PhD-FSTC-2015-22
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 04/05/2015 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Lautaro DOLBERG
Born on 15 March 1985 in Buenos Aires (Argentina)

NETWORK MANAGEMENT WITH DATA ANALYTICS

Dissertation defense committee

Dr Thomas Engel dissertation supervisor
Professor, Université du Luxembourg - SnT

Dr Raouf Boutaba
Univesity of Waterloo, Canada

Dr Ulrich Sorger, Chairman
Professor, Université du Luxembourg

Dr Jérôme François
Chercheur, Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA) - INRIA

Dr Radu State, Vice Chairman
Dr Habil, Université du Luxembourg - SnT

Acknowledgements

This doctoral thesis was created at the SECAN-LAB of the Interdisciplinary Centre for Security, Reliability and Trust (SnT) and the University of Luxembourg. It would have been impossible to achieve the goals of this thesis without the encouragement of a very patient and supportive environment.

Firstly, I should like to thank my supervisor Prof. Dr. Thomas Engel for giving me the opportunity to participate in his research group since December 2012. Also, I am sincerely thankful to Dr Jerome Francois for his wise guidance and orientation over almost four years. I also owe sincere thanks to Dr. habil Radu State for his vision and coordination. I want to thank Prof.Dr. Raouf Boutaba for his support and for being a member of my committee and jury. I also owe sincere thanks to Prof. Dr. Ulrich Sorger for being part of my jury.

Furthermore, I would like to thank all the members of the Vehicular Lab who provided insight and expertise that greatly assisted my research. In particular I would like to thank Dr. Raphael Frank. I would like to thank the Luxembourg National Research Fund (FNR) for providing financial support through LORE 2010 Move project.

I would like to warmly thank Dominic Dunlop for comments that greatly improved this manuscript and my publications carried out in the last years. Special thanks go to Lara Codeca for sharing the office with me for the last years but also for her support and understanding.

Additionally, I owe gratitude to my family and my girlfriend who supported me at all times

Abstract

Networks are an essential part of the everyday life of million of individuals and organizations around the planet. Management and security are critical to ensure the correct functioning of networks and applications using them. Additionally, the requirements in terms of performance but also security have greatly increased in recent years. Simultaneously, the high adoption rate of distributed computing paradigms and cloud-based applications are dependent on the performance of data centre and computing facilities. This scenario shows the need for strong data analytic techniques capable of contextualizing multiple events in order to identify security events efficiently. The emergence of Software Defined Networks (SDN) has suggested promising improvements in network performance improvement. In this context, data analytic methods for network management require support in order to be be efficient.

This thesis addresses the topic of data analytics for network security, including a multidimensional monitoring approach to detecting anomalies of distributed applications in loosely restricted access networks, as for example: TCP/IP, Domain Name System and crowd-sourced location-based applications for vehicular routing. As a complement, we propose an approach to leverage the performance of SDN-based data centre environments by raising application-level awareness to the network management level.

In this thesis, we validate our multidimensional aggregated model for representing network events and its associated metrics involved in the detection of anomalies and malicious activity. For the validation experiments we use real-world data and proposed an approach for identifying and counteracting network threats in particular circumstances. Last but not least, we have implemented a framework to raise application-awareness as a proof-of-concept for improving the performance at data centre environments.

Contents

Chapter 1

Introduction

1.1 Context

In recent years Internet access has become a global phenomenon, and the use of Internet-based services has grown exceptionally. Access to this wide spectrum of Internet-based services is leveraged by the increasing use of personal communication devices with network access capabilities. According to a study carried out by Cisco Inc [1], in the past five years we have witnessed more than fivefold increase of global Internet traffic. In 2013, mobile data traffic accounted for 30 times the volume of the global Internet traffic in 2000, which reached two and half exabytes per month at the end of 2014 [2], as shown in Figure 1.1. The high adoption rate of personal devices for using Internet-based services has impacted on networks in term of information volume for routing and storage but also on security. The volume, frequency and complexity of attacks have increased, as suggested by a security report conducted by Arbor Networks in 2014 [3], as for example, the volume of Distributed Denial of Service (DDoS) attacks in the 2013 increased by 50%. This is explained by the fact that a bigger surface of attack and a larger number of potential victims are now available.

This evolution represents a challenge of paramount importance for information security. Security mechanisms are not only required for establishing confidential and reliable channels of communication between users and applications encompassing all intermediary points, but also because an exceptional increase in data volumes in terms both of transfer and of storage. The magnitude of this increase has affected negatively web hosting servers, name servers, and data centres by multiplying the potential risks of attacks against infrastructure and magnifying the havoc caused by security threats.

According to a survey published by Kaspersky Security [4], in 2014 more than 90% of the companies encountered information security issues, which, their top concerns in security were: protecting highly sensitive data, preventing IT security breaches and protecting business-critical infrastructure (*e.g.* preventing DDoS attacks). Data loss due attacks and security issues is carried out using a variety of strategies, as illustrated in Figure 1.2. Not all the attacks are carried out aiming at large organizations, the motivations from malicious users can be financially driven, therefore a vast portion of attacks are targeted to individuals employing Phishing or Malware strategies.

Phishing and Spam activities have also become of relevant importance. According to a recent report conducted by Anti-Phishing Working Group [5], payment services continue to be the most targeted industry sector. Also, it was pointed out that, not less than 30% of Internet connected

computers were infected with Phishing or Spam related malware. This type of security threat follows a global distribution suggesting The United States, China and Russia as the top countries hosting phishing sites. Only in the first quarter of 2014 more than 100 thousand new phishing domains were detected [5]. Kaspersky Labs has recently reported that malware is becoming multifunctional, not only sending spam, but also performing more complex tasks as participating in DDoS attacks [6].

This scenario sets out the need for monitoring techniques to leverage the detection of the diverse and complex techniques carried out by the attackers. Monitoring a the traffic that passes through an Internet Service Provider (ISP) can be a costly process, therefore approaches for aggregated analysis offer a cost-effective solution. Due to the vast diversity of attacks against services and applications, using individual approaches for detection could not to be scalable enough. Therefore, a realistic approach is to employ methods for analysis as general as possible, capable of handling unstructured data from multiple sources, which could differ significantly from the approaches traditionally applied.

Significant changes in the data analytic methods for information security are discussed in a forecast published recently by Gartner [7]. This changes are explained by the emergence of centralized storage for multiple sources of events, followed by the advances in techniques for the analysis of aggregated data, which were not practicable before due performance limitations of computing centres. Hence, data analytic approaches are needed to cope with the large and heterogeneous data sets, and also to keep pace with the increases on volume that security applications are facing [8]. From the perspective of Intrusion Detection System, data analytic approaches can be applied with diverse purposes (*.i.e* forensics, detection, monitoring) to reduce the time for correlating and contextualizing events. Recent advances in computing infrastructure have made possible to implement distributed computing patterns

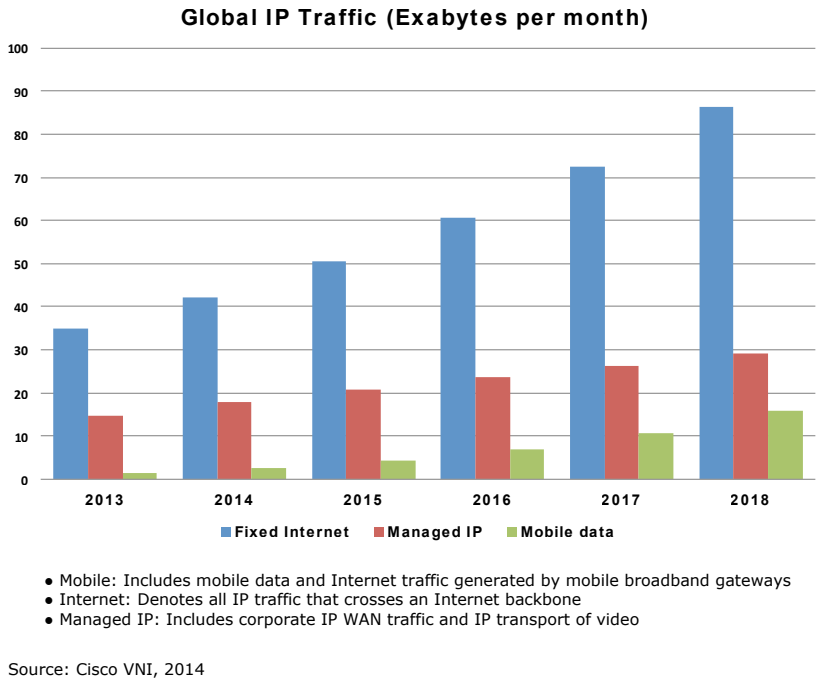


Figure 1.1: Global IP traffic growth forecast for the years 2013-2018 by Cisco Inc [1]

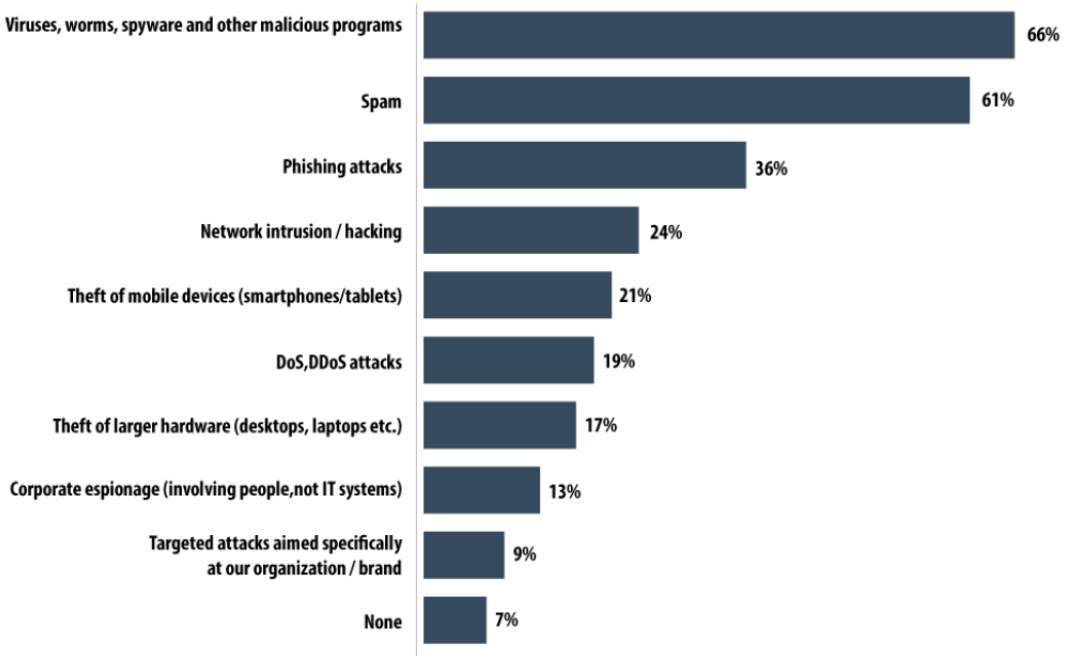


Figure 1.2: External threats reported during the year 2014 [4]

and storage methods for empowering data analytic approaches [7].

Distributed computing has evolved from local and small environments to larger, and even global installations with a significantly larger user base. In 2013 global data centre traffic accounted 3.1 zettabytes (ZB) per year (255 exabytes per month) [9]. For the period 2013-2018 is expected a 23% compound annual growth rate (CAGR) increase on data centre global traffic, as illustrated in Figure 1.3. At the same time, the emergent use of Software Defined Networks (SDN) may lead to a significant improvement on the performance of networks at dedicated computing facilities [10, 11]. This can be in part explained by the increase in the adoption rate of virtualization in data centres, as suggested recently in [12]. From the user perspective, information availability and storage media have become a of less concern in comparison to fast and ubiquitous network connectivity. One of the main reasons behind this phenomenon is the emergence of the Cloud computing and its applications, which allows computing, accessing and storing information in a globally distributed fashion[1].

The emergence of distributed-based services for public and private storage and computing, referred currently as Cloud computing, sets out as top priority for enterprises the need of significant infrastructure changes [8]. This change has not only affected the requirements in terms of infrastructure, but also it evidences a change in the use of distributed computing towards centralized services [7]. Despite bandwidth exceptional increase in recent years [9], along with throughput and latency are among the major concerns of many organizations [8]. This can be explained by the fact that most of organizations' infrastructure was not designed to respond to the requirements of Cloud-based applications.

[1] As for example services such as Google Apps (https://www.google.com/work/apps/business/), Apple Cloud (http://www.apple.com) or Dropbox (http://www.dropbox.com)

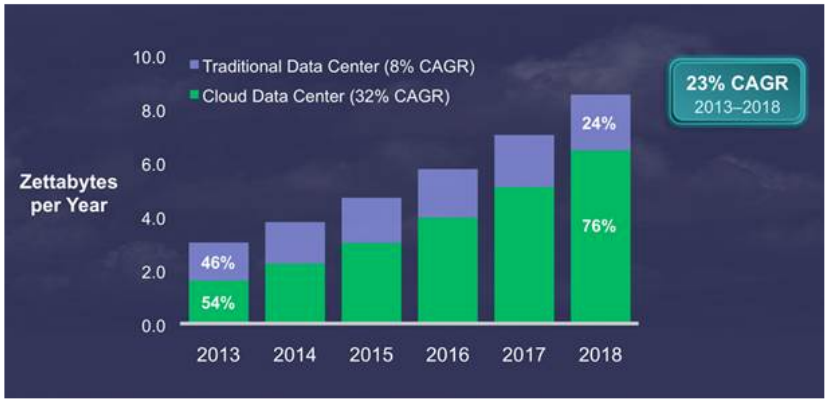


Figure 1.3: Data centre traffic growth forecast for the years 2013-2018 [9].

This sets out a vast field for performance improvements at specially data centres where Cloud-based applications are hosted. The use of SDN in data centres has recently showed promising advances in terms of performance and security, as recently pointed out by authors in [13].

The adoption of SDN allows obtaining valuable traffic volume metrics that can be studied to achieve a greater performance in data centres. Approaches considering the specific needs of applications to allocate and guarantee resources are a suitable solution. These types of approaches combined to the virtualization techniques and technological advances may lead to significant improvements in performance in terms of bandwidth, throughput and latency.

1.2 Problem Description and Contributions

This thesis addresses the problems and challenges that arise for monitoring distributed applications over the Internet and managing network to support these applications or services. To fulfill our goal, we focus on aggregation techniques via the usage of data analytics that will help in reducing the volume of the data, making it easier to perform observations and draw preliminary conclusions about data patterns corresponding to network anomalies, which could lead to the detection of network attacks or malicious users.

Additionally, we address also the problems for implementing efficiently in a distributed fashion (in the Cloud) data analytic methodologies for management and security.

Network Security

In this thesis we study address the inherent relationships of hierarchical data belonging to applications distributed over large-scale networks or the Internet from the perspective of network security in the following areas:

- In the context of network monitoring, how information from multiple heterogeneous sources can be extracted, stored and, further on correlated to analyze significant events leading to the

detection of anomalies?

As pointed out by many authors [14–18], aggregation approaches have been applied for diverse purposes in network security. However, some approaches are specific to a certain type of network protocols or may encounter severe limitations in terms of performance while considering multiple sources of information. In addition, another limitation in the mentioned approaches is the need of human interaction for defining the granularity of aggregation.

We introduce an approach for handling heterogeneous-formed data, with diverse sources. We propose a methodology for temporal and spatial aggregation for reducing the scale of data, with out loosing the relevancy of information. A first aspect of our approach is considering multiple features or dimensions by design. A second attribute of our approach is that the granularity is given by the data aggregation process, where only relevant events are held. Since smaller events are aggregated into larger ones, the information is not lost but collapsed into more relevant entities. Our approach considers every feature independently without giving prominence to any specific one. In addition, we provided an implementation of our methodology, which has been validated using several real world scenarios.

- How the dynamics of multidimensional data in IP networks can be studied using aggregated traffic patterns for the detection of anomalies?

  Over recent years network monitoring became a fundamental part in network security, at ISP levels. In the scenario of a large network, the volume of data available for analysis may also become big, thus, this have a negative impact on the time for post-processing time and further analysis. An efficient method for reducing the scale of the data is to aggregate network traffic, as for example with the use NetFlow protocol [19]. Traffic analysis on aggregated traffic has proven to be effective for detecting anomalies using aggregated flows, as suggested by authors in [14,16]. However, the attacks observed recently have become more diverse, frequent and complex. Despite stronger methods for scanning, monitoring and accounting have been developed, detection of network attacks is still of a major concern [4, 20–23]. In particular, large network attacks such DDoS and Botnets are still a major threat to Internet operators and the general public.

  Our approach consists in studying the quantitative changes over time of the network activity present in NefFlow records. Our methodology is leveraged using multidimensional tree-based aggregation, that enables us, to apply metrics considering multiple features simultaneously (*e.g.* source, destination, port number, etc.). To validate our approach, we conducted experiments for detecting TCP Flood attacks present in network traffic offered by one major ISP in Luxembourg [187].

  A major improve to this approach is to analyze not only the IP traffic by also consider another widely used protocols as sources such as Domain Name Service (DNS), HTTP Secure (HTTPS) or position-based applications. This points out to a subsequent scenario that we address in the next research question.

- How to apply metrics for the analysis of the match between DNS names and IP subnetworks to find out potential malicious websites?

  The correct functioning of many Internet distributed applications is crucial for wide variety of large-scale services. Basic Internet infrastructure has become a target for hackers, as suggested

in [20]. In particular, applications such DNS or HTTPS are essential for an immense number of people using end-user applications.Public concert about security issues has also experienced changes in recent years as exposed in [20], which also points out the evolution of the attackers goals. DNS traffic monitoring can help to identify two major threats, FastFlux and phishing, and can also provide insights into other malicious activities [24–26]. A common characteristic of malicious websites is their rapid changes of IP addresses and DNS names, due the reactive actions of blacklists and malware reporting websites.

In this scenario, we propose our approach for identifying anomalies and detection of malicious web sites using a metric based on the association time between an IP subnetworks and a DNS names. Our approach was carried out by studying the persistence over time of DNS records, which were included into the repertory of Multidimensional Aggregation Monitoring (MAM) for multidimensional aggregation. Our detection method was leveraged by aggregation to reduce the scale of the data set and also to consider multiple sources of information such as a Passive DNS database [27] and blacklists [28]. To validate our approach we conducted macroscopic (over the course of one year) and microscopic (during one week) experiments. Our results suggest that malicious domains can be detected efficiently using our approach [188].

Network monitoring approaches still encounter limitations when facing mobile technologies. Communication channels have become more heterogeneous with the advent of new network technologies such as Vehicular Ad Hoc Networks (VANETs), and the Internet of Things along with rapid advances in cellular network technologies. Car communications in VANETs are expected to experience rapid growth, for the year 2020, 220 million cars are expected to be manufactured with connectivity devices, according to a recent forecast [29]. The emergent technologies for communication and computing platforms for Vehicular Ad Hoc Network (VANET) are nowadays very diverse. Hence, we believe that our contributions to this field should be from a general perspective, thus the technology have not yet been standardized. This points to a following research problem addressed in this thesis.

- In the context of security for crowd-sourced applications. How the dynamics of positional information in an unrestricted VANET could lead to the detection of traffic anomalies?

  Position-based services are essential for an immense number of vehicles using applications for routing and safety. According to a report published in 2014 by the U.S. Department of Transportation, National Highway Traffic Safety Administration, safety applications would on an annual basis potentially prevent 250 thousand crashes on average [30]. Assuming a crowd-sourced application, there is no strong restriction on users, everybody is basically identified with pseudonyms. Therefore, an attacker or a malfunctioning device can produce erroneous data and inject it. This topic is of major concern among the researchers [31–36]. Our approach consisted in leverageing detection with multidimensional aggregated analysis, which consists in studying the plausibility of the reported positions of moving cars. In addition, we developed a mechanism for malicious information removal from a compromised data set reducing the information loss. To validate our approach we conducted simulations using a state of the art simulator [37].

