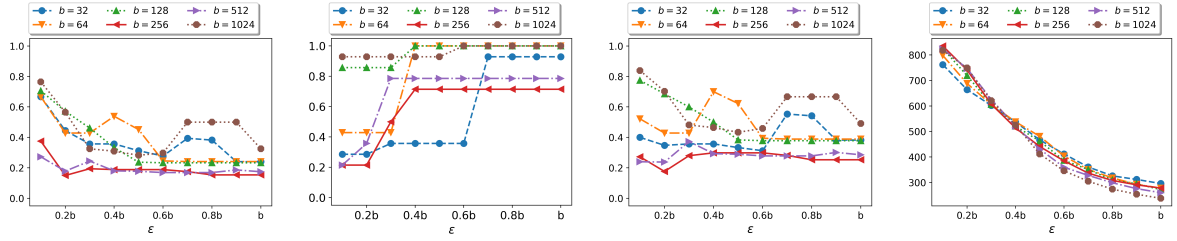
For the following experiments, to favor the recall, we choose $m = 150$, because the precision is almost as high as for $m = 200$, but the recall is more often equal to 100%. Fig. 4.8 shows for regular bins the precision (Fig. 4.8a), the recall (Fig. 4.8b), the F1-score (Fig. 4.8c) and the number of clusters (Fig. 4.8d). Fig. 4.9 shows for adaptive bins the precision (Fig. 4.9a), the

recall (Fig. 4.9b), the F1-score (Fig. 4.9c) and the number of clusters (Fig. 4.9d). Multiple $\epsilon$ values (DBSCAN parameter) are tested in $[0.1 \times b, 0.2 \times b, ..., b]$.



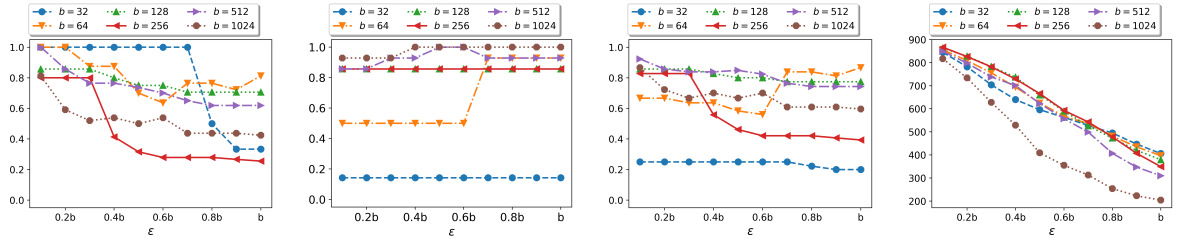

(a) Precision. (b) Recall.

Figure 4.7: Impact of parameter $m$ on the precision and recall, for 512 adaptive bins.




(a) Precision. (b) Recall. (c) F1-score. (d) Number of clusters.

Figure 4.8: Regular bins: precision, recall, F1-score and number of clusters (*BotFP-Clus*).




(a) Precision. (b) Recall. (c) F1-score. (d) Number of clusters.

Figure 4.9: Adaptive bins: precision, recall, F1-score and number of clusters (*BotFP-Clus*).

### DBSCAN $\epsilon$ choice

We use the $\ell 1$-norm as distance function; it increases linearly with the number of elements in the vector. Therefore, taking the parameter $m$ as a constant, we can establish a general relationship between $\epsilon$ and $b$, that is why we chose $\epsilon$ as a fraction of $b$ in Fig. 4.8 and 4.9.

A large value of $\epsilon$ may produce too large clusters resulting in false positives, while a too small value may overfit the data and miss bots. One way to choose a good value for $\epsilon$ can be to take

the one for which the recall and precision are the highest; overall, we favor the recall as we want to detect the most bots as possible. For regular bins, $\epsilon_{reg} = 0.4 \times b$ seems the best choice as the recall is high and we get an acceptable precision and F1-score. For adaptive bins, the best value is $\epsilon_{\text{ad}} = 0.5 \times b$, with the highest recall and a quite good F1-score, for all values of $b$.

#### Comparison between regular and adaptive bins

We observe that formatting the data by handling adaptive bins gives more consistent results and eases the process of clustering. Fig. 4.8b shows the recall for regular bins: we observe that the recall values are quite unstable even with high $\epsilon$ values. For adaptive bins (Fig. 4.9b) on the contrary, the recall oscillates between 85% and 100% (i.e., between 0 and 2 undetected bots) for $\epsilon$ starting from 150 and all values of $b$, which is quite stable.

Only looking to the recall is not sufficient, we also need to know the precision (number of false positives). We observe that using adaptive bins (Fig. 4.9a) presents a far higher precision compared to regular ones (Fig. 4.8a).

For these two reasons, we could confirm the intuition that using adaptive bins grants a more accurate view and therefore leads to better results.

#### General observations

Let us draw further observations from these preliminary results.

- *Benefits of clustering*

In Fig. 4.8 and 4.9, $\epsilon = 0$ is equivalent to not clustering the data, i.e., comparing each host from the test set to labeled hosts from the training set. Fig. 4.8b and 4.9b show that the recall never reaches 100% in this case, no matter $b$ and the quantification technique, as the classification is too specific and we overfit the data. However, increasing $\epsilon$ (then forming larger and larger clusters) enables one to detect all bots in some setups.

Clustering the data also reduces the complexity of the classification, by limiting the number of comparisons to do: we compare hosts from the test set to a limited set of clusters, rather than to all hosts from the training set.

- *Number of bins b*

The number of bins $b$ is the third parameter to choose, which determines our suggested $\epsilon$ values as above discussed. The objective is to find a setup with a recall equal to 100% (i.e., all bots detected) and a precision as low as possible.

Using adaptive bins, the recall reaches 100% for nearly all values of $b$. However there is a strong correlation between $b$ and the precision: the higher $b$, the higher the precision (thus the lower the number of false positives). Therefore the best solution is reached for a high number of bins ($b = 512$), for which the recall is equal to 100% and the precision is high.

- *Number of clusters*

Finally, Fig. 4.8d and 4.9d show the number of clusters respectively for regular and adaptive bins. This shows the benefits in clustering the data: 550 clusters for $b = 512$ (Fig. 4.9d) is approximately 60% less than the 910 initial hosts.

### 4.5.2 Comparison between *BotFP-Clus* and *BotFP-ML*

We compare *BotFP-Clus* and *BotFP-ML* in terms of precision, recall and F1-score. For the former, we use the $\epsilon_{reg} = 0.4b$ and $\epsilon_{\text{ad}} = 0.5b$ settings identified in the previous section. For the latter, we tune the hyperparameters with Gridsearch by favoring the recall. In addition, we show in Appendix B the most relevant features in the classification process.

Fig. 4.10 and 4.11 compare the precision, the recall and the F1-score for *BotFP-Clus* and *BotFP-ML*, respectively for regular and adaptive bins, and for $b$ between 8 and 1024.

Let us look first into *BotFP-ML* algorithms. For Random Forest, both for regular and adaptive bins, the precision and recall values are too unpredictable and varying with $b$. For SVM, it is also too varying for adaptive bins, but for regular ones the precision and the recall are very high (both 93%) starting from $b = 256$. For the MLP classifier and Logistic Regression, the precision and recall values for both types of bins are quite high for all values of $b$. The recall for the MLP classifier is higher for adaptive bins, reaching 90% for 64 bins. For Logistic Regression, there is no notable difference between the use of regular and adaptive bins. In all cases, the recall never reaches 100% no matter $b$ (or with very low precision), which means that there are still some undetected bots.

For *BotFP-Clus*, the precision is very low for regular bins. We observe a correlation between $b$ and the precision and recall: the higher $b$, the lower the precision and the higher the recall. Thus we may opt for the *BotFP-Clus* with a high value of $b$. In particular, for 512 and 1024 bins, the recall reaches 100% (i.e., all bots detected) and 55 to 75% precision. Actually, a precision equal to 75% represents very few false positives: 4 benign hosts classified as bots, out of the 712 benign ones.

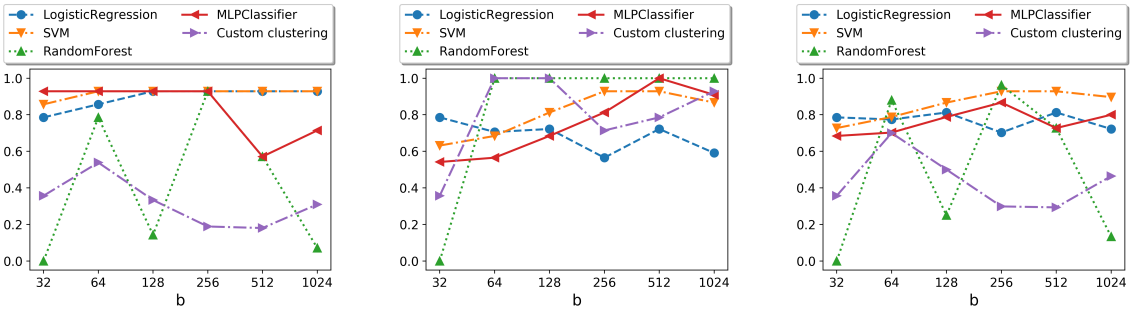
To sum up, we selected four best-performing solutions summed up in Table 4.3 according to the parameter we want to favor. Grey cases show the values for which a given parameter is the best one across all solutions. Let us further comment on the following goals:

- *Maximize true bot detection*: We recommend *BotFP-Clus* with 512 adaptive bins, for which the recall is equal to 100% and the precision to 75%. Contrary to others, this method enables to detect all bots while keeping a good precision;

- *Balance the precision and the recall, thus maximize the F1-score*: *BotFP-SVM* with a linear kernel and 256 regular bins is ideal in this case. The recall and the precision are both equal to 93%. Moreover, this method does not require to compute adaptive bins, therefore is more lightweight than others in this respect;

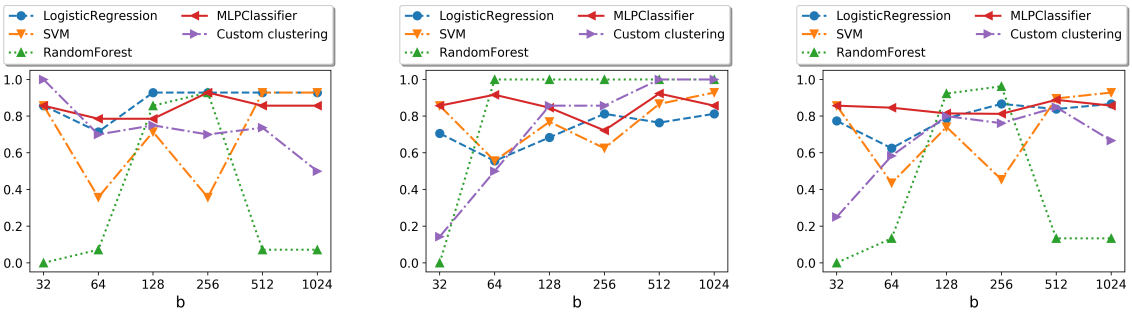
- *Minimize the memory usage*: *BotFP-MLP* with 32 adaptive bins best suits this goal. As $b$ is lower than 256 (see Section 4.3.5), we can apply PCA beforehand to keep only 50 out of the 288 initial factors. The recall and the precision are still good, both equal to 84%. Note that we obtain exactly the same precision and recall values as without applying PCA. Therefore this solution is efficient and above all very lightweight;

- *Maximize precision:* One may choose to favor a high precision, i.e. a low false detection rate ($FDR$) defined as $FDR = 1 - precision$, especially in production environments where administrators want to receive as few alerts as possible. In this case, we recommend *BotFP-Clus* with 512 adaptive bins and $\epsilon = 0.1b$ instead of the $\epsilon$ value previously tuned to favor the recall. Using this setting, the precision is equal to 100%, the recall to 85%, and the F1-score to 93%.




(a) Precision. (b) Recall. (c) F1-score.

Figure 4.10: Regular bins: precision, recall and F1-score for both approaches.




(a) Precision. (b) Recall. (c) F1-score.

Figure 4.11: Adaptive bins: precision, recall and F1-score for both approaches.

Table 4.4 shows the confusion matrix for scenarios 1, 2, 6, 8, 9 from the test set, for the best cases that we identified, i.e. the *BotFP-Clus* algorithm with $b = 512$ adaptive bins and $\epsilon_{ad} = 0.5 \times b$, the *BotFP-MLP* algorithm with $b = 32$ adaptive bins, the *BotFP-SVM* algorithm with $b = 256$ regular bins, and the *BotFP-Clus* algorithm with $b = 512$ adaptive bins and $\epsilon_{ad} = 0.1 \times b$ For *BotFP-Clus* and $\epsilon_{ad}$, we detected bots from all scenarios (1 bot for scenarios 1, 2, 6, 8, and 10 bots for scenario 9), which makes the recall equal to 100%. In total, we labeled 9 benign hosts as bots, which results in a very high precision equal to 74%. For *BotFP-MLP*, 2 bots have not been detected, while it remains 2 false positives. For *BotFP-SVM*, there are only 3 false positives, but it remains one undetected bot. For *BotFP-Clus* and $\epsilon_{ad} = 0.1 \times b$, there are no false positives as we favor the precision, but there are 5 undetected bots.

### 4.5.3 Comparison to state-of-the-art detection techniques

We now compare the four selected *BotFP* settings to other state-of-the-art detection methods, namely BClus [70], CAMNEP [70], BotHunter [69], *BotGM* [75] and [82] described in Section 2.3.

To compare *BotFP* to other methods, we compute the accuracy for scenarios from the test set, as proposed in [70]. Table 4.5 reports the results for each solution and all scenarios from the test set, as proposed in [70]. Our results are very competitive as we reach an accuracy between 97% and 100% with the MLP classifier, SVM, and *BotFP-Clus*, while other algorithms provide an accuracy between 30% and 95%. Only [82] achieves up to 100% accuracy for scenario #9 but it tested only that one and trained on the 12 other scenarios. Note also that our algorithm has

| Solution | Bins type | $\epsilon$ | $b$ | Precision | Recall | F1-score | PCA | Complexity |
|---|---|---|---|---|---|---|---|---|
| *BotFP-Clus* | adaptive | $0.5 \times b$ | 512 | 74% | 100% | 80% | X | low |
| *BotFP-MLP* | adaptive | $0.5 \times b$ | 32 | 85% | 85% | 85% | ✓ | high |
| *BotFP-SVM* | regular | $0.4 \times b$ | 256 | 93% | 93% | 93% | X | medium |
| *BotFP-Clus* | adaptive | $0.1 \times b$ | 512 | 100% | 85% | 80% | X | low |

Table 4.3: Summary of the best solutions according to the detection performance to favour.

| Id | TP | TN | FP | FN |
|---|---|---|---|---|
| 1 | 1 | 164 | 3 | 0 |
| 2 | 1 | 131 | 0 | 0 |
| 6 | 1 | 112 | 0 | 0 |
| 8 | 1 | 163 | 4 | 0 |
| 9 | 10 | 133 | 1 | 0 |

(a) *BotFP-Clus*, $b = 512$ adaptive bins ($\epsilon_{ad}$).

| Id | TP | TN | FP | FN |
|---|---|---|---|---|
| 1 | 1 | 166 | 0 | 0 |
| 2 | 0 | 131 | 0 | 1 |
| 6 | 1 | 111 | 1 | 0 |
| 8 | 1 | 164 | 1 | 0 |
| 9 | 9 | 134 | 0 | 1 |

(b) *BotFP-MLP*, $b = 32$ adaptive bins.

| Id | TP | TN | FP | FN |
|---|---|---|---|---|
| 1 | 1 | 166 | 0 | 0 |
| 2 | 1 | 131 | 0 | 0 |
| 6 | 1 | 111 | 1 | 0 |
| 8 | 1 | 165 | 2 | 0 |
| 9 | 9 | 134 | 0 | 1 |

(c) *BotFP-SVM*, $b = 256$ regular bins.

| Id | TP | TN | FP | FN |
|---|---|---|---|---|
| 1 | 1 | 166 | 0 | 0 |
| 2 | 0 | 131 | 0 | 1 |
| 6 | 1 | 111 | 0 | 0 |
| 8 | 1 | 165 | 0 | 0 |
| 9 | 7 | 134 | 0 | 3 |

(d) *BotFP-Clus*, $b = 512$ adaptive bins ($\epsilon = 0.1b$).

Table 4.4: Confusion matrix for scenarios from the test set, for four *BotFP* settings.

far lower complexity (cf. Section 4.6).

| Metrics | Recall | | | | | Precision | | | | | Accuracy | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Algorithm** | **1** | **2** | **6** | **8** | **9** | **1** | **2** | **6** | **8** | **9** | **1** | **2** | **6** | **8** | **9** |
| *BClus* [70] (2014) | 0.4 | 0.3 | <0.0 | 0.1 | 0.1 | 0.5 | 0.6 | 0.4 | 0.2 | 0.4 | 0.5 | 0.5 | 0.4 | 0.3 | 0.4 |
| *CAMNEP* [70] (2014) | 0 | <0.0 | <0.0 | <0.0 | <0.0 | <0.0 | 0.8 | 0.9 | 0.9 | 0.9 | 0.5 | 0.4 | 0.4 | 0.5 | 0.5 |
| *BotHunter* [69] (2007) | 0.01 | 0.02 | 0.06 | 0 | 0.02 | 0.8 | 0.9 | 0.98 | 0 | 0.9 | 0.4 | 0.3 | 0.38 | 0.42 | 0.4 |
| *BotGM*[a] [75] (2017) | X | X | X | X | X | X | X | X | X | X | 0.91 | 0.78 | 0.95 | 0.89 | 0.83 |
| *Graph-based ML*[b] [82] (2019) | X | X | X | X | 1 | X | X | X | X | 0.91 | X | X | X | X | 1 |
| ***BotFP-Clus*** ($b = 512$) | **1** | **1** | **1** | **1** | **1** | **0.25** | **1** | **1** | **0.2** | **0.91** | **0.98** | **1** | **1** | **0.97** | **0.99** |
| ***BotFP-MLP*** ($b = 32$) | **1** | **0** | **1** | **1** | **0.9** | **1** | **0** | **0.5** | **0.5** | **1** | **1** | **0.98** | **0.99** | **0.98** | **0.99** |
| ***BotFP-SVM*** ($b = 256$) | **1** | **1** | **1** | **1** | **0.9** | **1** | **1** | **0.5** | **0.33** | **1** | **1** | **1** | **1** | **1** | **0.99** |
| ***BotFP-Clus*** ($b = 512$ - $\epsilon = 0.1b$) | **1** | **0** | **1** | **1** | **0.7** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0.99** |

Table 4.5: Recall, precision and accuracy of different algorithms compared to *BotFP*.

[a] Note that BotGM [75] did not provide per-scenario recall and precision values; however, they provided ROC curves showing a TPR (recall) equal to 80% for FPR=0, but we have no information about which scenario they used for this plot.

[b] The training was done on 12 scenarios (including 1, 2, 6 and 8) and the evaluation only on scenario 9.

## 4.6 Complexity

We qualify the space and time complexity of *BotFP*, considering its three steps: attribute frequency distributions computation, training, and classification. We consider that the classification takes all the elements in the test set $\mathcal{E}$, even if hosts may also be processed individually in practice. We also compare it to other recent bot detection methods.

### 4.6.1 Attribute frequency distributions computation

First, we need to compute the fingerprint $\sigma_j$ for all hosts.

**Space complexity.** Given a host and $|A|$ attributes, we need to store arrays of $b$ bins for all the attributes, then the per-host space complexity is equal to $\mathcal{O}(|A| \cdot b)$. The overall process is

a one-shot operation over all hosts, resulting in a complexity $\mathcal{O}(|\mathcal{T} \bigcup \mathcal{E}| \cdot |A| \cdot b)$. In our setting we have $|A| = 9$, which can be reduced to $|A'| = 1$ when using PCA.

**Time complexity.** For a given host $i$, the computation of each attribute vector comes with $|a_i|$ entry readings, before bin aggregation, thus the worst-case time complexity is equal to $O(|A| \cdot max_i|a_i| \cdot |\mathcal{T} \bigcup \mathcal{E}|)$.

### 4.6.2 Training

The training phase depends on the implemented supervised learning algorithm. For *BotFP-Clus*, the training consists of building host clusters from the training set, each host being characterized by its fingerprint $\sigma_j$.

**Space complexity.** For *BotFP-MLP*, a one-dimensional neuron input array grows linearly with the number of neurons, which send their outputs as inputs to a given neuron [145], thus $\mathcal{O}(h)$ with $h$ the number of neurons. The space complexity of *BotFP-SVM* is around $\mathcal{O}(|\mathcal{T}|^2)$ [146]. For *BotFP-Clus*, DBSCAN presents a space complexity of $O(k|\mathcal{T}|)$, where $k$ is a fixed memory cost to store the positions and labels of each among the $|\mathcal{T}|$ points, their labels and the neighbors of the current point.

**Time complexity.** For *BotFP-MLP*, the time complexity of the training (backpropagation) for a single iteration is $\mathcal{O}(|\mathcal{T}| \cdot |A| \cdot b \cdot h^k)$, for $k$ hidden layers containing $h$ neurons. For *BotFP-SVM*, the time complexity of the training is $\mathcal{O}(|\mathcal{T}|^2 \cdot |A| \cdot b)$ [147]. For *BotFP-Clus*, DBSCAN presents a worst-case time complexity of $O(|\mathcal{T}|^2)$ (without the use of an accelerating index structure). For each point of the database, we have to visit each other point to query their neighborhood.

### 4.6.3 Classification

For *BotFP-ML*, the classification technique depends on the implemented supervised learning algorithm. For *BotFP-Clus*, the classification determines the closest cluster to each host to classify, and assign its label to the host.

