POLITECNICO DI TORINO

Department of Electronics and Telecommunications

Master's Degree in *"Communications and Computer Networks Engineering"*

Master's Thesis

Design and Engineering of System for Large-scale Internet Traffic Visualisation

Supervisors

Prof. Marco Mellia

Prof. Idilio Drago

Candidate

Claudia Carletti

Contents

List of Figures

Chapter 1

Introduction and problem description

1.1 Motivation

Anomalies in the network traffic can arise due to faults or, especially, cyberattacks. When the data set is large, a graphical representation of the traffic can make it easier to read, interpret and understand, by translating large amounts of text-based data into a visual representation. This operation can be particularly important and helpful in security applications, to visualize and identify abnormal patterns within the network traffic, reducing the time and the effort required by system administrators to check and analyse, otherwise, all the traffic saved in text logs. A visual representation of traffic data allows to have a better overview of network activity, useful to both take actions in real-time and to discover compromised hosts within the network.

A great variety of intrusion detection systems are currently employed to detect unauthorized access to computers or networks. Historically, intrusion detection can be classified in signature-based and anomaly detection. While the systems in the first category work by comparing the traffic to test with samples of known malicious traffic, the second class relies on the concept of anomaly, or outlier, as deviation from the “normal” behaviour. The biggest advantage is that, while signature-based detection systems fail in detecting new attacks, anomaly detection systems can, if recognize them as anomalies. Nowadays, consequently to the growing in the number of attacks and their increasing power, anomaly detection is a field widely studied and several techniques have been investigated and proposed over the past years. These proposals are based on concepts coming from several different fields of study, like statistics and information theory. Most intrusion detection methods with visualization are anomaly-based.

Traditional anomaly detection approaches applied to network environments work well in detecting outliers inside the traffic. However, generating an alert for any suspicious event appearing in the network traffic isn’t always the good strategy: a lot of frequent alerts always

related to the same event could divert the attention of the analyst from most important ones. Moreover, several anomaly detection techniques fail in situations in which the traffic profile is not very well behaved. So we want a more robust way to detect anomalies, less sensitive to seasonality effects that can arise in network traces.

1.2 Goal

In such context, the objective of this work is to develop a system that allows to visualize in a flexible way the traffic and the anomalies inside a network. While traditional anomaly detection systems aim to detect the appearance of any deviation, we wish to focus on the identification of only the most relevant ones.

The system is composed by 6 modules. TCP network traffic is first captured and collected in flow-based logs, differentiating the connections that fail from the successful ones. All this data is processed through a script to extract meaningful network features (traffic volume, number of unique source IP addresses and number of unique destination IP addresses), helpful to delineate the traffic profile, and to aggregate them in hourly time series per destination port. Since the amount of data is considerable, this processing step is performed exploiting a big data approach. The output is elaborated with Prophet, a library developed at Facebook to forecast time series data. From the data passed, Prophet is able to make a prediction for the future trend, compare it with the actual data and detect the points in which the trend changes. The output is sent and stored in a distributed database for time series, and successively retrieved to provide the interactive graphical representation of our time series through a dashboard.

We test the performance of the system with a case study, using traces of traffic collected within the Politecnico di Torino network. These traces are related to failed connections in a month of traffic. As first, we made a general analysis of the traffic to understand some statistics about it, like the top ten ports in the month for destination traffic volume. In this way, we process the traffic to obtain the time series related to these ports. Then, anomalies in the trends are visualized with Prophet. In particular, we can spot anomalies under two forms: the change in the trend and the presence of dots falling outside the uncertainty interval for the forecast. By

combining these two features, we are able to detect anomalies in network traffic with Prophet. Once an anomaly is detected, it can be investigated to identify the threat.

1.3 Thesis organization

The following topics will be addressed in each chapter.

Chapter 2: State of the art

This section discusses the context in which this work is developed, the importance of the network dataset chosen, a description of the main categories which anomaly detection techniques can be classified, offering a general view of the most used and some reference to network traffic visualization.

Chapter 3: System Design and Architecture

This chapter discusses the proposed system architecture, focusing on explaining the single elements composing it and their purpose in the process. In particular, the system is composed by 6 modules, that will be individually explained.

Chapter 4: Dataset and Case study

This chapter describes the dataset used to test the proposed system and offers an overview on the results obtained, reporting as examples the cases related to four destination ports. At the end of this section, is also reported the investigation made upon an anomaly that has been found by applying the proposed system to our dataset.

Chapter 5: Conclusion and future work

In this last chapter, it is done a survey of the results with conclusions and considerations, and mentions about possible future works.

Chapter 2

State of the art

Nowadays Internet networks are wildly used in many different fields, among which we name: telecommunications, business, health, instruction, industry, agriculture, transportations and many others.

This leads to a huge amount of data crossing the network and, therefore, to an increasing number of cyber criminals that try to exploit the vulnerabilities in software and protocols for their personal or financial gain.

So, with the increase of the use of Internet, security incidents and attacks increased and became always more sophisticated and variate, so that it is harder to be immediately recognised.

According to a report released by Gemalto in 2018 *"944 data breaches led to 3.3 billion data records being compromised worldwide in the first half of 2018. Compared to the same period in 2017, the number of lost, stolen or compromised records increased by a staggering 72 percent, though the total number of breaches slightly decreased over the same period, signalling an increase in the severity of each incident."* [1]

Nowadays networks show an increasing number of connected IoT devices, that in the last years also had a great influence on the number of cybersecurity incidents: over recent years the number of smart devices connected to the network, like security cameras, baby monitors, lights, fans, TVs, washing machines, cars, etc…has considerably increased besides the more traditional devices. This has posed an enormous threat in security: these devices have shown several security problems, due to unpatched vulnerabilities and weak/default password, that make them target for cyberattacks.

The first important IoT malware attack seen at global level goes back to October 2016, when an aggressive distributed denial of service attack left the major Internet platforms and services

unavailable in the east coast of US. This event was caused by the activity of the so called Mirai botnet, a self-propagating botnet malware designed to compromise IoT devices. [2] Since Mirai code has been published, several of variants have been developed and used up to now.

According to a report by researchers at F-Secure Labs, 19 new threats have been found in 2018 doubling the number of existing IoT threats**.** [3]
IoT devices are especially exploited for DDoS attacks (like in the case of the Mirai botnet attack and variants), or also for mining cryptocurrencies, with the goal to install cryptominers to generate virtual currency.

Finding a solution to the issues connected to IoT devices is becoming more and more stringent as, according to the previsions, the number of IoT devices is expected to grow to 10 billion by 2020 and 22 billion by 2025. [4] In this scenario, adopt intrusion detection solutions is always more significant.

2.1 Intrusion detection

With the growing in number of attacks and their increasing power, it is even more important to adopt intrusion detection solutions, that have the aim to detect unauthorized access to a computer or a network. [5]

There are two main possibilities to perform intrusion detection: ***signature base detection*** and ***anomaly detection***. The two approaches can also be combined to be used together in hybrid detection systems.

Signature-base detection is based on the recognition of the signature of an attack, that is the pattern of the threat: the traffic is compared with known signatures stored in a database, where each entry matches a certain threat. Because of that, signature-based systems can detect just known threats while they fail in recognize unknow attacks. However, the advantages of such

techniques are the presence of low false positive rate and the easy identification of the attacking threats. The system is however no more able to detect a malicious activity with a small variation from the known threat signature. So the signature database must constantly be updated. However, adding more and more signatures in the database increases the storage cost and the search cost required from the system to check all the possible matches. [6]

The ***anomaly detection*** approach instead tries to detect an event by building a traffic behaviour baseline and looking for any deviation from it. This unexpected behaviour is identified as anomaly, or outlier. Anomaly detection is not applied just for network applications. In fact, anomaly detection can be defined as "the identification of items, events or observations which is significantly different from the remaining data". [7] Therefore, it found place in multiple application domains: some examples are fraud detection for credit cards, identity thefts, sensor events, insurance or health care frauds, fault or damage detection and, as already mentioned, intrusion detection in cyber security. Anomalies, in a network scenario, can be either *performance* related anomalies, due for example to malfunctioning or overload, or *security* related anomalies.

In the case of security anomalies, anomaly detection systems can also detect new attacks, differently from the signature-based. However, they may sometimes lead to a high false positive or false negative rate. In the first case, a normal behaviour is detected as an attack since the normal behaviour itself changed. In the other case, a malicious event is not recognized as such because there is a very small variation in the trend that is not detected.

Since network attacks are always more sophisticated and frequent, network anomaly detection is a widely studied field and many approaches have been proposed with the goal of maximizing the ability to detect the greatest number of malicious activities in the traffic and minimize the false positive and false negative rates.

However, the "normal" profile for the traffic is not so trivial to be defined due to several reasons: boundaries among normal and bad activity, attacker trying to mime the traffic, high variability in the normal pattern itself, etc..and this can cause poor performance.

Moreover, when a deviation from the legitimate traffic is detected, it has also to be categorize since, differently from the signature base techniques, the newly found anomaly has not been labelled yet. [8]

2.2 Traffic monitoring for anomaly detection

Authors in [9] highlight how the detection of the various types of anomalies is dependent on the nature of the network data used.

A survey of possible data sets suitable for network anomaly detection is done in [10], focusing on their format and use. As reported, network data traditionally can be captured in two main formats: *packet*-based or *flow*-based. In the first case, data are commonly presented in pcap format and include the payload, and the information depend on the transport protocol used. The second case deals with information about network connections. It doesn't contain the payload since it aggregates packets according to the definition of flow: all packets that have the same source IP address, destination IP address, source port number, destination port number and transport protocol belong to the same flow. The packets can be aggregated in unidirectional or bidirectional flows. In the first case, the communication direction matters and, in the same connection, packets from the client to the server and from the server to the client are put in two different flows, while in the second case all packets in the same flow are put together, without caring about their direction.

A further distinction can be made considering the recording environment. For recording environment is intended the kind of traffic and the considered network. The traffic in the data set can be real network traffic captured by a network device within a production network environment. The capture environment should be considered too, as the collected dataset behaviour may vary if they are related to a small-medium size company network or to an internet service provider environment.

Once the data sets are passed as input, anomaly detection techniques can return as output an anomaly score or, like in most of the cases, a binary label, that simply tells if the data is considered anomalous or not.

For anomaly detection in network connections, the normal behaviour monitored is the one related to some network traffic characteristics.

For example, in volume-based detection techniques is observed the trend of the traffic volume in a network, looking for its significant changes. These systems work to detect some types of

anomalies like flooding attacks and some DoS attacks, that cause shifts in the network traffic load. On the contrary, attacks that don't affect the traffic volume, or not considerably, are not signalled from these systems. This situation occurs for example in presence of a port scan.

In feature-based anomaly detection, instead of considering just the traffic volume, are taken in consideration also other network traffic features. These features include some packets header fields like the source IP address, the destination IP address, the source port number, the destination port number, the TCP flag, the protocol number and packet size and measurements like for example the flow duration. These features can determine the presence of multiple abnormal situations: for examples, one or more IP addresses generating a great amount of traffic or connections can be due to worms, botnets or spoofing. While an increase in the number of destination IP addresses can show the presence of a port scan. In almost all the works, the goal is to understand the normal pattern of these network features, under normal traffic condition, and make a comparison with the features extracted from the traffic to test, in order to check if it is maintained or distorted by anomalies. [11]

2.3 Network Anomaly Detection methods

Another aspect of anomaly detection is the availability of supervision, that means the use of labels for input data. Accordingly to that, anomaly detection techniques can be classified in supervised, semi-supervised or unsupervised. These concepts come from machine learning, a branch artificial intelligence. **Machine learning** algorithms have ability to learn from data and, afterwards, make predictions based on previous data samples. [12]

In **supervised** anomaly detection techniques, labels are available for both normal and anomalies classes. The algorithm is fed with these labelled data instances so that it is able to build a general model of each category. Then, the data instance to categorize is processed to determine to which pre-learned classes it belongs. A difficulty in this approach is that it is not always easy to have a representative label for all the anomaly categories and, since a supervised algorithm only

knows the categories on which it has been trained, it is not able to categorize unknown data instances. So automated anomaly detection systems based on this kind of algorithms require information related to each possible type of anomaly. Classical supervised algorithms are neural networks, support vector machines and decision trees. Supervised anomaly detection techniques are similar to signature-based approaches. The main difference is that, while in signature-based a signature is given to be used as reference, in supervised approach the classifier in the system is trained, with the hope to generalize the attack.

There are several ways to define the **semi-supervised** approach. According to [7], semi-supervised anomaly detection algorithms are fed with labels for only normal data, for which the algorithm builds the general model. Data instances to test are then processed to determine whether they belong to the category or not. Since semi-supervised anomaly detection techniques don't require labels for the anomaly, they are more widely applicable than the supervised ones.

The most used way to perform classifications is through an **unsupervised** approach, since unsupervised anomaly detection algorithms don't need labelled training data set. Such techniques are based on two assumptions: first, it is presumed that most of the network connections are normal traffic and only a small percentage is abnormal. Second, the malicious traffic is statistically different from normal traffic. In such a way, groups of similar data instances that appear frequently are assumed to be normal traffic and those data groups that are infrequent are considered anomalous. In this way, unsupervised anomaly detection techniques are able to detect any types of anomaly, including those never encountered before. But, as already said, in this case the main difficulty is to determine what is part of "normal" behaviour.

Among the mentioned categories, we here describe the most widely used. We recall:

- **Statistical-based** anomaly detection techniques, that statistically identify traffic that deviates from normal traffic. The base idea is that normal data instances are generated with high probability from the stochastic model assumed, while anomalies with low

probability. Statistical techniques can be divided mainly in parametric and non-parametric. In parametric statistic techniques, the model structure is given by parameters. For example, the parameters that follow the Gaussian distribution and are estimated through the Maximum Likelihood estimated. In non-parametric statistical-based techniques, instead, the model is not built upon parameters, but it is derived directly from the data. The advantage of this approach is a fast computation. Examples of non-parametric models applied to network anomaly detections are histograms and kernel-density estimators (KDE). [13]

- **Clustering-based** anomaly detection techniques, whose idea is to group similar data instances in clusters, larger clusters often represent the normal data behaviour, while are considered anomalies data instances that don't belong to any cluster or belonging to a smaller or less dense cluster. Clustering-based techniques are examples of unsupervised anomaly detection. Some examples of clustering-based approaches applied to anomaly detection in network traffic are K-means [14]**,** DBScan [15] or nearest-neighbor [16].