Network Management New uses of technology have emerged recently, which centralize the computing and storage of information into cloud-based applications [38]. Simultaneously, performance

requirements of such applications have become critical due the increase in the number of users, in particular using mobile devices [2]. Therefore, network management has become of a paramount importance [7, 9]. This scenario sets out the need for approaches towards efficient network management. In such a context, the network plays a vital role to guarantee QoS. This scenario motivates the following research question:

- How to leverage network awareness in a SDN cloud-based environment to apply flexible applications-based policies for security requests, availability and performance?

Software Defined Networking clearly helps in defining and applying advanced flow based policy. However, an SDN controller is not able to take fully optimized decision without knowing about the context of the flows. Our approach takes a step to fill up this gap by proposing a framework that allows the SDN controller to learn about the flows context and take more optimized decision. We also demonstrate the feasibility of our approach by implementing the framework and making it work with existing SDN controller. Our evaluation results also reveal that our proposed framework has very little footprint, while enabling the network to take much optimized decision to meet QoS requirements of applications.

1.3 Structure of the thesis

The structure of this thesis follows the order of the research questions listed below. The contributions of this thesis are set out in chapters 5 and 6. In chapter 5 explain in details the contribution to the methods of data analytics proposed. In chapter 6 our approach for performing data analytics using the methods as enablers is detailed. The first part, about State of the art is organized in three chapters. The first two correspond to the State of the art of data analytics for network security, and the last chapter is for data analytics for network management, in particular on the field of data centre networks.

The remainder of this thesis is organized as follows:

- Part II - State of the Art: In chapter 2 we first cover the data structures and methodologies relevant to the field of network monitoring. The second part of the chapter describes the principles of network security and sets out as the state of the art. Additionally, we introduce network management concerns for big data applications in chapter 3.

- Part III - Contributions: In chapter 5 we present two data analytic methods for improving network security and management. The first one, is a multidimensional data structure for representing large scale network traffic in order to reduce its volume before further monitoring techniques [187]. The second enabler is employed to implement flexible network policies for security requests, availability and performance in Cloud infrastructure. In chapter 6 we introduce several methodologies for the analysis of large-scale distributed applications on unrestricted access networks. These include NetFlow analysis, DNS and positioning services on vehicular networks.

- Part IV - Evaluation: In chapter 7 we present our evaluation of the proposed techniques for large scale monitoring, malicious activity detection and recovery for NetFlow, DNS and position-based crowd-sourced services [187–190]. In particular, results of experiments which validate our

approach on real world data. In chapter 8 we present our experimental evidence on performance and security policy enforcement in Software Defined Networks.

- Part V - Conclusion: In chapter 9 we conclude this thesis with a discussion of the most relevant contributions and highlights perspectives open by our contributions.

Part I

State of the Art

Chapter 2

Data Analytics: Concepts & Methods

Ways to store, retrieve and analyze information have always been among the most frequently addressed problems in computer science. Since the early days of computer science many outstanding contributions have been made into this field, including new algorithms and data structures.

Hierarchical data is present in most scientific disciplines. For instance, any taxonomic classification system (biology, astronomy, computer science) uses a hierarchical underlying model. In particular in computer science and computer networks, most address spaces (*e,g* IPv4, DNS, MAC, among others) for locating computers in a network follow a hierarchical model. However, these address spaces are so vast that reducing their scale is necessary to facilitate analysis and representation. An example is using subnets to group and study the packet traffic in a network. It is important to note that reducing the scale in order to observe data allows a more compact representation, but could result in some small-scale phenomena, not being represented due to the magnitude of the scaling. For example, a representation of network traffic by network class can summarize large groups of hosts, but renders individual traffic patterns unobservable.

In the following sections we will address several contributions to the field of data structures and methods employed in the analytics of network and applications.

2.1 Data Structures

The data structures covered in this section are those that serve to efficiently store aggregated data, in order to support or enable network monitoring tools. For example, a common use case in monitoring is to efficiently retrieve the volume of data for a certain portion of a network or user group, in order to determine the bandwidth consumed per host at a given time. In this case, efficiently maintaining an aggregated data structure with its intermediate values (elements of the hierarchy) is critical. Additionally, scalability becomes a crucial factor when the volume of data grows almost without bound as it may, for example, with network users, bandwidth or simultaneous requests to an application server. For aggregation over a single dimension, a binary tree structure is suitable [39–41]; however it is also possible to aggregate single-dimensional data into a k-ary tree [42]. For more than one dimension, this chapter covers the data structures employed in state-of-the-art approaches for network monitoring and data analytics [14, 16, 18]. These structures are: Fenwick trees, Quadtrees, Octtrees and K-D trees.

The authors of [39] describe the Fenwick, or Binary Indexed, tree, a data structure designed to

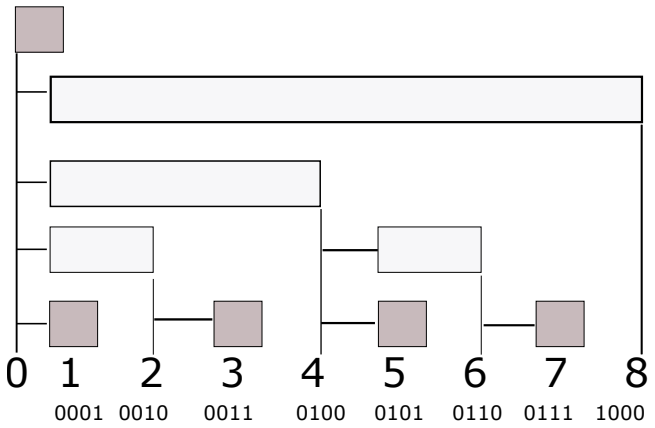


Figure 2.1: Fenwick tree representation of a 4 bit coded frequency [39]

efficiently store aggregated cumulative frequencies. The basic principle behind Fenwick trees is: As an integer can be represented as a decomposed sum of smaller integers, cumulative frequency can also be represented as a sum of its subfrequencies. Each node is indexed with a positive integer; the node at the index i stores the sums from i to $i - 2^k + 1$ where k is the greatest integer that satisfies $i \; mod \; 2^k = 0$. For example, a Fenwick tree using a 4-bit index, is illustrated in Figure 2.1. Node 7, binary 0111, carries the sums from 7 to (6+1), and is a terminal node. Node 6, binary 0110 (so $k = 2$), stores the sums from 6 to (4+1). Therefore, node 8 accumulates the sums of all its predecessors.

One of the main advantages of Fenwick trees is their temporal and spatial complexity for retrieval. Retrieval of a node can be done in logarithmic or constant time, where the worst case complexity is $\mathcal{O}(\log(MaxVal))$, with $MaxVal$ the maximum integer in the tree [39]. The memory requirements of a Binary Indexed tree are linear in the number of nodes to be stored, also allows the storage of n-dimensional arrays.

While Binary Indexed trees were created for data compression, [43] shows how they may be used in network monitoring tools. Here, the authors propose using a Fenwick tree to store the cumulative percentage of activity of a network (or a part of it) in terms of data consumption, using the prefix length in the network address as an index. For example, the tree would partition the 192.168.1.0/24 subnet into smaller subnets - /26, /27 and so on.

Quadtrees [41] were introduced in the late 70's to model bi-dimensional data. A Quadtree is defined as a tree with four children per node, each child representing a recursive subdivision of the data space. Each node has a defined capacity. As illustrated in Figure 2.2, the base case for a Quadtree is to represent an undivided 2D plane, then four nodes are created at each step by subdiving the 2D plane into four equal sub-parts. This recursive subdivisions continues until the capacity of the node is reached.

While, traditionally Quadtrees are used for spatial representation of 2D objects in the X,Y plane, their use is not limited this field. However, the authors of [44] propose a TCP packet classification approach based on a priority-based quad-tree (PQT); this will be described in detail in Section 5, on page 55. [45] shows how Quadtrees may be use to index Peer-to-Peer (P2P) networks.

Just as Quadtrees are used to represent bi-dimensional spaces, Octtrees [46] are used for three dimensional spaces. This approach can be generalized to an M-dimensional space, as [47] shows.

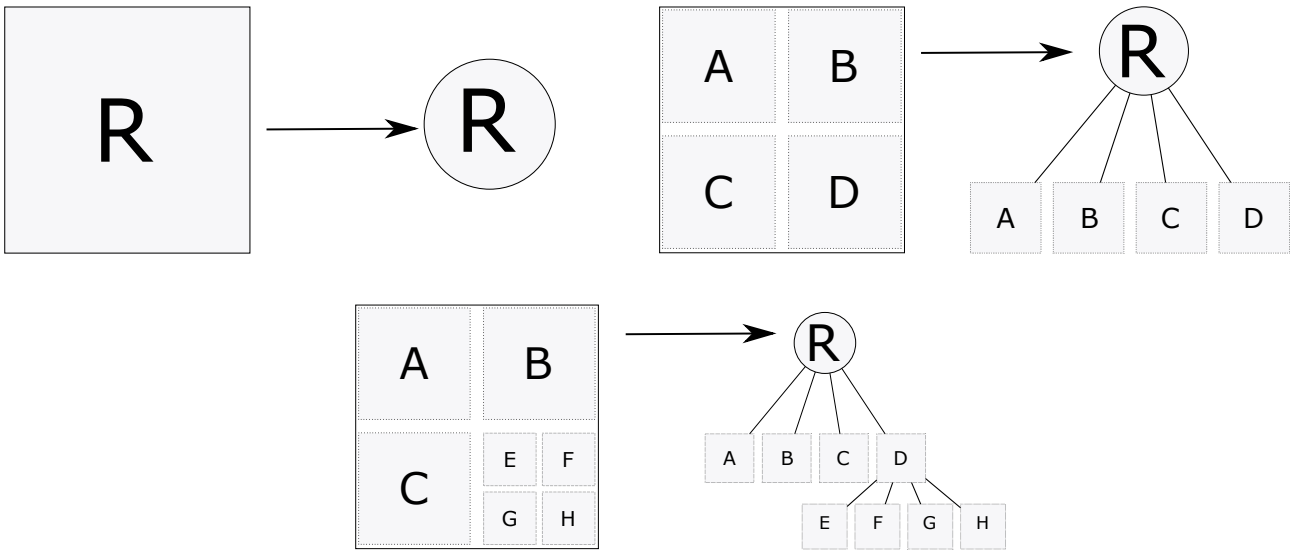


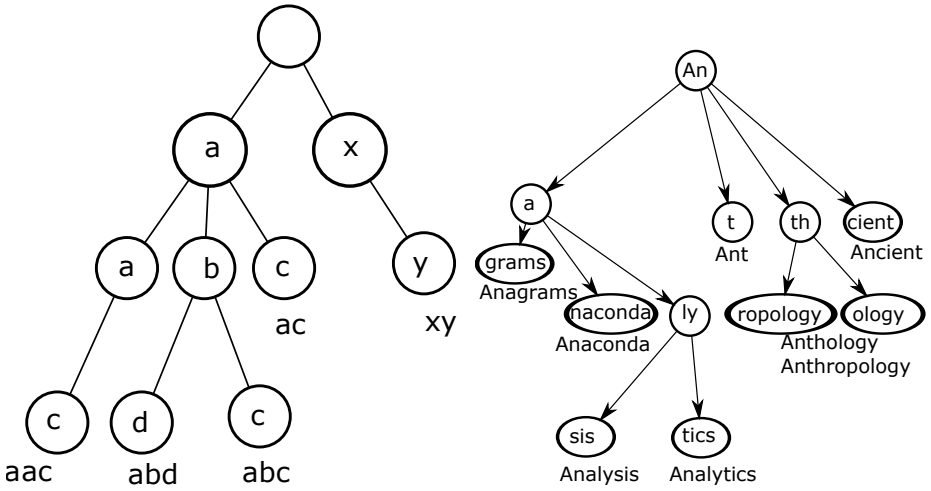
Figure 2.2: Quad tree representation of a pixel matrix [41]

Patricia Tries [40] played a part in the implementation of critical networking tasks, for example, in the BSD kernel routing table [48]. A trie is the simplest version of a prefix tree, as illustrated in Figure 2.3 (a), where only the leaves store the values (full words); it is used for string storage. It offers faster retrieval than a classic Binary Search tree. A string look-up operation on a trie has a complexity of $\mathcal{O}(m)$, where m is the length of the string, while on a Binary Search tree, it has a complexity of $\mathcal{O}(m \log n)$ with n the number of nodes in the Binary Search tree. Patricia tries or radix trees are a compact version of the trie [49] data structure. Nodes with a single child are merged together, resulting in a tree structure where every internal node has at least two child nodes and a maximum of r, where r is a positive integer and a power of Two (the size of the prefix space), as illustrated in Figure 2.3 (b). One of the differences that makes Patricia trees more efficient than regular tries is their support for prefix search (by comparing chunks instead of single characters).

Limitations One of the most significant limitations of the above-mentioned data structures is their lack of flexibility for multidimensional problems. Despite their use in network monitoring, Fenwick trees are limited to only one dimension, meaning that each node can aggregate only one dimension at a time. As an example, a bi-dimensional tree representing source and destination of IPv4 address cannot be used to aggregate both dimensions simultaneously.

Quadtrees have been proven to be efficient for spatial representation; however, their limitation concerns dynamic partitioning, because once a node size has been defined, it must remain as defined throughout the life cycle of the data structure instance. Another key limitation is the lack of granularity for aggregation: once a structure has been created with a previously defined aggregation parameter, it must remain as it was created. As explored in [50], limitation particularly affects tools using prefix-or-suffix based trees such as Aguri.

The generic issue of storing multidimensional data was investigated in the past. While data structures cited in [41, 47] are plausible model tools for K-Dimensional spaces, they are recommended



(a) Trie Example (b) Patricia Trie Example

Figure 2.3: Examples of string storage using Tries

only for use on regular partitioned spaces, with the previously mentioned data structures implying a requirement for a pre-established order among dimensions. However, in our context, the dimensional space division is not known in advance and instead is determined on the fly when the tree is created.

2.2 Algorithms & Methods

2.2.1 Data Mining

Knowledge Discovery in Databases (KDD) may be defined as "the subject of the emerging field of knowledge discovery in databases" [51]; it consists in extracting information not known in advance from a dataset. The application of KDD is crucial in large volume data analysis where manual techniques are not feasible. For example, given a data set of users and their call records for a whole country, KDD might be used to extract trends concerning users' calling habits. Knowledge Discovery, as defined by [51] is a multi-step process, entailing:

- Selection: The initial phase of the process consists in selecting a subset of the original data. For example appling a filter to select only data containing a specific attribute.
- Pre-processing: During this step, processing takes place if any noise or missing fields are present in the selected data.
- Transformation: After pre-processing, data is projected or reduced to match the data representation required for data mining.
- Data mining: During this step the search for patterns takes place.
- Evaluation: In this step the extracted patterns are interpreted and also may be visualized.

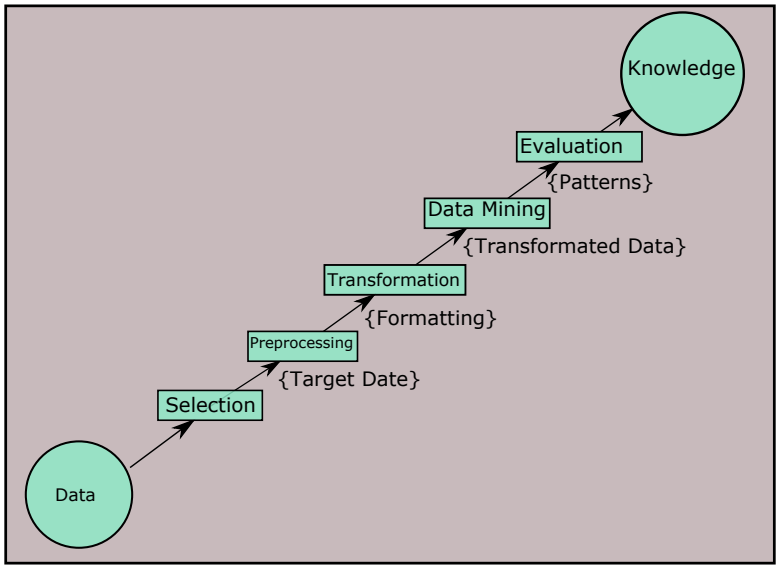


Figure 2.4: An overview of the original KDD process

This process is illustrated in Figure 2.4. Data mining stage is responsible for identifying patterns present in the original data. In recent years, some modifications to the original KDD have been proposed, including a business-oriented variant [52] and an automated version of the process [53].

Data mining methods and algorithms are intended to predict and describe data. Prediction employs variables to intuit new or unknown values, while description relies on human interpretation of patterns to explain data. However, the boundary between these two sub-disciplines is sometimes loose [54].

2.2.2 Machine Learning

Among the most powerful methods and algorithms used for data mining are learning algorithms. These are used as a tool to generalize a function over a domain for which there is no pre-existing data. For instance, given a series of values for a stock market index, a learning algorithm can use the existing data to estimate future values. As set out in Section 2.2.1 on page 14, there is usually a strong link between machine learning, statistics and to data mining. Some of the most prominent algorithms are illustrated in Figure 2.5.

In order to estimate the values of one or multiple variables, a prediction model is needed. A model is a representation of pre-existing knowledge used to generate predictions or estimates. In essence, the model is fed through a learning process using pre-existing data to establish a decision criterion not fixed in advance. Some algorithms use a supervised learning model, while others require no supervision during learning. The supervision process gives feedback to the algorithm by corresponding to a given input. One advantage of unsupervised learning is that it can be fully automated [54, 55]. A third type of learning is reinforcement learning, where an algorithm must learn unsupervised from the environment or the real world. Such learning algorithms are used in intrusion detection systems, autonomic behaviour in robots or automatic game playing against human opponents [56].

Problems with Machine Learning

- Classification: A classifier is a function that partitions the input space into two (or sometimes more) predefined classes. For example, a spam filter indicates whether a message should be labelled as unwanted or not, based on previous messages of both classes. There is a large family of classifiers, each having variable performance according to the data and the particular problem. For example, as illustrated in Figure 2.5(a), a linear classifier can be used to predict whether a new point belongs to "black" or "white".

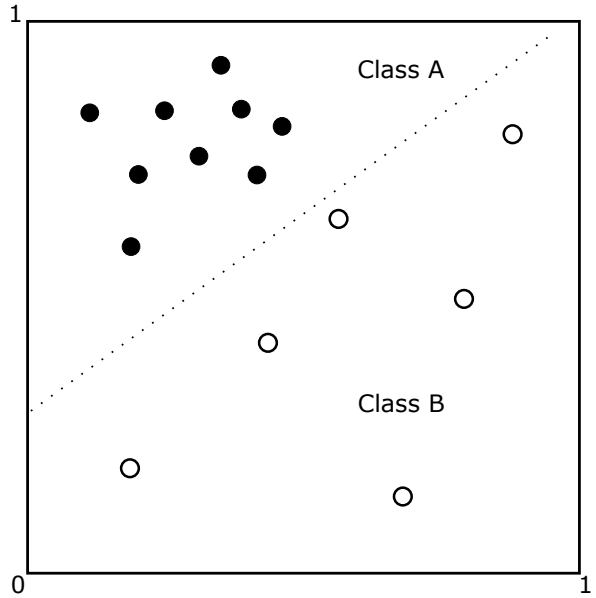
- Regression: In contrast to a classifier, where the output is binary (in the case where there are only two classes), a regression estimates the value of a variable with a real range. A common for regression methods is to provide support for extrapolation or interpolation. Regression methods are also used for estimating probabilities. Hence, in practice there is no great difference between the models used for regression and classification. For example, a regression method can be used to calculate the probability of an email being spam, while a classifier can be used to label an email as spam, when its probability of being unwanted exceeds a certain threshold. Regression methods support both linear and nonlinear models to fit series of values.