**Space complexity.** For testing as well, *BotFP-MLP* presents a worst-case space complexity of $\mathcal{O}(h)$ with $h$ the number of neurons. The test space complexity for a linear SVM is $\mathcal{O}(|\mathcal{E}|)$ [148]. For *BotFP-Clus*, we have to store the positions of all clusters. Also for each host, we need to store the distance between its signature and each cluster. Therefore the total space complexity is $\mathcal{O}(|\mathcal{C}| \cdot |A| \cdot b + |\mathcal{E}| \cdot |\mathcal{C}|)$.

**Time complexity.** For a trained MLP, the overall complexity of the classification (forward propagation) is $\mathcal{O}(|\mathcal{E}| \cdot |A| \cdot b)$ [149]. For a trained SVM, the overall complexity of the classification is $\mathcal{O}(|\mathcal{E}|^3)$ [147]. For *BotFP-Clus*, we need to parse all hosts from the test set, then to compare each of them to all clusters with a $\ell 1$-norm, thus the time complexity is equal to $\mathcal{O}(|A| \cdot b \cdot |\mathcal{E}| \cdot |\mathcal{C}|)$.

### 4.6.4 Comparison to other techniques

The time complexity of *BotFP* feature computation is therefore linear with the number of nodes, then the training is linear for *BotFP-Clus*, quadratic for *BotFP-SVM* and up to exponential for *BotFP-MLP*. For classification, if one considers that it would in practice run in runtime on a per host basis (i.e., $|\mathcal{E}| = 1$), *BotFP-Clus* and *BotFP-MLP* are linear with the number of hosts, knowing that $|\mathcal{C}| < |\mathcal{T}|$, hence very competitive; and *BotFP-SVM* time complexity is cubic.

Let us compare the time complexity of our method to recent bot detection techniques [75, 82], for each main step:

*Features computation: BotGM* [75] creates graphs of communications between two hosts as features to feed their algorithms. The time complexity is thus $\mathcal{O}(|\mathcal{P}| \cdot |\mathcal{E}_\mathcal{P}| \cdot ln(|\mathcal{V}_\mathcal{P}|))$ with $\mathcal{P}$ the set of IP addresses pairs, $\mathcal{E}_\mathcal{P}$ the set of edges per pair (i.e., the set of source-destination port pairs per IP pair) and $\mathcal{V}_\mathcal{P}$ the set of vertices per pair (i.e., each communication from one pair to another). For [82], graphs of communications for all possible source-destination IP addresses are computed, which generates a complexity of $\mathcal{O}(|\mathcal{P}| \cdot ln(|\mathcal{V}|))$ with $|\mathcal{V}|$ the set of vertices between each edge in $|\mathcal{P}|$. Then the complexity needed to compute features over the graphs is different depending on the feature, and is up to quadratic for features like Betweenness Centrality. Both [75] and [82] compute features for each pair of hosts, while we work on individual hosts which thus implies linear instead of quadratic processing.

| Algorithm | Features | Training | Classification |
|---|---|---|---|
| *BotGM* [75] | $\mathcal{O}(\|\mathcal{P}\| \cdot \|\mathcal{E}_\mathcal{P}\| \cdot ln(\|\mathcal{V}_\mathcal{P}\|))$ | X | $\mathcal{O}(\|\mathcal{P}\|^2 \cdot \max(n_1, n_2)^3)$ |
| *Graph-based ML* [82] | $\mathcal{O}(\|\mathcal{P}\| \cdot ln(\|\mathcal{V}\|))$ | $\mathcal{O}((\|\mathcal{T}\| + k) \cdot \|A\| \cdot b + \|A\| \cdot b \cdot k \cdot \|\mathcal{T}\|)$ | $\mathcal{O}(\|\mathcal{E}\|)$ |
| *BotFP-Clus* (512 adaptive bins) | $O(\|A\| \cdot max_i\|a_i\| \cdot \|\mathcal{T} \bigcup \mathcal{E}\|)$ | $\mathcal{O}(k\|\mathcal{T}\|)$ | $\mathcal{O}(\|A\| \cdot b \cdot \|\mathcal{E}\| \cdot \|\mathcal{C}\|)$ |
| *BotFP-MLP* (32 adaptive bins) | $O(\|A'\| \cdot max_i\|a_i\| \cdot \|\mathcal{T} \bigcup \mathcal{E}\|)$ | $\mathcal{O}(\|A'\| \cdot b \cdot \|\mathcal{T}\| \cdot h^k)$ | $\mathcal{O}(\|A'\| \cdot b \cdot \|\mathcal{E}\|)$ |
| *BotFP-SVM* (256 regular bins) | $O(\|A\| \cdot max_i\|a_i\| \cdot \|\mathcal{T} \bigcup \mathcal{E}\|)$ | $\mathcal{O}(\|A\| \cdot b \cdot \|\mathcal{T}\|^2)$ | $\mathcal{O}(\|\mathcal{E}\|^3)$ |

Table 4.6: Time complexity of different bot detection algorithms.

*Training:* [75] uses an unsupervised method thus does not require training. For [82], techniques used for training are quite heavy as they use unsupervised (mainly clustering algorithms including $k$-Means which is NP-hard) then unsupervised (various classifiers) learning algorithms which are heavy too. Looking at the classification results of their algorithm, we assess that they better use $k$-Means followed by DecisionTree. $k$-Means is known to have a quadratic time complexity $\mathcal{O}(|\mathcal{T}|^2)$, and the standard decision-tree has a time complexity of $\mathcal{O}(|A| \cdot b \cdot k \cdot |\mathcal{T}|)$, with $N$ the number of training examples and $d$ the depth of the decision tree.

*Classification:* The classification step in [75] is very costly, they compute for each possible pair of graphs ($O(|\mathcal{P}|^2)$ the GED which is NP-hard [150]. Thus the total time complexity is $O(|\mathcal{P}|^2 \cdot \max(n_1, n_2)^3)$ with $n_1$ and $n_2$ the number of elements in the two graphs to compare. They compute these distances for each possible pair of graphs, then $|\mathcal{P}|^2$ times with $n$ the number of graphs. Classification in [82] consists of the same process as training, i.e., unsupervised then supervised learning algorithms. The training and classification phases are simultaneous in $k$-Means and Decision Tree, thus their time complexity is simply $\mathcal{O}(|\mathcal{E}|)$.

Hereafter, Table 4.6 compares the time complexity for recent bot detection techniques, namely [75, 82], and ours.

Above all, *BotFP* is quite lightweight with respect to recent bot detection techniques: it deals with features consisting in vectors, easy to compute, and not graphs. For *BotFP-Clus*, the training is very lightweight, then the classification consists in computing inexpensive $\ell$1-norm distances. For *BotFP-MLP* and *BotFP-SVM*, the training is a bit more complex and depends on the number of layers and nodes. However, as for other algorithms, the training is made only once then the classification step is quite lightweight. It is hard to establish the exact time complexity of other algorithms [75, 82] because we do not know the details of their implementations. We know that [75] works on every possible pair of nodes then draws expensive graphs, and once again studies every possible graph combination. Also, [82] uses expensive centrality measurements.

## 4.7 Conclusion

Botnet attacks are constantly more sophisticated, and this is expected to get even worse with the massive increase of connected objects and virtualized infrastructures. Therefore the quick identification of such bots is crucial to Internet security. Our technique *BotFP* focuses on the detection of botnets that infect thousands of machines and perform malicious actions such as launching port scanning and DDoS attacks. We propose an attribute frequency distribution design to characterize the hosts' communication, where bots exhibit specific behaviors. We design two *BotFP* variants for the training and classification. *BotFP-Clus* clusters similar host signatures of each host to group similar instances of traffic, hence avoiding data overfitting and reducing the complexity. *BotFP-ML* applies a supervised ML algorithm to learn from the signatures and detect new bots. The detection results are very promising, since *BotFP* detected all bots from the CTU-13 dataset with very few false positives, outperforming alternative techniques at the state of the art. With both techniques, *BotFP* achieves an accuracy close to 100% while being very lightweight compared to graph-based techniques. The said variant is chosen according to the parameter we want to favor, such as a high recall or precision, a low complexity, a small number of features.

This chapter ends our two contributions to botnet detection, the first one at the ISP carrier-level and the second one at the network-level. The next chapter goes one step further in network anomaly detection, but this time working on two dimensions strongly impacting the characterization of the elements, not only time but also space. Working on cellular network traffic data, the objective is to identify and model spatiotemporal characteristics of per-mobile application usages, and to propose a methodology to automatically detect unusual phenomena that may occur in a city. In this context, we are able to detect a wider variety of events and not only botnets.

# Chapter 5

# Group anomaly detection in mobile apps usages

Analyzing mobile apps communications can unleash significant information about the current social and infrastructure states. A wide variety of events can engender unusual mobile communication patterns that may be studied for pervasive computing applications, e.g., in smart cities. Among them, local events, national events, and network outages can produce anomalies in the mobile access network load. We focus on the detection of such anomalies by decomposing cellular data usage features time series, then detecting first raw anomalies and then grouping them in a spatiotemporal convex hull, further refining the anomaly detection logic, using an algorithmic approach we propose. We can so unveil details about mobile events timeline, their spatiotemporal spreading, and their impacted apps, by clustering them into broad categories. We apply our technique to extensive real-world data and open source our code. By linkage with ground-truth special events that happened in the observed period, we show how our methodology is able to detect them. We also evidence the existence of five main categories of anomalies, characterizing them. Finally, we study how the nature of impacted apps evolves temporally and spatially during an event.

## 5.1 Introduction

Through mobile apps, a telecom operator can witness a wide variety of anomaly events happening, ranging from local (e.g., football matches or concerts) to national events (e.g., political happenings), passing through network outages at both access and Internet/cloud levels, and app updates and failovers. Often, the detection of such events can allow reconfiguring the network to work around the negative effect a given event can have in terms of infrastructure overload and app interruption. Understanding the spatiotemporal characteristics of mobile app usages anomaly and related impacted apps can therefore be useful for both the mobile network operator and application providers.

First, the network operator can anticipate the demands specific to a special event known to happen in advance, and take into account its spatiotemporal dynamics (e.g., during a match there is more traffic around the stadium, while before and after it, the supporters coming from and going back home being split in different neighborhoods). Knowing in advance the typology of impacted mobile app related to a given category of events can support efficient resource allocation. For instance, it can be useful to allow more bandwidth for streaming apps during bank holidays and lock-downs. The operator can also distinguish between known and unknown

events, and in the latter case can understand the root causes and take countermeasures.

Second, understanding and predicting per-app spatiotemporal traffic peaks can serve existing and forthcoming pervasive computing applications [151, 87]. For example, navigation systems could avoid predicted congested areas, and social apps could lead people to or away from the crowds. Other apps include emergency plans, accident or crime zone prediction, and the inference of points of interest. The characterization of neighborhoods based on mobile app usages could also allow for real-time maps exploitable for the advertisement or construction industries.

In this chapter, we propose a group anomaly detection methodology named *ASTECH* for Anomaly SpatioTEmporal Convex Hull. We define a 3-step algorithm that isolates and groups raw singleton (per: app, cell, time-slot, feature) anomalies as follows:

- It decomposes the features time series, extracts the residual component, and detects anomalies as usage drops and peaks through the use of z-score.

- Anomalies are thus grouped into snapshots, that represent the network state at a given place and time. Upon spatiotemporal categorization of anomalies, it groups the abnormal snapshots both spatially and temporally to get spatiotemporal convex hulls, we call *group anomalies*. This enables us to qualify them, unveiling details about their timeline, spatiotemporal spreading, and impacted apps.

- It applies a further clustering to the set of detected group anomalies, to partition them into categories.

We apply it to extensive real-world data collected by a major mobile network operator in a medium-size city from France over three months. We show how our algorithm is able to detect known and unknown anomalies impacting the mobile traffic data during these three months. In particular, leveraging on additional ground-truth event information, we evidence the existence of five categories of anomalies: *local events*, *national events*, *outages*, *bank holiday*, and *app updates*. We also found specific typologies of impacted apps for each category inducing a traffic increase. For the sake of reproducibility, our source code is publicly available at [152].

The contributions can be summarized as follows.

- We propose a novel methodology to detect group anomaly in mobile apps usage data and classify them via per-app profiles. Contrary to previous approaches, the information about app profiles enables us to classify events in addition to detecting them.

- As a result, we are able to identify and group the anomalies which impacted mobile traffic in a given area during a given period. Such anomalies include local events, national events, app failovers or updates, Internet outages, and bank holidays. We rely on three main characteristics to characterize an event: the variation of traffic (i.e., more or less traffic than usual), its spatial spreading, and the list of impacted apps during the event.

- We map a specific set of impacted apps to a given category of anomaly. For events impacted by a single app due to failovers or updates, we simply map the specific app to the category of events. Other events usually cover a whole set of impacted apps. In this set, we investigate if there is a recurrent subset of apps impacted by a given category of events, i.e., if there is a typology of abnormal apps specific to each category. For local and national events, a specific typology was observed depending on the event category, for example messaging and streaming apps for matches; however, there is no typology for bank holidays and Internet outages as there is less traffic than usual thus all of the apps are (more or less) impacted.

- We study the evolution of the sets of impacted apps for a given event, through different places and times. As per *iii)*, for a category of events there is a specific impacted apps typology, however, they vary depending on the time and place for a given event. We observed two distinct patterns of temporal variations depending on the event category: for bank holidays, there is a rotation between break/commuting and working hours; for local events, we observe a commuting pre/post event regime, and another regime with messaging and streaming apps during the event. We identified no specific pattern of spatial variations.

The chapter is organized as follows. Section 5.2 introduces the dataset we employed in our study. In Section 5.3, we describe our methodology. In Section 5.4 we focus on the extraction of raw anomalies through time series decomposition. Section 5.5 presents how we extract and refine group anomalies. In Section 5.6, we present results from the numerical evaluation, with a detailed characterization of detected events. Finally, Section 5.7 concludes the chapter.

## 5.2 Measurements and dataset

In this section, we describe the dataset we employed.

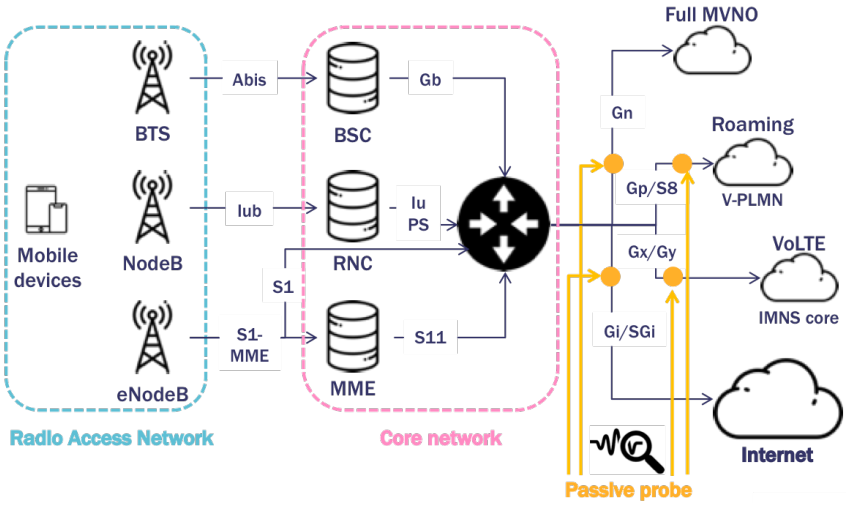



Figure 5.1: Simplified 2G/3G/4G network with passive probes.

Our dataset was collected at the mobile core for user sessions by Orange, a major European mobile operator, in the frame of the CANCAN project [153]. These data, collected at the Gateway GPRS Support Node, allow knowing the app used during an Internet connection (e.g., mail, video streaming, VoIP, gaming) and the amount of data consumed. Raw data are organized in TCP sessions with timestamps and cell locations of the start and end of the session. We aggregate the TCP sessions per 30-minute timeslots, for each Voronoi cell (composed of co-located base stations), each app, and each feature. Our final data set is thus composed of a set of time series (see Section 5.3.2). Fig. 5.1 shows all capture points of the probe; in our collection, we monitor Orange France traffic only, i.e., seen at Gi/SGi and Gx/Gy interfaces.

The dataset describes the mobile traffic generated by the Orange subscriber base in the area around Saint-Denis, a medium-size city from France. It covers almost 3 months, from Mar. 16, 2019, to June 6, 2019. The time frame allows capturing the vast majority of mobile traffic dynamics, which are known to occur over weekly timescales, while avoiding that the dataset size becomes unmanageable.

## 5.3 *ASTECH* Methodology

In this section, we introduce our methodology to detect group anomalies from mobile app usage data, which we name *ASTECH* for Anomaly SpatioTEmporal Convex Hull detection. We present the network model and the main parameters used.

### 5.3.1 Algorithmic approach

For the sake of presentation, let us illustrate our methodology with the reference map of the space covered in our tests, the city Saint-Denis (Paris suburbs); it is a medium-sized city containing various land uses including a residential area in the north district, a large employment area (called *La Plaine Saint-Denis*) in the south section, and the *Stade de France* (a national stadium). First and foremost, we compute the Voronoi tessellation from the list of the antennas' coordinates. The Voronoi tessellation of a set of points is dual to its Delaunay triangulation, so that each cell in the diagram is adjacent to its neighbors. Fig. 5.2 depicts the Voronoi tessellation of the studied area. The color of each cell of the diagram varies with the number of packets during the 3-month period that we study, from yellow for the smallest number of packets to purple for the largest one, and passing through green then blue.

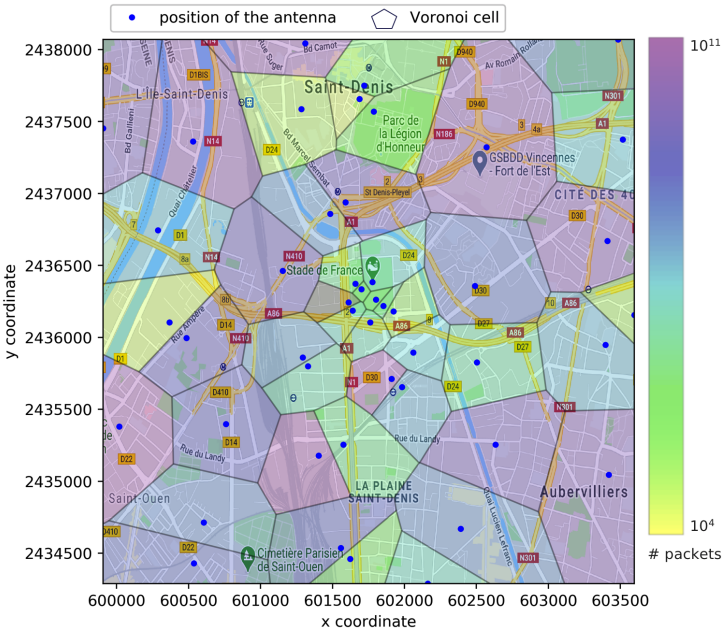



Figure 5.2: Voronoi tessellation of the Saint-Denis area. The color gradient represents the amount of packets accounted in the cell over the 3-month period.

Different variables and dimensions characterize spatiotemporal mobile app usage data: besides space and time interval (structured in Voronoi cells and 30' slots in our setting), based on which one can aggregate both temporally and spatially, apps can be grouped together, and that as a function of multiple features. In order to scale with these dimensions while taking into consideration useful variables for anomaly detection, we propose the following algorithmic 3-stage approach:

- **Step 1**: Time series decomposition, extraction of residual component and anomaly detection;

- **Step 2.1**: Aggregation of multi-source anomalies into snapshots;
- **Step 2.2**: Detection of group anomalies via a spatiotemporal grouping of abnormal snapshots;
- **Step 3**: Classification of group anomalies into several categories through feature-based clustering.

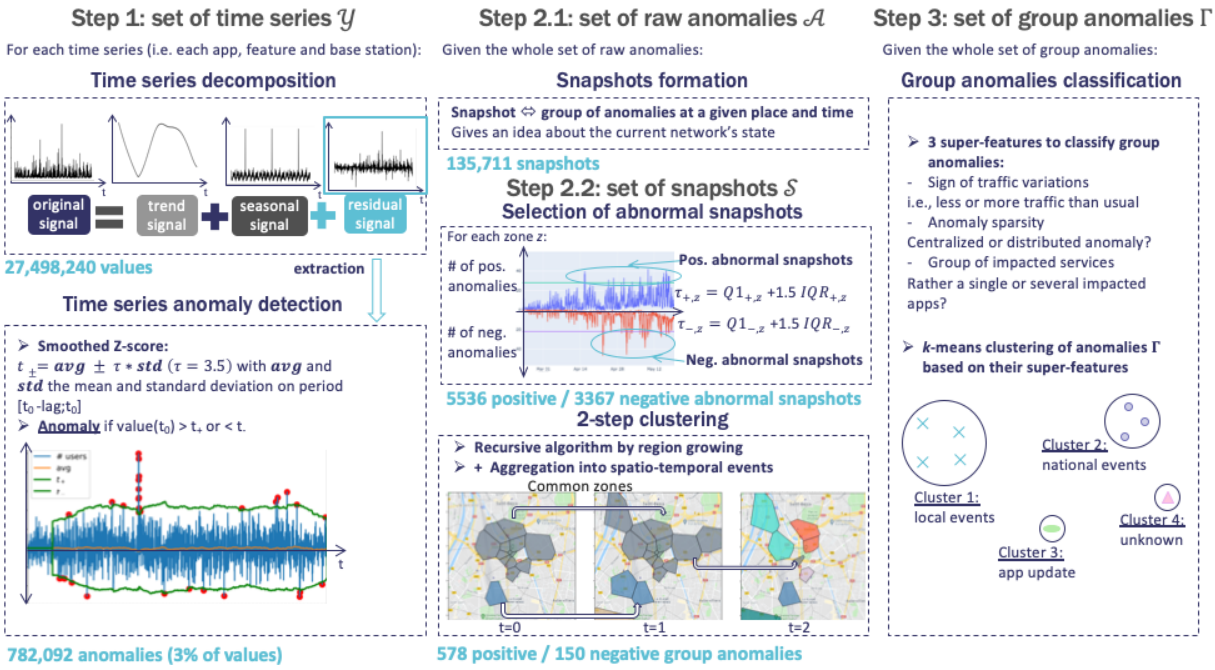



Figure 5.3: ASTECH processing steps. **Step 1**: collection of time series, one for each unique app, feature, and Voronoi cell; extraction of the residual component over which a change point detection (z-score) algorithm is applied to collect sudden time series changes leading to a set of anomalies. **Step 2.1**: anomalies are aggregated into snapshots, then **Step 2.2** only the most abnormal ones are aggregated into spatial groups, which are in turn grouped into spatiotemporal groups that we call *group anomalies*. Finally, **Step 3** provides the categorization of such group anomalies through three super-features, by using the $k$-means clustering algorithm.