2.4 Network Traffic Visualization

Visualization techniques aim to graphically represent sets of data. This is particularly important when the data set is large, since a graphical representation can make it easier to be read, interpreted or understood.

In network environments, this is particularly true. In fact, we have large-scale traffic data. All these data, if displayed, allow to visualize abnormal visual behaviour patterns in an easier way, reducing the time and the effort required by system administrators to check and analyze, otherwise, all the traffic saved in text logs. Therefore, a visual representation of traffic data allows to have a better overview of network activity, useful to both take actions in real-time and to later discover compromised hosts within the network.

The main challenge is to visualize all the information that are relevant to the particular situation. Since in network environments there are large amount of data, this data must be aggregated before to be put in visual form.

The need for such visualization tools that can promptly and efficiently catch the attention on important events occurring in the network is at the base of several works. Some visualization tools for network traffic are for troubleshooting but, especially in past years, they have been developed to focus in particular for intrusion detection applications, to monitor, discover and investigate security related events.

One example is VISUAL [17], a visual tool home-centric based on packet traces to represent hosts linkages in a network. Another packet-level visualization tool that has been proposed is TNV [18], based on a matrix to show hosts network activity over time and allows to visualize their interactions, port activity and, on demand, packets details. In [19] the visualization tool proposed captures some packets information, which are analysed and visualized. Other examples of works in this context are NVisionIP [20] and NFlowVis [21].

Chapter 3

System Design and Architecture

In this chapter, we discuss the proposed system architecture, focusing on explaining the single elements composing it and their purpose in the process.

Figure 1 shows the architecture of the built system, composed by 6 modules. The system data source is Internet network traffic previously collected with Tstat, a passive network traffic sniffer. It is processed, exploiting the Spark framework on the BigDataLab cluster, with the aim to extract a set of meaningful features that characterize the traffic behaviour and aggregate them in time series, through a pre-processing script written in Python. The output is both stored in OpenTSDB, a distributed database for time series, and analysed with Prophet, a library developed at Facebook to forecast time series data. All the time series stored in OpenTSDB are retrieved and visualized through Grafana, a visualization tool, to have the graphical representation of the trends. All pieces of the proposed system are described in detail in the following.

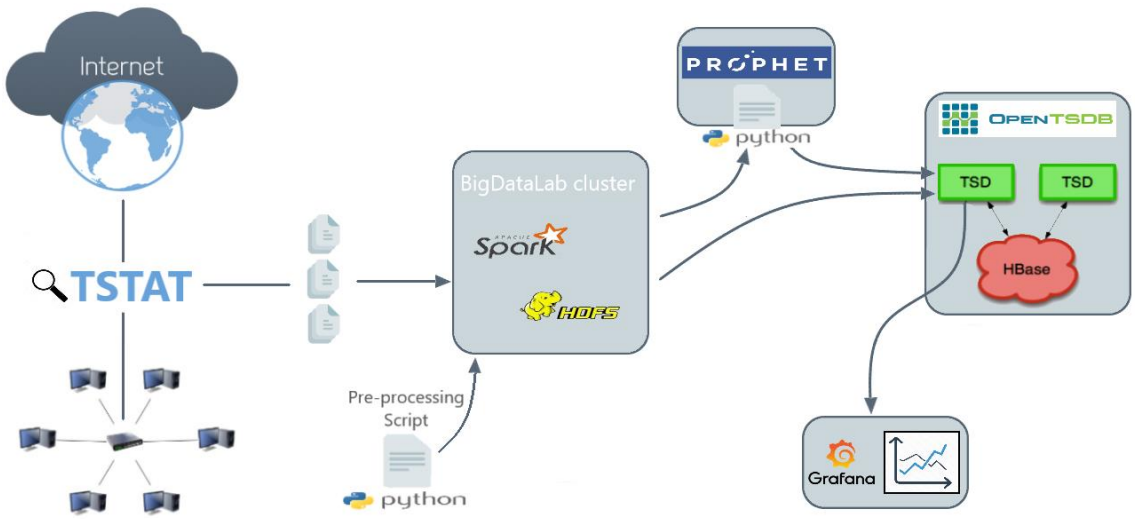


Figure 1: Architecture of the proposed system

3.1 Tstat

As first stage in the system, the traffic is captured by Tstat.

Tstat stands for *TCP Statistic and Analysis Tool* and it is a passive network traffic sniffer developed by the Politecnico di Torino. [22]

As shown in figure 2, Tstat is able to intercept network traffic from/toward an internal network toward/from the Internet.

Figure 2: Tstat

TSTAT generates different flow-level measurement collections, that are saved in log files. The logs partition the analyzed flows according to several selected protocols (i.e. TCP, UDP, HTTP). In our case, we focus on the log_tcp_nocomplete and log_tcp_complete logs, that collect the TCP traffic. The data in these logs are organized in rows and columns: each row represents a flow and each column indicates a measure. According to the definition of flow, all the packets that have the same source and destination IP address and source and destination port number are considered for the same flow row. A TCP connection is tracked by TSTAT as soon as a SYN segment is observed and it is stopped when either a FIN/ACK or RST is seen or no data packet is observed after a default time. The log_tcp_complete stores all the connections that are correctly opened, while all the others end up in the log_tcp_nocomplete. This means that log_tcp_nocomplete logs collect traffic that is anomalous by definition, since it is traffic that reached a machine that was no able to respond. In such way, traffic could be considered

anomalous without any further analysis. However, the volume of such traffic is very high, since it can be caused either by situations such as mis-configured systems or by attack attempts. Among the attack attempts, some of them are so normal that can be ignored too. From that, the need to detect just the most relevant anomalies.

Looking instead at the columns, the logs have 44 fields, as shown in figure 3.

C2S	S2C	Short description	Unit	Long description
1	15	Client/Server IP addr	-	IP addresses of the client/server
2	16	Client/Server TCP port	-	TCP port addresses for the client/server
3	17	packets	-	total number of packets observed form the client/server
4	18	RST sent	0/1	0 = no RST segment has been sent by the client/server
5	19	ACK sent	-	number of segments with the ACK field set to 1
6	20	PURE ACK sent	-	number of segments with ACK field set to 1 and no data
7	21	unique bytes	bytes	number of bytes sent in the payload
8	22	data pkts	-	number of segments with payload
9	23	data bytes	bytes	number of bytes transmitted in the payload, including retransmissions
10	24	rexmit pkts	-	number of retransmitted segments
11	25	rexmit bytes	bytes	number of retransmitted bytes
12	26	out seq pkts	-	number of segments observed out of sequence
13	27	SYN count	-	number of SYN segments observed (including rtx)
14	28	FIN count	-	number of FIN segments observed (including rtx)
29		First time abs	ms	Flow first packet absolute time (epoch)
30		Last time abs	ms	Flow last segment absolute time (epoch)
31		Completion time	ms	Flow duration since first packet to last packet
32		C first payload	ms	Client first segment with payload since the first flow segment
33		S first payload	ms	Server first segment with payload since the first flow segment
34		C last payload	ms	Client last segment with payload since the first flow segment
35		S last payload	ms	Server last segment with payload since the first flow segment
36		C first ack	ms	Client first ACK segment (without SYN) since the first flow segment
37		S first ack	ms	Server first ACK segment (without SYN) since the first flow segment
38		C Internal	0/1	1 = client has internal IP, 0 = client has external IP
39		S Internal	0/1	1 = server has internal IP, 0 = server has external IP
40		C anonymized	0/1	1 = client IP is CryptoPAn anonymized
41		S anonymized	0/1	1 = server IP is CryptoPAn anonymized
42		Connection type	-	Bitmap stating the connection type as identified by TCPL7 inspection engine (see protocol.h)
43		P2P type	-	Type of P2P protocol, as identified by the IPP2P engine (see ipp2p_tstat.h)
44		HTTP type	-	For HTTP flows, the identified Web2.0 content (see the http_content enum in struct.h)

Figure 3: Tstat logs columns fields. From [22]

As can be seen, columns are divided according to the traffic direction: C2S, client to server, and S2C, server to client. [23] Among all the column fields offered by Tstat in the logs, some of them are particularly useful to our purpose. In particular, we exploit the fields 38 and 39, to distinguish the connection attempts toward our network, the fields 1 and 15, to take and count the different source and destination IP addresses, the column 16, to distinguish our analysis basing on the destination port, and the field 29, to extract the needed timestamps.

3.2 BigDataLab cluster

In our scenario, we analyze multiple logs containing a great amount of data. This setting proves that Internet is a significant source of the so-called ***big data***. This expression is used to define "all the data sets whose size or type is beyond the ability of traditional relational databases to capture, manage and process the data with low latency". [24] The characteristics that distinguish the big data are volume, velocity, variety, veracity and value. **Volume** refers to the large dimension of the processed data, often not processable from traditional databases; **velocity** because data arrives with a fast rate and variety means that many different types of data are available. **Veracity** refers to the uncertainty of the data while **value** indicates the fact that these data have an intrinsic hidden value and, when properly processed to extract the information of interest, they can be exploited in a very useful way, so that data analytics is a very important and highly popular field in our days. For example, it is at the base of decision processes in many business areas. [25]

All these characteristics make these data sets arduous to be managed by traditional processing software. Working with big data requires in fact the ability to process large quantities of data, taking into account high availability and continuity of services. Thus, many big data frameworks have been devised for this purpose.

In our case, in order to process the network traffic traces, we make use of the BigDataLab cluster.

The BigDataLab cluster of Politecnico di Torino is a set of 30 servers running Hadoop.

Hadoop is a framework employed to process large data sets in a distributed way, to scale up from single servers to multiple of them, commonly referred as "nodes". Each node provides its computation and storage capability. [26] In BigDataCluster, the total storage available is 768 TB.

Hadoop is made of several layers of components, needed for its correct functioning:

- a distributed file system, **HDFS**.

  HDFS, that stands for **"**Hadoop Distributed File System", is the local file system available in all the nodes and it manages the data storage in Hadoop cluster. It differs from traditional file systems for some features like a highly fault-tolerant, a high throughput

access to application data and the fact that it is suitable for applications with large data sets. It is responsible for data storage, and so as source of data, and for data replication across the cluster nodes. Its architecture is of master/slave type: a master server, called NameNode, manages the file system namespace, by performing operations over files and directories, and the files access from clients. In HDFS each file stored is split in one or more blocks that are distributed in a set of DataNodes, the slaves in the architecture. DataNodes are usually one per node in the cluster and are responsible for the storage in their node. [27]

- a resource manager, **YARN**.

  YARN stands for "Yet Another Resource Negotiator" and its job is to coordinate and manage resources and schedule jobs. Yarn components are a ReosurceManager and a NodeManager agent per-machine. The ResourceManager globally supervises the resources in the whole system and allocates them to the running applications through the Scheduler and controls the acceptation of job-submissions and the restart of the ApplicationMaster in the container in case of failure through the ApplicationsManager. Each NodeManager instead controls the assigned container and, through the ApplicationMaster, is responsible for negotiating resources from the Scheduler, monitoring their usage and report it back to the Scheduler. [28]

- a processing engine (MapReduce, Test, HBase, Storm, Giraph, Spark and others), according to the specific framework. It runs on the top of the Hadoop ecosystem and its aim is to process large volumes of data in the cluster.

  The original version of Hadoop uses as processing engine MapReduce, which is batch based and makes use of the hard disk along each step. The BigDataCluster, instead, deploys **Spark** as processing engine. Spark is defined as an "in-memory" processing approach since, unlike MapReduce, it performs a full in-memory computation. Doing so, the storage level is used at the begin of the process to load the data into memory and

at the end to store the results while all the intermediate operations are instead handled using the random-access memory. [29] [30]

The architecture just descripted, implemented in the BigDataLab cluster environment, is shown in figure 4.

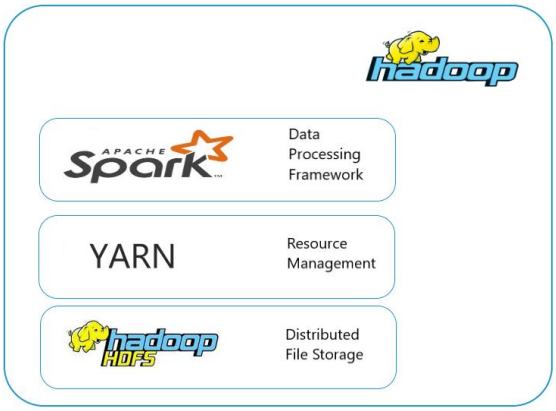


Figure 4: Hadoop ecosystem implemented in the BigDataLab cluster

3.3 Pre-processing Script

The script used to process data and aggregate them in time series has been written by exploiting Pyspark, which is the Python framework supported by Apache Spark. The script from command line takes at maximum three parameters: the log with the traffic data to process, the destination port number and the traffic feature to be returned. In fact, taking as model the feature-based anomaly detection approach, we select some features of interest that could be useful to delineate a per port traffic characterization. These features include incoming traffic volume, the number of unique external client IP addresses and the number of unique internal server IP addresses. In the case in which the port is not specified, the computation is done for all the traffic. In the script, through the `sc.textFile()` method available in PySpark, the log passed

from command line is read from HDFS and it is returned as an RDD of strings, where each line is an element. RDD stands for Resilient Distributed Dataset and it is the basic data structure in Spark. It represents an immutable partitioned collection of objects that can be operated in parallel, to achieve speed and efficient when performing operations. [31] Since for our analysis we want to consider all the incoming external failed connections, the initial RDD is filtered to obtain a second one containing only those elements in which the server IP address is internal to the network. This is can be done by considering all those flows that in the log show the field “Server internal” set to 1. From this new RDD, a further operation is done to consider among all elements those in which the client IP address is external, by considering all those flows having in the log the field “Client internal” equal to 1. The clients are also filtered in order to consider only those that are not anonymized. Then, different cases are considered according to the number of parameters passed. In the case in which the port number is specified, the RDD is filtered to obtain only those elements that are flows having this port as destination port. At this point, according to traffic feature wanted, different computations are made. A new RDD is obtained, reporting for each element the timestamp in the form 'Y-M-D H:00' that indicates year, month, day and hour at which that flow has been seen by Tstat, and the source IP address or the destination IP address or simply a 1, according to the considered feature. In case of source and destination IP address, a `distinct()` operation is also made, in order to consider once an IP address in each hour. Then, all the elements for the same hour are counted together. So at the end what is obtained is a time series having an hour timestamp, the number of source IP addresses or destination IP addresses or flows according to the required features to compute, and the destination port to which the time series is related to. Similar operations are done in the case in which the destination port is not specified. In this case, only the filtering part to select the flows related to a particular port is absent and therefore all the elements in each hour are considered.