- Clustering: Clustering algorithms are used to establish a grouping criteria to describe the data from the dataset. Groups are formed according to a distance metric given by the user. For example, given a set of points in a space, a clustering algorithm can be used to find groups that minimize the Euclidean distance within each group, as illustrated in Figure 2.5(b). Algorithms of this category are diverse in the method used for establishing a grouping criteria: hierarchical, centroid, density unsupervised, are among the possibilities [57]. A common use for clustering algorithms is in recommendation systems and behaviour prediction, for online marketing, in particular, but also for security policy grouping [55].

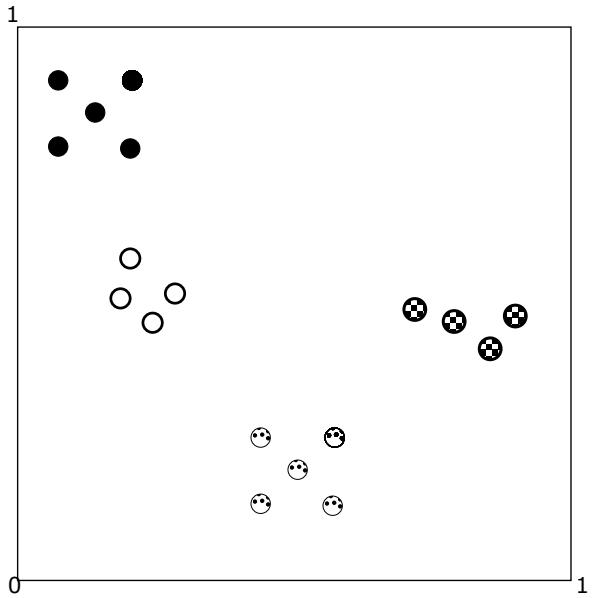
- Summarization: This method groups algorithms which processes a data set to extracting metrics in a compact format. For example, aggregating every field into standard deviation. More complex analysis made possible by summarization includes multiple variable visualization and interactive exploratory data reports.

- Dependency modelling: This technique finds dependencies among the variables of a model. Automatic tools focus on detecting three main kinds of dependencies: Structural (those dependencies observable in graphical plots), Quantitative (only observable by changing the scale or applying multiplying factors) and Multivariate interdependencies associated with probabilistic models. One of the most widely used techniques for testing dependency is the mutual information index of two random variables [55].

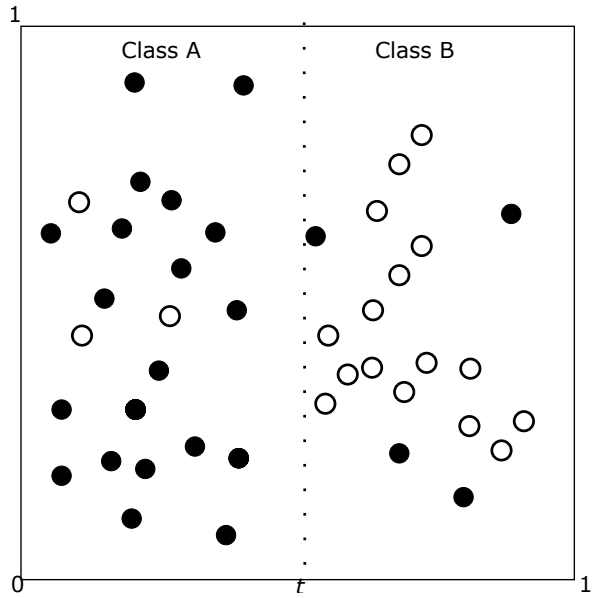
Learning Methods The choice of a method is usually determined by the nature of the dataset and the nature of the problem. For instance, given a set of data where there is a is large proportion is already labelled, as is the case for spam detection, a method using posteriori probability as a Bayesian classifier is a suitable choice. Well known methods for classification and regression mentioned in [55,58] are : bayesian methods,support vector machines, decision trees, neural networks, k-nearest neighbours, density-based clustering, threshold methods (as illustrated in Figure 2.5(c)) and non-linear methods.



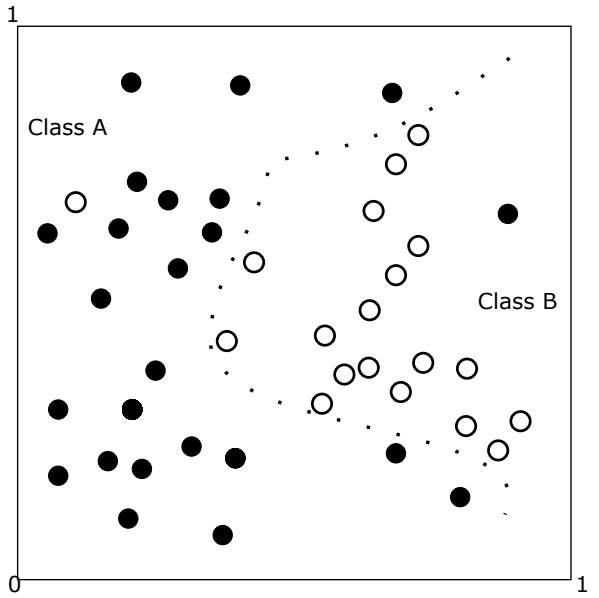
(a) Example of a linear classifier output. The points below the line belong to one category, and those above to the complementary category



(b) Example of 2D clustering algorithm output. Given a set of points in a space, the algorithm groups them in colour classes according their relative distance



(c) An example of a single threshold on the X axis to classify points on a 2D plane



(d) An example of non-linear classification for points on a 2D plane

Figure 2.5: Visual representation of output of different Machine Learning algorithms

In some cases, there is no linear function able to split a set of data in two without a great loss of specificity. This situation is illustrated in Figure 2.5(d). Several families of functions such as polynomials, splines and sigmoid functions can be adapted for classification and regression. Use of these functions are widely used in disciplines such as Neural Networks. [55].

		Predicted Class	
		Class A	Class B
Actual Class	Class A	True Positive	False Positive
	Class B	True Negative	False Negative

Table 2.1: Summary Table Example of the classification results for Machine Learning Algorithms of Figure 2.5

True Positive (TP) Black Dots in Class A (20)	False Positive (FP) White Points in Class B (3)
True Negative (TN) White Points in Class A (16)	False Negative (FN) Black Points in Class A (4)

Table 2.2: Confusion Table reporting the performance of the classification algorithm in Figure 2.5c

Evaluation of Learning Methods Learning algorithms are evaluated based on their performance in estimating values. For example, a classification algorithm for labelling emails as "spam" or "solicited" can be evaluated according to how many emails it labels as spam, but also counting how many "solicited" emails are mislabelled as "spam". In this manner, the performance of classification and clustering algorithms can be measured by several methods using the confusion matrix [55]. As illustrated in Table 2.1, which summarizes successful and unsuccessful classification cases. Based on the success/failure ratio of classification instances summary, a similar to Table 2.2 can be generated [55].

The following metrics are used to evaluate classification and recommendation algorithms [54,55,57].

- True Positive Rate (TPR) or Sensitivity: Represents the proportion of given data that was correctly classified among all the elements of the predicted class (including those classified wrongly). Defined formally as: $\frac{TP}{TP+FN}$
- True Negative Rate (TNR) or Specificity: Represents the proportion of negatives identified correctly. Defined formally as: $\frac{TN}{FP+TN}$
- Precision or Positive Predictive Value: Represents the proportion of correctly classified data among all the elements of the actual class. Defined formally as: $\frac{TP}{TP+FP}$
- False Positive Rate (FPR): Represent the proportion of elements that raise a "false alarm", elements from the negative class incorrectly classified as positive. Formally defined as $\frac{FP}{FP+TN}$ or $1-TNR$
- Accuracy: Represents the proportion of elements that have been correctly guessed. However, it may lead to misleading conclusions if the poroprtion of data in unbalanced. Defined formally as: $\frac{TP+TN}{TotalPopulation}$

- F1 Score: Represents a metric associated to the arithmetic mean of sensitivity and precision. Defined formally as: $\frac{2 \times TP}{2 \times TP + FP + FN}$
- Receiver Operating Characteristic (ROC): ROC Charts show the performance of a binary classifier as a function of its detection threshold. The graphic is generated by plotting the TPR against the FPR

The metrics cited above are usually used together to evaluate the performance of machine learning algorithms. Additional validation can be done by splitting the dataset into training and evaluation. This methodology is known as "Folding". A common practice is to N Fold the data set as follows:

1. Partition the dataset into N random sets having the same number of elements.
2. Use $N - 1$ subsets for learning, and the remainder for evaluation.
3. Repeat alternating all the possible combinations.
4. Calculate average confusion matrix and the performance metrics.

2.3 Data Analytics Computing Paradigms

2.3.1 Big data

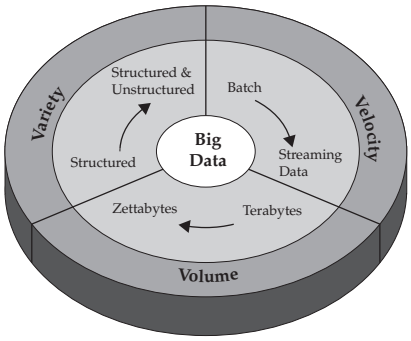


Figure 2.6: Characterization of Big Data by IBM [59]

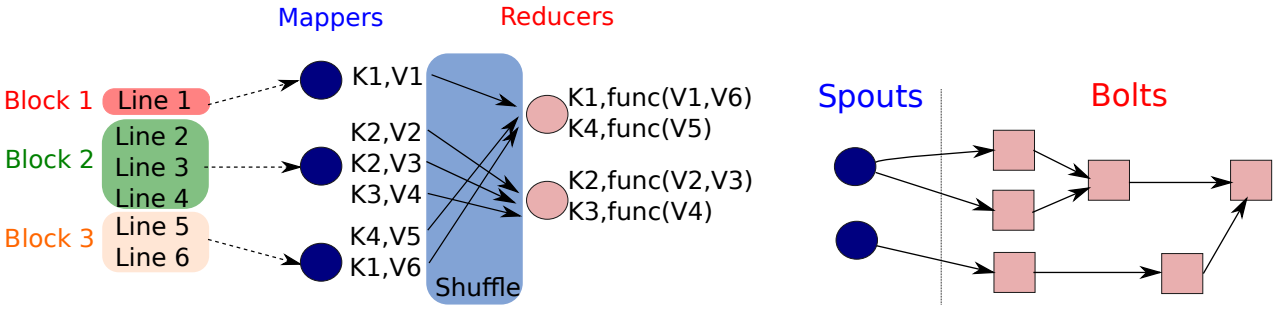
Big data refers to large datasets, that given their size, it is not feasible to perform computational tasks in a reasonable time. Among the scientific community there is not an established consensus about the definition of the term "big data" [60]. The definition can vary from sector to sector, with different considerations regarding the size of the datasets and the computing power required for processing. According to authors [61] big data can be defined as:

"Big data" refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyse. This definition is intentionally subjective and incorporates a moving definition of how big a dataset needs to be in order to be considered big data.

The term Big data can also be characterized according to the following description [59] (as shown in Figure 2.6):

Three characteristics define Big Data: volume, variety, and velocity

2.3.2 Distributed Computing Paradigms for Big Data



(a) Map-Reduce with K as key and V as value

(b) Storm

Figure 2.7: Big data computational model and the underlying network traffic as plain arrows

Distributed applications represent a large proportion of the usage of the cloud [38, 62, 63], which offers the distributed and on-line storage and elastic computing services. That, big data applications need to scale their computing and storage requirements on the fly. Building on recent improvements in virtual computing, data centres can offer a virtualized infrastructure in order to fit custom requirements. This flexibility has been a decisive enabler for the success of big data application. As an example, many applications of this type rely, directly or indirectly, on the MapReduce programming model [64] whose most popular implementation is Apache Hadoop[1]. The MapReduce paradigm just distributes computing tasks between mappers to produce intermediate results which are aggregated in a second stage by the reducers. This is illustrated in Figure 2.7(a) where the mappers send partial results (values) to specific reducers based on keys. Each reducer is in charge of applying a function to the whole set of values corresponding to a single key. For example, it can be an aggregating function like a summation.

Hadoop [65] is an open source MapReduce framework created by Apache Software Foundation[2], which implements distributed storage (using its own file system) and distributed computing. Hadoop's implementation of MapReduce is redundant, fault tolerant and scalable. It is offered as cloud hosting services by Amazon Elastic Cloud and Microsoft Azure, additionally it has been used by corporations as Facebook and Yahoo. The cycle of a MapReduce task in Hadoop is implemented as follows (as shown in Figure 2.7(a)): initially, all the data is stored in the Hadoop's file system; the data is split and the Map phase takes place; the output of the map phase is shuffled and transferred to the reducers; finally, the Reduce phase is executed and yields the output. An application may require a cycle of MapReduce tasks to calculate a given functionality.

To limit the volume of data transfer in the cluster the data is local to the executed code. However, large chunks of data are still transferred between the mappers and reducers in the shuffle phase

[1] hadoop.apache.org

[2] https://www.apache.org/

necessitating an efficient underlying network infrastructure. It is important to note that the shuffle phase can start as soon as the mappers begin producing (key, value) pairs; there is no need to delay shuffling it until mapping is complete. Since failures or bottlenecks can occur, Hadoop tasks are constantly monitored, and if one is not performing well (*e.g.* it does not progress as fast as others), it can be duplicated to another node (load balancing) at the cost of some additional data transfer.

Storm [66] is a Big data approach for low-latency data-analytics designed to operate in real time. Real-time applications such as Twitter[3] has been using Storm since 2011. It receives the input a stream, therefore it is able to implement interactive processes, unlike Hadoop which was originally designed for batch processing. Storm-based applications follow the topology of a directed acyclic graph (DAG), which represent the operations and the data path as shown in Figure 2.7(b). Storm comprises *spouts*, which read a data source to generate tuples to pass on to *bolts*, these *bolts* processing the tuples producing new tuples which are processed by further *bolts*.

The data transfer pattern during a Strom-based application is fairly constant, since the data is not local to the application (it arrives as a continuously as a stream). In contrast, the pattern of a Hadoop-based application is characteristic to the specific phases of a task, in particular the shuffle phase is the most traffic intensive.

There is also an approach to enable Storm applications and Hadoop applications to be hosted on a single cluster[4]. This approach may optimize cloud-storage allowing low-latency applications and batch applications to share the input data.

[3]https://www.twitter.com

[4]https://developer.yahoo.com/blogs/ydn/storm-hadoop-convergence-big-data-low-latency-processing-54503.html

Chapter 3

Data Analytics: Security of Distributed Applications

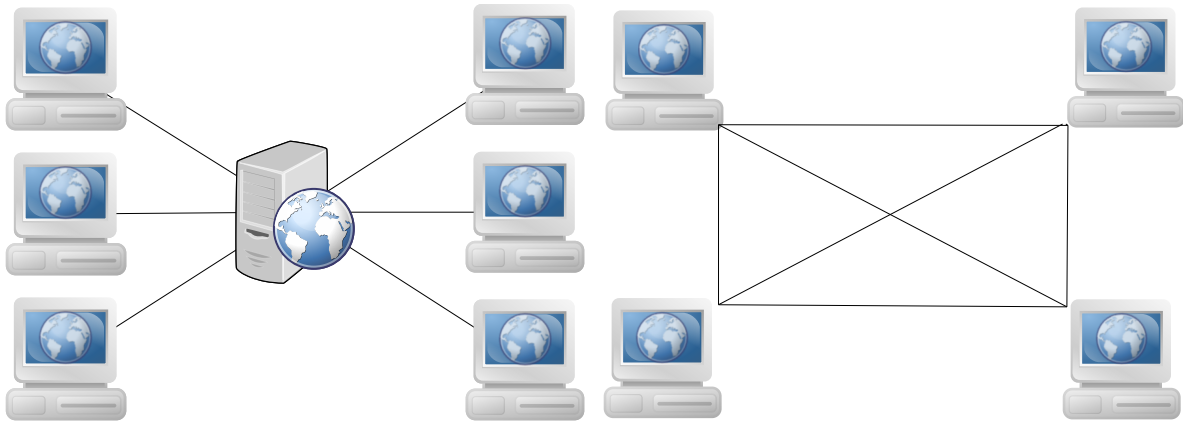
3.1 Overview

A distributed application is an ensemble of running software and/or network-interconnected hardware which is distributed logically (and physically) to implement a given functionality. A major class of distributed applications are those running on multiple machines interconnected through a network, *e.g.* in a data centre or over the Internet. For example, DNS [67,68] is a global distributed application which implements a name resolution service accessible from almost every device connected to the Internet.

The design patterns underlying distributed applications can differ substantially according to the functional and non-functional requirements of their users. For example, the design of an application hosted the world wide web (*e.g.* using Representational state transfer (REST)) is radically different from a peer-to-peer file sharing application operating over the Internet. The differences between design patterns include: the interconnection between components, the number of components and the separation of concerns among components.

One of the most significant advantages of using distributed applications is the avoidance of a single point of failure, providing grater robustness and fault tolerance than non-distributed systems. Additionally, in some cases, depending the Nature of the application, the data is produced in one location and consumed in other locations, for example, an application for the remote monitoring of sensors or distribution of images taken by security cameras. Only a distributed application approach can implement the requirements of those applications. In addition, when applications are distributed, other non-functional requirements might arise such as security concerns, and intellectual property considerations, among others.

Given the wide variety of distributed applications, the literature lacks an unified classification. However, the following classification groups the unrestricted access networks cases of distributed applications, which are relevant for this thesis. There is a conceptual difference between distributed applications and parallel applications. In the first, case each component has a private processor with its own memory, and communication among components is by message exchange. In the second case, communication uses shared resources such as shared memory or internal bus.



(a) Client-Server Architecture (b) Peer-to-Peer Architecture

Figure 3.1: The different types of architectural pattern for distributed applications

User access to distributed applications can also vary considerably, depending on the application. For example, DNS is an open access application that operates as a service and, despite the access control which can be enforced at network level (*i.e.* by IP addresses), the protocol specification does not provide any access control [67, 68]. However, other distributed applications or services such as telephony services over the Internet, for example Skype[1], require the user to be subscribed or previously registered. Additionally, security approaches using cryptography are usually employed for integrity and privacy but they are not efficient against network availability attacks such as DDoS attacks.

In this thesis, distributed applications are studied in this context. Examples include DNS, location services and crowd-sourced applications. Therefore, there is an crucial need to monitor the traffic of such networks in order to detect anomalies.

Architecture of Distributed Applications

Distributed applications on the Internet can be categorized using the following grouping based on their architectural design:

- Client-Server: In the Client-Server pattern, the information is held centrally on in one or more servers. Clients connect to the central server to request a service that provides data or performs an action. Important examples of this architecture are, the Simple Mail Transfer Protocol (SMTP), the Hypertext Transfer Protocol (HTTP) and Representational state transfer (REST)-based applications. For example, a typical HTTP application centralizes all the content in a web server which clients access through a browser, as illustrated in Figure a.

- Peer-to-Peer: In this scheme, the clients act also as servers, becoming active actors in content distribution and routing. Examples of this category are the BitTorrent protocol for file sharing and also some communication services, for example Skype, as illustrated in Figure 3.1(b).