Fig. 5.3 depicts the group anomaly detection methodology steps as well as its four components. We then describe each step in detail in the following two sections. We describe hereafter and in Table 5.1 the notations we use in the chapter.

### 5.3.2 Notations

Let $\Phi$ be a set of $n$ distinct base stations (BSs) on the Euclidean place $\mathbb{R}^2$. At a given antenna site, there are in practice several co-located antennas that cover different frequencies and especially 2G, 3G, and 4G: we assimilate such a group of co-located antennas to one single BS in this work.

The Voronoi diagram is a standard cell model for cellular networks; let $V(\Phi)$ denote a subdivision of the plane into $n$ cells such that: *(i)* each cell contains exactly one BS; *(ii)* letting $z \in \mathbb{R}^2$ be a variable space point that lies within a given cell of BS $X_i \in \Phi$, then the polygon representing all locations on the plane closer to $X_i$ than to any other point in $\Phi$ can be formed

| Variable | Description |
|---|---|
| $\Phi$ | Set of $n$ distinct base stations |
| $V(\Phi)$ | Voronoi diagram of $\Phi$ |
| $F$ | Set of features |
| $A$ | Set of apps |
| $T$ | Set of timeslots |
| $y^f_{c,a}$ | Time series for feature $f$, Voronoi cell $c$ and app $a$ |
| $\mathcal{R}^f_{c,a}$ | Residual component of time series for feature $f$, cell $c$ and app $a$ |
| $\mathcal{R}^f_{c,a}$ | Residual component of time series $y^f_{c,a}$ |
| $\mathcal{Y}$ | Set of original time series |
| $\mathcal{R}$ | Set of residual components from original time series |
| $\mathcal{A}$ | Set of anomalies |
| $\sigma_{c,t,*}$ | Snapshot of Voronoi cell $c$ at timeslot $t$ and traffic variation $*$ in $\{+,-\}$ |
| $\Gamma$ | Set of group anomalies |

Table 5.1: Notations

by imposing that $|z - X_i| < |X_j - X_i| \; \forall X_j \in \Phi, \forall X_i \in \Phi | X_i \neq X_j$. The Voronoi cell of the BS $X_i$ can so be denoted by:

$$c_i := \{z \in \mathbb{R}^2 : \; |z - X_i| \leq \min_{X_j \in \Phi} |X_j - X_i|\} \tag{5.1}$$

Two Voronoi regions are said to be *1-order neighbors* if they share boundaries of the Voronoi region.

Let $T$ be the set of time slots[1], and $A$ be the set of apps that can be associated with each session by the provider. We compute features on a per app ($a \in A$) and per cell ($c \in V(\Phi)$) basis; the features $f$ in $F$ we use are: the number of users; the uploaded traffic volume, measured as the total number of uploaded packets; the downloaded volume, measured as the total number of downloaded packets.

It is worth noting that we also tested the round-trip-time as an additional feature; however, its consideration in the group anomaly detection methodology revealed having no impact, because noticeable round-trip-time variations are much more seldom events than traffic volume and number of users dynamics. Also note that the apps can be detected by the mobile network operator via application port, Deep Packet Inspection, or fingerprinting techniques; in our tests, thousands of apps are used - to limit the number, we selected the top 40 apps totaling 80% of the traffic volume[2].

[1] In our tests, we use 30-minute time-slots in the period from Mar. 16, 2019, to June 6, 2019, where $n = 4320$.

[2] The list of selected apps from the most to the least used one is: Default HTTP 80, TLS, Facebook, iCloud Storage, iCloud command, Encrypted Videos, Google API, SnapChat, WhatsApp, Facebook Stream, Google Web, Instagram Videos, Web Advertising, Mail Microsoft, Youtube TLS, Apple Web, Apple push, Twitter, Google Play Store, Web Audience, Google NAV, Spotify, MMS, Other 443, e-Commerce, LinkedIn, Spotify, Youtube, Twitter Videos, browser downloads, iMessage Certificate, Facebook Messenger, HTTPS MAIL, Google+CDN, Images Web, NewsPaper, Web Microsoft, SIRI, Weather, Mail Google.

## 5.4 Time series anomaly detection

We further develop our methodology to detect raw anomalies from feature-based time series. We first decompose time series into several components (*sect. 5.4.1*). We then apply process the residual component derived from the decomposition to detect raw single-ton anomalies (*sect. 5.4.2*).

### 5.4.1 Time series decomposition

For each app $a \in A$ and each Voronoi cell $c$ in $V(\Phi)$, we denote the time series of feature $f$ in $F$ as: $y^f_{c,a} = \{y^f_{c,a,0},\ y^f_{c,a,1},\ ...,\ y^f_{c,a,n}\}$. Let $\mathcal{Y}$ be the set of time series $y$, defined for each feature $f$ in $F$, each Voronoi cell $c$ in $V(\Phi)$ and each app $a$ in $A$.

$$\mathcal{Y} = \{y^f_{c,a}\}, \forall f \in F, \forall c \in V(\Phi), \forall a \in A \tag{5.2}$$

The decomposition of time series [154] is the process of deconstructing a time series into several components, each representing one among many possible underlying pattern categories. A trend component and a cycle component are usually combined into a single trend-cycle component (often called the trend). We adopt a conventional set of three components, as follows:

- $\mathcal{T}^f_{c,a}(t)$, the trend-cycle component at time $t$, which reflects the long-term progression of the series (the secular variation), while the cyclical component reflects repeated but non-periodic fluctuations. The trend component may be linear or non-linear.
- $\mathcal{S}^f_{c,a}(t)$, the seasonal component at $t$, reflecting seasonality (seasonal variation). A seasonal pattern exists when a time series is influenced by seasonal factors. Seasonality occurs over a fixed and known period (e.g., the day of the week, the hour of the day).
- $\mathcal{R}^f_{c,a}(t)$, the residual component at $t$, which describes random, irregular influences. It represents the residual or remainder of the time series after the other components are removed.

Two types of decomposition are commonly used. The additive decomposition is:

$$y^f_{c,a}(t) = \mathcal{T}^f_{c,a}(t) + \mathcal{S}^f_{c,a}(t) + \mathcal{R}^f_{c,a}(t) \tag{5.3}$$

while the multiplicative decomposition is written as:

$$y^f_{c,a}(t) = \mathcal{T}^f_{c,a}(t) \times \mathcal{S}^f_{c,a}(t) \times \mathcal{R}^f_{c,a}(t). \tag{5.4}$$

The additive decomposition is appropriate if the magnitude of the seasonal fluctuations does not vary with the level of the time series. When the variation in the seasonal pattern appears to be proportional to the level of the time series, then a multiplicative decomposition is more convenient [155]. In our case, we use the additive decomposition because we do not experience a strong trend-cycle component that would significantly amplify the whole signal.

#### Decomposition technique

Let us review the main techniques for time series decomposition, to then justify the one we use.

The classical time series decomposition method originated in the 1930s, and widely used then, is the *Moving Average (MA)* one [154]. It is used as the basis of many time series decomposition methods. It uses as metric to qualify the trend-cycle at $t$ the average value of the time series, and that for $k$ periods of time. Observations that are nearby in time are also likely to be

close in value. Therefore, the average eliminates some of the randomness in the data, leaving a smooth trend-cycle component. This technique does not fit well data containing outliers. The trend component is computed as the moving average over the time series, then in the case of an outlier, the trend gets very high during a whole sliding window after the beginning of the outlier. This may perpetrate false positives, i.e., a very high amount of traffic, then an artificial drop in traffic (see more details in subsection 5.6.1).

Another time series decomposition techniques are those using month-granularity metrics such as X11 and SEATS. The X11 method originated in the US Census Bureau and Statistics Canada [155], and the SEATS ("Seasonal Extraction in ARIMA Time Series") decomposition was developed at the Bank of Spain and is now widely used by many government agencies [156]. The procedure works only with quarterly and monthly data, thus hourly, daily or weekly data would require an alternative approach.

Finally, another time series decomposition technique is the *Seasonal and Trend decomposition using LOESS (STL)* [157]. LOESS stands for LOcally Estimated Scatterplot Smoothing. This method estimates nonlinear relationships by combining multiple regression models based on $k$-nearest-neighbor models. It fits simple models, such as linear least squares regression, to localized subsets of the data that confer the flexibility of nonlinear regression. Taking into account the locality thus describes the deterministic part of the variation in the data, point by point. It presents several advantages over the MA, X11, and SEATS decomposition methods:

- Unlike X11 and SEATS, STL handles any type of seasonality, not only monthly and quarterly data.
- The change rate for the seasonal component as well as the smoothness of the trend-cycle can be chosen by the user.
- Contrary to other methods, STL can be robust to outliers, so that unusual observations do not affect the estimates of the trend-cycle and seasonal components. This alternative STL version uses LOWESS [158] (Locally Weighted Scatterplot Smoothing), that re-weights data when estimating the LOESS using a data-dependent function. Using the robust estimation allows the model to tolerate larger anomalies in the original signal.

We apply the robust *STL* decomposition to the time series denoted $y^f_{c,a}$ with the periodicity set to $7 \times 48 = 336$ (for 7 days multiplied by 48 30-minute timeslots in a day), to take into account the hourly and daily seasonality occurring during one week. We then retain the residual component $\mathcal{R}^f_{c,a}$ of $y^f_{c,a}$ in order not to be influenced by seasonal and trend variations. The whole set of residual components computed from the set $\mathcal{Y}$ of time series is written $\mathcal{R}$. Using the residual component rather than the original one can: *(i)* first, avoid seasonal anomalies, e.g., positive anomalies during rush hours; (ii) second, accentuate anomalies when framed in their context. We develop such benefits in Sect. 5.6.1.

### 5.4.2 Detection of raw anomalies

We detect the activity peaks and drops in the $\mathcal{R}^f_{c,a}$ time series using the *z-score* metric[15]. It compares the original signal versus its z-score, and tags elements whose absolute values are greater than the threshold as anomalies. The algorithm exploits the principle of dispersion: if a new data point is a given $x$ number of standard deviations away from some moving mean, it is marked as an anomaly. It takes three parameters as inputs: the **threshold** $\tau$, computed as the z-score at which the algorithm produces an anomaly; the **lag** $l$, computed as the number of

past samples in the one-week sliding window. We set $\tau$ to 3.5 as recommended in [15]. As we use time series with a 30-minute granularity in the weekly seasonality, the lag is equal to $48 \times 7 = 336$.

Let $Z(t)$ be the z-score at time $t$ computed as:

$$Z(t) = (|\mathcal{R}^f_{c,a}(t)| - \overline{\mathcal{R}^f_{c,a}})/\nu, \tag{5.5}$$

with $\overline{\mathcal{R}^f_{c,a}}$ and $\nu$ respectively the mean and standard deviation computed over the list of $\mathcal{R}^f_{c,a}$ values from $t-1-l$ to $t-1$. If $|Z(t)|$ is strictly greater than $\tau$, an anomaly can be denoted by the 5-tuple $(t, c, f, a, Z(t))$ as composition of several attributes: its intensity equal to $Z(t)$ which can be positive or negative (i.e., there is respectively an increase or a drop in the given app usage), the timeslot $t$, the Voronoi cell $c$, the feature $f$ and the app $a$.

Note that we handle time series only above a given number of non-null samplings. For example, some cells (like those covering the stadium stands) are active only when an event occurs. If there are too few values in the time series, the learning period is not relevant and this may produce false positives. We then apply the z-score only when the learning period contains at least 30 values (out of the 336 timeslots during one week), corresponding to 15 hours coverage on the week-wide window. We chose 30 as the least value because, the side cases with 30 quasi-consecutive non-null samplings (15 hours) can cover one full day, or two long evenings, which we esteem sufficient enough for detecting only major anomalies lasting hours. By applying this filter, we go from 848,688 to 92,940 anomalies, eliminating those ones that were caused due to too few values in the time series.

The set of raw anomalies $\mathcal{A}$ found by applying the z-score on the set of residual time series $\mathcal{R}$ is thus:

$$\begin{aligned} \mathcal{A} = \{(t, c, f, a, Z(t)) \; \forall t \in T \text{ if } |Z(t)| > \tau\}, \\ \forall \; \mathcal{R}^f_{c,a}(t) \in \mathcal{R}, c \in V(\Phi), f \in F, a \in A \end{aligned} \tag{5.6}$$

We refer to the elements of $\mathcal{A}$ as raw anomalies because related to a single app and not yet grouped, which is covered by the next steps.

## 5.5 Group anomalies

In this section, we present our methodology to detect group anomalies, meant as groups of raw anomalies that are spatiotemporally adjacent, i.e., they form what we call a 'convex hull'of anomalies adjacent in time and space. First, we pass through a grouping into 'snapshots', then grouped in turn spatially and temporally to form group anomalies.

### 5.5.1 Identification of abnormal snapshots

A snapshot is defined as a group of anomalies pertaining to the same timeslot $t$ and Voronoi cell $c$. We distinguish between snapshots of positive and negative anomalies, i.e., the anomalies whose z-score is respectively positive and negative. A snapshot $\sigma_{c,t,*}$ is thus defined by four elements:

- the Voronoi cell $c$;
- the timeslot $t$ (30-minute in our tests);
- the operator $*$ in $\{+, -\}$, respectively for positive and negative anomalies;

- the set of impacted apps denoted $\Sigma_{c,t,*}$.

Our objective is to identify 'abnormal snapshots', we mean as those ones with an abnormally high number of anomalies. To identify extreme values in our dataset, we use the interquartile range on the snapshot cardinality. Above a threshold we set to the conventional 1.5 times the interquartile range, we spot snapshots with an abnormally high number of anomalies in a given cell as abnormal ones.

### 5.5.2 Detection of group anomalies

Our objective is to identify group anomalies, defined as a group of abnormal snapshots in a spatiotemporal convex hull, that is a compact spatiotemporal area of adjacent cells and time slots: we group together abnormal snapshots that are close in terms of space and time. We propose a 2-phase grouping process: *(1)* group nearby abnormal snapshots at the same timeslot to create spatial groups, then *(2)* group spatial groups which are temporally close in order to form spatiotemporal events, the so-called **group anomalies**.

**Phase 1: Creation of spatial groups.** The goal of this phase is to form groups of nearby abnormal snapshots at timeslot $t$. Phase 3 below shows the recursive process to form spatial groups at $t$ from the set of abnormal snapshots $\{\Sigma\}_{t,*}$. The function *Get_spatial_groups* starts from an abnormal snapshot, then applies a **region growing** algorithm which recursively inspects all of the cells in its neighborhood which are also abnormal, then their own abnormal neighbors, etc, until there are no more abnormal neighboring cells to study. The function in charge of recursively finding all of the recursive neighbors of a cell is denoted *Get_neighbors*. It takes into account: the set of abnormal snapshots $S_{t,*}$ at $t$ of sign $*$, the neighbors to study $\mathcal{N}$ (i.e., if they belong to an abnormal snapshot, we add them to the given group), the list *studied* of cells which have already been inspected (whether they belong to an abnormal snapshot or not) and the dictionary $\mathcal{D}$ which contains the cells in $V(\Phi)$ as keys and their neighbors in Voronoi diagram as values. We then repeat the operation for abnormal snapshots at $t$ which are not yet grouped.

**Phase 2: Creation of group anomalies.** This phase consists in grouping spatial groups at successive timeslots into spatiotemporal group anomalies. Phase 4 depicts the function *Get_group_anomalies* grouping spatial groups to form group anomalies. The objective is to group spatial groups at two successive timeslots that have at least one Voronoi cell in common. If a group at $t$ does not have any cell in common with a group anomaly occurring at $t-1$, then we initialize a new group. From the set of spatial groups $\mathcal{G}$, we obtain the set of group anomalies denoted $\Gamma$. Each group anomaly $\gamma \in \Gamma$ is defined by: its starting timeslot $t_{\text{start}}$, its end timeslot $t_{\text{end}}$ and the list *cells* of the sets of impacted cells, one for each intermediate timeslot.

### 5.5.3 Fine-grained characterization of group anomalies

A precise characterization of the detected group anomalies is needed to understand possible usages. We identified several broad categories of anomalies (including local events, national events, outages, bank holidays, app updates), however, we lack ground-truth labels. Therefore we chose to apply a clustering algorithm to the set of group anomalies, of positive and negative anomalies separately. Clustering algorithms are designed to group similar vectors into clusters and identify isolated ones as outliers. The similarity between two vectors is commonly evaluated using a distance function like the Euclidean distance. Two vectors are defined as similar if they are close to each other, else dissimilar.

**Phase 1**: Creation of spatial groups

**function** GET_NEIGHBORS($S_{t,*}, \mathcal{N}, studied, \mathcal{D}$) ▷ Get all related neighbors of a given cell
  new neighbors to study $new \leftarrow \emptyset$
  **for** neighbor $n \in \mathcal{N}$ **do**
    **if** $n \in$ abnormal snapshots $S_{t,*}$ **then**
      $studied$.add($n$)
      $new$.add($\mathcal{D}[n] - studied$) ▷ We add all of the neighbor's neighbors to inspect them later
  **if** $new$ is empty **then**
    **return** $studied$
  **else**
    **return** get_neighbors($S_{t,*}, new, studied, \mathcal{D}$)

**function** GET_SPATIAL_GROUPS($S_{t,*}$) ▷ Get the spatial groups $\mathcal{G}_t$ at $t$ from the set of abnormal snapshots $S_{t,*}$
  spatial groups $\mathcal{G}_t \leftarrow \emptyset$
  dictionary $\mathcal{D}$: key = cell, value = cell's neighbors
  **for** abnormal snapshot $\sigma_{c,t,*} \in S_{t,*}$ **do**
    $neighbors \leftarrow get\_neighbors(S_{t,*}, \mathcal{D}(c), \emptyset, \mathcal{D})$
    **if** $\exists\ g \in \mathcal{G}_t$ so that $\exists\ n \in neighbors$ belonging to $g$ **then**
      $g$.extend($neighbors$)
    **else**
      $\mathcal{G}_t$.add(Spatial_group($t, neighbors$))
  **return** $\mathcal{G}_t$

**Phase 2**: Creation of group anomalies

**function** GET_GROUP_ANOMALIES($\mathcal{G}$) ▷ Form group anomalies $\Gamma$ from the set of spatial groups $\mathcal{G}$
  group anomalies $\Gamma \leftarrow \emptyset$
  **for** spatial group $g \in \mathcal{G}$ **do**
    $cell\_in\_common \leftarrow$ None
    **for** $c \in g.cells$ **do**
      **if** not $cell\_in\_common$ **then**
        **for** group anomaly $\gamma \in \Gamma$ **do**
          **if** $g.timeslot == \gamma.end\_time$ + 30' **and** $c \in \gamma.list\_cells[-1]$ **then**
            $cell\_in\_common \leftarrow c$
            $\gamma$.update($g.cells, g.timeslot$)
            **break**
    **if** not $cell\_in\_common$ **then**
      $\Gamma$.append(Group_anomaly($g.time\_slot, g.cells$))
  **return** group anomalies $\Gamma$

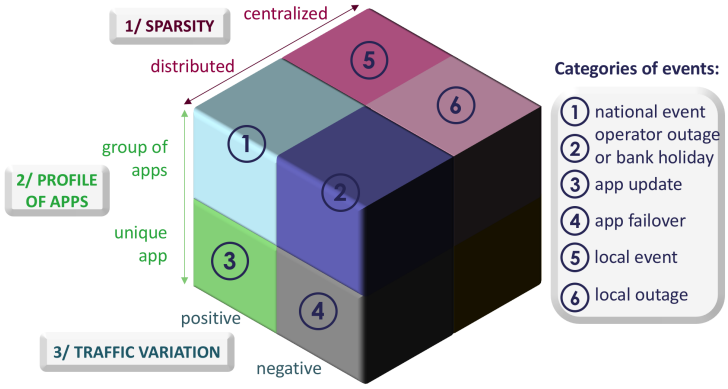
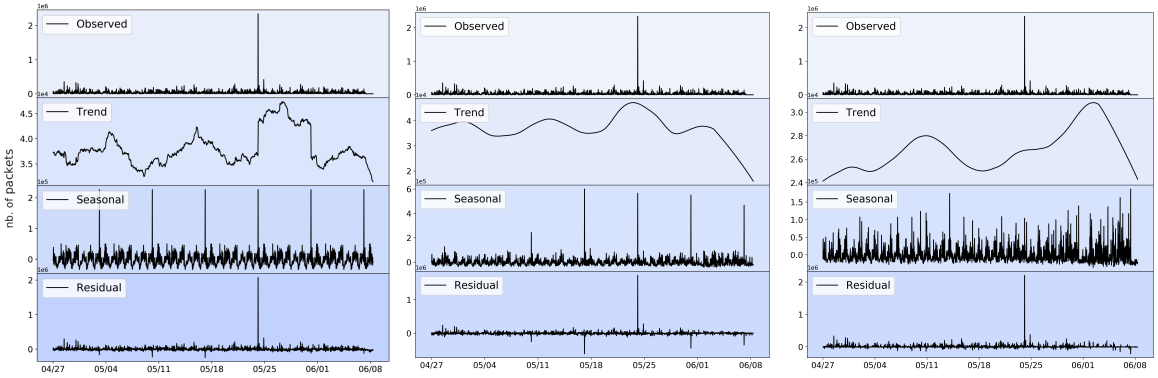



Figure 5.4: 3 super-features to classify group anomalies: *(i)* spatial spreading, *(ii)* profile of impacted apps, *(iii)* variations in mobile traffic; and 6 broad categories of events.