The time series returned as output is saved in a csv file. Data is organized in the form timestamp, values, port number as tag, according to the format wanted by OpenTSDB as input.

3.4 Prophet

In our system, we want also to be able to visualize anomalies inside the traffic. As discussed in chapter 2, several anomaly detection solutions have been proposed over the years. In this case, we choose to make use of the forecasting tool Prophet [32], that is specifically designed to output analyses and predictions on time series. The techniques such tool uses fall in statistical-based category, since they are based on a nonparametric regression method called additive model.

Prophet is an open source software released from a team at Facebook, available in Python and R, and developed to make automated time series forecasts. Since it deals with time series predictions, it well suits our scenario. [33]

Prophet tries to overcome some difficulties met in producing reliable forecasts due to seasonable effects, changes in trend and data set with outliers, aiming to create high quality forecasts. In fact, like shown in [33], other forecasting tools, like ARIMA, snaive, TBATS, fail in captured any seasonality or longer-term seasonality and can't model properly yearly seasonality. Prophet, instead, is thought for data set with hourly, daily or weekly observations, showing holidays effect, having several outliers and with trends that are non-liner growth curves.

The fitting phase allows Prophet to understand the data model, in order to make then a prediction for the future trend. This is done by employing a curve fitting regression model instead of a traditional time series one, since it gives some advantages like more modelling flexibility, an easier way to fit the model and a better way to handle missing data and outliers.

The model is a decomposable time series one, similar to a Generalized Additive Model. There are three main components that are trend, seasonality and holidays, and the model is obtained as:

$$y(t) = g(t) + s(t) + h(t) + \in_t$$

where $g(t)$ is the trend function, modelling non-periodic changes in the trend of the time series (i.e. growth over time), $s(t)$ is the seasonal component representing periodic changes (i.e. daily, weekly, yearly seasonality) and $h(t)$, that gives the holidays effect, to model all the events occurring every year in a different day or set of days, while $\in_t$ is an error term to represent all the other changes. $y(t)$ is the time series data observed at time t. In this way, new components can be easily placed when required, for example when a new source of seasonality is identified. The classic Generalized Additive Model, on which it is based, is "a class of regression models with potentially non-linear smoothers applied to the regressor". In Prophet, the time is used as regressor.

For the trend $g(t)$, Prophet provides two possible trend models: a saturating growth model and a piecewise linear model. The first trend model is used in non-linear growth with saturation situations. The second trend model, instead, is for cases in which the growth rate is constant or there is no saturating growth.

Instead, the seasonal component $s(t)$ models periodic changes due to weekly or yearly seasonality using Fourier series.

This model in Prophet is realized in Stan [34], a programming language for statistical interference.

Beside the capability to handle the common features of time series, another important advantage of Prophet is that it allows the analyst to adjust parameters. Prophet automatically detects changes in the time series trends by selecting the changepoints and let the trend to adapt to them appropriately. Changepoints are abrupt changes in trajectories. The changepoints detection depends on the parameter δ, that controls the rate adjustment in the equation for the trend model function $g(t)$, with $\delta_j = \text{Laplace}(0, \tau)$. The changepoints control the flexibility of the trend, that, according to them, can be overfit or underfit. By default, the parameter that controls the strength of the sparse prior, τ, is set to 0.05, but its value can be increased, to make the trend more flexible, or decreased, to make it less flexible, according to the scenario. By default, Prophet detects 25 changepoints uniformly placed in the first 80% of the time series. Also these settings can be changed to better adapt the different scenarios. [35]

Along with the forecasts, Prophet returns also the uncertainty intervals, returning upper and lower values for the predictions. This is done by extending forward the generative model, which is given by the number of changepoints S over a history of T points, each with a rate change given by $\delta_j = \text{Laplace}(0, \tau)$, to simulate possible future trends which are then used to compute uncertainty intervals. Prophet estimates the uncertainty intervals using Monte Carlo simulation. The uncertainty interval is controlled by two parameters: the number of samples used to estimate the uncertainty interval and the interval width. Prophet allows also to change these two parameters, to better suit the different scenarios.

As seen, Prophet is a tool developed to forecast time series data, especially useful for business applications. However, some of its features are particularly suitable for our purpose in detecting anomalies inside network data. In particular, we can exploit its ability to detect changes in the trends and to signal the presence of dots that don't match the trend that is has predicted. Therefore, by applying Prophet to our case, we can interpret the appearance of changepoints in the trend and dots falling outside the uncertainty interval as anomalies in the traffic feature under observation. Looking to the changepoints, we can detect the "relevant anomalies" by looking at those changepoints appearing in more than one plot related to the same port. About the dots, Prophet finds and visualizes them. They can fit the prediction made by Prophet or not. According to their variation from the predicted values, we can evaluate them as anomalies.

In conclusion, Prophet looked to well suit for our scenario and purpose. Other forecasting tools, in fact, fail in captured any seasonality or longer-term seasonality and can't model properly yearly seasonality. As first, has been supposed to apply also some traditional anomaly detection approaches like the **box plot** [41] and the **Kolmogorov-Smirnov test** [42]. However, by reading about, what has been thought is that, in our scenario, in this way the deviations could be well detected just in some cases, while for those in which the trend looks very regular, due to the day-night effect, or very noisy, the box plot and Kolmogorov-Smirnov test could fail in visualize anomalies. Also, we want that not all the anomalies are signalled, but just the more relevant among the all set. Therefore, also traditional anomaly detection approaches, such as the box-plot and the Kolmogorov-Smirnov test, don't suit well for our particular scenario.

3.5 OpenTSDB and Grafana

The last part of the process has the aim to visualize the time series data so far obtained. In order to store the processed data and create plots, we make use, respectively, of **OpenTSDB** and **Grafana**.

OpenTSDB [36] is a distributed and scalable database particularly thought for time series data, on purpose to store this kind of data generated by endpoints and make them accessible and graphable. In time series data are reported at regular intervals of time and the dataset has two dimensions: the time, the independent variable, and the value, as dependent variable. The base working point of OpenTSDB is based on TSD, Time Series Daemon, that allows the interaction with the open source database HBase, therefore users never have to access it directly. HBase is the Hadoop database, distributed and scalable. It is useful when a random and real time read/write access to Big Data is needed. [37] Each TSD is independent from the others and the number of them used depends on the load to handle. Several options let the user to communicate with the TSD: a simple Telnet-style protocol, an HTTP API or a built-in GUI. Then, all communications take place on the same port. This OpenTSDB architecture is shown in figure 5. The use of OpenTSDB in our scenario is due to the fact that we have a lot of data to deal with: we have a value for each hour, for each feature and for a total of 65.536 ports. By considering a year, this means a lot of data.

The two base operations are writing and reading.

The writing step allows to send time series data into OpenTSDB through the TSD. Then, OpenTSDB formats the data and stores it in HBase. Time series data points in OpenTSDB are stored in a format that consists in a UNIX timestamp, a metric name and a tag, to distinguish similar data points produced by different sources or related entities.

In the reading step the users access the system to retrieve the required time series data. To achieve this task, OpenTSDB provides either a built-in user interface for selecting one or more metrics and tags to generate a plot as an image or an HTTP API to connect OpenTSDB with external systems such as monitoring frameworks, dashboards, statistics packages or automation tools. [38]

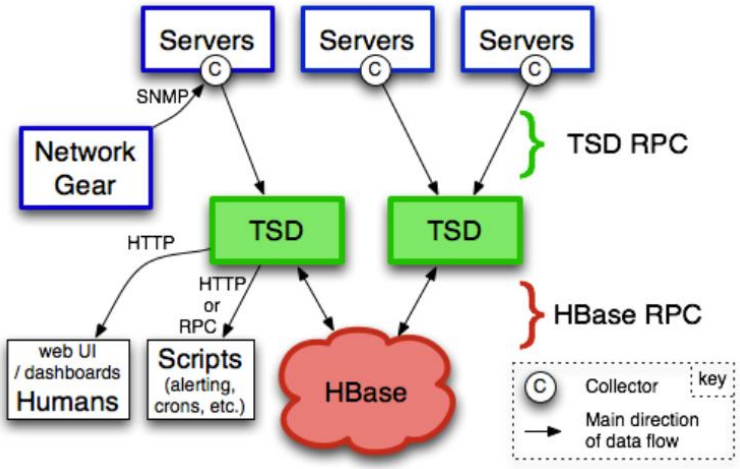


Figure 5: OpenTSDB architecture. From [38]

In our case, we use Grafana as external dashboard system connected to OpenTSDB, so that the time series data retrieved from the database are shown in an interactive form.

Grafana is an open source visualization tool, that can be employed to run on top of a variety of different data store systems. Grafana was developed for time series analytics and monitoring. [39]

The Query Editor in Grafana allows to query the metrics contained in the Data Source, in our case OpenTSDB, by building one or more SQL queries for the time series stored in the database.

The basic visualization building block is the Panel. Each Panel provides a Query Editor, to allow to extract the proper data to be visualized. One or more Panels can then be placed in the Dashboard.

Since we have extracted three network features for each destination port, we have three panels arranged in the dashboard. For each graph, Grafana provides also some metrices: the maximum value registered in the time interval under observation, the average value computed over the time interval, the current value and the total sum of all the values in the time interval.

The Dashboard allows to explore data in real time and in an interactive way, for example by zooming a certain period of interest we can have a more detailed visualization.

Figure 6 shows a shot taken from the last step in our system: the final visualization made with Grafana of the traffic features related to a chosen destination port, in this case port 3389. The red plot is related to the number of flows, the yellow one to the number of sources IP addresses and the green one to the number of destination IP addresses. The destination port can be changed by selecting the chosen one from the drop-down arrow in the top left. On the top right instead can be selected the period of interest. In this figure, the data is referred to the month of June 2019.

Figure 6 - Grafana Dashboard in the system

Chapter 4

Dataset and Case study

We illustrate the performance of the proposed system with a case study. We describe at first the dataset used to test the system and then we offer an overview on the results obtained, reporting as examples the cases related to four destination ports. It is also reported the investigation made upon an anomaly that has been found by applying the proposed system to our dataset.

4.1 Dataset

To test the performance of the proposed system, we use traces of traffic collected within the Politecnico di Torino network. These traces are related to a month of traffic previously collected. We start considering the log_tcp_nocomplete logs and we take the traffic related to March 2018. Log_tcp_nocomplete logs collect all the traffic produced by failed connections: they can be due either to errors in the connections or to malicious activities.

As first, we made a general analysis of the traffic to understand some statistics about it, like the top ten destination ports in the month for traffic volume. To this purpose, the traffic in the logs is filtered by considering all the incoming attempts of connection from external IP addresses, the clients, toward internal IP addresses, the servers. The client IP address in each flow is also only considered in the case in which it is not encrypted. Next, the so previous obtained flows are grouped by destination port and divided by the total number of flows, so that the percentage of connections that are directed toward each port is obtained. All the values are ranked in decreasing percentage values and the first ten in the month are considered.

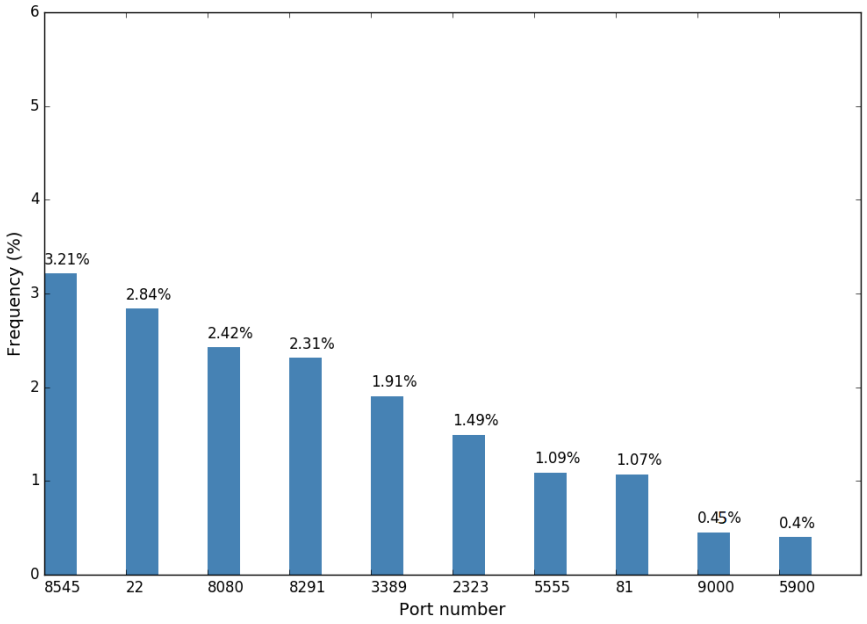


Figure 7: Top-10 ports per traffic volume, March 2018

Figure 7 shows the histogram so obtained, considering the top 10 destination ports ranked per traffic volume percentage in the month of March 2018, in Politecnico di Torino network.