[1] http://www.skype.com

- Hybrid patterns: Certain applications such as those based on Apache Hadoop, normally used for big data applications, are designed with a hybrid approach. Here, there is central controller that distributes the tasks to be executed by a cluster of slave computers. The data required for performing the computations is distributed along the network-interconnected slave devices, which can communicate with each other to exchange data chunks.

The security aspects of a distributed application is affect to all its layers and components. While, security can be applied specifically to each structural layer, such as network security for low-level communication components, a global approach to security requirements is of paramount importance.

Often demonstrated, it is possible to trigger an attack with devastating consequences by exploiting a localized vulnerability. Consequently a key role of monitoring is to analyze the traces and traffic of distributed applications with the aim of identifying anomalies. These anomalies in some cases are symptom of distributed attacks, or a less serious scenario, can point out malfunctioning components. Usage trends could also be identified by monitoring the traffic of distributed applications.

There is a need to observe the activity traces of distributed applications at a global level. Normally, this is done in by aggregating data, collecting traces from communications and studding them in a compressed form in which only aggregated metrics and the most relevant features are retained. However, due to the critical role of some applications, such as DNS, WWW or email, it is sometimes necessary to take a step further and look beyond aggregated data derived solely from the communication patterns. In the following section, Internet Monitoring and Application Specific Monitoring, the concepts behind monitoring will be explored.

To some applications availability is critical for economic reasons, as for instance, major Denial Of Service (DOS) attacks can affect the normal functioning of payment gateways or content distribution websites. According to an Arbor Networks report [69] there are more than 6,000 DDOS attacks each day.

3.2 Internet Monitoring

In recent years, the number Internet connections has experienced a dramatic increase. According to recent studies, the number of Internet connections has expanded in almost all countries% [70, 71] between year 2000 and 2015. In Europe, North America and Oceania, more than 60% of the population has access to a domestic Internet connection. This trend is illustrated in Figure 3.2. Although this trend is far from reaching its peak, the usage of the Internet has changed drastically. During the 90's the most significant application of Internet was the email; however, with the boom in social media applications and widespread adoption of mobile computing (smartphones, tablets and handheld devices) the variety of applications is now a rich and heterogeneous ecosystem [2, 9].

This phenomenon has multiplied the variety of security threats and exploitable vulnerabilities in this vast ecosystem of distributed applications. The complexity of attacks can also be correlated with the increase in the variety of Internet use and applications and with the growth of reachable hosts. A good starting point for monitoring traffic for anomalies and subsequent attacks is the ISP level. Depending on to the data retention and privacy regulations of each country, an ISP may inspect the traffic of its clients and store the resulting data for an extended period.

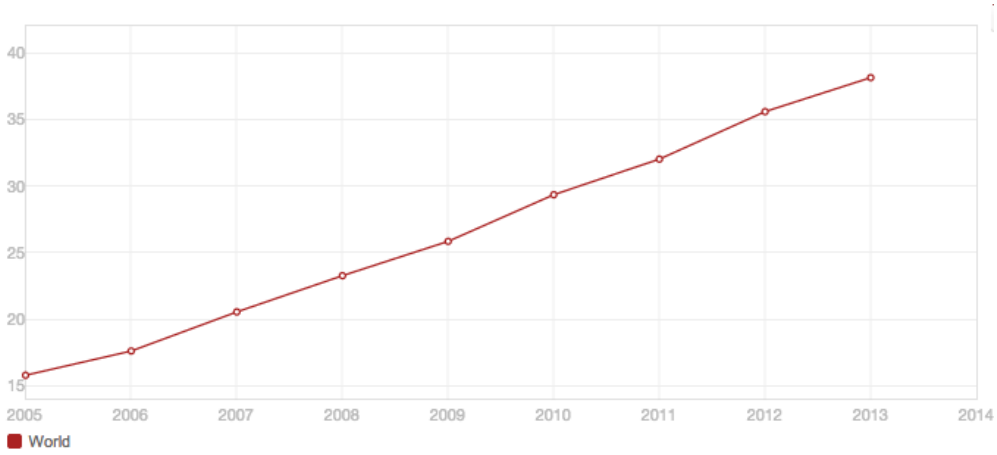


Figure 3.2: Internet users of the world, global average (per 100 people) according to International Telecommunication Union, World Telecommunication/ICT Development Report and database, and World Bank estimates. [72]

Flow-based monitoring

Monitoring techniques can be applied over several scopes. For instance, is possible to analyze a packet exchange between two hosts on the Internet or within the IP address space of an ISP. However, it is not practical to make such a fine-grained analysis for every connection because this approach does not scale. A solution which reduces the volume of communication packet exchange data is to summarize the whole communication as an IP Flow.

While the reader may refer to source document [19] and IETF IPFIX standard (RFC 5101) [73] for the full exposition of the term, the RFC 3954 [74] states:

A flow is defined as a unidirectional sequence of packets with some common properties that pass through a network device. These collected flows are exported to an external device, the NetFlow collector. Network flows are highly granular; for example, flow records include details such as IP addresses, packet and byte counts, timestamps, Type of Service (ToS), application ports, input and output interfaces

A flow is uniquely described by a combination of the seven fields instances. Given two flows, if any of the field values differs, there are treated as different flows.

- IP source address
- IP destination address
- Source port
- Destination port
- Layer 3 protocol type
- Class of Service

- Router or switch interface

Version 9 of the NetFlow protocol [19] add the following fields accounting:

- Flow timestamps to understand the life of a flow; timestamps are useful for calculating packets and bytes per second
- Next hop IP addresses including BGP routing Autonomous Systems (AS)
- Subnet mask for the source and destination addresses to calculate prefixes
- TCP flags to examine TCP handshakes

Normally, flows are collected centrally, as show in in Figure 3.3. Each NetFlow enabled device router can summarize the traffic it handles as NetFlow records and forward them to a NetFlow collector server. NetFlow collectors can be deployed at remote locations of the cost of the bandwidth resources required to transfer the records, which typically have a volume of 1-5% of the traffic that they describe [75].

The use of NetFlow to monitor large networks has some limitations and shortcomings. In most cases, NetFlow records are stored and analyzed in "off-line" mode, although there are some commercial solutions offering "on-line" or real-time NetFlow analysis [76]. The first issue is the storage requirements resulting from high traffic deployments; another is the time and infrastructure required for NetFlow processing. Normally, an ISP might have to handle 60,000 flows/second [76]. Some commercial solutions such as [77–79] offer data compression to reduce the storage requirement. Compression only reduces the scale of storage requirements: in the long run, flow records will deplete all available resources. Another solution is to discard part of the data by selecting only the most relevant parts of the information, this in itself is a non-trivial problem. Several authors [80, 81] propose storing only a sample of the total number of records. However, the effectiveness of this method depends on the sampling rate and strategy chosen by the user.

Using efficient data structures to optimize storage and access has also been explored. Giura et al. [82] introduce NetStore method, an efficient storage infrastructure for network flow data using a column-oriented storage that outperforms row-based techniques.

Security

Data analysis makes it feasible to detect and monitor network security threats by observing NetFlow records. The major issue is small quantity of information embedded in a single flow record, which makes correlation between multiple flows necessary. In recent work, authors [76, 78, 83–87] propose various techniques including anomaly detection leveraging machine learning techniques. Using the building blocks described in literature [78,88], the attacks and security threats that are most frequently detected by IP intrusion detection systems can be classified into the following taxonomy:

- Denial of Service (DOS): This attack does not exploit a vulnerability, but rather a technical limitation. It consists of compromising the availability of a given service or application. In

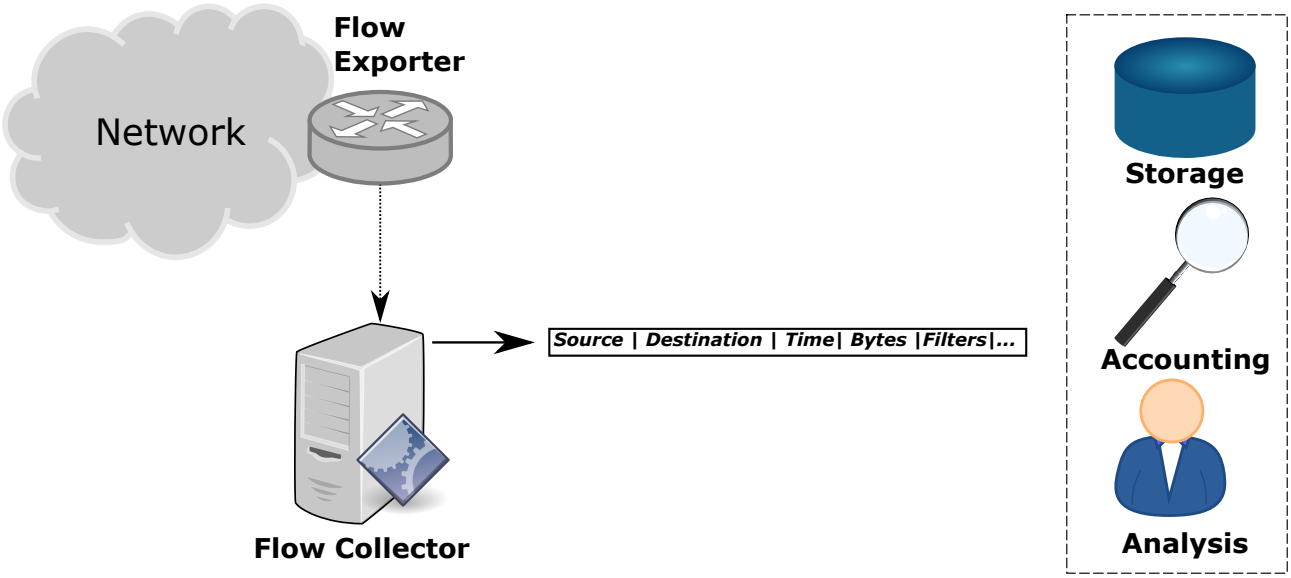


Figure 3.3: Example of NetFlow Collection

practice, this attack can be implemented by causing congestion in the network, or saturating the computing resources where a service or application is running. When carried out by a very large number of clients (far greater from the number of legitimate users expected) simultaneously accessing a given service, it is known a Distributed Denial of Service (DDOS). For example, the *Smurf* attack [21] uses a broadcast Internet Control Message Protocol (ICMP) with a spoofed address to trigger a very large number of ICMP responses back to a targeted victim host.

- Malware propagation (worms): This category consists of programs that have a mechanism for self replication over the Internet. In the early 90's a well-known technique was to infect the victims computer and then scan the address book to send emails containing a copy of the virus [89]. Worms are sometimes used to seed botnets, normally remaining with low network activity until fully activated remotely. On July 19, 2001, not less than 359,000 hosts with Internet connection were infected with the Code-Red in nearly half a day [90].

- Botnets: Botnets are collections of hosts infected with a malicious application that can be controlled remotely. This allows a single attacker to control and orchestrate thousands of hosts simultaneously. Normally, the host do not show any evidence of being compromised until the remote controller initiates an attack. Botnets are often responsible for DDoS attacks, spam distribution and espionage [91]. Botnet operations usually support criminal activities such as phishing, malware and child exploitation. One common technique to hide Botnets from detection tools is FastFlux [92, 93]; this particular attack is covered in Section 3.3 on page 34.

- Network Attacks: Network attacks [88] are not targeted to weaknesses in specific applications, such as a buffer overflow or a privilege escalation. Instead, they exploit flaws in communication protocols through techniques such as the Man in the middle. Some attackers will first "sniff" a large network, looking for potential victims with open ports and running vulnerable or outdated

software. Port scanning is considered to be attack if it is done with malicious intent; however, intentions are intuit in advance. TCP/UDP flooding is also regarded as a network attack; they can only be successful against software implementations of the TCP/UDP Stack. Generally, network attacks are a forewarning of a future application and user-targeted attack.

- Spam: Unsolicited bulk email affects a broad spectrum of Internet users. Initially, unsolicited messages were delivered only through email, but nowadays the appear in other sources such as social media, online gaming, blogs and instant messaging. From a security perspective, the mere distribution of advertising does not represent a threat by itself; however it may serve as a vector for other attacks such as phishing, XSS and virus distribution. In addition, it has been found that infected computers in botnets are responsible for Spam distribution [94].

- Application-Specific: There is broad range of application-specific attacks such as buffer overflow, SQL injection, SIP session forging, password cracking, Transport Layer Security (TLS) exploits and many more, which can compromise specific applications, that possibly leading to a leak of sensitive information or a privilege escalation. Compromised systems that are victims of such attacks are likely to eventually download or upload content, making themselves visible to a network intrusion detection system.

  - Web: XSS or Cross-Site Scripting is a technique for injecting a chunk of malicious HTML or JavaScript code into the content delivered website, exploiting a sanitation vulnerability. Whenever, whenever a compromised website is visited, it triggers a redirection to a malicious site. [95]
  - DNS: DNS Spoofing or DNS Cache poisoning consists in compromising the database of names used for IP resolution, and usually targets the resolver's cache. Victims are redirected to malicious websites or compromised servers. A variant of this attack injects malicious data when the name databases are copied from one DNS server to another. To mitigate and prevent this attack, a variant of DNS called Secure DNS (DNSSEC), which incorporates a public/private key authentication mechanism, is set out in RFC 3383 [96].

It is important to note that many of the attacks found in the wild fall into multiple categories. For instance, an attacker might use a worm to distribute a malicious binary that serves as a seed for a botnet. Once the botnet is successfully deployed, the attacker is ready to launch a DDoS attack.

In the recent years, flow monitoring has been an active topic in the security community, both to monitor anomalies, and to develop detection techniques that apply attacks listed above.

Many researchers have proposed approaches to monitoring large scale networks, as well as to tackle storage and scalability problems. The creators of Aguri [18] describe an aggregation tool with a variable granularity over the IP space. Their most significant contribution is a tree-shaped data structure, which aggregates TCP/IP packets spatially and temporally. By defining a custom time unit, the IP activity can be aggregated in subnets. The aggregation technique is guaranteed to capture at least a specified portion of network activity. In [100] uses the Aguri trees to detect attacks such as TCP Flood and DDoS attacks. As an example, in Figure 3.4 an instance of an Aguri Tree represents the traffic of the volume of traffic per source for the private network 192.168.0.0/17 during a time window t.

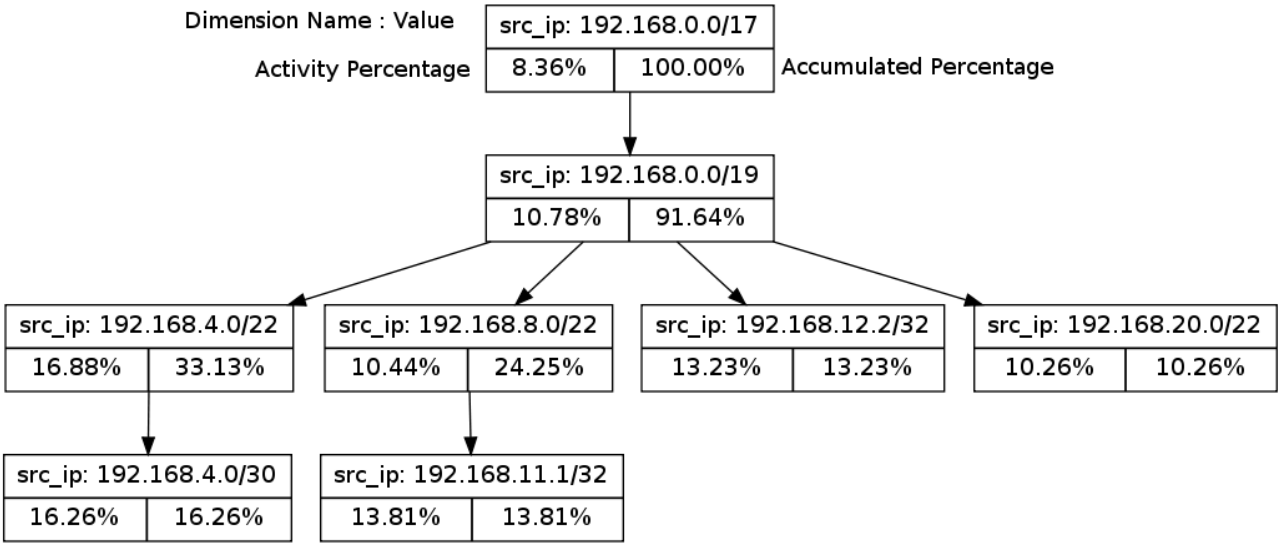


Figure 3.4: Example of an aggregated Aguri Tree representing source traffic volume of the private network 192.168.0.0/17

Authors	References	(D)DOS	Botnets	Malware or Spam	Network Attacks	Application Specific (DNS & Web)
Cho at al.	[18]	x			x	
Wagner et al.	[14, 97]	x	x	x		
Francois et al.	[15, 98]		x			
Moreira et al.	[50]			x		x
Lee et al.	[99]	x				

Table 3.1: Comparison of approaches for mitigating or preventing attacks according to taxonomic classification

Using a similar approach, BadHoods [50] reduces the volume of data, performing aggregation on a subnet basis to demonstrate that malicious hosts can be referenced locally in the IP space (*i.e.* malicious nodes may be close to each other in the IP space). Fenwick trees [39] handle single dimensions by efficiently storing prefix sums for given values represented as a table. For example, an IP subnetwork's hash tables can be indexed using a Fenwick tree, for rapid access to its aggregated traffic volume. However, the authors note that is difficult to arrive at an appropriate setting for the prefix size of the subnets to be monitored. The authors of [16] leverage the aggregation of attack flows by measuring the entropy of flow tables, rejecting flows which appears to be intended to overflow the cache tables of routing devices. This is done by studying the source and destination fields in flow tables using information theory algorithms.

In addition, Machine Learning techniques have been used recently to detect of anomalies. Significant contributions have been made by the authors of [14], who described to employ Support Vector Machines (SVM) to assess and classify IP Flows. Their classification-based approach has proven ef-

fective against port scanning and TCP and UDP flood attacks. Sequential Pattern mining is used by the authors of [99] to generalize attack patterns applied to an Intrusion Detection System.

Botnet detection is also a recurrent topic in the security community. A notable contribution made by [15,98] uses the MapReduce computing paradigm to cope with large IP Flow data sets. The authors propose a community detection approach for P2P Botnets using an adapted distributed PageRank algorithm [101].

The approaches described in this section are summarized in Table 3.1 on page 30.

3.3 Domain Name System

Overview DNS is an essential service for the Internet [102]. Its main function is to associate domain names with IP addresses but it can also be used to associate IP addresses with names (*i.e.* reverse search). Almost every device connected to the Internet uses DNS on a daily basis.