(a) MA decomposition. (b) STL decomposition. (c) Robust STL decomposition.

Figure 5.5: Comparison of time series decomposition techniques: *MA*, *STL*, *robust STL* in a 6-week period.

To cluster group anomalies, we use the $k$-means clustering algorithm. Unlike density-based clustering algorithms as DBSCAN and OPTICS that require the maximal density between data-points, $k$-means only takes the number of clusters as parameter, which is equal to the number of categories. It first divides the input points into $k$ initialization groups, then it associates points with the closest cluster according to the center point of each cluster. Our goal is to find a match between clusters resulting from $k$-means and the broad categories of events that we identified.

Fig. 5.4 illustrates the three super-features we leveraged on to classify group anomalies and the categories of events we observe. We first evaluate the group sparsity (**super-feature 1** in the figure), to determine if the group is rather localized (happening in a specific place) or distributed (happening in many far places at the same time). Then we estimate whether the group covers rather a single app or a whole set of apps (**super-feature 2**). These two first super-features are determined by continuous variables, i.e., the group can be more or less sparse, and more or less split among a group of apps, as meant by the arrows in the figure. The last super-feature is the sign of the traffic variation (**super-feature 3**), i.e., whether it is a group of positive or negative anomalies. This time, this super-feature is binary. According to these three super-features, the list of attributes in the $k$-means algorithm includes:

- for **super-feature 1**, the weighted spatial 2D barycenter of the impacted cells for the

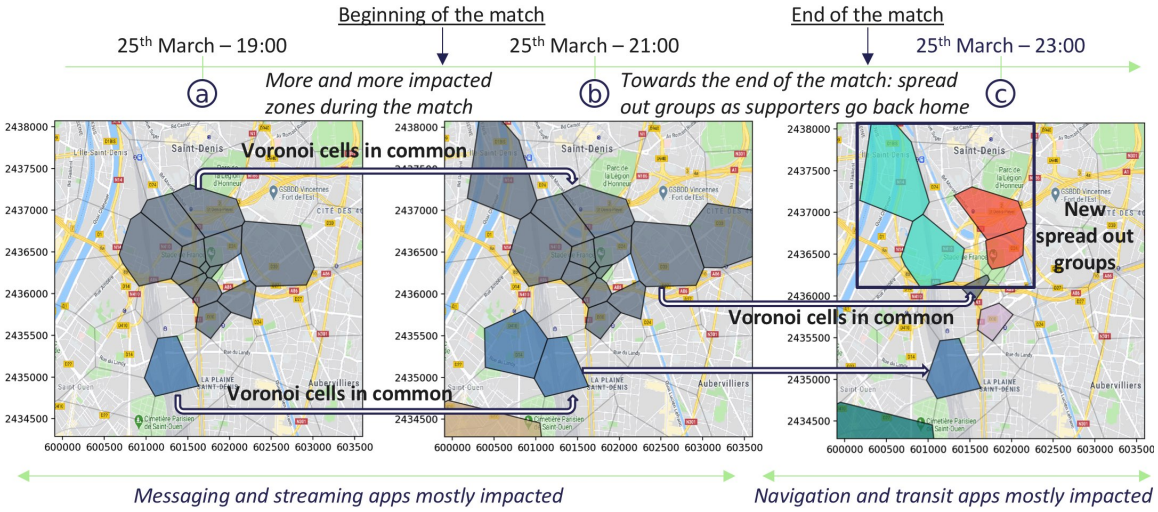



Figure 5.6: Timeline of the football match between France and Iceland on March 25, 2019 in *Stade de France*. Colored cells on the map represent abnormal snapshots at a given instant. A group of adjacent cells of the same color represents a spatial group. Two spatial groups that share at least one cell at two consecutive timeslots form what we call a **group anomaly**, represented by the same color. New spatial groups at a given instant have a different color.

anomaly;

- for **super-feature 2**, the parts represented by the 2 most recurrent apps in the set of anomalies and a one-hot encoded vector containing the 3 most recurrent apps during the given anomaly;

- we leverage on **super-feature 3** for clustering of positive and negative anomalies, separately.

From the combination of super-features in Fig. 5.4, we spot six broad categories of group anomalies: local event, national event, app update, app malfunction, operator outage/bank holiday, and local outage. Note that there are eight possible combinations from the set of super-features, however, we labeled only six groups categories as the two combinations made from "localized/unique app/positive" and "localized/unique app/negative" were never observed and not quite relevant.

In order to get satisfying results for the clustering, we retain only major group anomalies, using ad-hoc thresholds for the anomalies' spatial spreading. Then, after the clustering, we map unclassified groups (i.e., below the threshold) to spot anomalies occurring at the same time to classify a larger range of anomalies.

## 5.6 Numerical results

In this section, we test the *ASTECH* detection methodology against data related to the Saint-Denis area, in France, for the period from Mar. 16 to June 6, 2019. We first analyze raw anomalies (related to Step 1 in Fig. 5.3, Section 5.4), then group anomalies (related to the other Steps, Section 5.5). Finally, we classify and characterize the found group anomalies. The source code used for the detection and evaluation is available in [152].

### 5.6.1 Raw anomalies

We present results justifying Step 1 of our methodology.

#### Time series decompositions (MA, STL, robust STL)

Fig. 5.5 illustrates the decomposition of the time series composed of the number of WhatsApp downloaded packets overtime, using three decomposition techniques: Moving Average (MA) (Fig. 5.5a), Seasonal-Trend Decomposition using Loess (STL) (Fig. 5.5b), and robust STL (Fig. 5.5c).

As previously discussed in Section 5.4.1, MA can be influenced by extreme values (i.e., strong anomalies) and thus may perpetrate false anomalies during a whole sliding window after a large outlier: in Fig. 5.5a, there is a large increase in traffic on May 23 due to a football match at *Stade de France*; then the trend component is abnormally large during one whole week after this large outlier. The seasonal component is computed by applying a convolution filter to the data (to remove the seasonality) and by computing the average of this smoothed series for each period: in Fig. 5.5a, we notice that it is usually large on Saturday because of the large outlier produced by the football match on a single evening.

With STL, the seasonal component changes over time at a rate controlled by the user. Therefore, we observe in Fig. 5.5b that the seasonal component is impacted during a couple of Saturdays next to the Saturday of the match. The trend is also lightly impacted by the large outlier. Finally, some drops in traffic occur on Saturday evenings before and after the match in the residual series, which are false positives induced by the match.

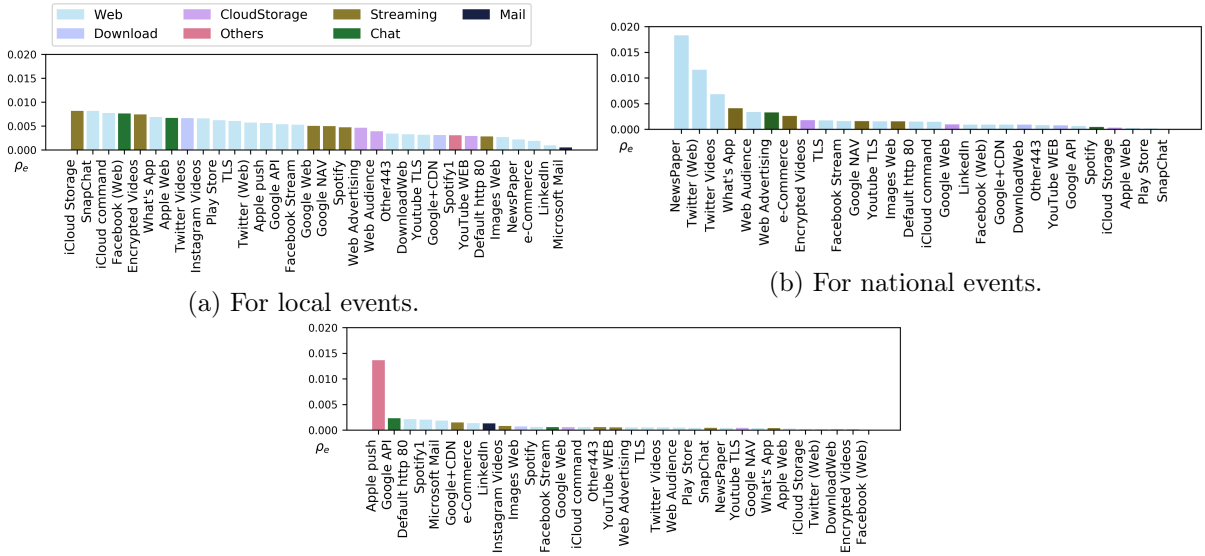
With respect to the robust STL decomposition (using LOWESS), in Fig. 5.5c, the trend component is not impacted by the outlier, the seasonal component is recomputed each week and thus the residual component does not contain any false positive. Given that we observe these patterns for every time series in the dataset, as anticipated we choose the robust STL as it proved to be robust against outliers.

#### Advantages in using the residual signal

As seen in Section 5.4.1, we handle the residual component instead of the original one in our analysis. This presents several advantages. First, we avoid anomalies caused by seasonality, i.e., anomalies produced by seasonal variations like traffic peaks during rush hours and traffic drops during weekends. In addition, this enhances sudden traffic variations and makes the anomalies more visible (i.e., the z-score of the detected anomalies is even greater when using the residual component). Table 5.2 shows the number of positive anomalies, i.e., anomalies whose z-score is greater than the threshold $\tau$, and the number of negative anomalies, i.e., anomalies whose z-score is lower than $-\tau$. $\tau$ has been set to 3.5 (cf. Section 5.4.2). We observe less positive anomalies when using the residual series rather than the observed one, as anomalies during rush hours are less visible when put in the context. Therefore using the residual component significantly reduces the number of false positives. On the contrary, there are no negative anomalies when using the observed component, for all time series (no matter the feature, the app, and the Voronoi cell). This means that there is no significant drop in the original series. If we lower $\tau$ to 1.5, we notice drops at night, when there is almost no traffic anymore, but still no drops during bank holidays and massive outages. However, using the residual series enables one to emphasize these variations and thus to produce negative anomalies during outages and bank holidays. We observe 137,954 negative anomalies using the residual component. In total, we spotted 782,092 anomalies, which represents 2.8% of the total number of values in the time series equal to 27,498,240.

| Component | # of positive anomalies | # of negative anomalies |
|---|---|---|
| **Observed** | 915,710 | 0 |
| **Residual** | 644,138 | 137,954 |

Table 5.2: Number of positive and negative anomalies, respectively for the observed and residual components.




(a) For local events.

(b) For national events.

(c) For app updates.

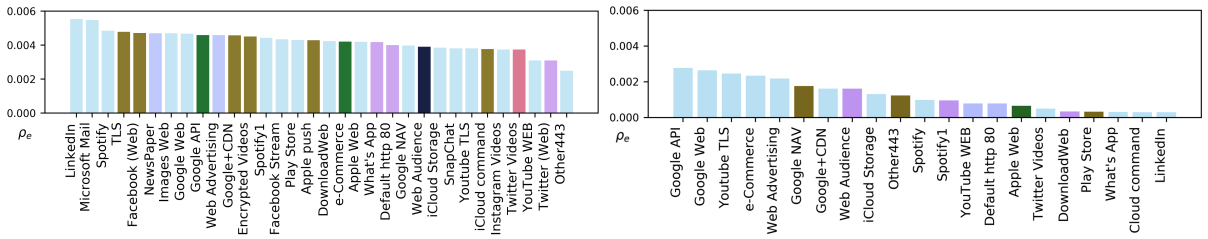
Figure 5.7: $\rho_e$ coefficient of each app for different categories $e$ of groups of **positive** anomalies.

### 5.6.2 Group anomalies

We analyze group anomalies built from raw anomalies.

#### Representation of a given event

Fig. 5.6 represents the timeline of the football match between France and Iceland in *Stade de France* on Mar. 25, 2019, generating group anomalies. First, in **(a)**, at 19:00, before the match, we observe several impacted cells centered around the stadium. The colored Voronoi cells represent cells whose snapshot is abnormal at that time (i.e., with an abnormally high number of anomalies). At that time there are two spatial groups composed of adjacent cells of the same color (blue and grey). The most impacted apps are messaging apps (like WhatsApp and Messenger) and streaming ones (like iCloud Streaming and Storage). In **(b)**, two hours later, at 21:00, we observe spatial groups of the same blue and grey colors: they represent the evolution in time of the same two group anomalies. The number of impacted cells slightly increases as the match starts, while the nature of the impacted apps remains the same. In **(c)**, two hours later, the match is ending and supporters go back home. We then observe new group anomalies: groups of a different color appear, spread out in the city, and not centered around the stadium anymore. This time, the most impacted apps are transit and navigation apps (like Uber, Google Maps).




(a) For bank holidays.

(b) For outages.

Figure 5.8: $\rho_e$ coefficient of each app for different categories $e$ of groups of **negative** anomalies.

### Typology characterization of abnormal apps

We now investigate whether there is a typology of the impacted apps specific to each category of events (e.g., local event, national event). Our first intuition is to study for each category the support of each app. The support of app $a$ for category $e$ is computed as the ratio of instances from $e$ that contain at least one anomaly produced by app $a$. However, there may be a correlation between the app popularity (i.e., the number of packets for the given app) and the number of anomalies produced by this app. Therefore the support of a given app may be high because the app is highly used in general (thus often produces anomalies), and not because it is highly impacted during this category of group. Therefore, instead of computing the support of app $a$, we use the following probability term for the group category $e$: $\Pr(a \in \mathcal{A} \mid e) / \Pr(a \in \mathcal{A})$.

This way, we measure how it is likely that $a$ belongs to the set of impacted apps knowing that the category of events is $e \in E$, where $E$ is the set of categories, divided by the probability that $a$ belongs to the set of impacted apps no matter the category of events. We now measure, for a category $e$ of events, the average of the aforementioned metric for all events of this category. We define the rarity coefficient $\rho_e$ for the category $e$ of events as:

$$\rho_e = \frac{1}{|E|} \sum_{e \in E} \left\{ \frac{\Pr(a \in \mathcal{A} \mid e)}{\Pr(a \in \mathcal{A})} \right\} \tag{5.7}$$

Fig. 5.7 shows the value of the $\rho_e$ coefficient for each app within each category $e$ of groups of **positive** anomalies. The color of each bar represents the category of app, as defined by Orange. During *local events* (Fig. 5.7a), a whole subset of apps is impacted, including streaming, web, and chat apps. During *national events* (Fig. 5.7b), the subset of impacted apps is composed of NewsPaper and Twitter (Web). During *app updates* (Fig. 5.7c), there is clearly a single impacted app which is Apple push. Therefore for groups of positive anomalies, we can induce a typology of the impacted apps for each category of groups. Local events (matches or concerts), national events (like the Notre-Dame de Paris fire), and app updates have a different signature, each one being represented by a specific set of apps.

Fig. 5.8 shows, for each category $e$ of groups of **negative** anomalies, the $\rho_e$ coefficient of each app. We observe that during bank holidays (Fig. 5.8a), all apps are (more or less) impacted because they are less used than usual. The three most impacted apps are LinkedIn, Mail Microsoft, and Spotify as they are widely used during working days. For outages (Fig. 5.8b), the most impacted apps come from Google as Google API and Google Web. Therefore, contrary to groups of positive anomalies, we do not notice a specific typology for groups of negative anomalies. Actually, all apps are lightly impacted during network outages and bank holidays as there is a bit less traffic than usual.

**Spatiotemporal pattern analysis of abnormal apps**

We study for a given event how the sets of impacted apps evolve in time and space. The Jaccard distance [159] between $\Sigma_{c,t1,*}$ and $\Sigma_{c,t2,*}$ measures the dissimilarity between the sets of impacted apps from two snapshots of a same group, at two distinct timeslots $t1$ and $t2$:

$$\text{dist}(\Sigma_{c,t1,*}, \Sigma_{c,t2,*}) = \frac{|(\Sigma_{c,t1,*} \cup \Sigma_{c,t2,*}) - (\Sigma_{c,t1,*} \cap \Sigma_{c,t2,*})|}{|\Sigma_{c,t1,*} \cup \Sigma_{c,t2,*}|} \tag{5.8}$$

Similarly, the Jaccard distance between $\Sigma_{c1,t,*}$ and $\Sigma_{c2,t,*}$ measures the dissimilarity between the sets of impacted apps from two snapshots of a same group at $t$, in two distinct cells $c1$ and $c2$:

$$\text{dist}(\Sigma_{c1,t,*}, \Sigma_{c2,t,*}) = \frac{|(\Sigma_{c1,t,*} \cup \Sigma_{c2,t,*}) - (\Sigma_{c1,t,*} \cap \Sigma_{c2,t,*})|}{|\Sigma_{c1,t,*} \cup \Sigma_{c2,t,*}|} \tag{5.9}$$

Following the evolution of such distances through time and space enables one to identify temporal and spatial patterns for the evolution of the sets of impacted apps for any group. Fig. 5.9a and Fig. 5.9b show the **temporal** evolution of the sets of impacted apps respectively for the match on Apr. 27 in cell *Cornillon Stade* and during Easter on Apr. 22 from 07:30 to 19:00 in cell *Saint-Denis canal*. Fig. 5.10 shows an example of the **spatial** evolution of the sets of impacted apps for the concert on May 13, 2019 at 00:30. The heatmaps are matrices of Jaccard distances between every possible combination of times for a given cell for the **temporal** evolution, and of cells for a given time for the **spatial** one. The darkest areas highlight two sets of very similar impacted apps.

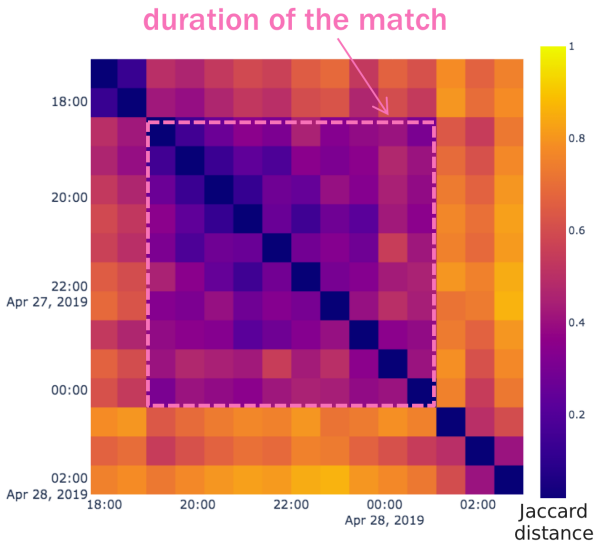
We first investigate the spatiotemporal evolution of *local events* such as concerts and matches. For the **temporal** evolution (Fig. 5.9a), we observe that the sets of impacted apps are very close to each other during the heart of the match, i.e., between 19:00 and 00:00. Before and after the match, the dissimilarity is greater. We find this pattern for a majority of *local events*: the similarity within sets of apps slightly increases, then is stable during the event, then slightly decreases at the end of the event. For the **spatial** evolution (Fig. 5.10), we observe some cells very similar to one another and quite dissimilar to the others. Some cells located around the stadium (the ones covering the stands) are very similar in terms of impacted apps in comparison to those farther away in the suburbs.

We then study the spatiotemporal evolution of **bank holidays**. For the **temporal** evolution (Fig. 5.9b), we notice an interesting pattern: first, we note some dark squares during commuting hours; the time ranges [08:00; 09:30], [12:30; 14:00], and [17:30; 19:00] exhibit very similar sets of apps, including mostly Spotify and social networks. Then, the working hours [09:30; 12:30] and [13:00; 16:00] exhibit very similar sets of apps, with mail servers and LinkedIn mainly impacted. This heatmap shows drops in usage during Easter; meaning that these apps are less used than during normal working days. We observed this pattern for a majority of groups happening during bank holidays. For the **spatial** evolution, we do not observe any specific pattern in this case.

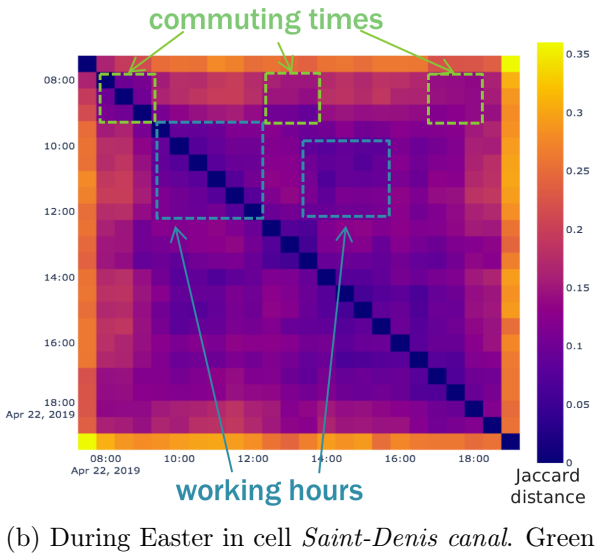
To sum up, we show that what our algorithm detects is the exceptional behavior from users, that we consider anomalous. We can then observe through these visualizations the extent in time and space of these types of behaviors, and for example notice localized events, their timeline, population displacement, etc.

**Spatiotemporal spreading**

Fig. 5.11 shows the number of groups with a given mean number of impacted cells over time (on the x-axis) and a given duration (on the y-axis). The width of the circle gives the number of such




(a) During the match on Apr. 27 in cell *Cornillon Stade*. The pink square highlights the heart of the match, while the impacted apps are very close to each other.