Ports, and services running on them, have vulnerabilities. Very likely all of them can be targets of attacks, since none of them is totally out of the risk. However, some ports are more targets than others related on the type of services running on the port, the service version, its correct configuration, if there is a password required for the services and if they are strong enough. An example of that is Telnet service running on port 23. Telnet is an unsecure remote connection since it is in clear, so that is has been substitute with SSH. The use of default or weak passwords for it, makes this service vulnerable to brute force attacks. The well-known ports assigned for services can be aim of attacks since they are potentially open, waiting for incoming requests of connections. So, scan ports attacks are widespread and commonly made in order to verify an active port on the target hosts and, consecutively, exploit a known vulnerability of the service.

In our case, the histogram shows that the port 8545 has been the one with the most amount of traffic toward, with 3.21% of connection attempts. Port 8545 is the standard port for the JSON RPC (Remote Procedure Calls) interface, in Ethereum. The second target port is 22, where the SSH service runs on. Port 8080 is the default port used by proxy servers for client connections. Port 8291 is used for the WinBox service in MikroTik RouterOS based devices. Port 3389 is in use mainly for Windows Remote Desktop and Remote Assistance connections, but also by Windows terminal Server. Port 2323 is an alternative for port 23 Telnet service, particularly it is used for Telnet IoT. Port 5555 is an unofficial port used by several services, for example it is the default port for Softether VPN, HP Data Protector, SAP and others. Port 81 is an alternative port for HTTP service, which use by default port 80. Port 9000 is another unofficial port, used by multiple different web servers. Finally, port 5900 is officially registered for RFB protocol and it is also used for VNC for RFB protocol. [40]

Surprisingly, ports 23 and 80 don't appear in the histogram. This is due to the firewall configurations in our network, that limit the traffic toward these at-risk ports.

Particular cases are also the ports 53, for the DNS service, and 443, for HTTPS. In fact, without limiting the clients to be external, we can see a greater percentage of traffic towards these two ports. These because these two ports offer services to clients in internal Politecnico di Torino networks.

Next, similarly, the previous analysis has been repeated considering the daily top ten ports for traffic percentage, instead of considering their sum over all the month. The time series so obtained is shown in figure 8 and it gives a first idea of the ports traffic behaviour and displays, in some cases, the presence of peaks that could be thought to be caused by an anomalous activity and, therefore, further investigated. For example, what can be noticed is the fluctuating traffic volume trend for port 8080 and the appearance, at the end of the month, of a high volume of traffic toward a destination port not seen so far. The traffic in this case is considerable, since it doubles the one toward the top 3 ports in the same period and in the previous days. Also, it can be spotted an increasing in the traffic volume toward port 3389 at the end of the month.

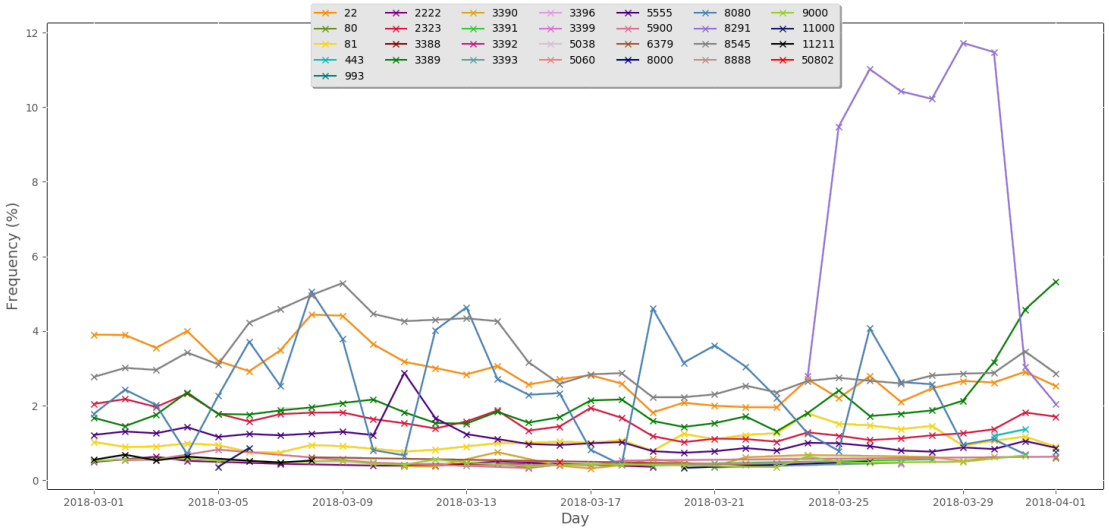


Figure 8: Traffic percentage of the top 10 ports per day per traffic volume

Another possible representation is obtained by considering just the ports' rank positions day by day. This rank is reported in figure 10. We can see that the top 3 positions are almost occupied by the same ports, with the exception of the last days, when the port 8291 appears in first position, as also already observed before.

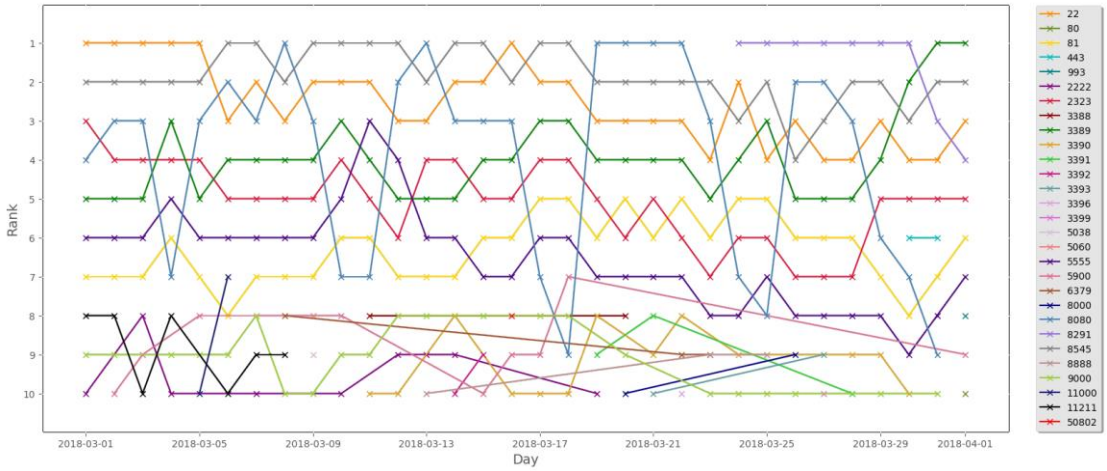


Figure 9: Rank of the top 10 ports per day per traffic volume

4.2 Traffic Visualization

So far, we have visualized the daily traffic volume per destination port. The same log_tcp_nocomplete logs for the traffic in the month of March 2018 are now analysed through our proposed system, described before. In this case, data are aggregated for hours, offering a more detailed representation. As parameters from command line are passed the logs with the data to process, the port and the feature of interest to extract. Looking at the histogram, we have a rank of the ports toward which the percentage of traffic is the most in the month. Using this histogram as reference, we submit the script to be processed by the cluster, considering successively these ports to get their trends regarding the different features.
The time series output is stored in OpenTSDB and then retrieving to be represented with Grafana. In this way, we get the traffic visual representation that we wanted.

We report the plots obtained in this way and related to four among the ten ports previously mentioned: port 8291,3389, 2323 and 81. We choice these four ports because they show all different trends in the plots. In this way, we can test the efficiency of our anomaly detection approach in different situations.

First, we can look at the plots to make some general observations. Figure 10 shows the plots related to the representation of the number of internal destination IP addresses for the destination ports above mentioned, taken from the Grafana dashboard in our proposed system. We can spot the presence of port scans targeting the entire network when, as shown in the plots, peaks are present. In particular, we can notice that the peaks in every plot are approximated at a maximum value that goes from 34200 to 34800. This range is due to the fact that a different number of internal IP addresses could have replied to the port scan toward the different destination ports and, in this case, the relative flow is stored in the log_tcp_complete log instead in the one that we are currently analysing. We can also notice that different trends are shown by the different destination ports.

These port scans, and in general any malicious activity, can be the work of a loner IP address or of a botnet. An idea of the situation can be given looking at the number of external source IP addresses in correspondence of the scanned IP addresses peaks. Figure 11 displays the time series obtained with our proposed system related to the number of source IP addresses, always considering the same destination ports above mentioned. Also here, according to the specific destination port under observation, different behaviours arise.

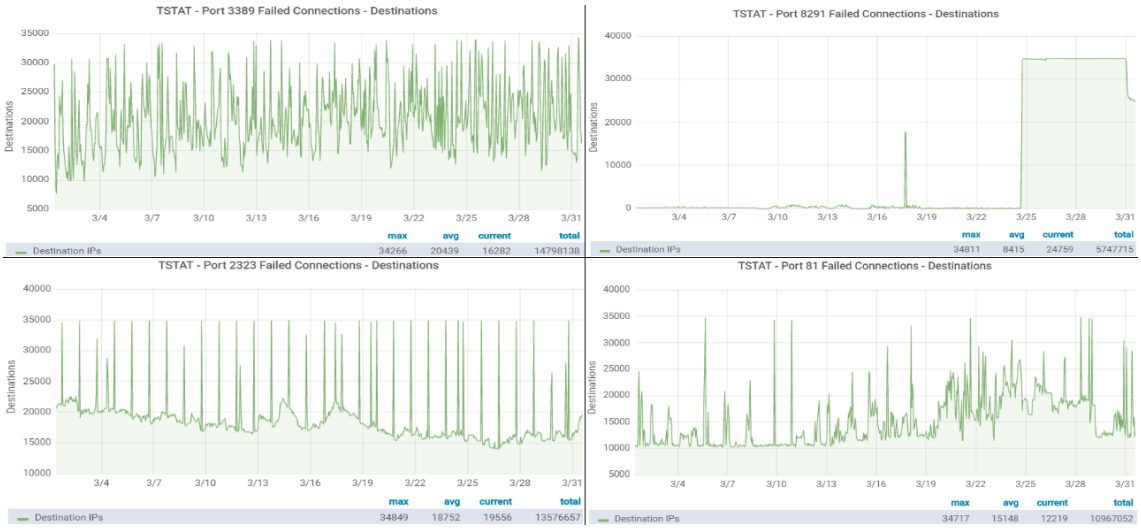


Figure 10: Number of destination IP addresses per hour in the month of March 2018 for destination ports 3389 (top left), 8291 (top right), 2323 (bottom left) and 81 (bottom right)

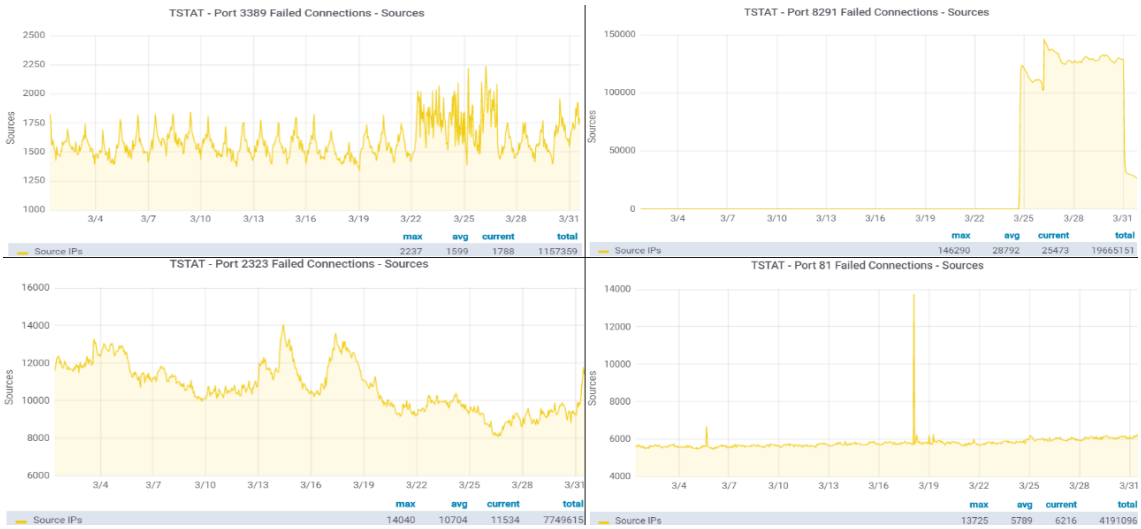


Figure 11: Number of source IP addresses per hour in the month of March 2018 for destination ports 3389 (top left), 8291 (top right), 2323 (bottom left) and 81 (bottom right)

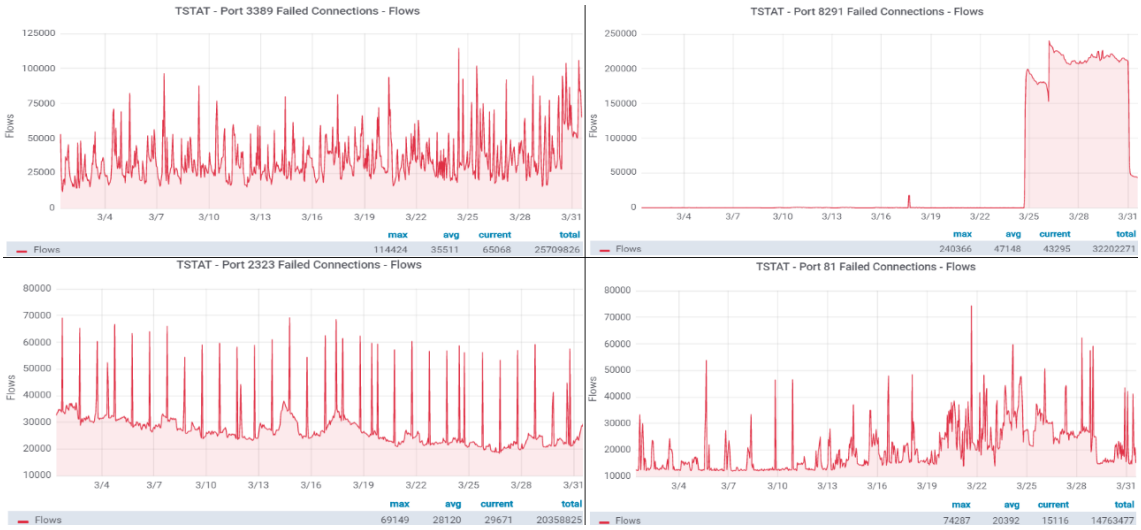


Figure 12: Number of flows per hour in the month of March 2018 for destination ports 3389 (top left), 8291 (top right), 2323 (bottom left) and 81 (bottom right)

Lastly, in figure 12 is represented the time series related to our third extracted network feature: the number of flows toward our usual four destination ports. Also in this case, different trends are shown according to the destination port at which we are looking.