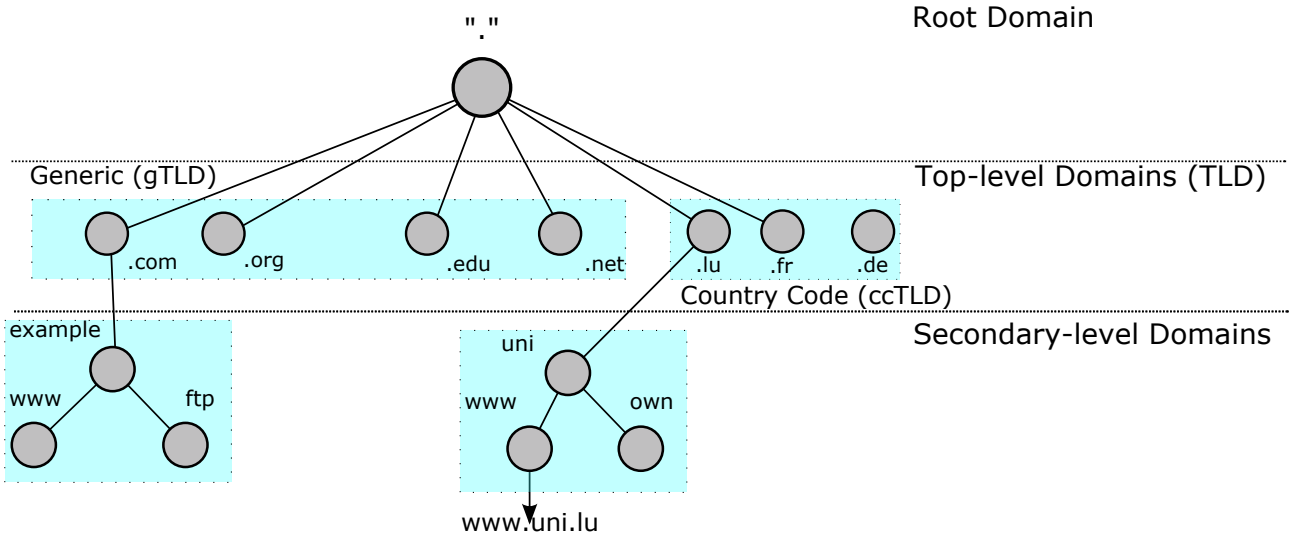


Figure 3.5: Example of domain name levels: The three essential layers

Architecture

DNS follows a strict hierarchical architecture. DNS can be tiered into three essential levels: Root Servers; Top Level Domain (TLD) or Authoritative Servers; and Delegated Zone Servers. There are 13 root servers distributed globally [67, 68, 102]. Each root server is responsible for answering root zone requests and also provides TLD server lists (TLDs are split into two categories: generic such as *com, org, edu, mil, gov, info* and country code such as *.lu, .fr, .de, .tv*). The DNS service delegates the responsibility of matching IP addresses to names to Authoritative Name Servers, which are in charge of their assigned domains. Authoritative Servers can also delegate authority over subdomains, (*i.e.* subexample.example.com). This pattern is illustrated in Figure 3.5. Each Secondary Level Domain is managed by the entity responsible for the registration of names in that domain. Thus, the delegation of zones contributes significantly to the DNS service robustness by decentralizing the database of

names and avoiding a single point of failure for domain namesresolution. Formally defined by RFCs 1034-5 [67,68], a domain name is a list of labels separated by a dot "." character. Each label is limited to a length of 63 characters, traditionally ASCII-encoded. The number of labels is limited to 127.

DNS resolution takes place every time a user or program requires to access an Internet resource by its domain name, for example *http://www.uni.lu* points to the World Wide Web home page of the University of Luxembourg. The resolution of a DNS query can be carried out in many possible ways; A client can resolve using its own cache stored locally (*e.g.* in Unix-based operative systems located in /etc/hosts), also a DNS server can maintain its own cache of resource record information to minimize the number of recursive request. Typically, during the resolution process extensive cache mechanism are be deployed at each level to cut down on traffic and speed operation. Caches are updated periodically, during the update process an attacker can aim at poisoning the cache with non legitimate data if caution is not taken [103].

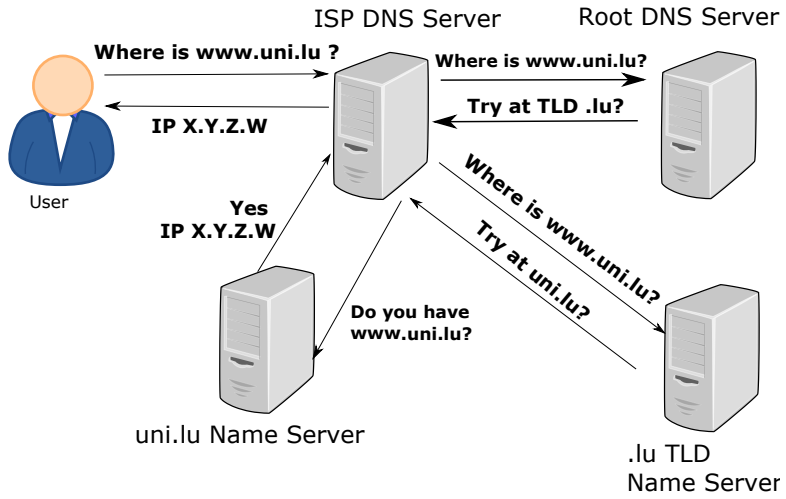


Figure 3.6: Example of the DNS resolution process

The following name resolution is illustrated in Figure 3.6:

1. A user wants to access a certain Internet host
2. The user's device makes a DNS request to the recursive resolver
3. The recursive resolver queries the root DNS server requesting for the corresponding TLD server
4. The TLD returns the address of the name server where the given domain is registered
5. The recursive server queries the delegated name server for the Uniform Resource Locator (URL) the user wanted to browse

DNS Security: Monitoring and Detection of Malicious Activity

DNS security is of paramount importance due the critic role of DNS in the Internet. Passive

monitoring of DNS is one of the pillars of DNS monitoring; this collection mechanism described in [27], opened a vast spectrum of research into monitoring and its applications. A passive DNS server is placed near recursive servers and gathers the requests to and the responses from authoritative servers. The passive DNS collection mechanism is illustrated in Figure 3.7.

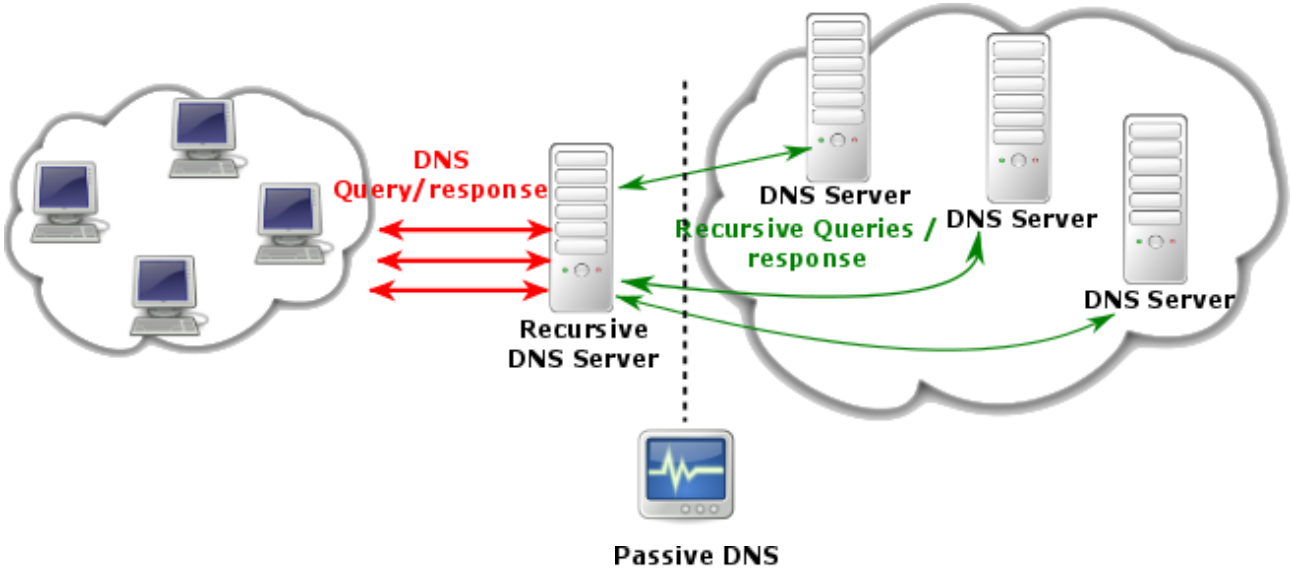


Figure 3.7: Example of a Passive DNS database gathering process

Passive DNS collection works as follows:

1. A client initiates a query requesting domain name resolution.
2. The DNS server receives the client's query and performs a recursive lookup.
3. The passive DNS sensor tracks the recursive lookup, storing the type of each answer in the recursive loop (Figure 3.6 illustrates the resolution step by step), as well as IP addresses and types of records
4. The client receives its response
5. The Passive DNS stores the lookup and request in its database.

Passive DNS has provided the basis several research works such as [24, 104–106].

DNS Security: Taxonomic Classification of Threats and Malicious activities

The following taxonomic classification, sets out the threats and malicious activities that are relevant to DNS monitoring; these activities are all observable through their DNS activity.

- Botnets: Networks of infected computers can be commanded and controlled using the DNS protocol. Among the most complex attacks in this category are FastFlux attacks, used to hide a hive of infected computers controlled to perform DDOS attacks, Spam distribution or other

malicious activities. FastFlux points infected computers to a TLD and changes the association of the corresponding A records frequently (every 300 seconds or less), whith the result that that a particular host acts as a reverse proxy for the zombies for only a short time before being replaced by other [92, 93, 107, 108]. This technique makes tracing the reverse proxy difficult, and blocking or shutting them down non-effective.

- Phishing: This threat tricks a user into accessing a malicious website impersonating a legitimate one. The malicious site attains to obtain login credentials or certificates for use in other malicious activities such as credit card and banking fraud [109].

- Malware Distribution: Malware and Virus are distributed through websites forged or compromised to infect visitors' computers in order to take control or perform another types of attack. This category also includes "Spyware". Such programs generally do not harm the victim's computer but instead collect personal data such as websites visited, login credentials or other sensitive information. [110]

- Application Specific: DNS cache poisoning consists in an inaccurate resolution of an Fully Qualified Domain Name (FQDN) at the cache of a DNS resolver. Usually, is part of a Machine to Machine (M2M) attack and it can alter the normal functioning of almost any application connected to the Internet. However, is observable through DNS monitoring techniques [107] such as Passive DNS.

DNS Security: Detection of attacks

In recent years researchers have made important contributions to the detection of the Fast-Flux networks. Using FastFlux, attackers can deliver malicious content through a distributed, fault tolerant architecture.

A Fast-Flux attack rotates the IP address assigned to a FQDN with a very short Time-To-Live (TTL) (such as three minutes). The IP addresses rotated (using *e.g.* round-robin scheme) belong to compromised hosts. The complexity can be increased by choosing infected nodes with the best available bandwidth and by using multiple mapping to balance the load across the hosts. This scheme is illustrated in Figure 3.8 on page 35.

A second step referred to in literature as Double Flux can be added to this attack [111]. In practice, the infected computers in the IP address pool act as reverse proxies, allowing an attacker to hide behind redirection. Essentially, the malicious content is no longer stored at the IP address resolved in the first step; instead it is accessible by traffic forwarding to an ever-changing pool of servers. The controller component of the Fast-Flux network is known as the "mothership"; it acts as a hidden upstream node, delivering malicious content to the content in response to user request.

While botnets are operated over Internet Relay Chat (IRC) or with a command and control system, however, the more powerful mothership concept allows FastFlux network to have more features than a regular botnet. One mothership can serve thousands of infected machines.

Despite the high fault-tolerance of FastFlux networks there are significant techniques for detecting them. In [92] the authors show how Fast Flux networks may be unmasked assessing revealing features (such as IP Addresses, Name Servers and Autonomous System Numbers). Another approach to detecting FastFlux networks is to monitor the TTL field associated to a given DNS Record. The

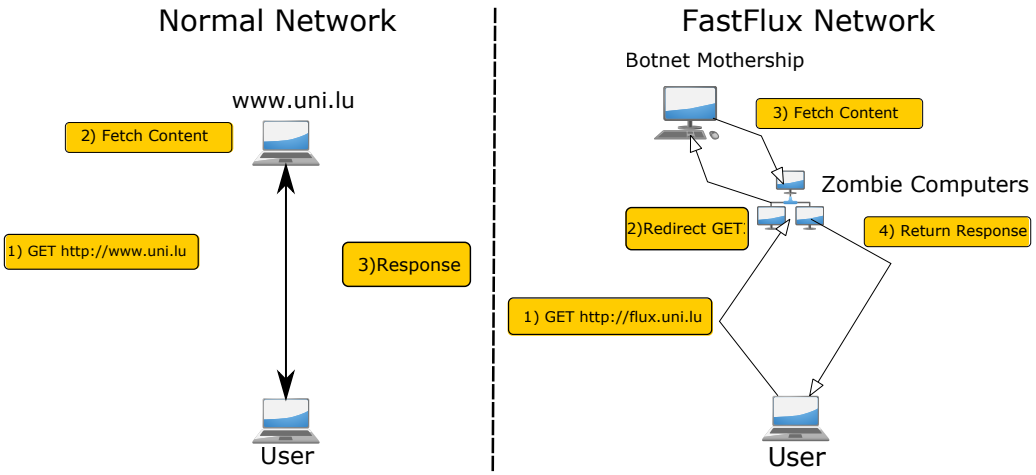


Figure 3.8: Example of the FastFlux resolution process

authors of [112] also considered features such as country of origin and type of Internet connection (cable, DSL, etc.), in addition to the TTL feature.

EXPOSURE [26] relies only on pure passive DNS analysis to detect any kind of malicious activity. Anomalies related to malicious activity are reflected in the name resolution data extracted as features from the Passive DNS collection. Request features are complemented with other features like the usage of numerical characters and time-based (metrics *e.g.* short-lived domains and traffic volume per unit time). The authors also considered looking for the longest meaningful words in a domain name. In [110], the system assess the DNS data of DNS servers near the top in the top hierarchy (root and TLDs) to determine the IP addresses of the recursive servers relaying the requests of their clients. Some methods also take in account shared behaviours between hosts and domains [22, 104] and so cannot be used to analyze a single domain. Failure graphs are helpful for locating suspicious activities because they link domains and the hosts from which requests originate. However, standard passive DNS cannot provide access to the request originator due to legal constraints steaming from privacy considerations. Studying user trends and in particular country-based behaviours is explored in [113].

A technique to detect botnets by monitoring DNS traffic and differentiating DNS query origin is described in [23]. The novelty of this approach is that it uses evidence on the usage of certain features that might are absent in legitimate DNS queries and present in a botnet-originated traffic.

Another major concern for DNS monitoring is the detection of phishing and malware download websites. In this context the use of a Passive DNS is a reactive method, leading to proactive alternatives being proposed in recent years. Another reason for discovering phishing domain names actively is that Passive DNS monitoring has specific limitations. Only websites visited by users of the ISP where the Passive DNS is installed are monitored. Thus, the data from Passive DNS Databases is only a sample of global DNS activity.

Another monitoring technique not limited by passive monitoring's restrictions is the semantic exploration of domain names. A variety of strategies for exploring the DNS name space have been proposed. Notable among these is brute-forcing. Is important to note that the same IP address may host more than one domain name, meaning that brute forcing of the IPv4 address space is not a priori

a solution for domain name brute-forcing. In any case, in the IPv6 address space, brute-forcing is not an option due the enormous grow of the address.

The authors of [24, 114, 115] take a proactive approach for detecting malicious websites. Using a semantic engine, they build a semantic tree to probe the name space. Semantic exploration leads to the discovery of malicious websites, which may then be categorized.

Heuristic-based models provide a significant alternative to approaches based on semantics. These approaches rely on classification algorithms leveraged by machine learning such as SVM, Bayes and logic regression. The features used to classify FQDN and websites can be extracted from either the host information (WHOIS, AS Number, IP Address) or from the website contents and attributes including its URL. The authors of [116] rely on the occurrences of certain terms in the composition of the domain name. In contrast, [117, 118] use features such as protocol, host name, TLD, domain length, the length of the URL. Table 3.2 summarizes the approaches to detect or mitigate DNS-based attacks.

Authors	References	Botnets (FastFlux)	Phishing & Malware (Passive)	Phishing & Malware (Active)	Application Specific
Holz et al.	[92]	x			
Antonakakis et al.	[119]	x			
Perdisci et al.	[112]	x			
Zdrnja et al.	[120]	x			
Choi et al.	[23]	x		x	
Blum et al.	[117, 118]			x	x
Prakash et al.	[121]		x		x
Marchal et al.	[24, 114, 115]		x	x	

Table 3.2: Comparison for mitigating or preventing DNS and network-based attacks by taxonomic classification

Last but not least, as an alternative to methods based on heuristics, several authors have worked on the active blacklisting of malicious websites. In [122] blacklists are compiled by monitoring the activity of newly-registered domain names, particularly the activity immediately following the registration. However, this approach cannot be widely applied because domain zone information is not always available and requires prior knowledge. Alternatively [121] describes PhishNet, a tool that uses existing phishing URL as a basis for generating and validating new ones. If the attempt is successful, then the entry goes into a dynamically confectioned blacklist.

As a final remark, on the DNS monitoring approaches cited focus on single properties of malicious sites, or aspects of malicious behaviour. While they perform with certain degree of accuracy and effectiveness when analyzing DNS data traces to detect malicious activity, there is scope for additional contributions in the area of multidimensional analysis.

3.4 Crowd-sourced Applications: Position Reporting Services

In recent years with the advances in mobile technologies and the proliferation of devices equipped with a GPS receiver, applications based on position reporting and urban routing have become available for widespread use. There is a wide range of applications[2] offering urban and road navigation services based on both Wi-Fi and GPS for positioning. These applications also use the positions reported by their users to estimate road conditions and make traffic predictions. Additionally, there is wide variety of aggregation service applications based on personal position. These applications work by gathering multiples sources of information and combining them to provide recommendations based on distinguishing features, such as geographical location. These include, applications to find specific shops, free parking places or which promote encounters with other users within a given distance of the user's position. Position reporting services are an essential foundation for urban navigation, routing applications and aggregated service applications. Hence, it is crucial important to monitor position reporting services for sings of abuse if databases of reported positions are compiled collectively without enforcing strict access control. This section addresses methods for detecting malfunctioning devices and malicious users.

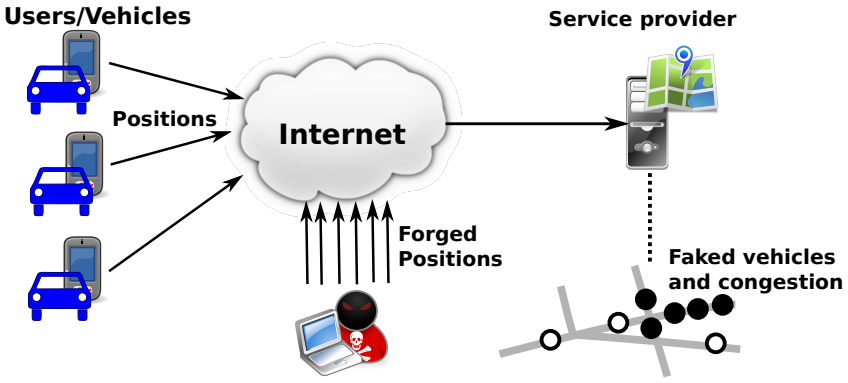


Figure 3.9: Example of a forged position attack on a VANET

Most of the applications offering navigation and social services are free to use. Therefore, without strong user authentication, crucial information like vehicle identity or position can essentially be forged to create multiple realistic, but false, driven paths or even single positions as shown in Figure 3.9. In [35], the authors describe the environment and conditions required to perpetuate a position-spoofing attack. Forged positions and Sybil vehicles are highlighted as major vulnerabilities for VANETs according to a risk study of roadside attackers in [123], however, the authors suggest no countermeasures verifying locations, there are several approaches which rely on the infrastructure to check the agreement between the claimed location and the actual existence of the vehicle at this location. For example, the distance between the claimed position and a fixed probing station can be estimated using challenge-response time [124, 125]. The deployment of roadside units is also a prerequisite in [126, 127]. For these approaches to be effective, the service provider must be in control of the infrastructure. However, in many applications for urban navigation and aggregation services based

[2] such as Google Maps (http://maps.google.com), OpenStreetNav, TomTom[3], among others

on geographical position, this might not be the case. Therefore, the effectiveness of the approaches described in [126–128] is predicated on specific implementations of VANETs.