(b) During Easter in cell *Saint-Denis canal*. Green squares highlight dark areas showing similar patters during different commuting hours, while blue ones highlight similar patterns during different working hours.

Figure 5.9: Heatmap of the Jaccard distances between every possible combination of two timeslots.

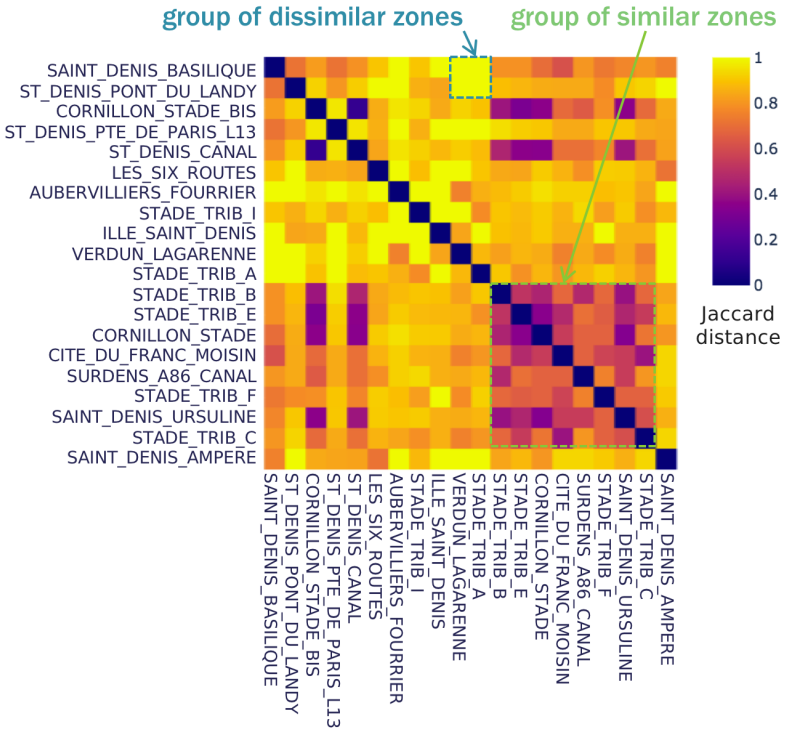


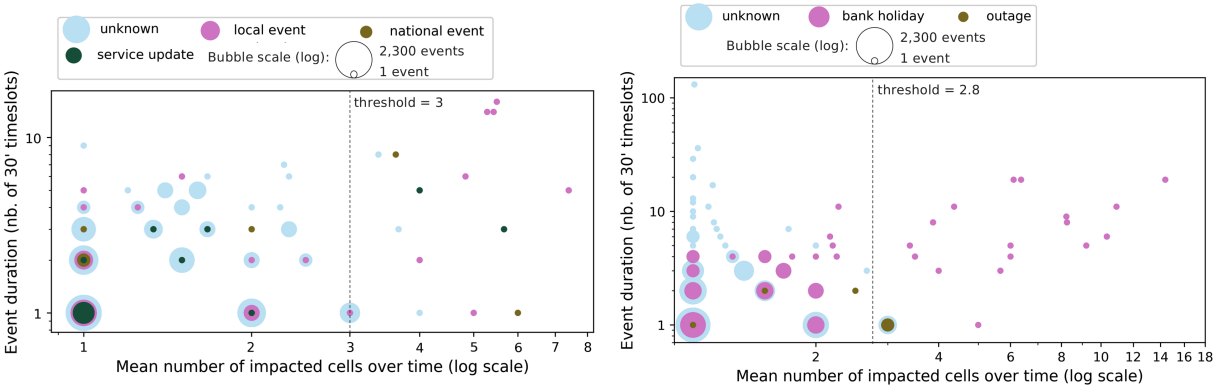

Figure 5.10: Heatmap of the Jaccard distances between every possible combination of two cells, on May 13, 2019 at 00:30.

groups for each couple of coordinates, in logarithmic scale. The smallest circle represents one group of anomalies, while the largest one around 2,000 groups. This representation enables one to distinguish between insignificant groups (e.g., with one targeted cell on average and lasting one timeslot) and meaningful groups (e.g., spatially and/or temporally distributed).

For groups of positive anomalies (Fig. 5.11a), we observe that the group anomalies the most spatially and temporally distributed (upper right on the figure) are mostly known events (including 7 local events, 2 national events, and 2 app updates) and 3 unknown events. Overall, we notice that the most relevant characteristic of known events (i.e., not labeled as *unknown*) is the spatial spreading (i.e., the mean number of impacted cells over time on the x-axis). In fact, strictly above **3** impacted cells on average, most events are known. We later use this value in order to retain only the most meaningful events. We notice that such known events are more or less temporally distributed, between 1 and 11 timeslots, thus the temporal spreading is not a sufficient criterion for retaining the meaningful events.

For groups of negative anomalies (Fig. 5.11b), the events labeled as *bank holiday* distinguish themselves through their high mean number of impacted cells. In addition, the three groups labeled as *outage* are also recognizable and only 1 unknown event exceeds two average impacted cells. We retain the group anomalies with at least **2.8** impacted cells on average. This time, known events that are spatially spread last at least 2 timeslots (i.e., 1 hour) most of the time.

We use these two thresholds the remaining of the chapter to retain only the most meaningful group anomalies, both for positive and negative ones.




(a) For groups of positive anomalies.

(b) For groups of negative anomalies.

Figure 5.11: Group anomaly duration vs. mean number of impacted cells.

### 5.6.3 Group anomalies classification

Table 5.3 sums up the whole list of group anomalies that we detected in our dataset, containing mobile traffic data in Saint-Denis during the period from Mar. 16, 2019, to June 6, 2019. For each group we indicate: the date, the name, the broad category (i.e., the ground-truth label), and the number of the $k$-means cluster to which it belongs (see next paragraph). Table 5.3a covers the groups of **positive anomalies**, with at least 3 impacted cells on average. We detected the totality of the events that took place at the national stadium during this period [160], including the UEFA football match between France and Iceland, the final game of the French football cup, and the Metallica concert. In addition, we detected the Notre-Dame de Paris fire and app updates from Apple push. We detected 3 unidentified anomalies as well. Table 5.3b lists groups of **negative anomalies**, with at least 3 impacted cells on average. We detected all of the bank

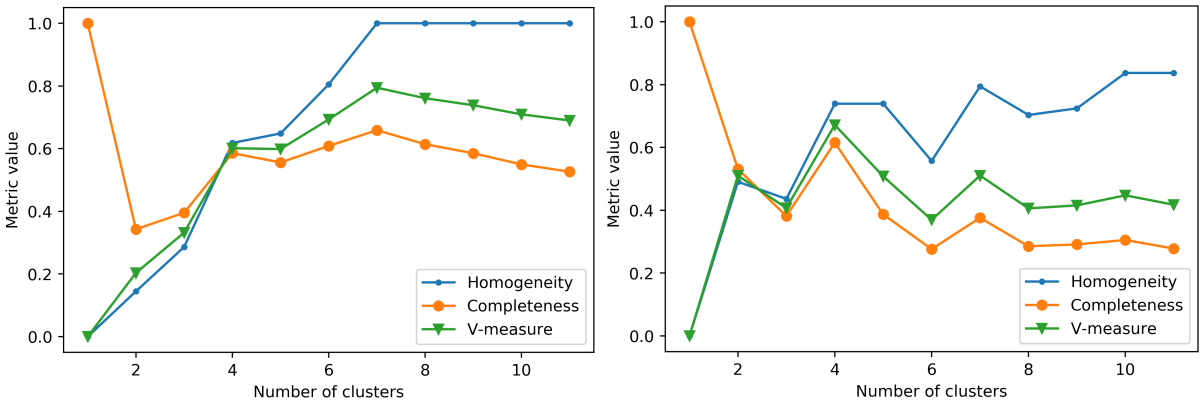
| Date | Event | Ground-truth label | #cluster |
|---|---|---|---|
| Mar. 25 17:30→00:00 | UEFA match | local event | c1 |
| Apr. 9 16:00→17:00 | Apple Push notification | app update | c2 |
| Apr. 15 19:00→22:30 | Notre-Dame de Paris fire | national event | c3 |
| Apr. 15 20:00 | Notre-Dame de Paris fire | national event | c3 |
| Apr. 24 16:30→18:30 | Apple Push notification | app update | c2 |
| Apr. 27 16:00→23:30 | French football cup | local event | c1 |
| Apr. 28 00:00→02:00 | French football cup | local event | c1 |
| Apr. 28 01:30→02:00 | French football cup | local event | c1 |
| May 12 15:00→22:00 | Metallica concert | local event | c1 |
| May 12 22:30→01:00 | Metallica concert | local event | c5 |
| May 13 01:00 | Metallica concert | local event | c6 |
| May 15 09:30 | Unidentified | unknown | c7 |
| May 15 15:00→18:30 | Unidentified | unknown | c7 |
| May 23 18:00→19:00 | Unidentified | unknown | c7 |

(a) Groups of **positive** anomalies, with $k = 7$ clusters.

| Date | Event | Ground-truth label | #cluster |
|---|---|---|---|
| Apr.15 12:00 | Orange outage | outage | c1 |
| Apr.15 12:00 | Orange outage | outage | c1 |
| Apr.22 07:30→10:00 | Easter | bank holiday | c2 |
| Apr.22 10:00→19:00 | Easter | bank holiday | c2 |
| Apr.22 11:00 | Easter | bank holiday | c2 |
| Apr.22 07:00→16:00 | Easter | bank holiday | c2 |
| May 1 07:00→16:00 | Labour Day | bank holiday | c2 |
| May 1 14:30→15:30 | Labour Day | bank holiday | c2 |
| May 1 15:30→19:30 | Labour Day | bank holiday | c2 |
| May 8 07:30→09:30 | VE Day [a] | bank holiday | c2 |
| May 8 08:00→11:30 | VE Day | bank holiday | c2 |
| May 8 11:00→12:30 | VE Day | bank holiday | c2 |
| May 8 12:00→14:00 | VE Day | bank holiday | c2 |
| May 8 14:00→19:00 | VE Day | bank holiday | c2 |
| May 8 19:00 | VE Day | bank holiday | c2 |
| May 27 12:00 | Unidentified | unknown | c1 |
| May 29 18:30 | Unidentified | unknown | c1 |
| May 30 07:00→12:00 | Ascension day | bank holiday | c2 |
| May 30 07:30→08:30 | Ascension day | bank holiday | c2 |
| May 30 07:30→16:30 | Ascension day | bank holiday | c2 |
| May 30 14:00→17:30 | Ascension day | bank holiday | c2 |
| May 30 17:30→19:30 | Ascension day | bank holiday | c2 |
| May 30 18:30→20:00 | Ascension day | bank holiday | c2 |
| June 2 01:00 | Unidentified | unknown | c3 |
| June 2 20:30 | Unidentified | unknown | c4 |
| June 3 22:00 | Unidentified | unknown | c3 |

(b) Groups of **negative** anomalies with $k = 4$ clusters. [a]: Victory in Europe Day.

Table 5.3: List of group anomalies detected in Saint-Denis from Mar. 16 to June 6, 2019, for those ones respectively containing at least 3 or 2.8 impacted cells on average over time.




(a) For groups of positive anomalies.

(b) For groups of negative anomalies.

Figure 5.12: Homogeneity, completeness and V-measure values depending on the number $k$ of clusters in the $k$-means algorithm.

holidays occurring during this period and in particular Easter, Labour Day, Victory in Europe Day, and Ascension Day. We also detected an outage on the Orange 4G network on Apr. 15 [161]. Finally, there are 5 anomalies that we could not identify. The unidentified anomalies on June 2 and 3 could be related to the massive outage from Google Cloud [162].

We now perform the clustering operation on the most meaningful events that we retained above the ad-hoc thresholds for the group spatial spreading defined in Section 5.6.2. Given the knowledge of the ground-truth class assignments of the samples, three key related metrics reflect the quality of a clustering operation: the homogeneity, the completeness, and the V-measure. The homogeneity is a measure of the ratio of samples of a single class pertaining to a single cluster. The fewer different classes included in one cluster, the better. The completeness measures the ratio of the member of a given class that is assigned to the same cluster. The V-measure is computed as the harmonic mean of the homogeneity, and the completeness. We choose the number $k$ of clusters in the $k$-means algorithm so that the V-measure is the largest. Fig. 5.12 shows the homogeneity, completeness and V-measure values depending on $k$. For the groups of positive anomalies (Fig. 5.12a), we set $k$ to 7 for which the homogeneity equals 0.796, the completeness 0.757, and the V-measure 0.776. Note that if we do not know the number of clusters (i.e., the ground-truth labels) in advance, we are not able to tune $k$. Nevertheless, the V-measure is quite high starting from 4 clusters, hence the $k$ value is not essential to get satisfying results. For the groups of negative anomalies (Fig. 5.12b), unsurprisingly the clustering is not efficient as we concluded earlier that there is no specific typology of impacted apps for the groups of negative anomalies, and outages as bank holidays are spatially distributed. We can use the ground-truth on bank holidays to distinguish between both categories of groups. Nevertheless in Table 5.3b we set the clusters numbers for $k = 4$ clusters for which the V-measure is equal to 0.68, the homogeneity to 0.78 and the completeness to 0.6.

To understand the composition of the 7 $k$-means clusters for groups of positive anomalies (while there are 4 ground-truth labels in our dataset), we indicate the cluster numbers assigned to the groups by $k$-means in the last column of Table 5.3a. The homogeneity equals 1, thus instances from different classes never pertain to the same cluster. However, *local events* (especially matches) are split into four different clusters. The first one concerns groups covering the whole match, while the three other ones represent groups at the end of the match when supporters go back home (around 01:00). These latter groups involve only one or two transit apps like Uber

and Google Navigation, thus look like the *app update* cluster which covers a single app. Finally, the categories *national event*, *app update* and *unknown* are well separated into three dedicated clusters. For the groups of negative anomalies (Table 5.3b), four distinct clusters give the highest V-measure value: the first cluster includes the totality of *outages* as well as three *bank holidays* similar to outages. The second one contains all other *bank holidays* and some *unknown* events. Finally, the two last clusters cover the remaining unknown events.

## 5.7 Conclusion

Getting a better understanding of the spatiotemporal dynamics of the group anomalies occurring in a city can help to anticipate the load for such events. We analyzed 3-month real-world mobile traffic data in which we detected the two matches and the concert that occurred in the studied area, the Notre-Dame de Paris fire, a network outage, and the four bank holidays happening during this period.

Our main contributions can be summarized as follows:

*(i)* We showed that there exist specific typologies of impacted apps depending on the anomaly category. We found that local events rather impact streaming and messaging apps, national events impact NewsPaper and Twitter apps, and updates impact the *Apple push* app. However, for groups of negative anomalies (like outages or bank holidays), no specific typology was identified as the anomalies consist of slight decreases in traffic compared to normal working days, thus all apps are (more or less) impacted.

*(ii)* We identified two distinct patterns of temporal variations for the sets of impacted apps: one for local and national events where the similarity slightly increases, then is stable during the anomaly, and slightly decreases at the end; and the other for bank holidays composed of an alternation of commuting/break hours and working hours. We found no explicit evolution pattern in space, but most of the time the Voronoi cells are more or less dissimilar from one another.

*(iii)* We categorized these group anomalies through a $k$-means clustering operation applied to a set of super-features composed of the sign of traffic variations, the spatial spreading, and the profile of impacted apps. We apply a clustering operation to the detected groups of positive anomalies (whose categories are local event, national event, app update and other) and another to the ones of negative anomalies (whose categories are Internet outages and bank holidays). In the best case, the V-measure reaches 0.8 for groups of *positive* anomalies, which means that the clustering is 80% correct compared to the ground-truth labels, and 0.7 for groups of *negative* anomalies.

This concludes our contribution to exploring possible applications of anomaly detection to networking. The next chapter concludes this dissertation.

# Chapter 6

# Conclusion

In this thesis entitled "*Novel anomaly detection and classification algorithms for IP and mobile networks*", we presented three detection systems for network security, traffic prediction, or pervasive computing purposes: *Split-and-Merge*, *BotFingerPrinting*, and *ASTECH*. We conclude this thesis by giving a summary of our contributions (Section 6.1) and proposing several perspectives for each of them (Section 6.2).

## 6.1 Summary of contributions

Each of our contributions provides new insights into anomaly detection and classification approaches, and their relevance to networking. In *Split-and-Merge*, we provided a technique for the early detection of emerging botnets that are slightly spreading on the Internet to infect other hosts. Our technique aims to split the detection process over different network segments to retain only the distributed anomalies at the correlation module. We used a simple yet efficient change-detection algorithm based on a modified Z-score measure to monitor the usage of application ports. We argued how our technique can ensure the detection of large-scale zero-day attacks and drastically reduce false positives. As a result, we detected numerous known and unknown attacks targeting connected objects and servers around the world. Our port-based methodology to detect zero-day attacks, which was presented to the scientific community [102], then extended into a journal article [103], led us to the following question: how could we design a complementary approach, which would now detect botnets directly at the network-level?

We then focused on detecting botnet infected hosts, this time directly within a given network, for example at the enterprise-level. The quick identification of such bots is crucial to Internet security; botnet attacks are constantly more sophisticated, and this is expected to get even worse with the massive increase of connected objects. Our technique *BotFP* aims to detect botnet infected-hosts through a histogram-based algorithm that models host communication, where bots exhibit specific behaviors. Two *BotFP* variants are compared for the training and classification steps. Both techniques are very lightweight compared to graph-based techniques and achieved accuracy close to 100%. We may consider several parameters to favor, such as a high recall or precision, a low complexity, a small number of features, to choose the given *BotFP* variant. We first presented our approach to the scientific community [130], then extended it into a journal article [131].

Finally, we directed our attention to cellular traffic data, for the purpose of finely characterizing the timeline of events occurring in a city, in terms of volume of anomalies and impacted services. Getting a better understanding of such phenomena can help to anticipate the load for

such events. We proposed our *ASTECH* detection solution that aims to detect per-mobile app anomalies grouped into spatiotemporal convex hulls. As a result, we identified broad categories of events through a clustering operation applied to a set of super-features, including national events, local events, service updates and malfunctions, bank holidays, and Internet outages. We also revealed typologies of impacted services depending on the category of event and identified different patterns of temporal and spatial variations for the set of impacted services. Our approach was described and evaluated on 3-month real-world data, in a journal article that is currently under revision [163].

All of these detection systems are available to the scientific community; we make all our code publicly available.

## 6.2 Perspectives

This section concludes our dissertation and presents the future works that we identified for each of our contributions.

### 6.2.1 Detection of zero-day attacks

The objective of this work is to detect the early apparition of botnets, newly exploited vulnerabilities, and other diverse attacks within a given network. However, we noticed that we could obtain a comprehensive list of major botnets that emerged last years by coupling different data sources. For example, some preliminary scans target a limited IPv4 range only and thus are visible only on some datasets. As a result, we found that some attacks detected in one real traffic source (the MAWI dataset), containing both attack traffic and background traffic, were not found in another source containing only attack or unsolicited traffic (the UCSD dataset). We thus believe that it would be interesting in future work to leverage these different data sources in order to describe the landscape of major attacks and botnets in recent years.

As further work, we also plan to implement our proposal in a Software-Defined Networking (SDN) environment, using a controller and several switches running the algorithm. This way, the identified attacks could be mitigated, for example by patching the appropriate services or with network programming. These last years, numerous works have been undertaken for network built-in measurements. As discussed in our book chapter[104], network programmability and emerging technologies such as SDN and Network Functions Virtualization (NFV) can advance techniques for intrusion detection and protection. Combining data plane programming with control-level collaboration, we would propose a model to simplify the detection of large-scale, distributed network attacks. This model would simultaneously reduce the system overhead through direct mitigation at the network edge, and would enrich the detection process with corroboration of evidence from distributed sources. We also identified that a large range of attacks could be detected at the host-level, with no need for collaboration between hosts. We would then look to programmable network monitoring techniques that enable stateful data-plane designs in which it is possible to manipulate packet processing. We would then propose a collaborative SDNFV-based IDPS model that leverages programmable networks for local detection with collaboration between ISPs. This enables a system that combines both detection at the host-level and collaboration between hosts to detect large-scale attacks.

### 6.2.2 Botnet Fingerprinting

In our *BotFingerPrinting* algorithm, we showed that using only the information about 5-tuple flows is a very insightful way to characterize the communications of a host and enables us to efficiently detect bots. We plan as future work to consider additional features including: the payload size, as crafted packets from bots usually have a lower size than usual packets; the TCP flags, e.g., to detect SYN flood; the inter-packet time; the duration of the flows, e.g., the connection to the C&C server might be persistent; and DNS features inspired from [76].

Another meaningful improvement to our system would consist in developing a real-time implementation of our algorithm, which would require a shift from our current offline implementation to an online algorithm. The survey [164] on big data for network traffic monitoring and analysis gives us keys for doing so. Our model would thus run on a sliding time windows basis so that the model regularly updates. The major challenge is to design efficient mechanisms for online analysis of large streams of measurements. The key challenges associated with big data involve data volume, velocity, veracity, and variety. We might need to adapt our approach, favoring the versions of *BotFingerPrinting* that rely on lightweight features, and learning and classification algorithms with low computational complexity.

### 6.2.3 Group anomaly detection in mobile app usages

As for *BotFingerPrinting*, we plan to develop an online algorithm based on our existing system detecting group anomalies in cellular traffic data. Previous steps for doing so would consist in assessing the minimal learning period required to get relevant results, for two key steps of our algorithm, the time series decomposition, and the quartiles and interquartile computations. These computations currently need to learn on the whole data period, i.e., three months. We would also need to choose between stream and batch processing to handle large amounts of data regularly updated. Stream processing is the process of being able to almost instantaneously analyze data that is streaming from one device to another, whereas batch processing is the processing of a large volume of data all at once.