Looking at these plots we can see as, in some cases, a suspicious situation is easier to be deduced due to the appearing of peaks. For example, in the plots for port 8291. In such cases, using some traditional anomaly detection approaches like the **box plot** [41] and the **Kolmogorov-Smirnov test** [42], the deviations over the trends for this port are well detected**.** However, looking to the other ports in our scenario, we can notice that some trends look very regular, due to the day-night effect. Therefore, as first, the box plot and Kolmogorov-Smirnov test could fail in visualize anomalies in these cases. Second, we want that not all the anomalies are signalled, but just the more relevant among the all set. Therefore, the box-plot and the Kolmogorov-Smirnov test don't suit well for our particular scenario.

4.3 Anomaly Visualization with Prophet

As already previously discussed, ports feature trend can been very different according to the specific port under observation: in some cases the presence of peaks in the time series is evident while in others less due to the noise. As mentioned in the previous chapter, Prophet well fits all these cases. Since it works very well with time series that have strong seasonal effects, it is particularly suitable for our application case since we have ports that show an apparently regular behaviour related to the day-night and week-weekend effects, in which could be not easy to visualize outliers.

In Prophet, like in other forecasting tools, the starting point is samples of data, then the model is fit on theses samples and this obtained model is used to make forecasts. Since Prophet is available in Python, we use a second script written in Python to perform this step.

The input for the script is a csv file containing the time series obtained as output from the processing step. Starting from the time series in the csv file, in the script creates as first a ds and y data frame, since the input to Prophet must always be a data frame with these two columns. Ds stands for “datestamp”, therefore this column contains the time labels, while the y column hosts the numeric measurements we want to forecast. Then the data frame is passed to the Prophet `fit()` method, that allows to create the model for the trend. In order to obtain forecasts of our time series, we have to provide Prophet with a ds column containing the dates for which we want the predictions. This is done with the `make_future_dataframe`() method, that allows to create the time interval for the prediction.

In Prophet, is important to consider the frequency of our time series. Since we are working with data related to hours, we specify as parameters “periods” equals to 24 and and “freq” H, in order to obtain the right frequency of timestamps. In this way, the make_future_dataframe methods generates 24 hourly timestamps.

```
future_dates = m.make_future_dataframe(periods=24, freq='H')
```

Then, predictions are made passing the dataframe with the created timestamps to the `predict()` method. The method returns the predicted value, called *yhat*. Together with the

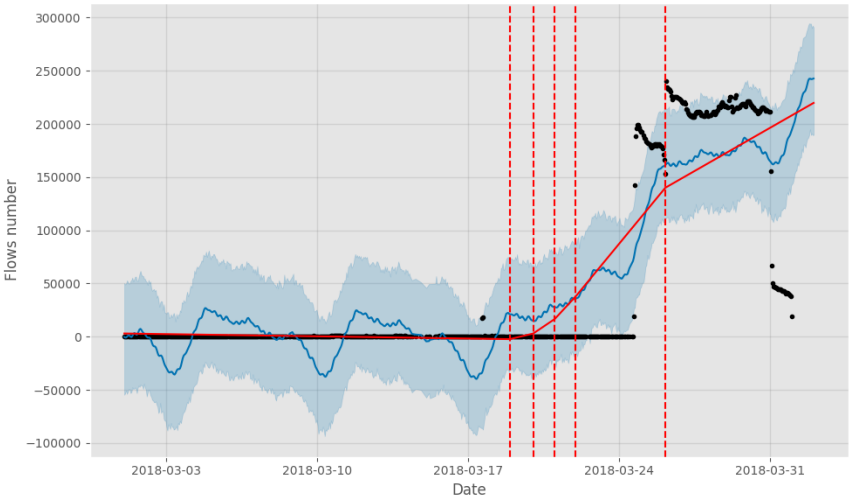
forecast yhat, other measurements are returned, like the trend. Among all these returned measurements, along with yhat, we take for our purpose, *yhat_lower* and *yhat_upper*, which together define the uncertainty intervals of the forecasts.

We report some examples of those forecasts made with Prophet for the time series related to the network features previously considered and for the same destination ports above examined: 8291, 3389, 2323 and 81. The blue lines indicate the forecasted values, the light blue area the uncertainty interval and the dots the actual values. The red dashed lines represent the changepoints, the points in which the trend changes, while the red line underlines the trend.

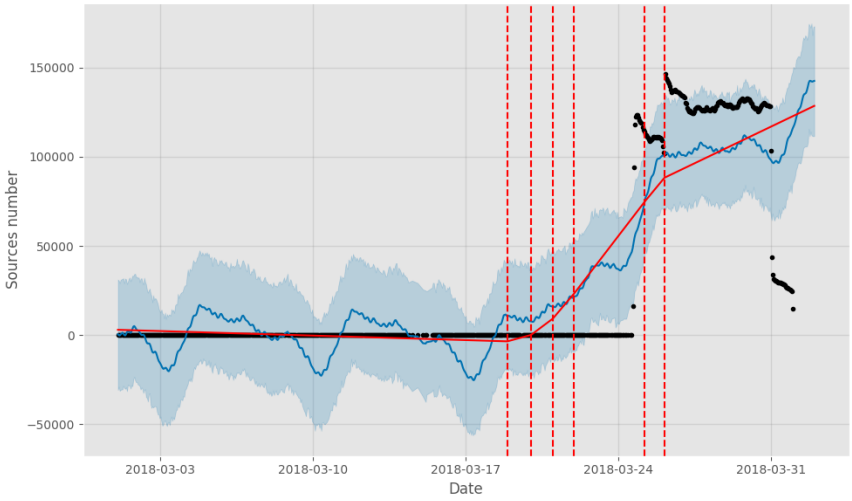
Looking at the graphs made with Prophet, we can spot anomalies under two forms: the change in the trend and the presence of dots falling outside the uncertainty interval. For what concerns the changepoints, Prophet allows to set their number. In the following plots, we left the default value, which is 25. This means that a maximum of 25 changepoints could be detected. Looking to the changepoints, we can detect the relevant anomalies. For “relevant anomalies” we mean those changepoints appearing in more than one plot related to the same port. About the dots, Prophet finds and visualizes them. The presence of dots outside the interval means traffic samples, always related to an hour, that don’t suit the trend forecasted by Prophet.

As first, figure 13 shows the three plots related to the three network features obtained by applying Prophet to the time series having destination port 8291. As we can see, in all the three plots we have almost the same changepoints and four in particular appear in all of them.

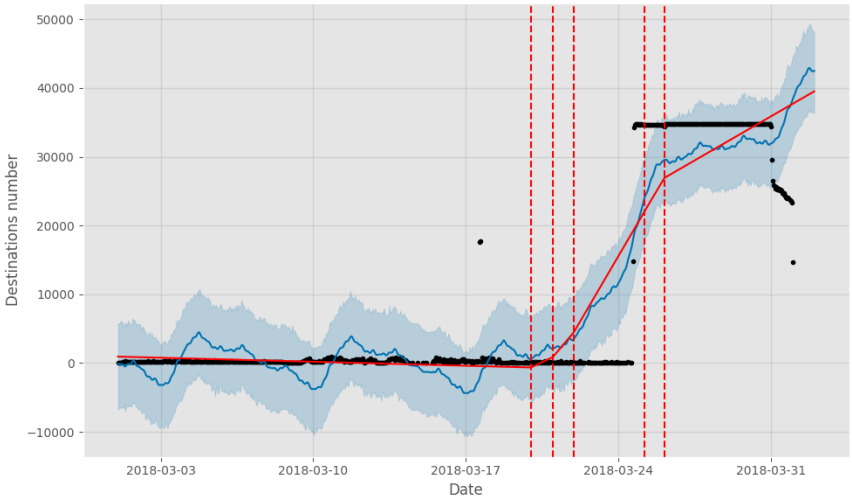
As shown, the prediction made by Prophet doesn’t follow exactly the actual samples trend, which is almost always constant, but they fall all inside the uncertainty interval. However, when the actual samples suddenly change their trend at the end of March, this deviation is detected in the plots. Looking instead at the dots, they almost always fall in the uncertainty interval. The relevant case in which this doesn’t happen occurs in the plot for the destination number, in which around the 17 March a sample deviates from the others.



(a)

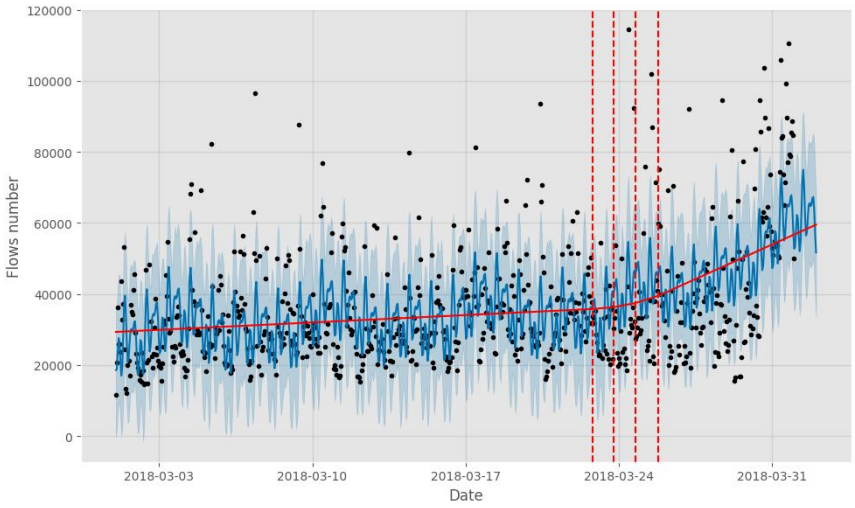


(b)

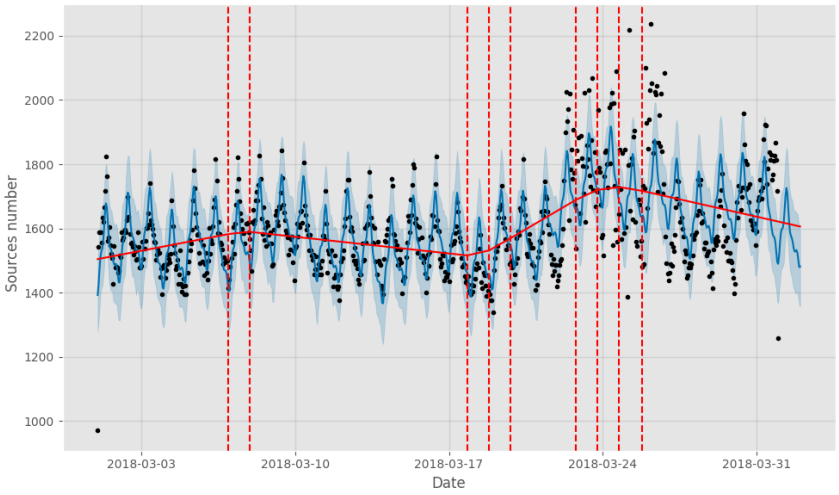


(c)

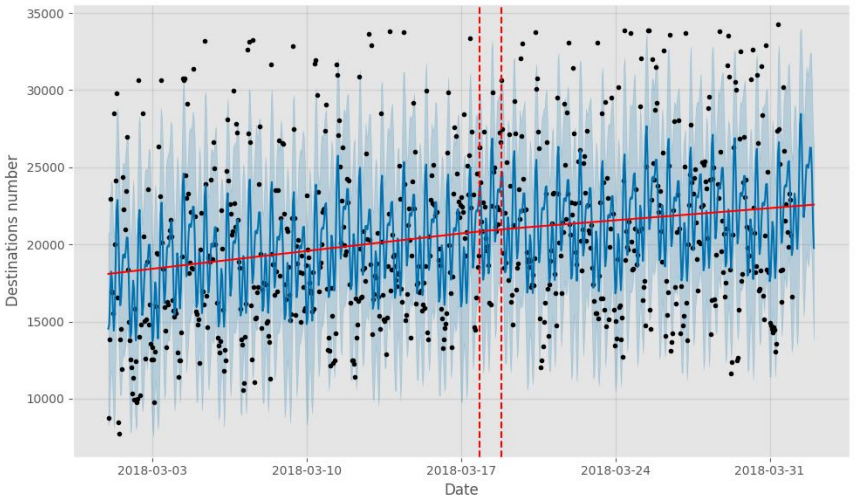
Figure 13: Prophet port 8291: number of flows (a), number of source IP addresses (b) and number of destination IP addresses (c)



(a)



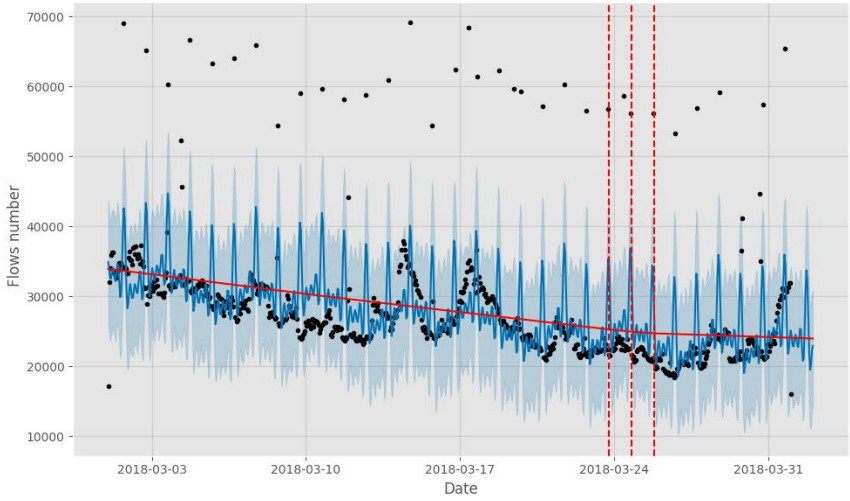
(b)