Authors	References	Plausibility	Routing	Trust management	Application Specific
Leinmuller et al.	[123]	x	x	x	x
Chen et al.	[127]	x		x	
Xiao et al.	[126]	x		x	
Song et al.	[129]	x			x
Hsiao et al.	[130]			x	x
Bissmeyer et al.	[131]	x	x	x	
Zhang et al.	[132]	x		x	x

Table 3.3: Comparison of approaches for mitigating or preventing distributed application attack on VANETs by taxonomic classification

Vehicles may be equipped with additional sensors to check the locations of other vehicles in their immediate neighbourhood [34], but this requires a vehicle modification, in contrast to the recent trend, which is to use network devices like smart phones. Properly authenticating users has also been proposed [130, 133–135], but this assumes a defined and trusted set of users, which is crowd-sourced applications, although this potentially allow the attacker to create a large number of fake vehicles. Another possible scenario assumes few misbehaving vehicles among population authenticated using a key management infrastructure [33, 136].

Some approaches [33,130,133–136] propose mechanisms based on properties like position and speed for authenticating vehicles before retrieving data. However, this is not practical in VANETs where users are not requested to identify themselves.

In recent years there has been much research into anomaly in VANETs based on metrics for the measurement of radio signals (*e.g.* signal strength, and beaconing rate) [123, 129, 132, 137, 138]. The underlying concept is to build a physical model for a plausibility check, of physical properties of moving vehicles (*i.e.* position, speed, acceleration). The authors in [139] described how position may be estimated based on radio signal strength. Based on radio signal for the positioning service, [137] sets out the following model: a node (a vehicle) performs a self-check of received information from other peers to validate its consistency. The authors show how the number of vehicles in a given space may be calculated using radio range, map topology, and speed between two consecutive beacons.

In [128] this approach is extended by enabling cooperation between nodes. A similar approach using an ellipse-based is taken in [129]. Self sensing methodologies have been also discussed in [138]. A final approach uses the time-of-flight of signals between cooperative nodes to complement positioning services, as presented [132].

As well as being useful for anomalies detection, vehicle positioning is also relevant to network routing. The different approaches for routing within a VANET are discussed in [140]. Traditionally, nodes in a network have a fix geographic location; however in a VANET, a node's position can be dynamic. The advantages and disadvantages of each of the available routing protocols for VANETs are compared in [141], which classifies routing protocols into the following categories:

1. Topology-based routing protocols
2. Position-based routing protocols
3. Cluster-based routing protocols
4. Geocast-based routing protocols
5. Broadcast-based routing protocols

Positioning is crucial for cases 2, 3 and 4, since the routing protocols need to know the vehicles positions to establish links between nodes [140, 141]. Hence, an attack using spoofed positions might compromise the decisions of the routing algorithms. For instance, an attacker could fake its position, so being selected as an intermediary able to perform a Man in the middle attack.

Another domain of particular importance is the trust management of the disseminating node in VANETs. The issues are discussed in [142]. As in the field of positioning services, radio frequency measurements like signal strength and timing differences provide the major metrics used to derive conclusions. A whistle-blowing approach is suggested in [131]. The nodes report detected misbehaviour to a centralized service, which aggregates them in order to implement countermeasures. This allow the impact of a Sybil attack to be mitigated, since the misbehaving nodes can be prohibited permanently or temporarily from disseminating their positions. In this case, location spoofing detection relies on end-user devices while the approach which we set out in the following chapter shifts location spoofing detection to service provider. In this way, resources are saved on the client side and a grater variety of client devices can be supported: standard devices, like smart phones or tablets, can be connected over the Internet, and do not require the ability to intercept the communications of other devices in vehicles around them.

Is important to remark that most of the actual approaches for plausibility checking and trust management involve microscopic analysis of VANETs position-based applications. This implies knowing the position and identity of each moving node within the VANET. One of the drawbacks of using a microscopic model is its scalability. Hence, for a very large number of peers or nodes a macroscopic approach is a valid alternative. This allows nodes to be studied in groups and metrics aggregated.

3.5 Conclusion

This section reviewed a relevant part of the state of the art approaches for the analysis of distributed applications on the Internet. A common thread to the three fields; IP Networks, DNS and position-based applications is that usually week access control is enforced to join these networks or applications. Under this conditions malicious users may take advantage of the loose access restrictions, to carry out various types of attacks. Despite the great efforts proposed, the variety of security threats continues growing, as introduced by authors in [4]. This is a major concern from the security point of view, therefore a strong monitoring technique with data analytics for anomaly detection is needed to cope with the variety and complexity of attacks. Computing complex data analytics involves correlating big volumes of data from multiple sources, which is usually undertaken in large computing premises or data centres.

In the next chapter, we focus on the management of networks, specially in data-centres to host distributed applications. The performance of this applications becomes a critical factor when from their output, sensitive information can be computed for security or safety.

Chapter 4

Data Analytics: Management of Distributed Applications

4.1 Overview

While technological improvements were mainly highlighted by a new computing design and approaches, like Hadoop, network optimizations are primordial to guarantee high performances. This section reviews existing approaches to configure network and schedule flows in such a context. In the following sub sections, the various optimization methods grouped according to their intrinsic features and their contributions will be detailed. In particular, recent network technologies such as SDN empowered the programmability of switching devices. Consequently, more complex network scheduling algorithms can be afforded to leverage the performances of Map-Reduce jobs. That is why this section focuses on SDN-based solutions but also introduces common networking approaches which could be applied as well as virtualization techniques. The latter are strongly coupled with the network design. For example, end-hosts in a datacentre are virtual machines which can be assigned to different tasks and so would lead to various traffic types, which can be better handled if the network is adaptive and so reconfigurable easily.

4.2 Topology design

Datacentres networks usually follow a scheme called Hierarchical Network Model [143, 144] with three defined layers:

- Core layer: This layer is the backbone of the network where high-end switches and fibers are deployed. In this layer only L2 forwarding takes place without any packet manipulation. The equipment for this layer is the more expensive among the hierarchical network model.
- Aggregation or distribution layer: in this layer takes place most of the L3 routing.
- Access layer: This layer provides connectivity to the end nodes and so are located at the top of the racks. They performs the last step of L3 packet routing and packet manipulation. Normally, those are the cheapest devices in the hierarchical network model.

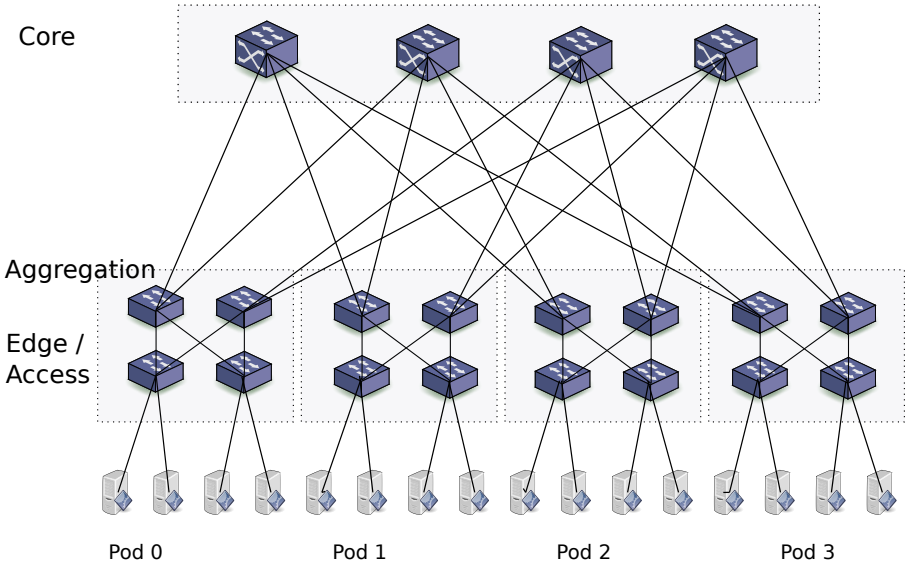


Figure 4.1: Example of a Hierarchical Network Model: Multi-rooted Network Topology

Thanks to this hierarchical model, a low latency is achieved for traffic between two nodes in the same rack. This explains why approaches like Hadoop leverage rack awareness to ensure fast replication of data by selecting nodes in the same rack for copying data (but also others out of the rack in order to guarantee data availability under a rack failure). In addition, this type of configuration supports a large number of ports at the access layer.

A specific instance of the hierarchical model is the fat tree proposed in [63] and illustrated in Figure 4.1 which enables fault-tolerance by ensuring redundant paths in a deterministic manner.

The fat tree or Clos topology was introduced more than 25 years ago [145] to reduce the cost on telephony switched networks. The topology layout is organized as k-ary trees, where in every branch of the tree there are k switches, grouped in pods. Actually, a pod consists in $(k/2)^2$ end-hosts and $k/2$ switches. At the edge level, switches must have at least k ports connected as follow: half of the ports are assigned to end nodes and the other half is connected to the upper aggregation layer of switches. In total, the topology supports $(k^2/2)$ k-port switches for connecting host nodes.

DCell [146] is a recursively interconnected architecture proposed by Microsoft. Compared to a Fat Tree Topology, DCell is a fully interconnected graph in order to be largely fault tolerant even under several link failures. In fact, high level DCell nodes are recursively connected to low level ones, implemented with mini switches to scale out as showed In Figure 4.2). Experimental results have showed that with a 20 nodes network can outperform by two times a large data-centrer used for Map-Reduce. As a downside, DCell requires a full degree of connectivity, making it in practice costly to maintain and deploy. To enhance network connectivity between servers, CamCube [147] is a torus topology where each server is interconnected to other 6 servers and all communications are going through them, without any switch for internal communication. Finally, recent propositions like [148] promote a high flexibility by alleviating the need for a well-defined fixed graph structure, as the fat-trees are, and so by introducing some randomness in the topology.

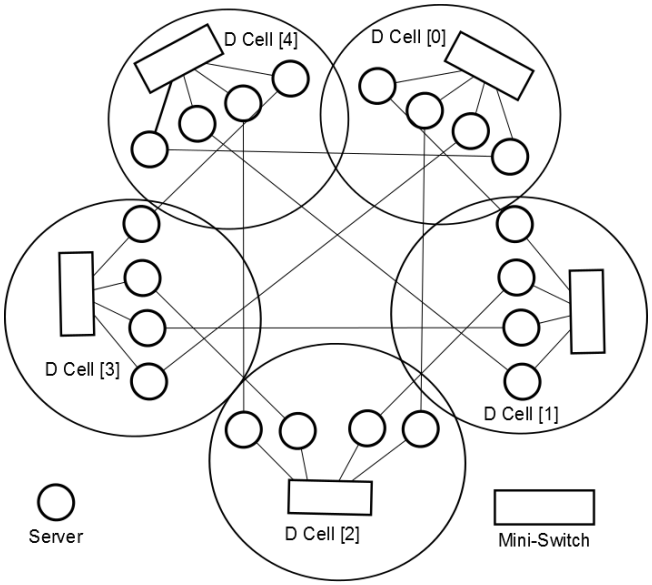


Figure 4.2: A DCell topology for 5 Cells of level 0, each containing 4 servers (src: [146])

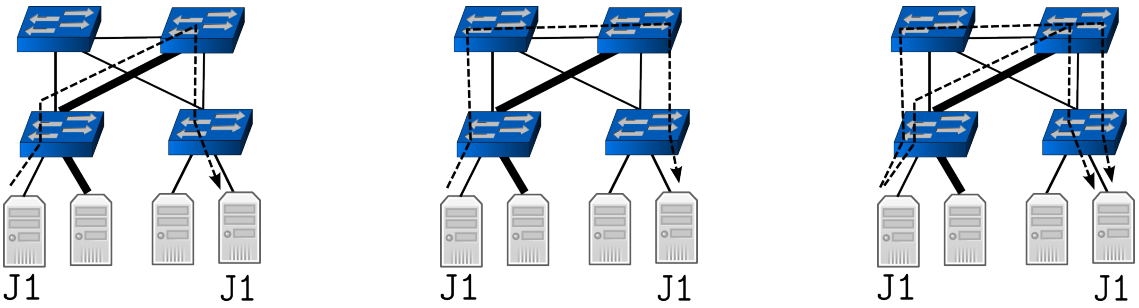
4.3 Conventional Networking

4.3.1 Routing

Datacentre network topologies like fat trees imply a large number of links leading to redundant paths. Therefore, routing algorithms can take benefit of that to achieve a higher bandwidth. As an illustrative example in Figure 4.3(a), the shortest path is used to route the traffic between the two tasks of the job *J1*. Unfortunately, it goes through a congested link. Hence, a redundant path can be used (Figure 4.3(b)) and even multiple of them conjointly (Figure 4.3(a)). Although these approaches have been proposed for routing in general, they are also used in datacentres to improve the performance of the Big Data applications. This is the reason why this section covers some propositions about how to use these principles in case of Big Data. However, the general issues are (1) to predict the traffic patterns and (2) to be able to rapidly change the configuration of the routing when the traffic suddenly changes, which is the case in a cloud infrastructure.

Nowadays, a major representative of such an approach is the Equal Cost Multi Path (ECMP) algorithm [149]. ECMP leverages the opportunity to route flows among multiple paths. Unlike traditional routing algorithms like Open Shortest Path First (OSPF) which are considering a single best path, ECMP consider all the best multi-paths according to any metric (as for example the number of hop) among which a single one is selected for a given flow through a load balancer. The number of multiple paths is dependent on the router implementation but usually bounded to 16. Hence, this may yield to a lower performance than expected for large datacentres. In fact, the amount of entries in the routing tables grows at exponential rate, increasing the latency of the routing algorithm. Commercial solutions promoting multi-path routing include FabricPath by Cisco Systems BCube [143], VL2 and Oracle Sun data-centre InfiniBand.

In addition to promote the fat tree topology usage for datacentres, the authors of [63] proposed a



(a) Shortest path routing (b) High throughput routing (c) Mutlipath routing

Figure 4.3: Routing decisions fro one job with two tasks. The width of a link represents its load.

dedicated routing algorithm based on an approach called *Two-Level Routing Tables*, where the routing tables are split into two hierarchical tables linked on the prefix length of the network address. A two layer table approach aims at leveraging the routing algorithm speed for establishing a route. This is possible because the authors introduced a private addressing system respecting a pre-established pattern like *8.pod.switch.host* assuming a class A network. The first table index entries use a left handed prefix length (eg, 8.1.2.0/24, 8.1.1.0/24, etc). The entries of the first table are linked to a smaller secondary table indexed by a right handed suffix. (eg, 0.0.0.1/4, 0.0.0.4/4). For example, to find the route to the address 8.8.8.8, the algorithm will lookup in the first table, find the corresponding entry for the first part of the network address 8.8.8.0/24, then jumps to the secondary table and find the remaining of the route. Since that each switch of the aggregation layer in a fat tree topology has always a $k/2$ degree of connectivity to the access layer, two-Level routing tables are bounded in the worst case to $k/2$ entries for suffixes and prefixes. Moreover, flows can be actually classified by duration and size. Then, the proposed algorithm in [63] minimizes the overlap between the paths of voluminous flows. To achieve that, a central scheduler is in charge of keeping track of used links in the network in order to assign a new flow to a non-used path. From this perspective, it fails into the category of centralized networking (see section 4.4.1) where a switch acts as the controller by informing other ones about the link to use to forward specific packets of a flow.

The flow establishment is also leveraged by the previously described routing lookup. In this approach, instead of routing traffic at a packet level, streams of data are grouped into flows and routed as a whole entity. One of the benefits of this approach is a faster route computation as it is reduced in a similar fashion as in circuit switching legacy technology. For example, if a host node requires to transfer a large data file as a part of a Big Data job, the whole stream will follow a pre-established route, reducing the latency of establishing a different route for each packet of the stream.

In order to enhance routing and network speed, hardware plays a core role. Therefore, there have been propositions to replace standard hardware. In particular, the authors in [150] argue for an hybrid optical-electric switch as optical links achieve higher throughput but has not well adapted to bursty traffic. Combining both technologies thus helps in obtaining good trade-off between accuracy and cost. Moreover, the technological availability of programmable circuits also lead to an approach for imple-

menting switching devices, specially in the aggregation and core layer using ASIC and FPGA devices. Authors of [151] propose an approach for implementing switching cards with a PCI-E interface.

A recent proposal [152] addresses dynamic routing by replacing the traditional DHCP address configuration for an another automated configuration address system. In this approach, the network is automatically blue printed as a graph. Then, by interpreting a set of labels assigned to each computing node, the system tries to find an isomorphism that minimize the traffic at the aggregation layer. From the preliminary results, this approach has yielded promising results. However, it actually runs only over BCube or DCell topologies because they have a fully connected topology.

4.3.2 Flow Scheduling

In order to provide a network service that fits to the service needs running over the network, network operators may perform traffic management. This consists in classifying the traffic according to the intrinsic characteristics of each service or application using the network directly. For example, using the IPv4 Type-of-Service field, it is already possible to define policies to give special treatments to Big Data applications. Similarly, the IPv6 Traffic Class includes the possibility of attaching information specific to a given application to the packet stream. Another types of support for enabling network infrastructure to perform management of the traffic is proposed in RFCs [153] and [154]. The first (DiffServ) proposes a protocol for differentiating services and its network behaviour. The latter, RSVP (Resource Reservation protocol), specifies also a protocol, that enables application to reserve network resources in advance of initiating a data transfer.

As highlighted in introduction, data analytics includes both batch processing and streaming analytics which are different by nature. In particular, batch processing are more prone to use the network heavily during certain phases while streaming uses the network constantly but still with various rates. Therefore, the apparition of a batch job (like Hadoop) may suddenly impact the network and so the other underlying applications. The authors of [155] researched on scheduling flows in a FIFO-LM (Limited Multiplexing) fashion. Therefore, flows are scheduled in the order of arrival with a certain degree of freedom since multiplexing may occur over a limited number of flows which thus allows small flows to be processed along with a large flow. Such a policy is then applied to switches using queues which are associated to application identifiers assuming the latter have been set by a central point of the network. Such an approach allows the co-execution of batch and streaming Big Data applications.

Limitations It is important to mention that, in common networking, only aggregation and core layer switches have the capability of scheduling flows. This is a limitation given by the hardware. To be able to exploit the full potential of flow scheduling, an additional network hardware is required. This is often implemented in a central controller, that takes the delegation of the computing algorithms. Thus, with a central controller, core and aggregation switches can be replaced by simple switches. One of the main advantages of using this approach is the reduced cost of switching and forwarding (L2) devices. Another disadvantage of common networking is that the network configuration remains static and so impacts on the maintenance cost of the infrastructure because any modification of the topology must be wired manually by the network administrators. Virtualized networks cope with the lack of flexibility of common networks, and became popular over the last years thanks to the emerging virtualization technologies and computing power to support them. Traditionally, data centre owners

offer their clients not only virtual machines (know as Virtual Private Servers (Virtual Private Servers (VPS))) but also the capability of virtualizing the network infrastructure. This allows VPS users to create customized topologies. Virtual LANs (VLAN) have been popular in the past decades for splitting large organizational networks into smaller ones. This approach fails to segregate application traffic because inside a VLAN all the traffic is bounded to the same routing system. A possible solution to this issue is to use a dynamic topology, adapting the paths to the specific needs of each application. In such a scope, the next Section 4.4.1 covers emerging methods to configure the network dynamically using a centralized approach.