In addition, we would like to extend our algorithm by grouping anomalies even disconnected between each other, i.e., not falling under the same convex hull. Another further work is running the analysis on a geographically larger coverage, with broader tessellation units than the base station Voronoi cell.

Finally, we would also like to explore supervised and semi-supervised learning approaches to enhance the performances of our algorithm. Data labeling typically requires human intervention, which may be very costly and fastidious if there is a large amount of data. Labeling only a subset of data with semi-supervised techniques is thus very practical, while significantly increases the performances compared to no labels at all.

# Publications

**International Journals with peer review**

- A. Blaise, M. Bouet, V. Conan and S. Secci, "Detection of zero-day attacks: An unsupervised port-based approach," in *Elsevier Computer Networks*, vol. 180, pp. 107391, Oct. 2020.
- A. Blaise, M. Bouet, V. Conan and S. Secci, "Botnet Fingerprinting: A Frequency Distributions Scheme for Lightweight Bot Detection," in *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1701-1714, Sept. 2020.

**International conferences with peer review**

- A. Blaise, M. Bouet, V. Conan and S. Secci, "BotFP: FingerPrints Clustering for Bot Detection," *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Budapest, Hungary, 2020, pp. 1-7.
- A. Blaise, M. Bouet, S. Secci and V. Conan, "Split-and-Merge: Detecting Unknown Botnets," *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Arlington, VA, USA, 2019, pp. 153-161.
- A. Blaise, S. Wong, A.H. Aghvami, "Virtual network function service chaining anomaly detection," *International Conference on Telecommunications (ICT)*, Saint-Malo, France, 2018, pp. 411-415.

**Book chapters**

- A. Blaise, S. Scott-Hayward, S. Secci, "Scalable and Collaborative Intrusion Detection and Prevention Systems Based on SDN and NFV," in: J. Rak, D. Hutchison (eds) *Guide to Disaster-Resilient Communication Networks. Computer Communications and Networks. Springer, Cham.*

**Under review**

- A. Blaise, M. Bouet, V. Conan and S. Secci, "Group anomaly detection in mobile app usages: a spatiotemporal convex hull methodology," submitted to *IEEE Transactions on Mobile Computing.*

# Appendix A

# Virtual network function service chaining anomaly detection

Network function virtualization (NFV) and virtual network function (VNF) service chaining are receiving significant attention from both academics and industry. However, most of the attentions has been concentrated on delivering the flexible network architecture and optimization of VNF placement across the network infrastructure. In this chapter, we focus on an important aspect of the network after its architecture is formed and its VNF placements are optimized. This aspect is related to the efficiency and effectiveness of VNF provisioning, the lack of visibilities on the location of VNF, the flexibility of VNF placement, and the VNF splitting into multiple sub-functions. This can be considered as a security issue covering the anomalies of the VNF orchestration and placement during the operation. We propose a VNF service chain anomalies detection method based on the Markov chain property in order to ensure the correctness of VNFs backward and forward placement and the K-means classification of VNF sequence patterns. This method identifies the patterns of the VNF service chaining sequence in correct behavior. This work is not just observing the existing network behavior, it also can be extended to identify the correctness of the sequence order of a new VNF service chaining request[1].

## A.1 Introduction

Fifth-generation (5G) telecommunication systems have focused on building an advanced flexible network architecture that is enabled by network function virtualization and software-defined mobile networking. This newly designed network architecture and infrastructure is: *(i)* expected to have a flexible radio access network (RAN) and to provide the necessary adaptability for handling fluctuations in traffic demands, *(ii)* aimed to support variable network workload, to provide elasticity of the network resources provisioning and deprovisioning in an autonomic manner based on virtualization, and *(iii)* intended to deliver an independent control of the logical network slices and to provide isolated network resources for tenants serving their plethoric network services.

However, the network service deployment and provisioning are traditionally coupled with network topology and physical network resources. Planning the whole provision and further the deployment processes requires a significant amount of time. Presently, programmable network technologies (e.g., OpenFlow, YANG and NETCONF, etc.) enable the network service deployment and provisioning, moving into a new era of decoupling with network topology and physical

[1]This work is issued from the master thesis and was presented at *IEEE ICT 2018* [165]

resources, dynamically creating network service chains on-the-fly [166, 167] and wholesaling multiple level services and network slices to the horizontal and vertical industry customers. A typical example of a network slice tenant would be a vehicle manufacturer or the performing art industry. Nevertheless, a traditional example is a mobile virtual network operator (MVNO) as a tenant.

Furthermore, this new generation of programmable networks opened the Pandora box of telecommunication network infrastructure, allowing the global cloud providers and their customers to easily become international mobile service providers. On the other hand, this programmable network introduces a number of security issues and threats to the telecommunication network. For instance, such behaviors like fast-flux malicious network function chaining requests [168], overloading virtual machine resources, and forcing virtual infrastructure providers (VIP) to create or move other tenant network functions to other virtual machines, are more likely to happen. These security issues could adversely reduce the overall network performance.

We propose a machine learning approach to detect anomalies and to identify the non-conforming patterns of network function generation in the network that can limit and/or throttle the overall network performance. This method is based on a decision-making process to confirm backward and forward VNFs in the right sequence. Precisely, a basic feature of this process is to identify known or unknown anomalous patterns of network functions chaining with sliding windows, whose outcome can trigger an alert to the administrator or even isolate the network function chaining immediately. Various clustering algorithms can be utilized for examining the scatter of different emerging patterns compared to the training set. A pattern recognition algorithm will be considered to identify the normalities and anomalies of the VNFs sequences or requests.

This chapter is organized as follows. Section A.2 discusses the virtual VNF service chaining properties, the formulation of the problem, and the possible scenarios of network function split. Section A.3 details the detection techniques under Markov chain. Section A.4 gives the pattern detection techniques under K-Means. Section A.5 presents the outcomes of our simulations. Finally, Section A.6 concludes.

## A.2 VNF Service Chaining Problematics

VNF service chaining [169, 170] takes an important role in delivering the 5G services. For instance, mobile service provider (MSP) provides a Network Slice as a Service (NSaaS) [171] to a vertical industry tenant. This tenant network slice has a number of different VNF service chains that can be picked from the network slice service catalog. These VNF service chains are the combination of a logical sequence of VNFs or instances of VNFs that provide an end-to-end network service. However, after the network slices are deployed, VNFs can be relocated due to runtime optimization or scale in and out purposes. Even a single network function could be divided into various locations, depending on the characteristics of the network functions and on the runtime network conditions. Especially, in runtime, the VNF service chain could dynamically relocate a specific VNF to another network node due to self-optimization adapting to the traffic demand. Furthermore, a VNF could be split into parallel or series forms in runtime. Fig. A.1 shows an example of a typical VNF service chain and network function split into parallel and series forms.

Nonetheless, these processes can be inadequately executed by a malicious tenant, and therefore the lack of visibilities in the virtual network infrastructure is a major issue. For instance, there is a possibility that a malicious tenant non-stop creates and terminates VNF in the VNF service chain until one of the virtual machine overloads and triggers the self-optimization process. This process might produce a side effect to reduce the overall MSP network performance and

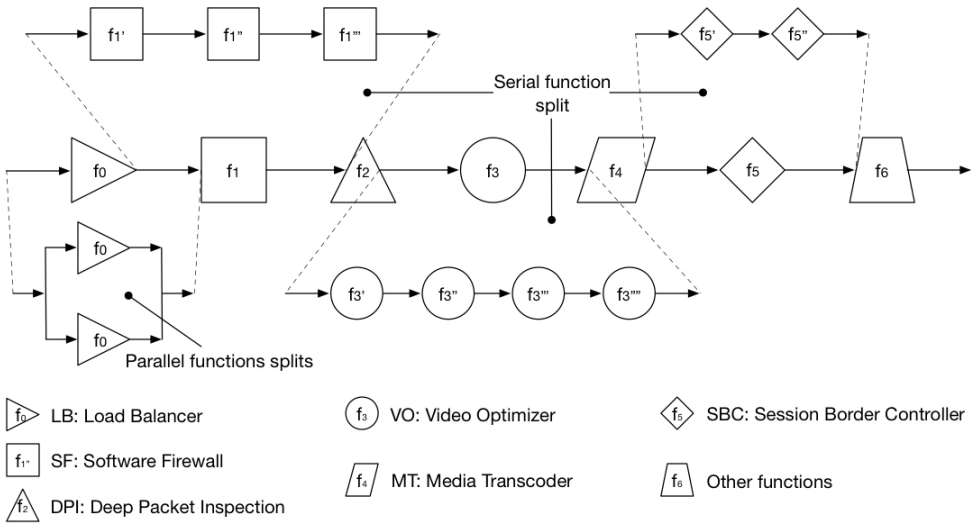



Figure A.1: VNF Service chaining splitting

computation power. In order to limit such malicious acts from tenants, we must increase the visibility of the virtual network infrastructure and continuously observe the tenant and tenant's VNF service chain behaviors. Therefore, we propose a method to ensure that the function before and function after are adequately placed in the VNF service chain and we set up a runtime observation of the VNF service chain behavior based on a sliding window approach when scale in and out or function split processes take place. Fig. A.2 illustrates a network infrastructure with 6 VNFs and 4 VNF service function chains. Based on this topology, the first phase algorithm is to analyze each VNF locally. The video optimizer (VO) is always placed before the session border controller (SBC) but the software firewall (SF) would not be appropriately placed before the SBC. In fact, there is an implicit sequence and logical order relationship between VNFs. Therefore, we can rule out the possibilities of SF being placed before SBC and consider such a chain as malicious, based on the chains we know as correct. In this chapter, we propose a two-phase algorithm to resolve this side effect: the first phase is to ensure the logical place of each VNF in the sequence of the service chain, and the second phase is to obtain the overall behavior of the VNF service chain based on the result of the first phase algorithm. In a software-defined network, the SDN controller has an overview of the whole network and communicates with the other VNFs so that it can easily classify the type and number of network functions and is able to know when new functions are added and old ones removed.

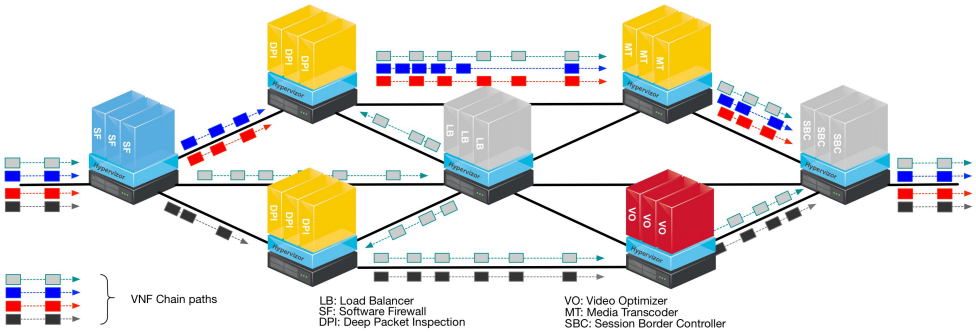



Figure A.2: VNF Service chaining network example

## A.3 VNF Service Markov Chain

A Markov chain is a discrete stochastic process. The Markov property is memoryless so that the future status of the system is only dependent upon the system's present state and is independent of the history of previous events. We apply the Markov chain backward and forward properties to the VNF service chain. Each VNF representing a state in the Markov chain, two transition matrices are used to reflect the link probabilities between the VNFs. By processing with a sliding window, we evaluate the relations between the VNFs and obtain the relevancy of the whole chain. Let $G = (S, E)$ be a graph with a set of states (or VNFs) and edges. Let $S = \{s_1, s_2, .., s_n\}$ be the set of VNFs in the network with $n$ the number of VNFs. Each VNF $s_i$ is considered as a state in the Markov chain. With each pair of states $(s_i, s_j)$ there are given:

- the probability $f_{ij}$ that the VNF $s_j$ is situated right forward of the VNF $s_i$ in a given chain,
- the probability $b_{ij}$ that the VNF $s_i$ is situated right backward of the VNF $s_j$ in a given chain.

The numbers $f_{ij}$, called the forward transition probabilities, and the numbers $b_{ij}$, called the backward transition probabilities, can be arranged in matrices like

$$F = \begin{bmatrix} f_{11} & f_{12} & \dots & f_{1n} \\ f_{21} & f_{22} & \dots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{n1} & f_{n2} & \dots & f_{nn} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}$$

The transition matrices $F$ and $B$ are stochastic matrices so that each $f_{ij}$ and $b_{ij}$ satisfies

$$\sum\nolimits_j f_{ij} = \sum\nolimits_j b_{ij} = 1, f_{ij} \geq 0 \text{ and } b_{ij} \geq 0 \tag{A.1}$$

The standard deviation of the probability vectors $f_{ij}, j = 1, ..., n$ and $b_{ij}, j = 1, ..., n$ has to be large enough to get significant results. Then these vectors must not be fully random but they are intentionally composed of some components very close to 0, some very close to 1 and the leftovers following a Dirichlet distribution verifying

$$x_1, ..., x_K \text{ where } x_i \in (0, 1) \text{ and } \sum\nolimits_{i=1}^{K} x_i = 1. \tag{A.2}$$

A probability $f_{ij}$ close to 0 means that the $j^{th}$ VNF has nearly no chance of being located right forward of the $i^{th}$ VNF whereas a probability close to 1 implies that it is very likely to happen. In the implementation of the algorithm, we first built the F matrix as described above. Then to simplify the algorithm, we build the $B$ matrix such as

$$B = \|F^T\|. \tag{A.3}$$

$B$ being the transpose of $F$ implies that $f_{ij} = b_{ji}$. Basically, the probability for VNF $s_j$ to be located right backward of the VNF $s_i$ equals the probability for VNF $s_i$ to be located right forward of the VNF $s_j$. Then we can compute the norm of the result in order to have vectors whose sum of the components is equal to 1. We define the probability pair $p(c)$ for the current VNF $c$ positioned between the forward VNF $f$ and the backward VNF $b$ as

$$p(c) = [B[b][c], F[c][f]]. \tag{A.4}$$

$p(c)$ reflects the weights of the associations between VNF $c$ and VNF $b$, and between VNF $c$ and VNF $f$. As a result, the probability pair is used to determine the behavior of each VNF. Let $ch_k$ be a VNF service chain which can be visualized as a Markov chain where $k$ is the index of the chain. This chain is an ordered list of VNFs which belong to the state space $S$. Each VNF may appear in the chain or may not. We define $V_{N,k}$ the vector containing all the probability pairs of the $N$ VNFs in the chain $ch_k$. Algorithm 5 is the calculation of the vector $V_{N,k}$ for a chain $ch_k$ of $N$ VNFs.

**Phase 1:** Calculate $V_{N,k}$

```
procedure CALCULATE_V(ch_k)
    N ← size(ch_k)
    list V_{N,k}
    for VNF c in chain ch_k do
        V_{N,k}.add(p(c))
    return V_{N,k}
```

## A.4 VNF Service Chain Classification

The VNF Service Markov Chain enables us to get an overall vision of the link probabilities between the VNFs of a given chain. Intuitively, the highest the probabilities are in the vector $V_{N,k}$ of a chain, the more relevant the chain is. Given this vector, the chain can be classified either as normal or abnormal. Any classification algorithm can be applied and we chose K-means as it is an unsupervised machine learning algorithm that aims at grouping correlated data into clusters [172]. In this case, it notifies the network administrator in case of a new abnormal VNF chain identified. Continuously monitoring K-means algorithm [173] is used to keeping analyzing the network behavior. The initialization phase consists in building the cluster of the probabilities vectors of the normal chains. To simplify the notation, we name these clusters the normal behavior clusters. Then the dataset used for K-means is the outputs of the Markov chain algorithm applied to the normal chains. The normal chains are the initial chains existing in the networks, created by the network administrators themselves before any scalability and optimization operations. The second phase involves the classification of new VNF chains, that are either automatically created to scale the traffic in the network or can be new chains requested by users. Here, K-means evaluates the distance between the cluster and the probability vector of a given chain to classify it.

### A.4.1 Normal Behavior Cluster

Let $C$ be a set of $m$ VNF service chains known as correct. The size of each chain, i.e., the number of VNFs on each chain, may vary.

$$C = \{ch_1, ..., ch_k, ..., ch_m\}, \qquad 1 < k < m. \tag{A.5}$$

The algorithm 6 hereafter seeks to build the normal behavior clusters depending on their size following 3 steps:

1. We compute the vector $V_{N,k}$ for each chain $ch_k$ of $N$ VNFs in $C$.

2. We dispose of a set of pairs (two-dimensional array) named *vectors* which associate a size (i.e., the number of VNFs in the chain) with a list containing all the vectors $V_{N,k}$ of the chains of that size. It is necessary to distinguish between sizes because K-means algorithm has to be applied to vectors of the same size.

3. K-means aims at minimizing an objective function known as squared error function. For each value of $N$, given the data set $(V_{N_1,1}, V_{N_2,2}, ..., V_{N_f,f})$, the function is given by

$$E(N) = \sum_{l=1}^{f} \|X_N - V_{N,l}\|^2, \tag{A.6}$$

where $X_N$ is the cluster center of the chains of $N$ VNFs. We then get a set of pairs named *clusters* which associate each value of $N$ to the cluster center $X_N$.

**Phase 2:** Apply K-means

**procedure** K_MEANS($C$)
  2-D array *vectors*
  **for** chain $ch_k$ in chains $C$ **do**
    $N \leftarrow size(ch_k)$
    $V_{N,k} \leftarrow$ calculate_V($ch_k$)
    $vectors[N]$.add($V_{N,k}$)
  2-D array *clusters*
  **for** $N$ in *vectors* **do**
    $clusters[N] \leftarrow KMeans(vectors[N], nb\_clust = 1)$
  **return** *clusters*

### A.4.2 VNF Chains Classification

Let $T$ be the new dataset made of the VNF service chains that we need to classify as normal or abnormal. Using a similar method as in [174], a chain $ch_k$ of $N$ VNFs is considered as normal if its probabilities vector $V_{N,k}$ belongs to the cluster $X_N$ of normal behavior, otherwise, it is abnormal. The chain is normal if the Euclidean distance between its vector and the cluster center is small enough, so smaller than a decision criterion $DC_N$ such as

$$d(V_{N,k}, X_N) \leq DC_N, \qquad DC_N > 0 \tag{A.7}$$

The decision criterion depends on the number $N$ of VNFs in the vector $X_N$. As a matter of fact, the most elements the chain contains, the highest the Euclidean distance will be, as we can see in this equation representing the Euclidean distance between two points $p$ and $q$

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

for an n-dimensional space (A.8)

It is not possible to find a mathematical formula for the decision criterion versus the number $N$ of VNFs in the chain. The experimental results are presented in Section A.5. Let's assume that the outcome of $DC_N$ is a function depending on the number $N$ of VNFs. For each chain

$ch_k$ of $N$ VNFs in $T$, let $M_k$ be the binary decision for the chain $ch_k$. The value of $M_k$ is true if the chain is considered as malicious, otherwise it is false.

$$M_k = \begin{cases} True & \text{if } d(V_{N,k}, X_N) > DC_N \\ False & \text{otherwise} \end{cases} \tag{A.9}$$

Using a set of chains $C$ known as normal, we can classify a set $T$ of chains as presented in the Algorithm 7 below.

**Phase 3:** Classify chains $T$

**procedure** CLASSIFY($C, T$)
  $clusters \leftarrow K\_means(C)$
  binary decisions $M$
  **for** chain $ch_k$ in $T$ **do**
    $N \leftarrow$ size($ch_k$)
    $V_{N,k} \leftarrow$ calculate_V($ch_k$)
    $dist \leftarrow$ distance($V_{N,k}, clusters[N]$)
    $M_k \leftarrow$ bool($dist > DC_N$)
    $M$.add($M_k$)
  **return** $M$

## A.5 Simulations and Performance Analysis

### A.5.1 Evaluation Metrics

A confusion matrix [175] is a table often used to evaluate the performance of a classification model. Let's now define the most basic terms, which are whole numbers and not rates. True Positive ($TP$) is the number of malicious chains correctly classified. True Negative ($TN$) is the number of normal chains correctly classified. False Positive ($FP$) is the number of normal chains incorrectly classified. False Negative ($FN$) is the number of malicious chains incorrectly classified.

### A.5.2 Resolution of the Decision Criterion

For the first simulation, we only work on the chains $C$ known as normal and used to build our model, we don't need to classify anything yet. We aim to find the correct decision criterion so that we get good results. We saw in Section A.4 that the criterion depends on the number $N$ of VNFs, then we could write it $DC_N$. Let $D$ be the Euclidean distance from point $V_{N,k}$ to cluster $X_N$.

$$D = d(V_{N,k}, X_N) \tag{A.10}$$

Let's plot $D$ versus the number of VNFs $N$ in the chain, so $D(N)$, to see the shape of this function. $D$ can be obtained by applying a transformation on the points of a cluster to obtain a cluster-distance space. The number of VNFs has to be large enough to get a significant view of the shape of the curve. It has been arbitrarily set to 15. The number of chains $C$ known as normal and useful to build the model has to be chosen larger than the number of VNFs to get at least one chain for each number of VNFs. The number of chains is then set to 30.