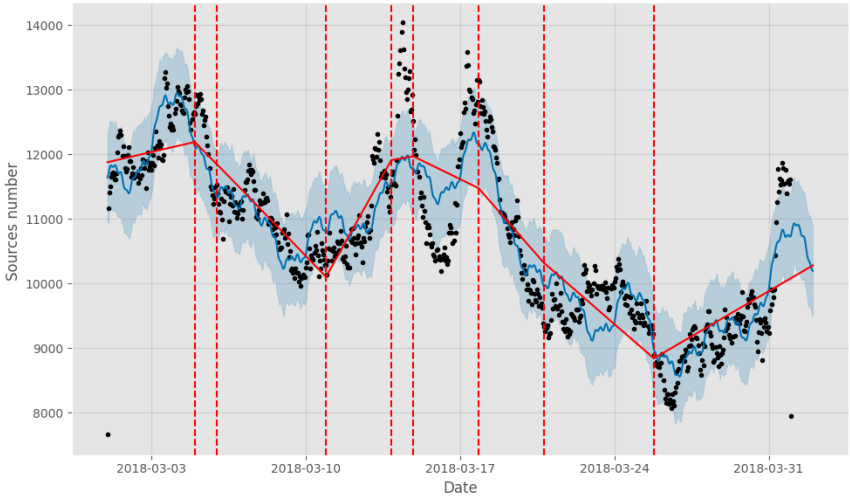
(c)

Figure 14: Prophet port 3389: number of flows (a), number of source IP addresses (b) and number of destination IP addresses (c)

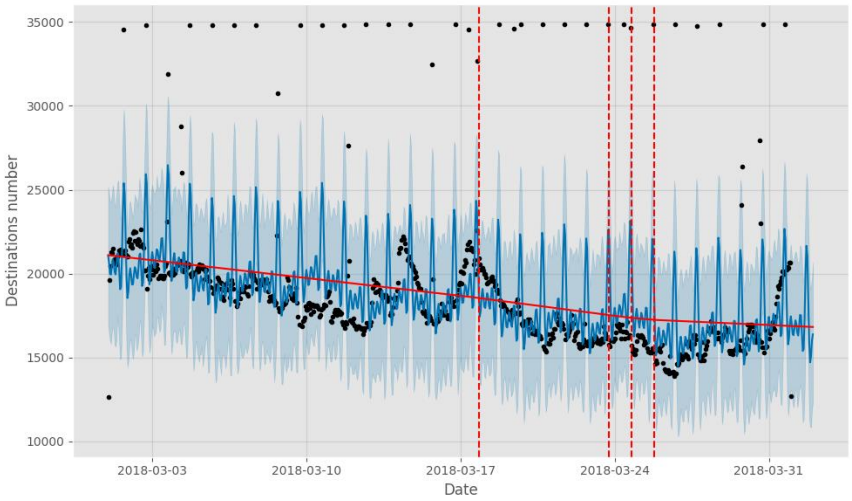
In figure 14, we show the same plots obtained with Prophet forecasts but having as destination port 3389. In this case, can be noticed a certain regularity in the behaviour, due to day-night effect, which is well captured by the forecasts made by Prophet. In the plot related to the number of flows, a certain number of dots fall outside the interval. Moreover, towards the end, a change in the trend is detected. This change is also detected in the plot representing the number of source address toward this port. In this plot, a greater number of changes in the trend is also detected. However, with respect the previous plot, the dots are almost always in the interval. In the number of destinations plot, the anomalies detected are more due to dots outside the interval. The trend, instead, almost doesn't change.



(a)



(b)

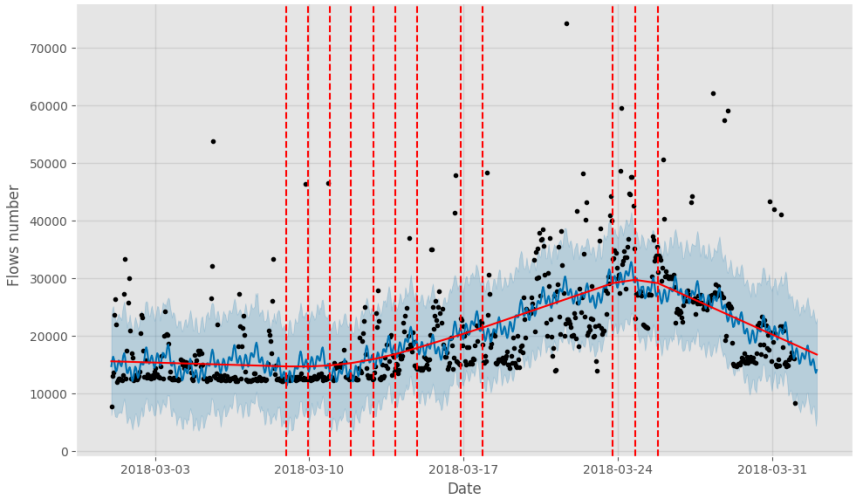


(c)

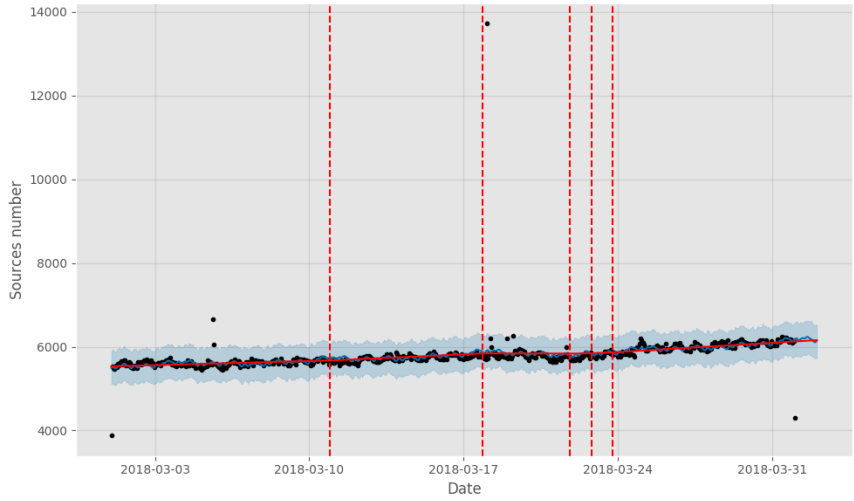
Figure 15: Prophet port 2323: number of flows (a), number of source IP addresses (b) and number of destination IP addresses (c)

The same plots related to destination port 2323 are shown in figure 15. This port also shows a kind of regular behaviour in the plots related to the number of destinations and the number of flows. The plot representing the number of sources addresses instead is more irregular and shows a lot of changes in its trend, with a certain number of dots outside the interval in correspondence of two of these changepoints. The other two plots instead have a trend almost constant, and a lot of dots falling outside the interval. In particular, in the plot for the number of destinations, dots are almost aligned on the same value, as previously discussed. The changepoints in these two plots are almost the same, while only one is appearing in all the three plots.

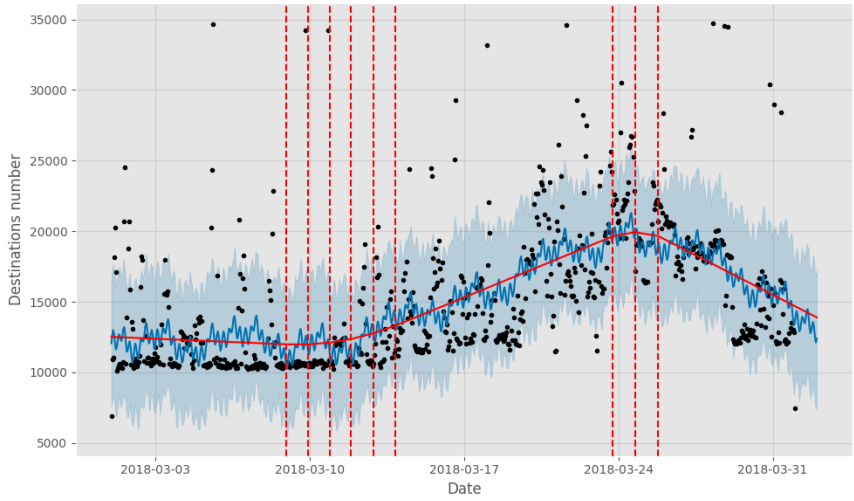
As last, in figure 16 we report the three plots related to the port 81. By looking at the first plot, a lot of little changes in the trend are detected and a certain number of dots outside the interval is spotted. By looking instead at the second plot, some changepoints are reported. However, the trend, represented by the red line, almost never changes. So in this visualization, a greater attention can be given to the dots rather than the changepoints. In the last plot, like in the first one, a certain number of changepoints and dots outside the interval are visualized.



(a)



(b)



(c)

Figure 16: Prophet port 81: number of flows (a), number of source IP addresses (b) and number of destination IP addresses (c)

The changepoints appearing in the plots are automatically detected by Prophet when a change in the trend occurs. The number of changepoints, in all those cases, was by default set to be at maximum equal to 25. Prophet however lets to change this number. By reducing the number of changepoints, Prophet shows a smaller number of them and so the most important are reported.

For example, we try to change it to 10. The comparison among these two cases is shown in figure17: in figure 17(a) we report the default case, so far used, while in figure 17(b) we can see how the changepoints changes when we modify the parameter that controls their number. In particular, we can see that the number of changepoints in this way is decreased.

So, by decreasing the number of changepoints, we can control the number of anomalies to be signalled, letting Prophet to report us only the most relevant changes in the trend.

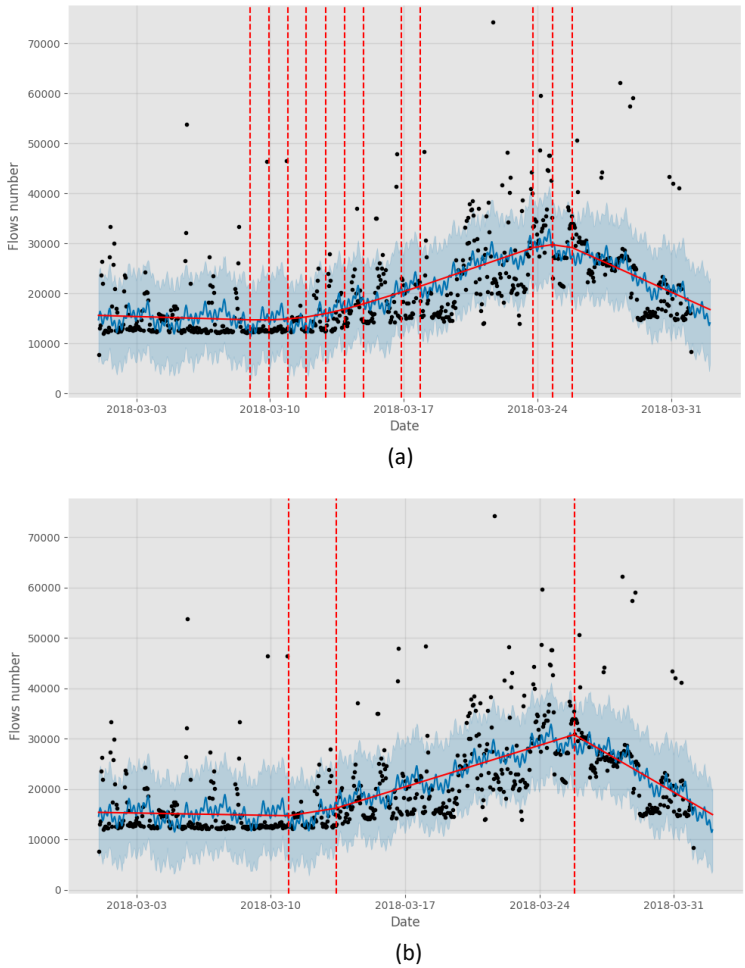
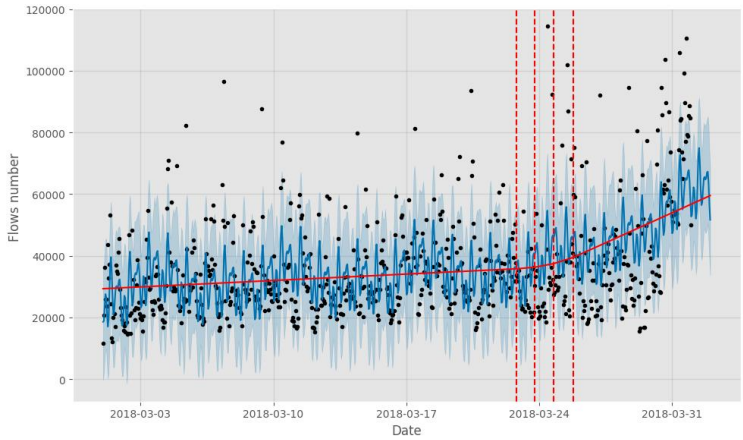


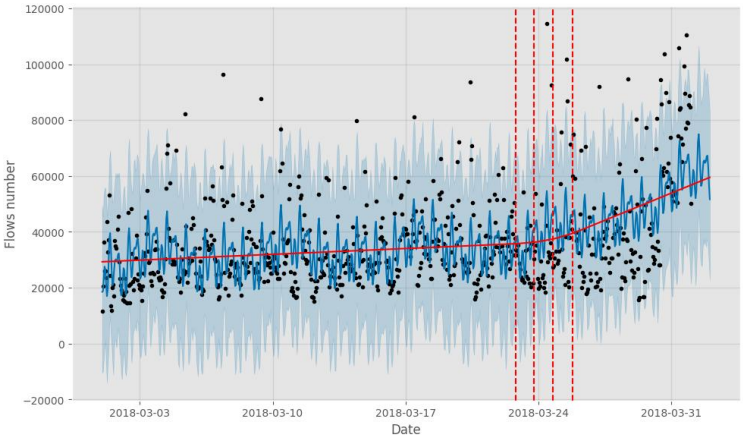
Figure 17: Default number of changepoints (a) and by decreasing their number to 10 (b)

Like the changepoints, also the interval of uncertainty width can be modified, by increasing or decreasing it.