4.4 Centralized Networking

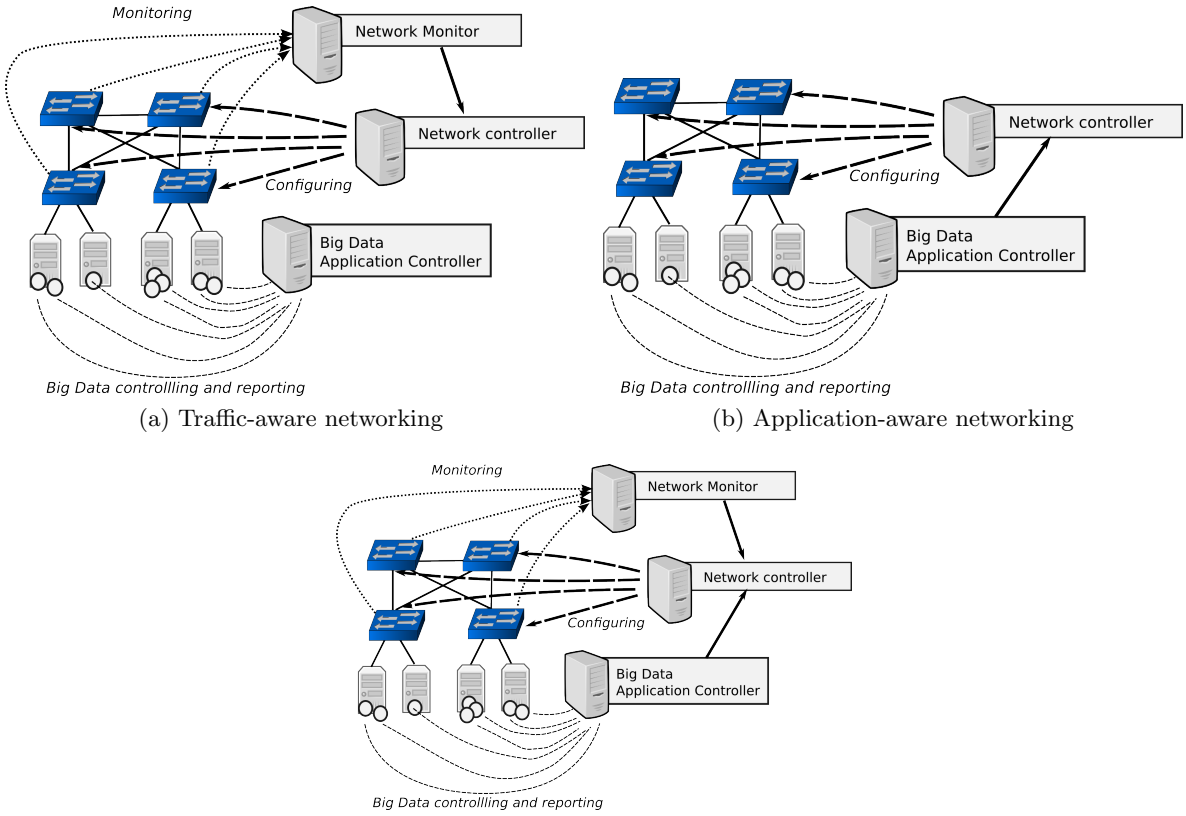
As dedicated to centralized solutions for networking, this section covers both formal approaches as well as practical implementations. Solutions highlighted in the following paragraphs combine three strong concepts: computational patterns present in most of Big Data services, data centres network architectural improvements such as hierarchical topologies (*e.g.* Fat-Trees) and dynamic routing algorithms leveraged by the adoption of technologies such as SDN. These three forces combined together allow to adapt the network configuration from the core to the aggregation infrastructure layer to suit better Big Data application needs.

Routing and scheduling decisions rely on the traffic matrix. Such a matrix can be observed in real-time at the network level but can also predicted in order to plan next actions. The traffic matrix usually reflects the flow's size, duration and frequency for each pair of nodes and eventually application instances or even between multiple tasks of a single job. Alternatively, Big Data applications can interact with a central controller to expose their current usage and needs. These two types of approaches are differentiated in Figures 4.4(a) and 4.4(b). In every cases, there is a Big Data application controller or manager (like the *jobtracker* or the *resource manager* in Hadoop) which is in charge of triggering and monitoring the tasks. In Figure 4.4(a), a monitoring service is gathering traffic statistics. In this figure, it gets it from forwarding devices. Then, this traffic monitor sends the information to the network controller itself which is in charge of taking routing decisions. The monitoring can even be done by OpenFlow [191] [156] (see next section) as an OpenFlow controller can request such statistics from OpenFlow switches. In such a case, both the monitor and controller are merged in a single entity. In a second case (Figure 4.4(b)), the Big Data controller sends itself information about the running applications to the network controller which can thus take proper configuration actions. Finally, it is also possible to imagine an hybrid approach (Figure 4.4(c)) where both types of information are gathered. It might be useful if the level of details from the Big Data controller is too coarse-grained.

As a brief summary, the different methods covered in the following paragraphs are, actually, similar to conventional networking (select better paths, minimizing congestion, etc.) but they rely on a higher and more dynamic coupling between the network configuration and applications (or the corresponding traffic).

4.4.1 Software Defined Networks

In the recent years, SDN technology has emerged. A new layer of abstraction is introduced for managing networks. Under this approach, switches are just forwarding devices while most of the



(a) Traffic-aware networking

(b) Application-aware networking

(c) Hybrid awareness

Figure 4.4: The different type of *-aware networking (Small circles represent a task of a Big Data process

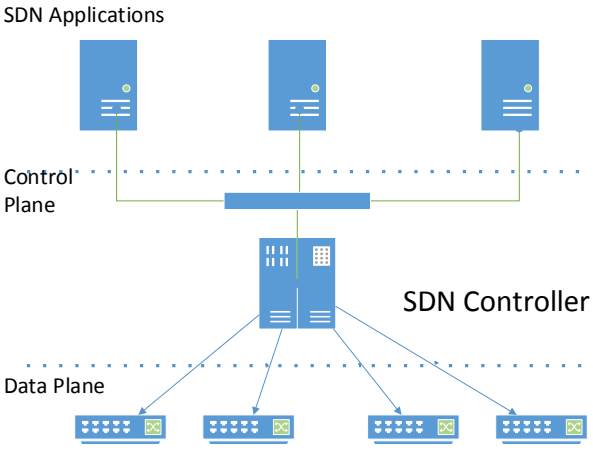


Figure 4.5: Software Defined Network Architecture Example

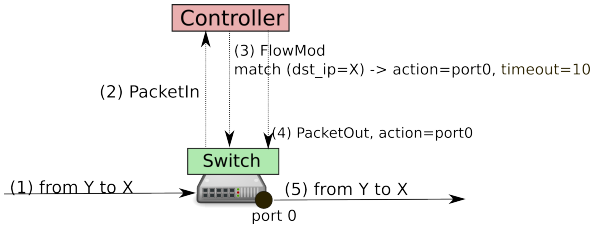


Figure 4.6: Software Defined Network with Open Flow rules

computing (*e.g.* routing decisions) is performed in a central controller. As a main improvement, forwarding devices are centrally controlled and the cost of administration is reduced. Also, the cost of forwarding devices can drop drastically.

As shown in Figure 4.5, when decoupling and abstracting the network control, two planes are usually distinguished:

- Control Plane: The concept of the control plane is to have a dedicated communication channel for exchanging signalization messages among forwarding and management devices. Most of the available products for SDN expose a so called North Bound API for applications to subscribe to real time statistics and service usage. Control plane messages have either as source or destination the central network controller.

- Data Plane: In this layer, also refereed as the Forwarding Plane, the traffic sent is the traffic to be routed or forwarded by the infrastructure. The traffic in this plane is accounted and measured and, also forwarding algorithms take place according to rules installed.

Additionally, the application layer is composed of custom made applications. The latter subscribe to the North Bound API of the SDN controller to enable extra functionality not provided by the hardware manufacturer. For example, these applications might be security oriented [157] or for routing purposes [158].

OpenFlow [159] is adopted as the most popular choice as the control protocol. OpenFlow acts as the communication channel between switches and controllers (*e.g.* NOX, Floodlight, POX, etc). An OpenFlow rule consists of two parts: a match field, that filters packet headers, and instructions, indicating what actions to take with the matched packets.

Upon arrival of a packet to a switch, the switch forwards this packet to a controller if it cannot find a rule in its cache. This event is known as *PacketIn*. When a *PacketIn* is sent to the controller, the highest priority rule that matches the packet header will indicate an action. If no match is produced, then a default action can be used. After looking up for an action to take, the controller forwards

the action to be taken to the switch. This event is known as *FlowMod*. Finally, the packet is sent (*PacketOut*). The figure 4.6, illustrate an example where a routing action is taken upon arrival a packet with destination X and source Y. Additionally, a controller can provision switches with flow tables entries in advance. Hence, a *PacketIn* message is not required to emit an event *FlowMod*. The rules also have soft (last seen packet) and hard (maximum absolute value) timeouts, after expiration of these time-outs the rule is removed.

While originally proposed for campus but not large networks, the modification proposed by authors of [160] consists in reducing the overhead induced by OpenFlow to enable a more efficient flow management for Big Data networking through the extensive use of wild cards rules within the switches to avoid to invoke the OpenFlow controller for each new flow. However, the extensive use of wild cards on OpenFlow might cause loss of granularity in the statistics derived from the counters on the controller and evidently on routing and scheduling decisions. As mentioned, [160], DevoFlow aims to devolve control by cloning rules whenever a flow is created using wildcards. The cloned rule, will replace the wild cards fields using the clone's specific information. Additionally, DevoFlow enriches OpenFlow rules by including local routing actions (without relying on the OpenFlow controller), such as, multi path routing. This last feature allows to rapidly reconfigure the route for a given flow leveraging the flow scheduling.

4.4.2 Traffic-aware networking

The Topology Switching approach [161] proposes a novel approach in network management by exposing several adaptive logical topologies on top of a single physical one. This approach targets the allocation every individual flow to a specific path to satisfy an optimization objective, as for example network capacity, trying to use as most as possible the available bandwidth. Considering a fat-tree Topology as showed in Figure 4.1, individual flows from Map-Reduce bisection traffic are considered as a separate *routing task*. Thus, each task runs an instance of a particular routing system. For each routing system, a pre-allocated bandwidth is established in the physical topology to ensure a certain total bandwidth. Topology Switching is implemented in a central topology server, responsible for allocating resources but also for subtracting unused resources and collecting metrics. The two metrics used in this approach are the bisection bandwidth and the all-to-all transfer. The bisection is used to measure the topology ability to handle concurrent transfers at the physical layer. The all-to-all metric is used to evaluate how the logical topologies react under a worst case scenario. Based on both metrics, the Topology Switching approach runs an adaptive algorithm for readjusting the logical configurations for the virtual networks. Topology Switching offers an alternative for "one-size fit all" data centre design, providing a good trade off between performance and isolation.

The Hedera [162] scheduler assigns the flows to non-conflicting paths similarly to [63], especially by aiming at not allocating more than one flow for routes that cannot satisfy the network requirements in terms of bandwidth. Hedera works by collecting flow information from the aggregation layer switches, then computing non-conflicting paths, and re-programming the aggregation layer to accommodate the network topology in order to fulfill the Map-Reduce jobs requirements. This approach yields an optimization at the aggregation layer in the specific case of the bisection. Unlike a local approach, bottlenecks can be identified based on a global overview of path states and traffic requirements.

4.5 Application-aware networking

The methods described in this section improve the network performance by rescheduling flows according to an application-level point of view. At the layer of transport, flows are not distinguishable from each other but groups of computing nodes in Big Data Application usually expose an application semantic. For example, a Hadoop-based task can be composed of several shuffle phases and each of them corresponds to a specific set of flows. Furthermore, a Big Data application can evaluate its current stage. For instance, the mapper status (completion time) is computed from the proportion of the data, from the source, which has been read and such a completion time can approximate the remaining data to transfer. Therefore, a mapper having read 50% of its data source and having already sent 1GB of data should approximatively sent again 1GB. This is an approximation and it cannot be guaranteed that the mapper will send as much information for the remaining data it has to read. For example, a usual example where a mapper sends a $< key, value >$ pair for each read line can also apply some filtering and so may emit nothing based on the line content.

Therefore, some methods build a semantic model reflecting the Big Data application needs, which used for these approaches associates the network traffic to be managed with the characteristics and the current state of the application it originates from. This model might differ among the different proposed works buts aims at assessing the state of the Big Data applications and their related flows.

In this context, the authors of [163] propose to optimize network performances by arranging QoS policies according to users requests. Host nodes running Big Data applications can exchange messages within this complementary framework called PANE to submit QoS policies similarly to what can be done with conventional networks (Section 4.3.2). Naturally, this approach will lead into traffic over subscription under high traffic demand circumstances. To solve this issue, users have also to provide conflict resolution rules for each QoS rule they submit into the system. Also, this approach can be employed for implementing security policies such as denial of service prevention by setting a top hierarchy policy triggered at the SDN controller.

OFScheduler [164] is a scheduler which assesses the network traffic while executing Map-Reduce jobs and then load-balance the traffic among the links and so aims at decreasing the finishing time of jobs based on the estimated demand matrix of Map-Reduce jobs. OFScheduler assumes that Map-Reduce flows can be marked (for example by Hadoop itself) to distinguish those related to the shuffle and those related to the load balancing (when a task is duplicated). The scheduling first searches for heavy loaded links and then selects flows which should be offloaded by giving the preference to (1) load-balancing flows and (2) larger flows in order to limit the impact on performances (cost of the offloading due to OpenFlow rule installation). The reason for (1) is that it corresponds to a duplicated task whose the original may finish somewhere else in the data centre unlike the others. The rational behind (2) is to minimize the global cost of offloading and so by moving a big flows, there are more chance to remedy the problem of the link load without re-scheduling additional ones.

Assuming optical links, authors of [11] describes an application-aware SDN controller that configures optical switches in real-time based on the traffic needs and patterns of Big Data applications. By enabling the Hadoop Job Scheduler to interact with the SDN controller they propose an aggregation methodology to optimize the use of optical links by leveraging intermediate nodes in the aggregation. In the most simple case, when a single aggregate has to gather data through N switches whereas the number of optical links is lower, it has to go through multiple rounds (optical switching) in order to

complete the job. The other switches only using a single connection to the aggregating switch can also be connected together to act as intermediate nodes to form a spanning tree rooted in the aggregator and so to avoid the multiple rounds. Such a principle (M-to-1) is extended towards general case with M-to-N jobs or when multiple single aggregation overlaps (*e.g.*, different sources overlap their aggregators). This thus requires more complex topologies such as torus. Other approaches addressed in this chapter such as DCell or CamCube also make use of high redundancy to build similar shaped topologies. Building a torus topology is more complicated than a tree because the search space for suitable neighbours is larger, a greedy heuristic is used to support as much as possible the traffic demand. The routing algorithm within the torus topology is meant to exploit all possible optical paths. Authors also propose to assign weights to the optical links for load-balancing purposes on the torus topology.

Although the previous approaches rely on the current status of an application, FlowComb [165] is a proactive and reactive method for flow scheduling. It allows the Hadoop controller to specify requirements but also promotes the use of a statistic-based method that predicts based on the network load of previous runs. Hence, this approach lies between application-aware, because it interacts with the application to monitor it, and traffic-aware, because it aims at predicting future flows similarly to a traffic matrix. Based on that, any routing or scheduling approaches described in section 4.4.2 could be applied, especially Hedera [162] which has been chosen by the authors. The central decision engine gathers all the job pertinent data and creates a set of Open Flow rules to be installed temporarily and erased after job completion. However, the main drawback of the proactive method using estimation is that circa 30% of jobs are detected after they start, and 56% before they finish.

Coflow [166] proposes a full reactive method, that only after receiving the Hadoop Job Scheduler network requirements is able to yield results. Its implementation exposes an API for declaring flows at application level. This API can be used for example from the Hadoop Job Scheduler as it is mentioned by the authors to express on demand bandwidth requirements at the different phases of a Map-Reduce job. Actually, CoFlow introduced an abstraction layer to model all dependencies between flows in order to schedule an entire application, i.e. a set of flows, and not only a single flow.

In contrast with the methods described previously, the authors [167] propose an approach for routing on a packet basis by splitting the flows in chunks similarly to TCP. These chunks are distributed to the available ports of a switch using different strategies: random, round robin and counter based. However, the main limitation of this approach is the necessity to reorder the chunks.

4.6 Conclusion

In this chapter we have discussed the most relevant approaches behind the recent technological improvements data centre computing, design and architecture for coping with large data sets. This set of technologies and methodologies for analyzing large data volumes has been recently referred as big data analytics. Data centre networks have recently included techniques for reconfigure the topology and schedule flows to support the needs of the big data applications. However, the traditional networks' limitations might impact on the performance of big data applications. With the emergence of SDN in recent years, approaches have been proposed for improving network management and enhancing network performance, as for example, using OpenFlow to speed up the traffic forwarding [159]. Specific approaches for big data applications in SDN-based networks have been also proposed, as for

instance enabling the network controllers to consider the state of MapReduce jobs in the forwarding rules [63, 162].

Simultaneously, traffic-aware approaches have been proposed by several authors to trigger rapid changes in virtual network topologies to fit the requirements of big data applications [162]. The approaches of this type are not free from limitations, since not every data centre is able to move onwards a fully virtualized network scheme without making significant changes in the infrastructure. Therefore, this scenario sets out the need of approaches focusing the application requirements to take routing and management decisions at the control level. Application-awareness approaches are of a paramount of importance, since they are able to focus on real application specific needs to perform network optimizations and leverage routing.

Part II

Contributions

Chapter 5

Methods for Data Analytics

5.1 Multidimensional Aggregation Monitoring

Problem Statement: Network monitoring is a critical tool for network management and security, in particular at large scale networks and at ISPs. The volume of data to analyze has experienced an exceptional growth in recent years, as suggested by [1, 12]. While, aggregated monitoring approaches have been proven to be effective [14, 16, 18, 19], existing approaches have their limitations when they consider multiple sources of information such as network traffic, user activity, network events from Intrusion Detection System. All of these produce a large volume of data which needs to be stored, analyzed and correlated. Scalability may become an issue while combining individual approaches. Therefore, an efficient aggregation approach combining the multiple sources into multidimensional data is a good solution in this scenario.

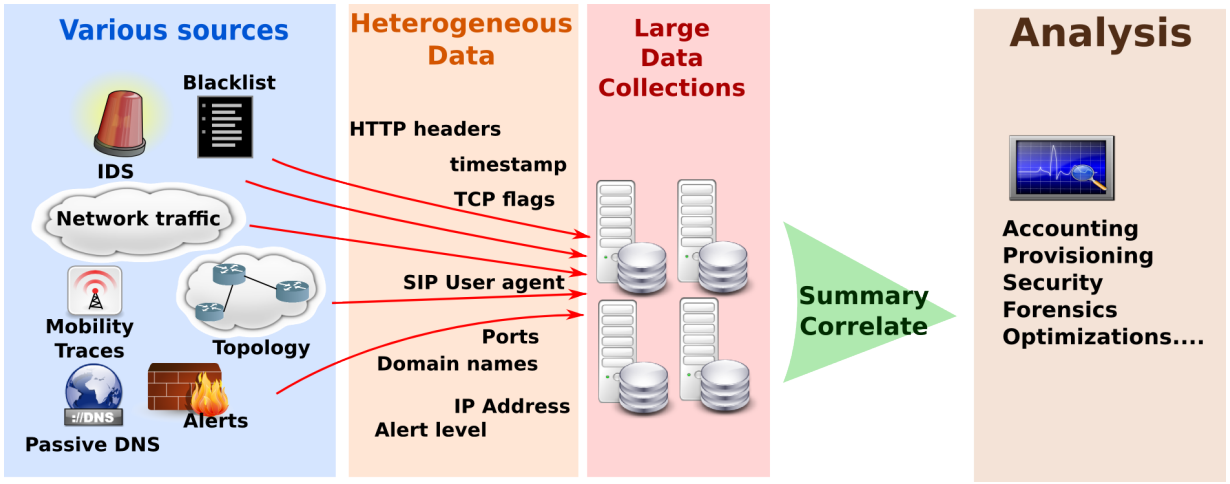


Figure 5.1: Schematic use of MAM

We targeted the design and implementation of a novel aggregation technique which is able to handle multiple kinds of dimension, *e.g.* features like traffic capture or host locations, without giving any preference a priori to any particular feature for ordering the aggregation process among dimensions.