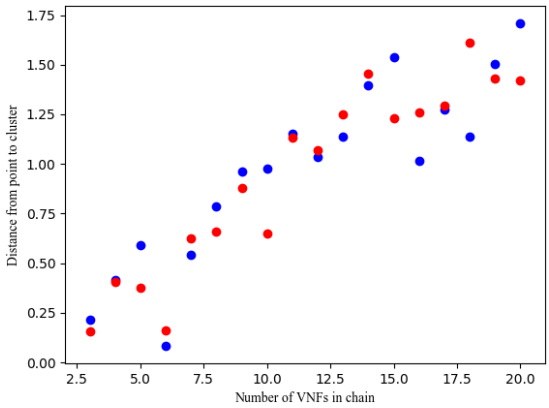



Figure A.3: Distance as a function of the number of VNFs in the chain

As observed in Figure 3, the Euclidean distance between a point $V_{Nk}$ and the cluster $X_N$ versus the number of VNFs seems to be a linear function so that

$$D(N) = a \times N + b \tag{A.11}$$

We determine the values of the two following constants

- $a$ the slope such as

$$a = \frac{\sum_{i=1}^{n}(N_i - \bar{N})(D_i - \bar{D})}{\sum_{i=1}^{n}(N_i - \bar{N})^2} \tag{A.12}$$

- $b$ the intercept (or constant term) such as

$$b = \bar{D} - a \times \bar{N} \tag{A.13}$$

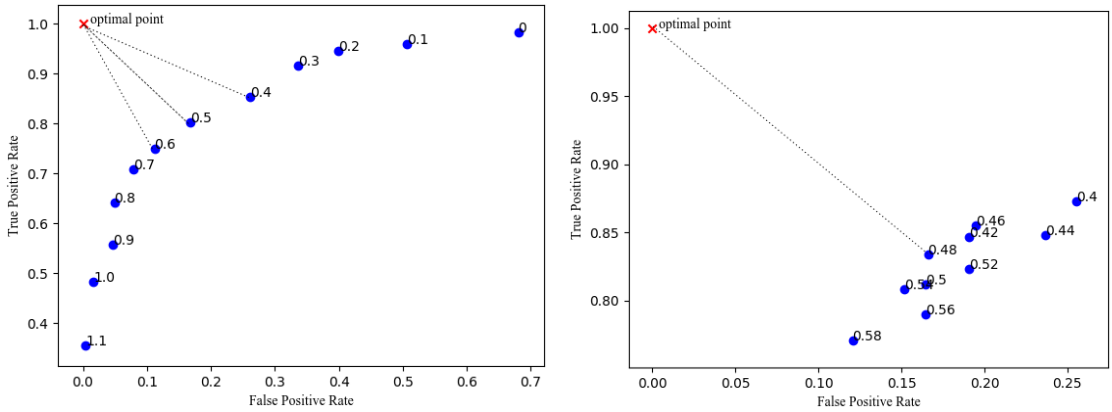
Some tools allow one to estimate $a$ and $b$ values by providing a set of discrete values. For our part, we used the Linear Regression library from Scipy designed for Python. The decision criterion $DC_N$ equals to the average Euclidean distance up to a positive constant $C$ such as $DC_N = a \times N + b + C$.

Fig. A.4a shows the TPR against the FPR. This type of graph has been inspired by the Receiver Operating Characteristic (ROC) curves that are used to assess models for a binary target in supervised learning [176]. We seek a high True Positive Rate ($TPR$) and a low False Positive Rate ($FPR$), so the optimal coordinates are $(0, 1)$. We make $C$ vary then select the value for which we get the closest to the optimal point.

To this end, we vary the value of constant $C$ from 0 to 1.2 with a step of 0.1, and for each value of $C$ we plot the point whose the x-axis is the $FPR$ and the y-axis the $TPR$. The goal is to find the smallest Euclidean distance between a point and the point of coordinates $(0, 1)$. As shown in Fig. A.4b, the best results are found for $C$ between 0.4 and 0.6. We refine the values of $C$ by making it vary between 0.4 to 0.6 with a step of 0.02 and we plot the results in Fig. A.4b. Among these values, the smallest distance has been found for $C = 0.48$.

The final equation for the decision criterion is then

$$DC_N = a \times N + b + 0.48 \tag{A.14}$$




(a) $C$ varying from 0 to 1.2 (b) Zoom for $C$ varying from 0.4 to 0.6

Figure A.4: True Positive Rate ($TPR$) for different values of $C$.

### A.5.3 Classification Results

In this subsection, we use the algorithm to classify a set of chains. The number of initial chains and the number of normal and abnormal chains to be classified are all set to 60 to correctly feed the algorithm and to find an acceptable cluster. Each chain contains between 4 and 20 VNFs, each VNF is split between 1 and 3 instances and one chain can go through any of the instances. To rely on a practical 5G configuration, we assume that there are only a few chains, whose functions have pretty often the same relations between them and are split into different instances depending on the network resources. Therefore we created the first matrix so that each function in the chain is strongly related (i.e., with a large probability, here higher than 0.5) to one or two other functions, is slightly related to some other functions (here the third of the remaining functions with probabilities following a Dirichlet distribution) and is never related (i.e with a probability equal to 0) to the leftover functions. The second matrix is equal to the transpose of the first matrix. Now, we create two types of new chains: some relevant normal chains based on the transition matrices and some random malicious chains. The goal of the simulation is to correctly classify the most chains as possible, either normal or abnormal. The number of iterations is set to 100 to get precise results.

Fig. A.5 shows the precision and the accuracy rates, the TPR and the FPR versus the number of VNFs in the network. The precision and the accuracy rates are very good as they are greater than 85 % for any number of functions. Furthermore, we observe that the accuracy, the TPR and the FPR are related and that all the rates depend on the number of VNFs. For a low number of VNFs, the FPR is low at the cost of the lowest TPR. This means that we have fewer false alarms but also fewer anomalous chains are detected. On the contrary, for a large number of functions, i.e., from 12 functions, the FPR increases as well as the TPR. This could be better in our case as we want to detect the most anomalous chains as possible.

## A.6 Conclusion

Virtualization, containerization, and softwarization increase the network programmability and agility that make the network provisioning and the deployment moving into a new era. The network self-optimization and network resource scale in and out on-the-fly might introduce a new type of threat and produce an unknown side effect to the virtual network infrastructure.

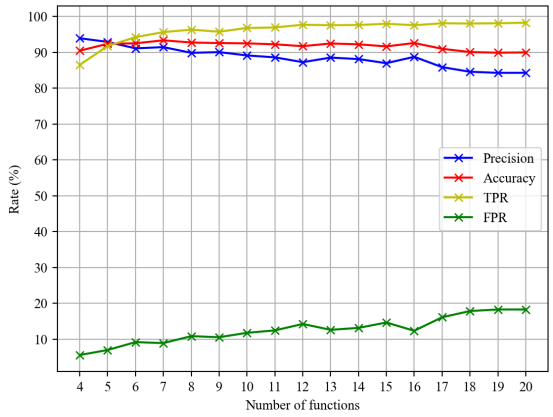



Figure A.5: Precision rate, accuracy rate, FPR and TPR versus the number of VNFs

Furthermore, the software-defined network induces a lack of visibility that increases the difficulty to prevent attacks. Therefore, we need a pro-active method to reduce the risks by constantly observing the VNF service chains and tenant provisioning behavior. In this chapter, we developed a two-step method for detecting anomalies in the VNF service chains. The first step uses Markov chains to ensure that the VNFs before and after are in the right sequence order. The second step applies K-means classification on observing the whole VNF service chain behavior. Moreover, our K-means classifier evaluation produces high accuracy in detecting anomalies. As a result, the algorithm can easily identify the normal and abnormal behaviors of the VNF service chains. We recommend updating the transition matrices values after the classification of new chains to gain a better outcome in the classification. The proposed model also shows that the accuracy and the precision rates are directly proportional to the number of VNFs in a service chain. Finding a more accurate way to determine the decision criterion, either mathematically or experimentally with a better regression, would also provide us better results.

# Appendix B

# Botnet Fingerprinting supplementary materials

## B.1 Observation of bots fingerprints

Fig. B.1a shows a typical example of a bot (`147.32.84.165`) from scenario #1 of CTU-13, where each point is representing one flow (bots from scenarios #2 and #6 show a similar behavior); through the graphs, we can infer its actions: scanning or spamming (looking to the whole range of IP addresses targeted), infection of other hosts by searching for their vulnerabilities (looking to the used destination ports corresponding to many vulnerable services), communication with the C&C server via a proxy. We also observe regular connections to the C&C server, using the exotic TCP/65000 port and the IRC protocol. Finally, the range for ephemeral ports is different from quite unusual, neither the range recommended by IANA or the typical Linux range.

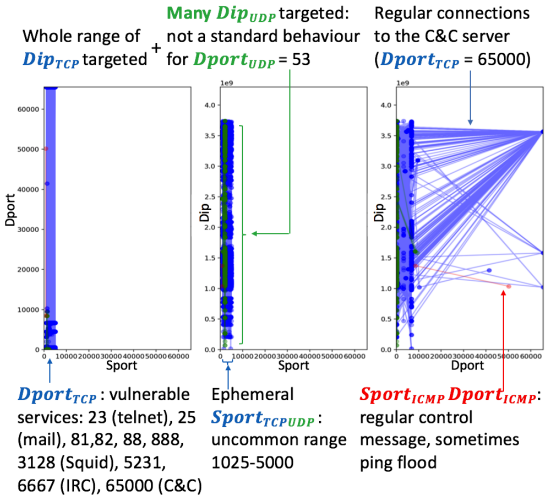
Fig. B.1b is an example of an infected host (`192.168.100.111`) from scenario #17 of IoT-23, performing several malicious activities; its behavior is close to hosts from scenarios #5, 7 from CTU-13 which performed port scanning. For this host, we observe usual TCP and UDP connections, but also port scanning: targeting port 8081 (alternative HTTP port) with the hard-coded source port number 17576, 80 and 8080 with source port 18088, 52869 (service Universal Plug and Play) with source port 18344, and 37215 (Huawei HG532 router port) with source port 17832, targeting the whole range of IP addresses. We observe that both hosts are performing network scanning: targeting specific ports known for their vulnerabilities and targeting the whole range of IPv4 addresses. We notice also some differences between both hosts: they did not run the same Operating System, as the range for ephemeral ports is different. Also, we do not observe C&C communications for the second host.

Fig. B.1c shows the fingerprinting of an infected host (`10.0.2.111`) from capture 112_2 from Malware Capture Facility Project. The host has normal TCP and UDP connections, but we also observe an ICMP DoS using a large variety of ICMP codes (noted destination ports in the graph) and targeting the DNS servers hosted at `8.8.4.4` and `8.8.8.8`. Its behavior is similar to hosts from scenarios #4, 10, 11 from CTU-13 performing ICMP DoS.

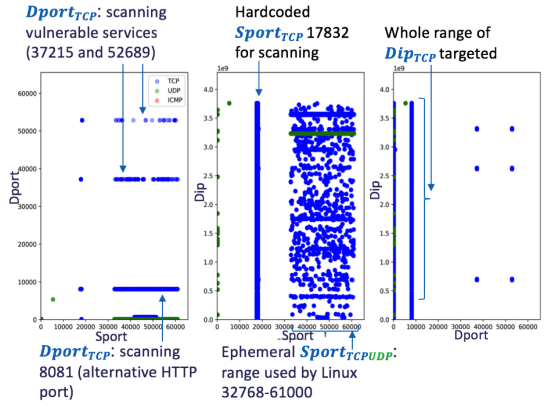
## B.2 Importance of features selection

To better understand the peculiarities of bots communications that enable us to detect them, we observe the most meaningful features in the classification process.

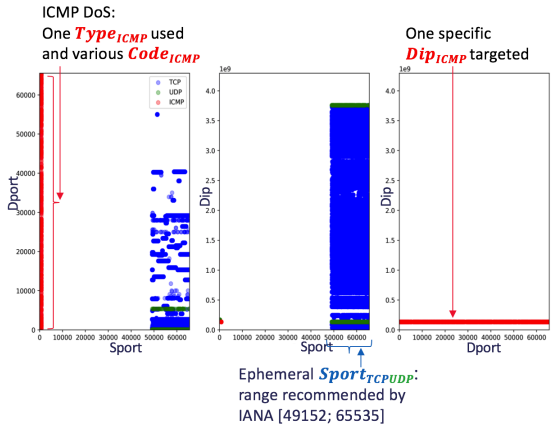
Once the linear SVM is fit to the data, with 256 regular bins (and the C parameter set to 100




(a) Host `147.32.84.165` from scenario #1 of CTU-13 performing network scanning.




(b) Host `192.168.100.111` from scenario #17 of IoT-23 performing port scanning.




(c) Host `10.0.2.111` from capture 112_2 of Malware Capture Facility Project.

Figure B.1: Fingerprinting of infected hosts.

according to Gridsearch), it creates a line or a hyper-plane which separates the data into classes. It uses support vectors to maximize the distance between two classes, and the weights obtained represent the vector coordinates which are orthogonal to the hyper-plane and their direction indicates the predicted class.

Table B.1 shows the ranking of the most meaningful attributes with, for each of them, the number of bins that have a weight not null, the sum of all per-bin weights, and the mean weight per bin. $\text{Type}_{\text{ICMP}}$ has the most significant impact of all attributes. It represents the type of ICMP message which, for a botnet, is often set to 3 for "Destination Unreachable" or to 8 for "Echo Request". The second and the third most important ones are $\text{Dip}_{\text{TCP}}$ and $\text{Dip}_{\text{UDP}}$, as during botnet spam or scan, nearly all destination IP addresses are targeted instead of some selected ASes. $\text{Sport}_{\text{UDP}}$ and $\text{Dport}_{\text{UDP}}$ follow, actually UDP is often used only for DNS, and in the case of a botnet is very differently used. Finally, the values of $\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{TCP}}$, $\text{Dport}_{\text{ICMP}}$ and $\text{Dip}_{\text{ICMP}}$ have nearly no impact on the results.

| **Attribute** | **Number of bins with a weight $> 0$** | **Sum of weights for all bins** | **Mean weight per bin** |
|---|---|---|---|
| $\text{Type}_{\text{ICMP}}$ | 148 | 0.8517 | 0.0058 |
| $\text{Dip}_{\text{TCP}}$ | 60 | 0.5328 | 0.0089 |
| $\text{Dip}_{\text{UDP}}$ | 52 | 0.3371 | 0.0058 |
| $\text{Sport}_{\text{UDP}}$ | 41 | 0.1663 | 0.0041 |
| $\text{Dport}_{\text{UDP}}$ | 100 | 0.1264 | 0.0013 |
| $\text{Sport}_{\text{TCP}}$ | 43 | 0.0046 | 0.0001 |
| $\text{Dport}_{\text{TCP}}$ | 10 | 0.0327 | 0.0032 |
| $\text{Code}_{\text{ICMP}}$ | 50 | 0.0014 | $2.8843 \cdot 10^{-5}$ |
| $\text{Dip}_{\text{ICMP}}$ | 5 | 0.0013 | 0.0003 |

Table B.1: Ranking of the attributes according to their importance in the classification, for *BotFP-SVM* with 256 regular bins.

# Bibliography


[1] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*. ACM Press, 2004.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai botnet. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, pages 1093–1110, 2017.

[3] Github. Mirai source code. `https://github.com/jgamblin/Mirai-Source-Code/`, 2016.

[4] ATLAS. Netscout threat intelligence report. `https://www.netscout.com/threatreport`, 2019.

[5] Kaspersky. DDoS attacks in Q2 2019. `https://securelist.com/ddos-report-q1-2019/90792/`.

[6] Wentao Chang, An Wang, Aziz Mohaisen, and Songqing Chen. Characterizing botnets-as-a-service. In *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*. ACM Press, 2014.

[7] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Surveying port scans and their detection methodologies. *The Computer Journal*, 54(10):1565–1581, April 2011.

[8] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, jul 1959.

[9] Gianluigi Folino and Pietro Sabatino. Ensemble based collaborative and distributed intrusion detection systems: A survey. *Journal of Network and Computer Applications*, 66:1–16, May 2016.

[10] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3):e0194889, mar 2018.

[11] Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1), 2018.

[12] Olivier Cappé, Eric Moulines, and Tobias Rydén. *Inference in Hidden Markov Models*. Springer New York, 2005.

[13] Davide Ariu, Roberto Tronci, and Giorgio Giacinto. HMMPayl: An intrusion detection system based on hidden markov models. *Computers & Security*, 30(4):221–241, jun 2011.

[14] Alexander G. Tartakovsky, Aleksey S. Polunchenko, and Grigory Sokolov. Efficient computer network anomaly detection by changepoint detection methods. *IEEE Journal of Selected Topics in Signal Processing*, 7(1):4–11, Feb 2013.

[15] Boris Iglewicz and David Hoaglin. How to detect and handle outliers. In Ph.D. Edward F. Mykytka, editor, *The ASQC Basic References in Quality Control: Statistical Techniques*, volume 16. 1993.

[16] Andreas Kind, Marc Stoecklin, and Xenofontas Dimitropoulos. Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, 6(2):110–121, jun 2009.

[17] Ming-Yang Su, Gwo-Jong Yu, and Chun-Yuen Lin. A real-time network intrusion detection system for large-scale attacks based on an incremental mining approach. *Computers & Security*, 28(5):301–309, jul 2009.

[18] FukudaLab. MAWILab database. `http://www.fukuda-lab.org/mawilab`, 2019.

[19] Christian Callegari, Stefano Giordano, and Michele Pagano. Entropy-based network anomaly detection. In *Proceedings of International Conference on Computing, Networking and Communications (ICNC)*, 2017.

[20] Juliette Dromard, Gilles Roudiere, and Philippe Owezarski. Misc and scalable unsupervised network anomaly detection method. *IEEE Transactions on Network and Service Management*, 14(1):34–47, 2017.

[21] Wei Lu and Hengjian Tong. Detecting network anomalies using CUSUM and EM clustering. In *Advances in Computation and Intelligence*, pages 297–308. 2009.

[22] Yi Li, Hong Liu, Wenjun Yang, Dianming Hu, and Wei Xu. Inter-data-center network traffic prediction with elephant flows. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, apr 2016.

[23] Zhitang Chen, Jiayao Wen, and Yanhui Geng. Predicting future traffic using hidden markov models. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, nov 2016.

[24] Muslim Elkotob, Daniel Grandlund, Karl Andersson, and Christer Ahlund. Multimedia QoE optimized management using prediction and statistical learning. In *IEEE Local Computer Network Conference*. IEEE, oct 2010.

[25] Yen-Liang Chen, Kwei Tang, Ren-Jie Shen, and Ya-Han Hu. Market basket analysis in a multiple store environment. *Decision Support Systems*, 40(2):339–354, aug 2005.

[26] Giuseppe Aceto, Alessio Botta, Pietro Marchetta, Valerio Persico, and Antonio Pescapé. A comprehensive survey on internet outages. *Journal of Network and Computer Applications*, 113:36–63, jul 2018.

[27] Ripe atlas. `https://atlas.ripe.net`.

[28] Alexandros Milolidakis, Romain Fontugne, and Xenofontas Dimitropoulos. Detecting network disruptions at colocation facilities. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE, apr 2019.

[29] Marco Di Bartolomeo, Valentino Di Donato, Maurizio Pizzonia, Claudio Squarcella, and Massimo Rimondini. Discovering high-impact routing events using traceroutes. In *2015 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, jul 2015.

[30] Philipp Richter, Ramakrishna Padmanabhan, Neil Spring, Arthur Berger, and David Clark. Advancing the art of internet edge outage detection. In *Proceedings of the Internet Measurement Conference 2018*. ACM, oct 2018.

[31] Karyn Benson, Alberto Dainotti, K. C. Claffy, and Emile Aben. Gaining insight into AS-level outages through analysis of internet background radiation. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, apr 2013.

[32] CAIDA. UCSD Network Telescope Aggregated Flow Dataset. `https://www.caida.org/data/passive/telescope-flowtuple.xml`, 2020.

[33] Mehdi Zakroum, Abdellah Houmz, Mounir Ghogho, Ghita Mezzour, Abdelkader Lahmadi, Jerome FranCois, and Mohammed El Koutbi. Exploratory data analysis of a network telescope traffic and prediction of port probing rates. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, nov 2018.

[34] MAWI. MAWI working group traffic archive. `http://mawi.wide.ad.jp/mawi/`, 2019.

[35] Stratosphere Lab. The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic. `www.stratosphereips.org/datasets-ctu13`.

[36] Paola Bermolen, Marco Mellia, Michela Meo, Dario Rossi, and Silvio Valenti. Abacus: Accurate behavioral classification of p2p-TV traffic. *Computer Networks*, 55(6):1394–1411, apr 2011.

[37] Dominik Schatzmann, Wolfgang Mühlbauer, Thrasyvoulos Spyropoulos, and Xenofontas Dimitropoulos. Digging into HTTPS. In *Proceedings of the 10th annual conference on Internet measurement - IMC '10*. ACM Press, 2010.

[38] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. ACAS. In *Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data - MineNet '05*. ACM Press, 2005.

[39] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected means of protocol inference. In *Proceedings of the 6th ACM SIGCOMM on Internet measurement - IMC '06*. ACM Press, 2006.

[40] Alessandro Finamore, Marco Mellia, Michela Meo, and Dario Rossi. KISS: Stochastic packet inspection classifier for UDP traffic. *IEEE/ACM Transactions on Networking*, 18(5):1505–1515, oct 2010.

[41] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1-3):37–52, aug 1987.

[42] Tarem Ahmed, Mark Coates, and Anukool Lakhina. Multivariate misc anomaly detection using kernel recursive least squares. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. IEEE, may 2007.

[43] Kun Xie, Lele Wang, Xin Wang, Gaogang Xie, Jigang Wen, Guangxing Zhang, Jiannong Cao, and Dafang Zhang. Accurate recovery of internet traffic data: A sequential tensor completion approach. *IEEE/ACM Transactions on Networking*, 26(2):793–806, apr 2018.

[44] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16(1):303–336, 2014.

[45] Cisco. Snort - network intrusion detection & prevention system. `https://www.snort.org/`, 2018.

[46] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, Dec 1999.

[47] Suricata | open source ids / ips / nsm engine. `https://suricata-ids.org/`.

[48] Antonio Gonzalez Pastana Lobato, Martin Andreoni Lopez, Igor Jochem Sanz, Alvaro A. Cardenas, Otto Carlos M. B. Duarte, and Guy Pujolle. An adaptive real-time architecture for zero-day threat detection. In *IEEE International Conference on Communications (ICC)*, 2018.