As already said in the previous chapter, Prophet estimates the uncertainty intervals using Monte Carlo simulation. The width of this interval is controlled by the percentage of the samples generated by the Monte Carlo simulation used to cover the uncertainty interval.



(a)



(b)

Figure 18: Default interval width (a) and by increasing it (b)

By default, Prophet set the width equal to the 80% of the generated samples. We try to change it, through the `interval_width` parameter, to set it equal to 95%. In this way, we obtain a

greater uncertainly interval. The plots in figure 18 report these two cases: in figure 18(a) the case with the default interval width values while in figure 18(b) the case obtained when we increase it. As can be seen, by increasing the interval width, we allow a greater number of dots to fall inside such interval, so that a smaller number of samples are identified as anomalies.

Therefore, the number of changepoints and the width of the uncertainty interval allow us to control the number of appearing anomalies in the plots, according to how many of them we want to visualize in our system.

4.4 The case of IoT attacks on port 8291

As already said, a difficulty in anomaly-based detection, differently from the signature-based approach, is to identify the kind of anomaly when it arises.

So as last step, the event that led to the alarm can be investigated to better understand if it is a false positive or a real threat and, in this case, study it in deeper to identify it. In order to perform this analysis, we make also use of the log_tcp_complete log.

As example, we report the case of the port 8291. The port 8291 is used for the WinBox service in MikroTik RouterOS based devices. As we can see from the graph describing the traffic trend over this port, it is always very low and, consequently, the enormous amount of traffic toward it exploded starting from the 24 March could only generates an alarm. But a significantly increase in the traffic addressed to this port, beginning from the 24 March (around 25 March 00:00, Beijing time), has been also observed worldwide. The attack was made by the Hajime based botnet, a variant of the Mirai one. As already mentioned, the Mirai botnet [2] was the first malware targeting IoT devices seen at global level. In the first phase, the devices already compromised scan pseudo-random IP addresses looking for open 23 and 2323 ports. Once discovered, they perform a brute force attack to discover the devices' login credentials, by trying a combination of default usernames and passwords commonly used for IoT devices. If this

operation succeeds, the information about the device, such as its operating system, are investigated so that the device is forced to download and run the specific malware version. At this point, the compromised device has joined the botnet, so it starts scanning in turn the network looking for targets and waits for commands from the control and command server, which controls and coordinates the botnet. Since Mirai code has been published, several of variants have been developed and used over the years, targeting different ports to exploit existing vulnerabilities. One of these cases is the Hajime based botnet, which targeted, in this case, the port 8291. As in the Mirai botnet, there is a first port scan phase to determine whether the port of interested is open over the target device and to exploit next its known vulnerabilities to infect and spread. The known Mikrotik vulnerabilities exploited by the botnet are HTTP and SMB. The figure 19, taken from a report related to this attack from [43], shows all the steps of the attack. As we can see, the first part of the attack is a SYN scan toward the port 8291, made to determine if the target is a MikroTik device, on which the port 8291 is open. If an IP address replies to the sent SYN with a SYN ACK segment, the client further scans looking for an open port among the common web ports 80, 81, 82, 8080, 8081, 8082, 8089, 8181 and 8880 on that device. If the IP address replies also this time, a connection is opened, and, through an HTTP request, the worm checks the device version to send the proper exploit carrying the shellcode. At this point, Hajime is downloaded and executed, and the device joins the botnet. [44]

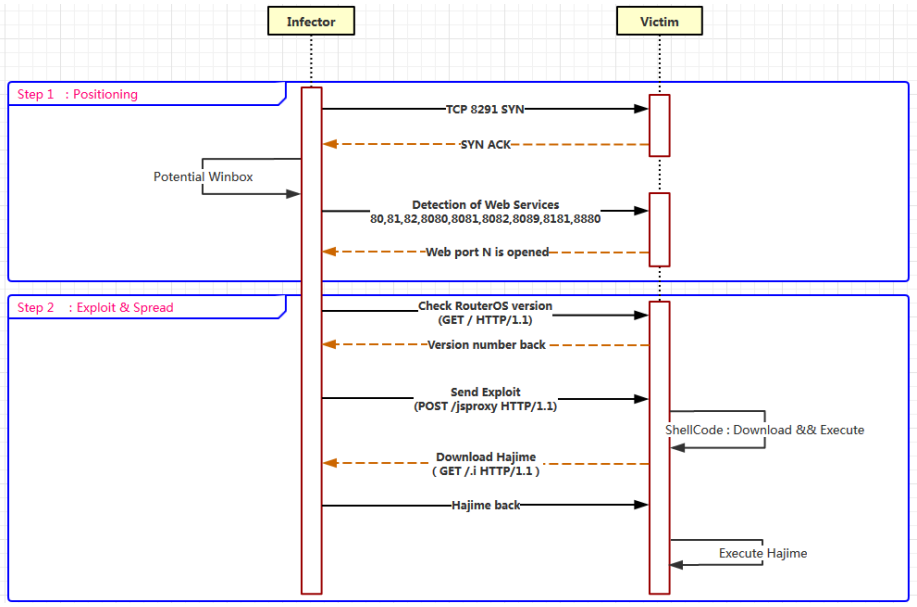


Figure 19: Attack pattern. From [43]

We want to see if this attack pattern is also found in our traffic traces.

At first glance, from the previous plots related to this port, reported in figure 10, 11 and 12, we can see just an evident anomaly trend. In all the three plots in figure 13, a change in the trend is detected with Prophet around the March 24.

These deviations, in this particular time period, could be linked to the port scan phase activity toward port 8291 made worldwide by the Hajime botnet attack: what we can derive from these plots is that there is a huge number of flows, made by a great number of different source IP addresses and against all the IP addresses in our network that, of course, look to be caused from a port scan activity, but it could be related to any malicious one.

So, from these graphs we can see just the aggressive port scan activity, without any other proof.

For this reason, in order to further search signs of this attack, a source IP address is selected among the set of the all that generate the traffic against the port under investigation in a day falling in the supposed attack interval of time. At the end, the chosen IP address is 222.88.37.66 since it is an IP address appearing both in the log_tcp_nocomplete and log_tcp_complete logs. This means that one or more internal IP addresses in the network replies to it. As first, the data in the log_tcp_nocomplete are processed so that, in the selected day, are returned all the connection attempts toward the port 8291 made by the source IP address 222.88.37.66 against the IP addresses in our network. The output displays an activity from this IP address from 2:10 a.m. until 5.25 a.m. for a total of about 34.840 SYN sent. A picture of this output is reported in figure 20(a), where the first field indicates the date, the second the source IP address, the third the destination IP address and the last the target port. This clearly shows the port scan phase made by the IP address 222.88.37.66. Next, the same processing procedure is applied to the log_tcp_complete. The output is shown in figure 20(b) and, from it, we can see the 8 internal IP addresses that replied to the SYN segment sent.

So at this point, the next step is to check whether this IP address makes other activity. Looking at the botnet attack model, we expect it will start a port scan against the common web ports. Therefore, similarly to what was done before, we process the data in the log_tcp_nocomplete belonging to the 29 March, filtering for our source IP address and destination port so that it is one among 80, 81, 82, 8080, 8081, 8082, 8089, 8181 and 8880. We found a port scan made by the IP address 222.88.37.66 toward these ports and, as expected, differently from before it doesn’t target all the network but just the 8 IP addresses previously found, checking one port

per hour. What is obtained is exactly reported in figure 20(c): starting from 6 a.m. the IP address checks for port 81, starting from 7 a.m. for port 8080, starting from 8 a.m. for port 8081 and so on for the next 2 hours for, respectively, port 82 and 8082. Since this data are taken from the log_tcp_nocomplete, it contains just the IP address that have been contacted with a SYN but that didn't reply to it. In fact, some IP addresses are missing with respect the ones found before. So, as before, we apply the same processing procedure but to the log_tcp_complete, looking for the successful connections made by the usual source IP address toward the same common web ports. What is obtained is shown in figure 20(d) and we can notice that there are the "missing" address from the previous picture, that, since replied to the scan, have been stored by Tstat in the other log.

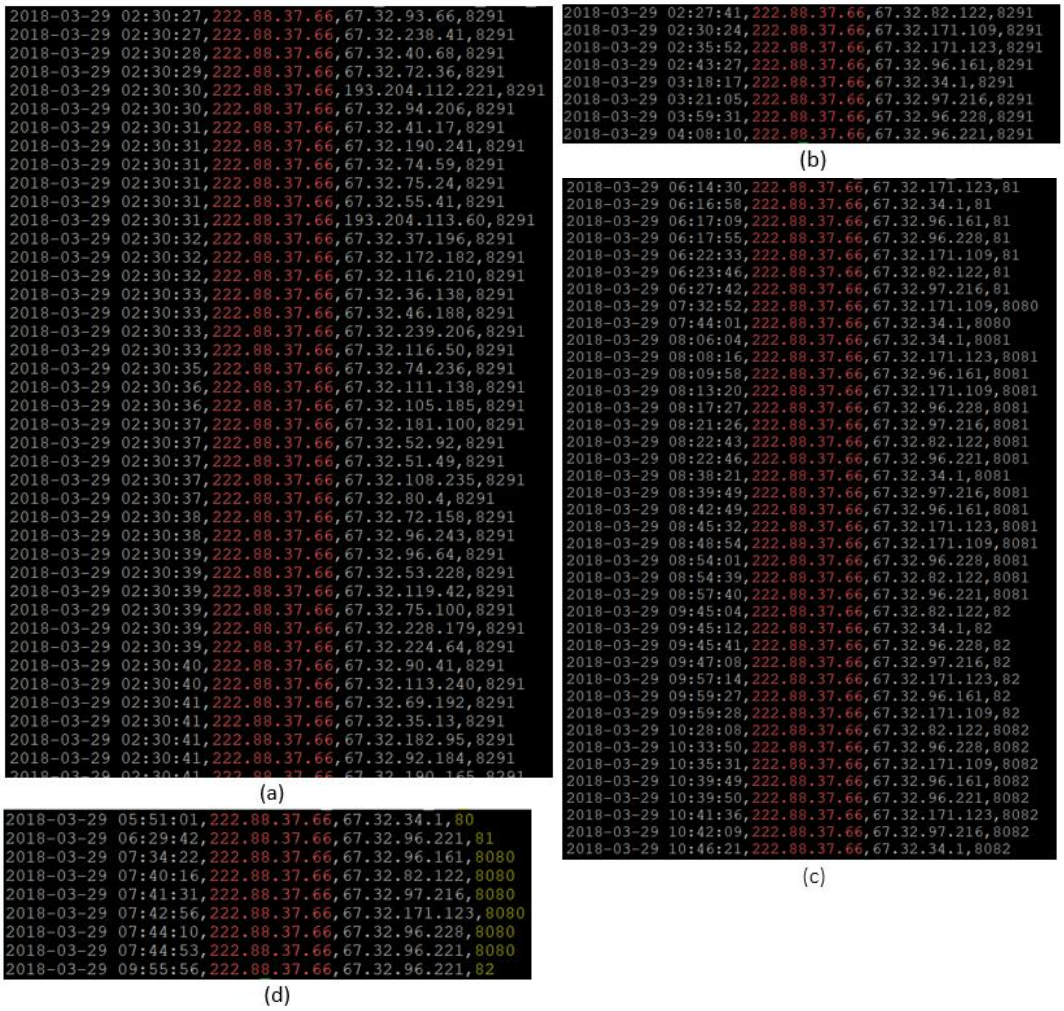


Figure 20: Portion of port scan toward port 8291 (a), internal IP addresses which replied to the scan (b), port scan toward common web ports (c) and internal IP addresses which replied to it (d)

So, by extracting the proper network features, aggregating them in time series per hours, applying Prophet to our time series data and representing them in a visual way, we are able to detect anomalies inside our network traffic. In the example case just reported, which takes in consideration the network traffic toward the destination port 8291 in the month of March 2018, we are able to detect the appearance of anomalies, due to a botnet activity, by looking at the changes in the trend in our time series plots.

Chapter 5

Conclusions and future work

Concluding remarks and future work

In this work we have presented a system to provide a possible solution to visualize network traffic and the most relevant anomalies inside it. It is mainly based on a graphical representation of the meaningful features that define the network traffic and a forecast component for the actual anomaly detection step, made with Prophet. The detection step is in particular based on the visualization of anomalies under two forms: the changes in the trend and the presence of samples falling outside the uncertainty interval of the forecasted trend. The use of Prophet well suits for our scenario, since the traffic profile toward some ports follow a certain seasonality trend, due to day-night and week-weekend effects, and Prophet is particularly thought to operate with time series data showing seasonality effects. The full process has been tested for a month of traffic and considering just the top ten ports in the month, according to their traffic volume. Future works can be focus on testing the traffic collected over a wider interval of time and verifying a bigger number of ports. Of course, due to the learning approach at the base of Prophet, by running it over data related to a wider interval of time, the precision for the forecast will be more accurate. Also, the analysis made with Prophet is currently not integrated with OpenTSDB and Grafana and it will be do in the future. For example, from Prophet it will be possible to get as output a time series with a field for the timestamp, a field to report the presence or not of an anomaly in that hour (1 anomaly, 0 otherwise) and the tag for the destination port. In this way, the timeseries can be stored in OpenTSDB and then used to mark the presence of anomalies directly on the graphs in the Grafana dashboard. As last observation, the script to process data traffic is made to extract three traffic features: traffic volume, number of source IP addresses and number of destination IP addresses. The system can be improved by adding other features to be extracted. This can be done easily, by adding the needed lines of

code to the script. A drawback in this system is that, in some cases, changepoints reported are due to small changes in the trend that could be not of interest, but they are signalled anyway. Therefore, we should need a way to specify when the change in the trend is enough significant to be signalled or not.