[49] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.

[50] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM*. ACM Press, 2005.

[51] Daniel Barbará, Julia Couto, Sushil Jajodia, and Ningning Wu. ADAM. *ACM SIGMOD Record*, 30(4):15–24, dec 2001.

[52] Gisung Kim, Seungmin Lee, and Sehun Kim. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, 41(4):1690–1700, mar 2014.

[53] Ozgur Depren, Murat Topallar, Emin Anarim, and M. Kemal Ciliz. An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*, 29(4):713–722, nov 2005.

[54] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security*, 29(1):124–140, Feb 2010.

[55] Praveen Kumar Shanmugam, Naveen Dasa Subramanyam, Joe Breen, Corey Roach, and Jacobus Van der Merwe. DEIDtect: towards distributed elastic intrusion detection. In *Proceedings of the ACM SIGCOMM workshop on Distributed cloud computing (DCC)*, 2014.

[56] Azeem Aqil, Karim Khalil, Ahmed O.F. Atya, Evangelos E. Papalexakis, Srikanth V. Krishnamurthy, Trent Jaeger, K. K. Ramakrishnan, Paul Yu, and Ananthram Swami. Jaal: Towards network intrusion detection at isp scale. In *Proceedings of the International COnference on emerging Networking EXperiments and Technologies - (CoNEXT)*, 2017.

[57] Dinesh Singh, Dhiren Patel, Bhavesh Borisaniya, and Chirag Modi. Collaborative IDS framework for cloud. *International Journal of Network Security*, 18:699–709, 2015.

[58] J. Francois, C. Wagner, R. State, and T. Engel. SAFEM: Scalable analysis of flows with entropic measures and SVM. In *2012 IEEE Network Operations and Management Symposium*. IEEE, apr 2012.

[59] Seyed Ali Mirheidari, Sajjad Arshad, and Rasool Jalili. Alert correlation algorithms: A survey and taxonomy. In *Cyberspace Safety and Security*, pages 183–197. 2013.

[60] Fariba Haddadi, Duc Le Cong, Laura Porter, and A. Nur Zincir-Heywood. On the effectiveness of different botnet detection approaches. In *Information Security Practice and Experience*, pages 121–135. Springer International Publishing, 2015.

[61] Wei Wang, Yaoyao Shang, Yongzhong He, Yidong Li, and Jiqiang Liu. BotMark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors. *Information Sciences*, 511:284–296, feb 2020.

[62] Javier Álvarez Cid-Fuentes, Claudia Szabo, and Katrina Falkner. An adaptive framework for the detection of novel botnets. *Computers & Security*, 79:148–161, nov 2018.

[63] S. Panjwani, S. Tan, K.M. Jarrin, and M. Cukier. An experimental evaluation to determine if port scans are precursors to an attack. In *2005 International Conference on Dependable Systems and Networks (DSN)*, 2005.

[64] Chia-Nan Kao, Yung-Cheng Chang, Nen-Fu Huang, I Salim S, I-Ju Liao, Rong-Tai Liu, and Hsien-Wei Hung. A predictive zero-day network defense using long-term port-scan recording. In *2015 IEEE Conference on Communications and Network Security (CNS)*, Sep 2015.

[65] Andreas Guillot, Romain Fontugne, Philipp Winter, Pascal Mérindol, Alberto Dainotti, and Cristel Pelsser. Chocolatine: Outage detection for internet background radiation. In *Network Traffic Measurement and Analysis Conference (TMA)*, 2019.

[66] Alberto Dainotti, Karyn Benson, Alistair King, kc claffy, Michael Kallitsis, Eduard Glatz, and Xenofontas Dimitropoulos. Estimating internet address space usage through passive measurements. *ACM SIGCOMM Computer Communication Review*, 44(1):42–49, dec 2013.

[67] Muhammad Mahmoud, Manjinder Nir, and Ashraf Matrawy. A survey on botnet architectures, detection and defences. *I. J. Network Security*, 17:264–281, 2015.

[68] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, 70:238–254, sep 2017.

[69] Guofei Gu, Phillip Porras, Vinod Yegneswaran, and Martin Fong. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the USENIX Security Symposium*. USENIX Association, 2007.

[70] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014.

[71] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM Computer Communication Review*, 34(4):219, oct 2004.

[72] Xuefeng Bai, Tiejun Zhang, Chuanjun Wang, Ahmed A. Abd El-Latif, and Xiamu Niu. A fully automatic player detection method based on one-class SVM. *IEICE Transactions on Information and Systems*, E96.D(2):387–391, 2013.

[73] Ahmed A. Abd El-Latif, Bassem Abd-El-Atty, Wojciech Mazurczyk, Carol Fung, and Salvador E. Venegas-Andraca. Secure data encryption based on quantum walks for 5g internet of things scenario. *IEEE Transactions on Network and Service Management*, 17(1):118–131, mar 2020.

[74] Randeep Bhatia, Steven Benno, Jairo Esteban, T. V. Lakshman, and John Grogan. Unsupervised machine learning for network-centric anomaly detection in IoT. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks - Big-DAMA '19*. ACM Press, 2019.

[75] Sofiane Lagraa, Jerome Francois, Abdelkader Lahmadi, Marine Miner, Christian Hammerschmidt, and Radu State. BotGM: Unsupervised graph mining to detect botnets in traffic flows. In *Proceedings of the Cyber Security in Networking Conference (CSNet)*. IEEE, 2017.

[76] Manmeet Singh, Maninder Singh, and Sanmeet Kaur. Detecting bot-infected machines using DNS fingerprinting. *Digital Investigation*, 28:14–33, mar 2019.

[77] Sudipta Chowdhury, Mojtaba Khanzadeh, Ravi Akula, Fangyan Zhang, Song Zhang, Hugh Medal, Mohammad Marufuzzaman, and Linkan Bian. Botnet detection using graph-based feature clustering. *Journal of Big Data*, 4(1), may 2017.

[78] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. BotGrep: Finding P2P Bots with Structured Graph Analysis. In *Proceedings of the USENIX Security Symposium*, pages 95–110, 2010.

[79] Hongling Jiang and Xiuli Shao. Detecting p2p botnets by discovering flow dependency in c&c traffic. *Peer-to-Peer Networking and Applications*, 7(4):320–331, jun 2012.

[80] Futai Zou, Siyu Zhang, Weixiong Rao, and Ping Yi. Detecting malware based on DNS graph mining. *International Journal of Distributed Sensor Networks*, 2015:1–12, 2015.

[81] Weikeng Chen, Xiao Luo, and A. Nur Zincir-Heywood. Exploring a service-based normal behaviour profiling system for botnet detection. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017.

[82] Abbas Abou Daya, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba. A graph-based machine learning approach for bot detection. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

[83] Jing Wang and Ioannis Ch. Paschalidis. Botnet detection based on anomaly and community detection. *IEEE Transactions on Control of Network Systems*, 4(2):392–404, jun 2017.

[84] Patrick Kalmbach, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. Themis: A data-driven approach to bot detection. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018.

[85] Diala Naboulsi, Marco Fiore, Stephane Ribot, and Razvan Stanica. Large-scale mobile traffic analysis: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):124–161, 2016.

[86] Julián Candia, Marta C González, Pu Wang, Timothy Schoenharl, Greg Madey, and Albert-László Barabási. Uncovering individual and collective human dynamics from mobile phone records. *Journal of Physics A: Mathematical and Theoretical*, 41(22):224015, may 2008.

[87] Francesco Calabrese, Francisco C. Pereira, Giusy Di Lorenzo, Liang Liu, and Carlo Ratti. The geography of taste: Analyzing cell-phone mobility and social events. In *Lecture Notes in Computer Science*, pages 22–37. Springer Berlin Heidelberg, 2010.

[88] Blerim Cici, Minas Gjoka, Athina Markopoulou, and Carter T. Butts. On the decomposition of cell phone activity patterns and their connection with urban ecology. In *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing - MobiHoc '15*. ACM Press, 2015.

[89] Diala Naboulsi, Razvan Stanica, and Marco Fiore. Classifying call profiles in large-scale mobile traffic datasets. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, apr 2014.

[90] James P. Bagrow, Dashun Wang, and Albert-László Barabási. Collective response of human populations to large-scale emergencies. *PLoS ONE*, 6(3):e17680, mar 2011.

[91] Cristina Marquez, Marco Gramaglia, Marco Fiore, Albert Banchs, Cezary Ziemlicki, and Zbigniew Smoreda. Not all apps are created equal. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, nov 2017.

[92] Ying Zhang and Ake Årvidsson. Understanding the characteristics of cellular data traffic. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design - CellNet '12*. ACM Press, 2012.

[93] Fengli Xu, Yong Li, Huandong Wang, Pengyu Zhang, and Depeng Jin. Understanding mobile traffic patterns of large scale cellular towers in urban environment. *IEEE/ACM Transactions on Networking*, 25(2):1147–1161, apr 2017.

[94] Angelo Furno, Marco Fiore, Razvan Stanica, Cezary Ziemlicki, and Zbigniew Smoreda. A tale of ten cities: Characterizing signatures of mobile traffic in urban areas. *IEEE Transactions on Mobile Computing*, 16(10):2682–2696, oct 2017.

[95] R. Adams and L. Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, jun 1994.

[96] Liang Xiong, Barnabás Póczos, and Jeff Schneider. Group anomaly detection using flexible genre models. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, 2011.

[97] Raghavendra Chalapathy, Edward Toth, and Sanjay Chawla. Group anomaly detection using deep generative models. In *Machine Learning and Knowledge Discovery in Databases*, pages 173–189. Springer International Publishing, 2019.

[98] Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, sep 2014.

[99] D. Birant and A. Kut. Spatio-temporal outlier detection in large databases. In *28th International Conference on Information Technology Interfaces, 2006.* IEEE, 2006.

[100] Tao Cheng and Zhilin Li. A multiscale approach for spatio-temporal outlier detection. *Transactions in GIS*, 10(2):253–263, mar 2006.

[101] Nabil R. Adam, Vandana Pursnani Janeja, and Vijayalakshmi Atluri. Neighborhood based detection of anomalies in high dimensional spatio-temporal sensor datasets. In *Proceedings of the 2004 ACM symposium on Applied computing - SAC '04*. ACM Press, 2004.

[102] Agathe Blaise, Mathieu Bouet, Stefano Secci, and Vania Conan. Split-and-Merge: detecting unknown botnets. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2019.

[103] Agathe Blaise, Mathieu Bouet, Vania Conan, and Stefano Secci. Detection of zero-day attacks: An unsupervised port-based approach. *Computer Networks*, 180:107391, oct 2020.

[104] Agathe Blaise, Sandra Scott-Hayward, and Stefano Secci. Scalable and collaborative intrusion detection and prevention systems based on SDN and NFV. In *Computer Communications and Networks*, pages 653–673. Springer International Publishing, 2020.

[105] Akamai. Memcached UDP reflection attacks. `https://blogs.akamai.com/2018/02/memcached-udp-reflection-attacks.html`, 2018.

[106] TechRepublic. Massive ransomware attack takes out 27,000 mongodb servers. `https://www.techrepublic.com/article/massive-ransomware-attack-takes-out-27000-mongodb-servers/`, 2017.

[107] An Wang, Wentao Chang, Songqing Chen, and Aziz Mohaisen. Delving into internet DDoS attacks by botnets: Characterization and analysis. *IEEE/ACM Transactions on Networking*, 26(6):2843–2855, 2018.

[108] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the International COnference on emerging Networking EXperiments and Technologies (Co-NEXT)*, 2010.

[109] Github. Source code for Split-and-Merge detection algorithm. `https://github.com/a-blaise/split-and-merge`, 2019.

[110] Aniket Mahanti, Niklas Carlsson, Anirban Mahanti, Martin Arlitt, and Carey Williamson. A tale of the tails: Power-laws in internet measurements. *IEEE Network*, 27(1):59–64, jan 2013.

[111] Mehmet Celenk, Thomas Conley, John Willis, and James Graham. Predictive network anomaly detection and visualization. *IEEE Transactions on Information Forensics and Security*, 5(2):288–299, jun 2010.

[112] Nimrod Aviram. DROWN Attack. `https://drownattack.com/`, 2016.

[113] Eric Wustrow Zakir Durumeric and J. Alex Halderman. ZMap: Fast internet-wide scanning and its security applications. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2013.

[114] Sam Edwards and Ioannis Profetis. Hajime: Analysis of a decentralized internet worm for IoT devices. `https://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf`, 2016.

[115] Symantec. Hajime worm battles mirai for control of the internet of things. `https://www.symantec.com/connect/blogs/hajime-worm-battles-mirai-control-internet-things`, 2017.

[116] Nmap. Nmap: the network Mapper. `https://nmap.org/`, 2018.

[117] Netlab360. New threat report: A new IoT botnet is spreading over http 81 on a large scale. `http://blog.netlab.360.com/a-new-threat-an-iot-botnet-scanning-internet-on-port-81-en/`, 2017.

[118] US Congress. Internet of Things (IoT) Cybersecurity Improvement Act of 2017. `https://www.congress.gov/115/bills/s1691/BILLS-115s1691is.pdf`, 2017.

[119] Netlab360. Warning: Satori, a Mirai branch is spreading in worm style on port 37215 and 52869. `http://blog.netlab.360.com/warning-satori-a-new-mirai-variant-is-spreading-in-worm-style-on-port-37215-and-52869-en/`, 2017.

[120] Netlab360. ADB.Miner: More information. `http://blog.netlab.360.com/adb-miner-more-information-en/`, Feb 2018.

[121] CheckPoint. IoTroop botnet: The full investigation. `https://research.checkpoint.com/iotroop-botnet-full-investigation/`, 2018.

[122] Radware. Why the world is under the spell of IoTReaper. `https://blog.radware.com/security/2017/10/iot_reaper-botnet/`, 2017.

[123] US CERT. Ics advisory (icsa-13-011-03), rockwell automation controllogix plc vulnerabilities. `https://www.us-cert.gov/ics/advisories/ICSA-13-011-03`, 2019.

[124] SANS ISC InfoSec Forums. Surge in exploit attempts for netis router backdoor (udp/53413). `https://isc.sans.edu/forums/diary/Surge+in+Exploit+Attempts+for+Netis+Router+Backdoor+UDP53413/21337/`, 2017.

[125] FireEye. Smb exploited: Wannacry use of "eternalblue". `https://www.fireeye.com/blog/threat-research/2017/05/smb-exploited-wannacry-use-of-eternalblue.html`, 2017.

[126] Radware. Satori iot botnet variant. `https://security.radware.com/ddos-threats-attacks/threat-advisories-attack-reports/satori-iot-botnet/`, 2018.

[127] Avira. 18000 routers taken hostage in less than a day. https://blog.avira.com/18000-routers-taken-hostage-in-less-than-a-day/, 2018.

[128] PentaSecurity. Top 5 botnets of 2017. https://www.pentasecurity.com/blog/top-5-botnets-2017/, 2017.

[129] ZDNet. A decade of malware: Top botnets of the 2010s. https://www.zdnet.com/article/a-decade-of-malware-top-botnets-of-the-2010s/, 2019.

[130] Agathe Blaise, Mathieu Bouet, Vania Conan, and Stefano Secci. Botfp: Fingerprints clustering for bot detection. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2020.

[131] Agathe Blaise, Mathieu Bouet, Vania Conan, and Stefano Secci. Botnet fingerprinting: A frequency distributions scheme for lightweight bot detection. *IEEE Transactions on Network and Service Management*, 17(3):1701–1714, sep 2020.

[132] ZDnet. Avast and french police take over malware botnet and disinfect 850,000 computers. https://www.zdnet.com/article/avast-and-french-police-take-over-malware-botnet-and-disinfect-850000-computers/.

[133] ZDNet. A hacking group is hijacking docker systems with exposed api endpoints. https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml.

[134] Mid-year update: 2019 sonicwall cyber threat report. https://blog.sonicwall.com/en-us/2019/07/mid-year-update-2019-sonicwall-cyber-threat-report/.

[135] Basil AsSadhan, Abdulmuneem Bashaiwth, Jalal Al-Muhtadi, and Saleh Alshebeili. Analysis of p2p, IRC and HTTP traffic for botnets detection. *Peer-to-Peer Networking and Applications*, 11(5):848–861, jul 2017.

[136] Source code for BotFP algorithm. https://github.com/BotFP/botFP-detection, 2020.

[137] Stratosphere Lab. Stratosphere Research Laboratory. https://www.stratosphereips.org/.

[138] Stratosphere Lab. Aposemat IoT-23. https://www.stratosphereips.org/datasets-iot23.

[139] Stratosphere Lab. Malware Capture Facility Project. https://www.stratosphereips.org/datasets-malware.

[140] Service name and transport protocol port number registry. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, 2013.

[141] Whois domain lookup. www.whois.com/whois/.

[142] IANA. Internet control message protocol (icmp) parameters. https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml.

[143] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[144] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)*, 2012.

[145] Gursel Serpen and Zhenning Gao. Complexity analysis of multilayer perceptron neural network embedded into a wireless sensor network. *Procedia Computer Science*, 36:192–197, 2014.

[146] Yong Hou and Xue Feng Zheng. SVM based MLP neural network algorithm and application in intrusion detection. In *Artificial Intelligence and Computational Intelligence*, pages 340–345. Springer Berlin Heidelberg, 2011.

[147] Abdiansah Abdiansah and Retantyo Wardoyo. Time complexity analysis of support vector machines (svm) in libsvm. *International Journal of Computer Applications*, 2015.

[148] Gaurav Sharma and Frederic Jurie. A novel approach for efficient SVM classification with histogram intersection kernel. In *Procedings of the British Machine Vision Conference 2013*. British Machine Vision Association, 2013.

[149] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *Proceedings of NIPS*, 2006.

[150] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

[151] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.

[152] Source code for special events detection. `https://github.com/a-blaise/ASTECH/tree/main`.

[153] Cancan project - content and context based adaptation in mobile networks. `https://cancan.roc.cnam.fr/`.

[154] C. H. P. Gifford and F. R. Macaulay. The movements of interest rates, bond yields and stock prices in the united states since 1856. *The Economic Journal*, 49(194):312, jun 1939.

[155] John C. Musgrave Julius Shiskin, Allan H. Young. The x-11 variant of the census method ii seasonal adjustment program. Technical report, Bureau of the Census, U.S. Department of Commerce, 1967.

[156] Victor Gómez and Agustin Maravall. Programs tramo and seats, instructions for the user. Technical report, Banco de España, 1996.

[157] Jean E. McRae Robert B. Cleveland, William S. Cleveland and Irma Terpenning. STL: A seasonal-trend decomposition. *Journal of Official Statistics*, 6(1):3–73, 1990.

[158] William S. Cleveland. LOWESS: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, 35(1):54, feb 1981.

[159] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et du jura. 1901.

[160] Past events - they occurred at the stade de france. https://www.stadefrance.com/en/ticket/archives, 2020.

[161] Marie Turcan. Panne chez orange : des problèmes de connexion internet et 4g sur toute la france. https://www.numerama.com/tech/481172-panne-chez-orange-des-problemes-de-connexion-internet-et-4g-sur-toute-la-france.html, 2019.

[162] Jonathan Shieber. Google cloud is down, affecting numerous applications and services. https://techcrunch.com/2019/06/02/google-cloud-is-down-affecting-numerous-applications-and-services/.

[163] Agathe Blaise, Mathieu Bouet, Vania Conan, and Stefano Secci. Group anomaly detection in mobile application usages: a decomposed time-series methodology. Submitted to IEEE Transactions on Mobile Computing.

[164] Alessandro D'Alconzo, Idilio Drago, Andrea Morichetta, Marco Mellia, and Pedro Casas. A survey on big data for network traffic monitoring and analysis. *IEEE Transactions on Network and Service Management*, 16(3):800–813, sep 2019.

[165] Agathe Blaise, Stan Wong, and A. Hamid Aghvami. Virtual network function service chaining anomaly detection. In *2018 25th International Conference on Telecommunications (ICT)*. IEEE, 2018.

[166] Gabriel Brown. Service chaining in carrier networks. *Heavy Reading, White Paper*, 2015.

[167] C. Pignataro J. Halpern. Service function chaining (sfc) architecture, rfc-7665, 2015.

[168] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A. L. Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with DNS traffic analysis. *IEEE/ACM Transactions on Networking*, 20(5):1663–1677, oct 2012.

[169] T. Nadeau P. Quinn. Problem statement for service function chaining, rfc-7498, 2015.

[170] Barbara Martini and Federica Paganelli. A service-oriented approach for dynamic chaining of virtual network functions over multi-provider software-defined networks. *Future Internet*, 8(4):24, jun 2016.

[171] Xuan Zhou, Rongpeng Li, Tao Chen, and Honggang Zhang. Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine*, 54(7):146–153, jul 2016.

[172] Lizhong Xiao, Zhiqing Shao, and Gang Liu. K-means algorithm based on particle swarm optimization algorithm for anomaly intrusion detection. In *2006 6th World Congress on Intelligent Control and Automation*. IEEE, 2006.

[173] Ming Hua, Man Ki Lau, Jian Pei, and Kui Wu. Continuous k-means monitoring with low reporting cost in sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 21(12):1679–1691, dec 2009.

[174] R. Kumari, Sheetanshu, M. K. Singh, R. Jha, and N.K. Singh. Anomaly detection in network traffic using k-mean clustering. In *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*. IEEE, mar 2016.

[175] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep learning approach for network intrusion detection in software defined networking. In *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, oct 2016.

[176] Michael H Ferris, Michael McLaughlin, Samuel Grieggs, Soundararajan Ezekiel, Erik Blasch, Mark Alford, Maria Cornacchia, and Adnan Bubalo. Using ROC curves and AUC to evaluate performance of no-reference image fusion metrics. In *2015 National Aerospace and Electronics Conference (NAECON)*. IEEE, jun 2015.