Appendix

A.1

```
#!/usr/bin/python
from pyspark import SparkConf, SparkContext
from datetime import datetime
import sys
import operator
from operator import add
from datetime import time, date, datetime, timedelta
import datetime


in_path = sys.argv[1]
port = sys.argv[2]


def server_interni_func(x):
	val = 0
	try:
		val = int(x.split(" ")[38])
	except:
		pass
	if val==1:
		return True
	else:
		return False
```

```
def client_esterni_func(x):
	val = 0
	try:
		val = int(x.split(" ")[37])
	except:
		pass
	if val==1:
		return False
	else:
		return True



def client_crypt_func(x):
	val = 0
	try:
		val = int(x.split(" ")[39])
	except:
		pass
	if val==1:
		return False
	else:
		return True



def client_ip_func(x):
	ip_c = x.split(" ")[0]
	strDate = x.split(" ")[28]
	convertedTime = datetime.datetime.fromtimestamp(float(strDate)/1000.)
	formatTime = convertedTime.strftime('%Y-%m-%d %H:00')
	return (formatTime, ip_c)



def server_ip_func(x):
	ip_s = x.split(" ")[14]
	strDate = x.split(" ")[28]
	convertedTime = datetime.datetime.fromtimestamp(float(strDate)/1000.)
```

```
	formatTime = convertedTime.strftime('%Y-%m-%d %H:00')
	return (formatTime, ip_s)


def port_func(x):
	val = 0
	try:
		val = x.split(" ")[15]
	except:
		pass
	if val==port:
		return True
	else:
		return False


def tempo_val(x):
	strDate = x.split(" ")[28]
	convertedTime = datetime.datetime.fromtimestamp(float(strDate)/1000.)
	formatTime = convertedTime.strftime('%Y-%m-%d %H:00')
	val = 1
	return (formatTime, val)


def replace_func(x):
	date = x[0]
	val = 1
	return (date, val)

def date_func(x):
	strDate = x.split(" ")[28]
	convertedTime = datetime.datetime.fromtimestamp(float(strDate)/1000.)
	date = convertedTime.strftime('%Y-%m-%d %H:00')
	return (date)
```

```
def main():

	#create a configuration object
	conf = (SparkConf()
		.setAppName("Log processing")
		.set("spark.dynamicAllocation.enabled", "false")
		.set("spark.task.maxFailures", 128)
		.set("spark.yarn.max.executor.failures", 128)
		.set("spark.executor.cores", "8")
		.set("spark.executor.memory", "7G")
		.set("spark.executor.instances", "50")
		.set("spark.network.timeout", "300"))

		   #create a Spark context object
	sc = SparkContext(conf = conf)

	log_tcp = sc.textFile(in_path + "/log_tcp_nocomplete.gz")

	description = log_tcp.first()
	log_tcp_nodescription = log_tcp.filter(lambda x: x != description)

	#internal servers
	server_interni_RDD = log_tcp_nodescription.filter(lambda x:
	server_interni_func(x))

       #external clients
	server_interni_client_esterni_RDD = server_interni_RDD.filter(lambda x:
	client_esterni_func(x))

	#clients no anonymized
	client_noan_RDD = server_interni_client_esterni_RDD.filter(lambda x:
	client_crypt_func(x))

	if len(sys.argv)==4:
		feature = sys.argv[3]
```

```
date_RDD = client_noan_RDD.map(lambda x: date_func(x))

uniq_date_RDD = date_RDD.distinct().collect()


lista_data = sorted(uniq_date_RDD, reverse = False)

filtered_RDD1 = client_noan_RDD.filter(lambda x: port_func(x))

if feature=='flows_numb':

       time_RDD = filtered_RDD1.map(lambda x: tempo_val(x))

       time_flow_RDD = time_RDD.reduceByKey(add)

       lista_flussi = time_flow_RDD.collect()

       for d in lista_data:

              for (k,v) in lista_flussi:

                     if (d==k):

                            print("%s" %d + ","  + "%s" %v + "," +
                            "%s" %port)

                            break

              if (d!=k):

                     print("%s" %d + ","  + "0" + "," + "%s"
                     %port)


elif feature=='src_ip':

       time_RDD = filtered_RDD1.map(lambda x:
       client_ip_func(x)).distinct()

       replace_IP_RDD = time_RDD.map(lambda x: replace_func(x))

       final_RDD = replace_IP_RDD.reduceByKey(add)

       lista_flussi = final_RDD.collect()


       for d in lista_data:

              for (k,v) in lista_flussi:

                     if (d==k):

                            print("%s" %d + ","  + "%s" %v + "," +
                            "%s" %port)

                            break

              if (d!=k):

                     print("%s" %d + ","  + "0" + "," + "%s"
                     %port)


elif feature=='dst_ip':

       time_RDD = filtered_RDD1.map(lambda x:
       server_ip_func(x)).distinct()
```

```
                replace_IP_RDD = time_RDD.map(lambda x: replace_func(x))

                final_RDD = replace_IP_RDD.reduceByKey(add)

                lista_flussi = final_RDD.collect()

                for d in lista_data:

                        for (k,v) in lista_flussi:

                                if (d==k):

                                        print("%s" %d + ","  + "%s" %v + "," +
                                        "%s" %port)

                                        break

                        if (d!=k):

                                print("%s" %d + ","  + "0" + "," + "%s"
                                %port)

        else:

                exit()


elif len(sys.argv)==3:

        feature = sys.argv[2]

        if feature=='flows_numb':

                time_RDD = client_noan_RDD.map(lambda x: tempo_val(x))

                time_flow_RDD = time_RDD.reduceByKey(add)

                lista_flussi = time_flow_RDD.sortBy(lambda x: x[0],
                True).collect()

                for (k,v) in lista_flussi:

                        print("%s" %k + ","  + "%s" %v)


        elif feature=='src_ip':

                time_RDD = client_noan_RDD.map(lambda x:
                client_ip_func(x)).distinct()

                replace_IP_RDD = time_RDD.map(lambda x: replace_func(x))

                final_RDD = replace_IP_RDD.reduceByKey(add)

                lista_flussi = final_RDD.sortBy(lambda x: x[0],
                True).collect()

                for (k,v) in lista_flussi:

                        print("%s" %k + ","  + "%s" %v)


        elif feature=='dst_ip':

                time_RDD = client_noan_RDD.map(lambda x:
                server_ip_func(x)).distinct()

                replace_IP_RDD = time_RDD.map(lambda x: replace_func(x))
```

```
                    final_RDD = replace_IP_RDD.reduceByKey(add)

                    lista_flussi = final_RDD.sortBy(lambda x: x[0],
                    True).collect()

                    for (k,v) in lista_flussi:

                          print("%s" %k + ","  + "%s" %v)

             else:

                    exit()

      else:

             exit()


main()
```

A.2

```
import datetime

import os

import pandas as pd

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

import numpy as np

import sys

import random

matplotlib.style.use('ggplot')

import matplotlib.pylab as pl


from fbprophet import Prophet

from fbprophet.plot import add_changepoints_to_plot




#input path

path_br_timestamp = sys.argv[1]

headers = ['ds', 'y', 'port']

df = pd.read_csv(path_br_timestamp, names=headers)

df.drop("port", axis=1, inplace=True)


df = df.sort_values(by="ds", ascending=True)
```

```
#costruttore

#classic

m = Prophet()


#to change the number of changepoints

#m = Prophet(n_changepoints=10)


#change the trend flexibility (changepoints)

#m = Prophet(changepoint_prior_scale=0.003)


#to change the interval

#m = Prophet(interval_width=0.97)


#fit

m.fit(df)


#dates on which we wish to make the prediction

future_dates = m.make_future_dataframe(periods=24, freq='H')



#the actual prediction made by Prophet

forecast = m.predict(future_dates)


a = add_changepoints_to_plot(fig.gca(), m, forecast)
```

Bibliography

[1] https://www.gemalto.com/press/Pages/Data-Breaches-Compromised-3-3-Billion-Records-in-First-Half-of-2018.aspx

[2] Mirai botnet, https://blog.attify.com/how-mirai-botnet-hijacks-your-devices/

[3] https://press.f-secure.com/2019/04/01/iot-threats-same-hacks-new-devices/

[4] https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[5] Animesh Patcha, Jung-Min Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends", Computer Networks, Volume 51, Issue 12, 2007, Pages 3448-3470, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2007.02.001.

[6] Handong Wu, Stephen Schwab, Robert Lom Peckham, *"Signature based network intrusion detection system and method"*

[7] Xuanfan Wu, "*Metrics, Techniques and Tools of Anomaly Detection: A Survey",*
(at) go.wustl.edu

[8] A. T. Tran, "Network Anomaly Detection," Proc. Semin. Futur. Internet Innov. Internet Technol. Mob. Commun., no. September, pp. 55–61, 2017.

[9] Thottan, Marina & ji, Chuanyi. (2003). Anomaly Detection in IP Networks. Signal Processing, IEEE Transactions on. 51. 2191 - 2204. 10.1109/TSP.2003.814797.

[10] Ring, Markus & Wunderlich, Sarah & Scheuring, Deniz & Landes, Dieter & Hotho, Andreas. (2019). A Survey of Network-based Intrusion Detection Data Sets.

[11] Kind, Andreas, Marc Ph. Stoecklin and Xenofontas A. Dimitropoulos. "Histogram-based traffic anomaly detection." *IEEE Transactions on Network and Service Management* 6 (2009): n. pag.

[12] Salima Omar, Asri Ngadi and Hamid H Jebur. Article: Machine Learning Techniques for Anomaly Detection: An Overview. *International Journal of Computer Applications* 79(2):33-41, October 2013.

[13] Goldstein, Markus & Dengel, Andreas. (2012). Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm.

[14] Gerhard Münz and Sa Li and Georg Carle. “Traffic Anomaly Detection Using KMeans Clustering”. In GI/ITG Workshop MMBne. 2007.

[15] Chen, Zhenguo & Fei Li, Yong. (2011). Anomaly Detection Based on Enhanced DBScan Algorithm. Procedia Engineering. 15. 178–182. 10.1016/j.proeng.2011.08.036.

[16] Jayshree S.Gosavi, Vinod S.Wadne. “Unsupervised Distance-Based Outlier Detection Using Nearest Neighbours Algorithm on Distributed Approach: Survey”. 10.15680/ijircce.2014.0212042

[17] Robert Ball, Glenn A. Fink, and Chris North. 2004. Home-centric visualization of network traffic for security administration. In Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security (VizSEC/DMSEC '04). ACM, New York, NY, USA, 55-64. DOI: https://doi.org/10.1145/1029208.1029217.

[18] J. R. Goodall, W. G. Lutters, P. Rheingans and A. Komlodi, "Preserving the big picture: visual network traffic analysis with TNV," *IEEE Workshop on Visualization for Computer Security, 2005. (VizSEC 05).*, Minneapolis, MN, USA, 2005, pp. 47-54. doi: 10.1109/VIZSEC.2005.1532065.

[19] C. L. Sowmya, C. D. Guruprakash and M. Siddappa, “Visualization of network traffic”, International Journal of Smart Sensors and Ad Hoc Networks (IJSSAN), ISSN No. 2248-9738 (Print), Vol-2, Iss-3,4, 2012.

[20] Kiran Lakkaraju, Ratna Bearavolu, A. Slagell and W. Yurcik, "Closing-the-loop: discovery and search in security visualizations," *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, West Point, NY, USA, 2005, pp. 58-63. doi: 10.1109/IAW.2005.1495934.

[21] Fischer F., Mansmann F., Keim D.A., Pietzko S., Waldvogel M. (2008) Large-Scale Network Monitoring for Visual Analysis of Attacks. In: Goodall J.R., Conti G., Ma KL. (eds) Visualization for Computer Security. VizSec 2008. Lecture Notes in Computer Science, vol 5210. Springer, Berlin, Heidelberg.

[22] Tstat, http://tstat.polito.it/

[23] Tstat measure, http://tstat.polito.it/measure.shtml#LOG

[24] Big Data analytics, IBM, https://www.ibm.com/analytics/hadoop/big-data-analytics

[25] Big data, https://www.oracle.com/big-data/guide/what-is-big-data.html

[26] Apache Hadoop, https://hadoop.apache.org/

[27] HDFS Architecture Guide, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Disk+Failure%2C+Heartbeats+and+Re-Replication

[28] Apache Hadoop YARN, https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[29] Apache Spark, https://spark.apache.org/

[30] Gurusamy, Vairaprakash & Kannan, Subbu & Nandhini, K. (2017). The Real Time Big Data Processing Framework: Advantages and Limitations. INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING. 5. 305-312. 10.26438/ijcse/v5i12.305312.

[31] Class RDD, https://spark.apache.org/docs/1.1.1/api/python/pyspark.rdd.RDD-class.html

[32] Facebook Prophet, https://facebook.github.io/prophet/

[33] J. Taylor, Sean & Letham, Benjamin. (2017). Forecasting at Scale. The American Statistician. 72. 10.1080/00031305.2017.1380080.

[34] Stan, https://mc-stan.org/

[35] Prophet trend changepoints, https://facebook.github.io/prophet/docs/trend_changepoints.html

[36] OpenTSDB, https://github.com/OpenTSDB/opentsdb

[37] HBase, https://hbase.apache.org/

[38] OpenTSDB overview, http://opentsdb.net/overview.html

[39] Grafana, https://grafana.com/

[40] Service Name and Transport Protocol Port Number Registry, https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[41] Krzywinski, Martin & Altman, Naomi. "Visualizing samples with box plots".

Nature Methods, 2014/01/30/online. VL 11 SP 119. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. https://doi.org/10.1038/nmeth.2813 10.1038/nmeth.2813

[42] Kolmorogov-Smirnov Test, https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test

[43] https://blog.netlab.360.com/quick-summary-port-8291-scan-en/

[44] https://blog.radware.com/security/2018/03/mikrotik-routeros-based-botnet/