Aggregation supports dynamic space granularity, which can be determined on the fly depending on the events which it is desired to monitor. In order to keep the computational overhead low, an efficient data structure is needed.

Our approach can be potentially used in many fields to leverage monitoring, such as at ISPs, Computer Emergency Response Team (CERT) organizations, or other forensic units for security incidents, as illustrated in in Figure 5.1. Also, since our approach allows any hierarchical type of data for aggregation, telephone operators for example could use multidimensional aggregation to consolidate call records and speeding up post processing analysis.

5.1.1 Overview

The general goal of Multidimensional Aggregation Monitoring (MAM) is to to aggregate multidimensional data spatially and temporally. To do this, MAM combines multidimensional unstructured data into a single tree structure associated with a time frame or window, the temporal dimension. Even if the data does not contain a temporal feature, MAM can still be used to create a single tree for the whole dataset. Figure 5.2 gives an overview of the procedure and its main components.

Our approach uses an aggregation tool which accepts network monitoring traffic data from heterogeneous sources and very large collections. For example, MAM can simultaneously consider the source and destination IP addresses and the ports of a network traffic capture, in particular from NetFlow records.

An important attribute of our approach is its capability to aggregate data, such as traffic load, over multiple dimensions without having to specify the granularity. Considering the NetFlow records case, this means that the data will not necessarily be aggregated using a fixed partitioning into subnetworks. With our tool, the space of a dimension is not split a priori, for example, by using a fixed-size subnets. The aggregation is thus able to keep track of information related to IP subnets having different sizes. This is illustrated in Figure 5.3, which exposes two possible partitions of a bidimensional IP address space. Figure 5.3(a) shows that the data's original distribution does not fit a regular partitioning (source, destination), preventing it from being partitioned in a simple way. In contrast, Figure 5.3(b) shows a dynamic partitioning of the space,where the data is grouped by a space partitioning built to fit the data's original pattern. The aggregation process is guided by the events which it is desired to monitor, in this case reaching a quantifiable visibility (5% of the traffic load in bytes or packets for instance).

Another important attribute relevant to our approach is to aggregate data without giving any preference to any particular view in advance. Considering the example of NetFlow records, this means that the data will not necessarily be aggregated first on source IP addresses and then on ports. This decision is usually made by human experts using their own skills. Some of will want to monitor the usage per service first (ports) and then per IP addresses, while others will prefer to have statistics per IP addresses first and then the details for each service. In the first case, statistics about global traffic of an IP address are not directly available and need to be reconstructed by iterating over all ports.

Aggregation can be used for post-analysis or for real time monitoring. In the first scenario, memory and computational cost can be rise considerably in order to reach a high level of precision. However in the case of real time monitoring execution time is a real constraint. Thus, two optimization strategies are introduced in Section 5.1.4 for real time computing, leading to a particular approach to monitoring.

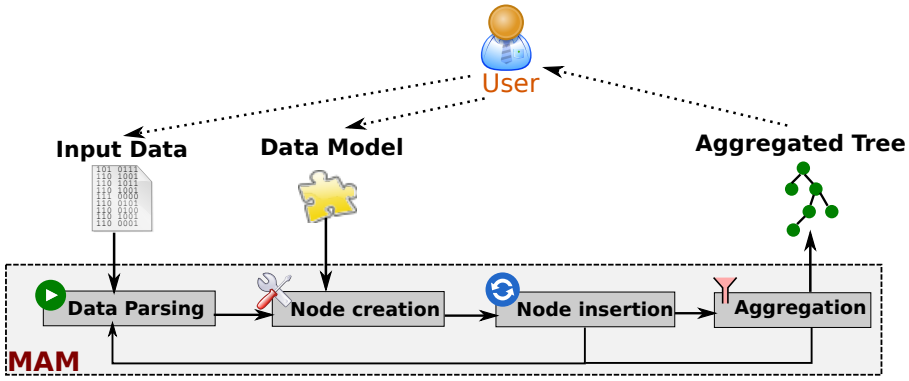
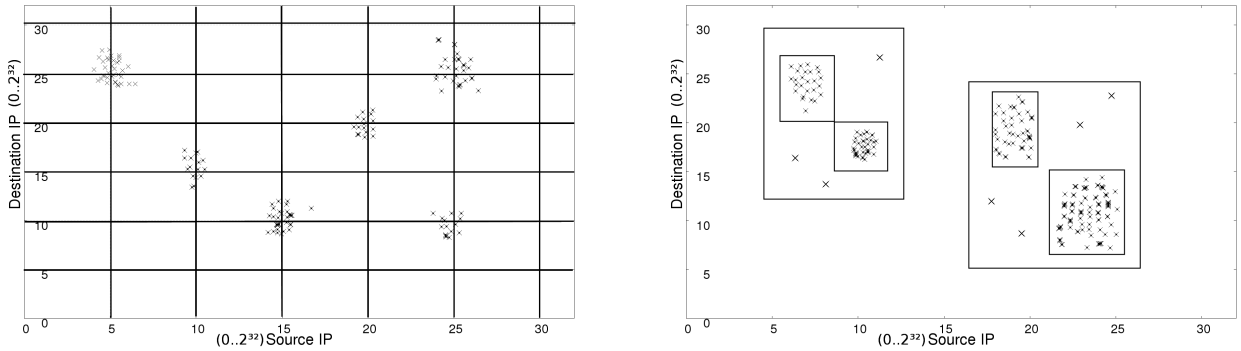


Figure 5.2: MAM overview



(a) Regular IP Address space partitioning

(b) Dynamic IP Address space partitioning

Figure 5.3: IP address space partitioning examples

Scalability can be achieved by optimizing the usage of storage data structures. MAM leverages a tree based structure to store data in a hierarchical representation of bounded size. Different strategies are described to satisfy this constraint.

MAM allows the user to provide a data model describing both the type of data and how to parse and aggregate. Some data models are already provided with the tool, for example for the IP address, services and geographical coordinates. As illustrated in Figure 5.2, the first step is to parse the input data. For each data item (usually a line in a file), MAM creates a corresponding node. Which is inserted into the current tree based on the hierarchy of each dimension (the following sections provide details). These steps are repeated until the tree reaches a preset size, at which point it is compressed by aggregating nodes in the tree from the leaves to the root in order to keep only relevant information. Compression takes place either when the size of the tree is too great (online aggregation), reducing resource consumption, or at the end of a time window (simple aggregation).

Following end-of-window compression, the tree is returned as a result to the user and MAM will continue the process by starting to parse the next time window.

5.1.2 Data Aggregation Structures

Aggregation reduce the granularity of data by grouping data instances according to some criterion, *e.g.* sharing similar properties. For instance, aggregation over the IP address space could consider the shared prefix as the property, leading to monitoring of network traffic by subnet. The parameter defining the size of the subnet, equivalent to the prefix length, is quite arbitrary like /24, /28, etc. Changing this parameter leads to different observations according the context [50]. This is illustrated in Figure 5.3 with a naive example where a regular partition of IP address space, in Figure 5.3(a), may not lead to identify subgroups of hosts, unlike the dynamic partitioning of Figure 5.3(b). For example, to monitor the hosts listed in Traffic Flow Table 1, monitoring /16 networks will give a general overview of the network activity which is probably too coarse; /24 seems more useful. However, using /24 or greater may be too fine-grained at the Internet level. Moreover, since traffic is probably not well-balanced between machines and subnets, some of them should be more carefully monitored, equivalent to use a greater prefix length, while others may be monitored with a coarse-grained view (smaller prefix length).

To counter this problem, aggregation can be guided by the nature of the events being monitored. For example, the traffic load can be aggregated in order to observe phenomena reaching a certain proportion of the entire traffic load or of Intrusion Detection System alerts.

Traffic Flow Table 1 Traffic flow example of a network nodes within 192.168.0.0/16 (Web and Mail services)

PORT	PROTO	KB	TIME	SOURCE	DEST
80	TCP	1491	2010-02-24 02:20:15	192.168.6.2	92.250.221.82
110	TCP	988	2010-02-24 02:20:19	192.168.8.2	92.250.223.87
443	TCP	902	2010-02-24 02:20:27	192.168.11.2	92.250.220.82
110	TCP	1513	2010-02-24 02:20:29	192.168.112.1	92.250.222.81
80	TCP	1205	2010-02-24 02:20:29	192.168.11.1	92.250.220.82
80	TCP	1491	2010-02-24 02:20:31	192.168.1.2	92.250.220.83
110	TCP	1467	2010-02-24 02:20:39	192.168.12.2	92.250.221.81
80	TCP	927	2010-02-24 02:20:39	192.168.12.2	92.250.220.82
443	TCP	1294	2010-02-24 02:20:39	192.168.11.1	92.250.223.82
110	TCP	940	2010-02-24 02:20:49	192.168.21.2	92.250.221.81
80	TCP	917	2010-02-24 02:20:49	192.168.23.1	92.250.220.82
443	TCP	460	2010-02-24 02:20:59	192.168.26.2	92.250.220.85

As set out in Section 2.1 on page 11, aggregation on a single dimension can be carried out using a tree structure, as proposed in [14, 18]. Spatial representation of a bi-dimensional space can be accomplished using a quad tree structure [41] where each internal node has exactly four children. Normally the space is recursively portioned into four quadrants or regions. A similar structure, an oct-tree, can be used to partition a three dimensional space. A general structure supporting M dimensions is a multidimensional tree (k-d Tree [47]) for k-dimensional space-partitioning. However, in our context, the dimensional space division is not known in advance and instead worked out on the fly as the tree is created.

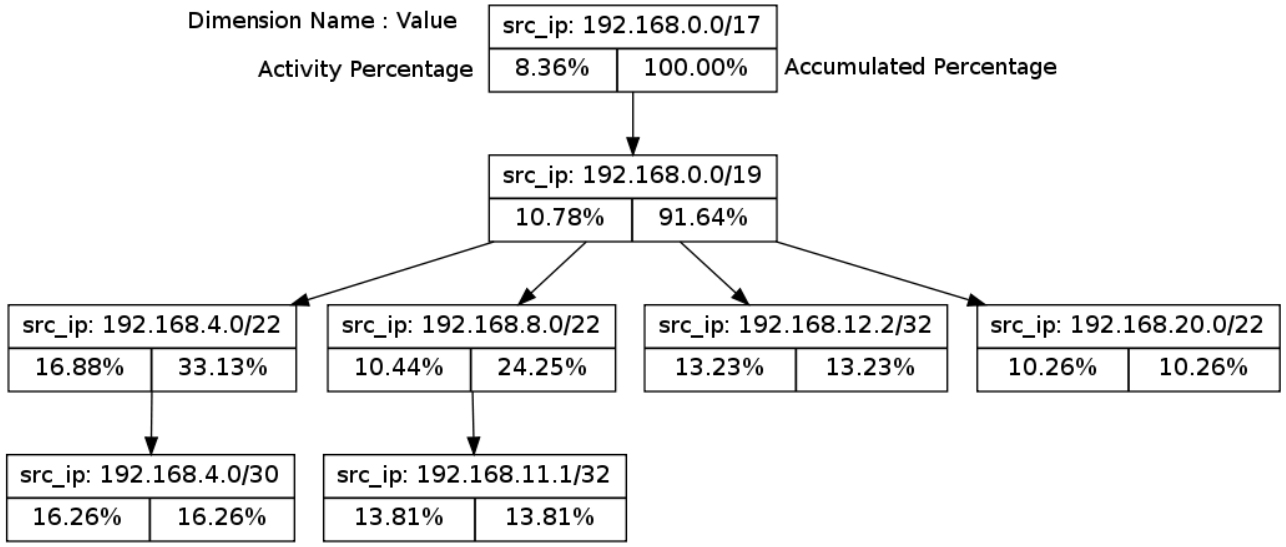


Figure 5.4: Single dimension tree (source IP addresses) based on Traffic Flow Table 1, activity volume: number of bytes, $\alpha = 10\%$

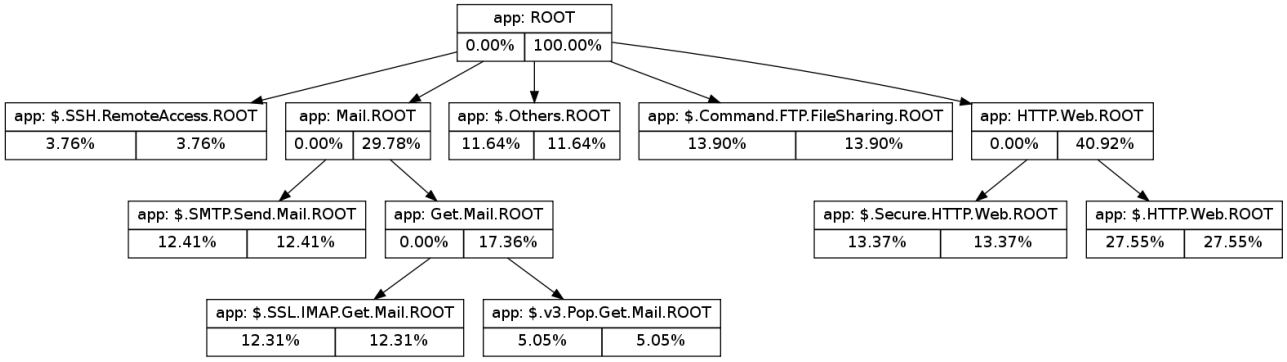


Figure 5.5: Single dimension tree (application) based on Traffic Flow Table 3, activity volume: number of bytes, $\alpha = 5\%$

We consider features where it is possible to derive underlying hierarchical relationships covering all potential data instances represented as nodes in a tree. Assuming two different data nodes, one is qualified as more specific or there is no relationship between them. Formally, the hierarchical relationship between two nodes n_i, n_j represents whether there is a path from the root of the tree to n_j that passes through n_i. In that case, n_i is more general than n_j. As for example, in Figure 5.5 the hierarchical relationship is given by the taxonomic classification of applications in Figure 5.6 on page 61.

Single Dimension

Spatial and temporal aggregation on a single dimension for traffic flows was proposed in Aguri [18] and Danak [14]. Temporal aggregation splits the dataset into fixed size time windows on which spatial aggregation is applied. We extend the notion of spatial aggregation to multiple and generic dimensions (features).

IP address aggregation is performed by extracting the traffic volume (bytes or packets) for the source or destination addresses. Aggregation is based on a tree structure following the common subnet hierarchy where each node represents an IP subnet or a single address. The total volume of traffic transmitted is decomposed into particular volumes expressed as absolute percentage values. Nodes with a proportion of traffic lower than a chosen threshold, α, are aggregated into their parents. An example of this from Traffic Flow Table 1 is given in Figure 5.4. In this small example, only nodes with more than 10% of the total traffic are kept. As shown, the root concentrates the global traffic of a /17 network. Each node contains the following information:

1. Dimension name and value (e.g. {app:ROOT}, {src_ip:0.0.0.0/0})
2. Percentage of aggregated activity (activity volume defined as *vol*) for the current node
3. Cumulated percentage of activity of the node and its subtree defined as *acc_vol*

Intuitively, a single-dimension tree represent a subset of the entire hierarchy (all the possible values) of a given feature, as illustrated in Figure 5.4.

Formally, an IP address single dimension tree of N nodes is [14]:

- A set of N nodes, where T = $\{n_0 \ \ldots n_N\}$ and $n_i =< prefix_i, prefix\_length_i, vol_i >$. IP subnets are decomposed using CIDR format [168]. $prefix_i$ and $prefix\_length_i$ are the prefix value and size of node n_i while vol_i is the entire traffic load associated to the IP addresses included in the subnet n_i.
- A parent-child relationship where a $child : T \rightarrow \mathcal{P}(T)$ returns a set of child nodes for a given node.

Single dimension aggregation can also be done for many other attributes such as protocol messages, port numbers, and spatial coordinates. Aggregation for TCP port numbers, using data from Traffic Flow Table 3, is set out in Figure 5.5. In this case, every node represents an application family or a specific application, defined by the taxonomic classification of Figure 5.6. We consider a dimension as a feature for which the values may be represented in a hierarchical tree with final values at the leaf nodes and group values as internal nodes.

This definition can be extended to any generic dimension as follows:

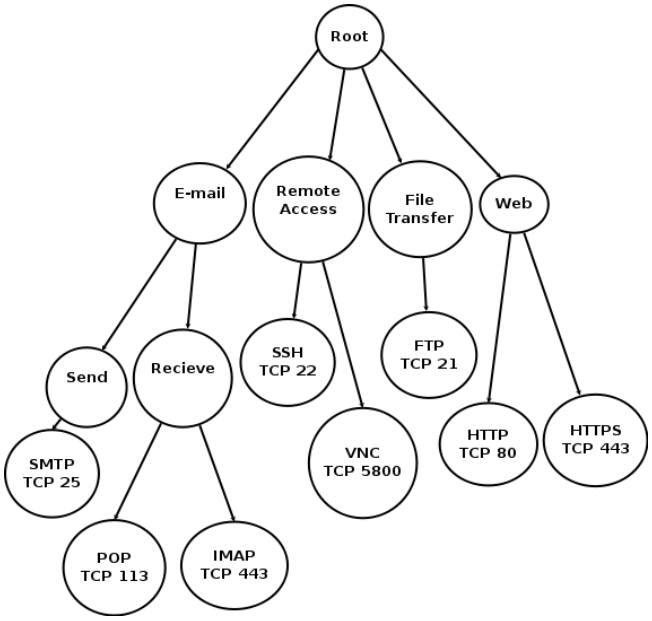


Figure 5.6: Example Application Taxonomy by TCP Port Numbers

- A set of N nodes, where T = $\{n_0 \ \ldots n_N\}$ and $n_i \ = < f_i, vol_i >$, where f_i is an associative array modeling a dimension from a given traffic flow. For example, in the case of application port $f_i \ = \ \{app : value_i\}$ where *app* is a label and $value_i$ is a string modelling a path in the taxonomic tree described in Figure 5.6. As mentioned above, this can be a full path to a leaf or an intermediate branch describing a subfamily of applications. For IP addresses, f_i is $\{prefix : \ prefix_i \ , \ prefix\_length : \ prefix\_length_i\}$.
- A parent-child relationship where a $child : \ T \rightarrow \mathcal{P}(T)$ returns a set of child nodes for a given node.

Single dimension aggregation has proven to be an effective technique for practical network analysis methods and anomalous network traffic detection [14, 18]. However, information from network traffic includes more than one dimension. Furthermore, the anomalies can be present in a combination of dimensions.

Port scanning consider the applications/service features, while IP scanning monitors IP addresses. Assuming a botnet carrying out port scanning from and to multiple IP addresses, all these features (source and destination IP addresses, destination ports) must be monitored to observe the attack activity globally. Consequently, predicting the single dimension or the combination of dimensions to monitor is hard.

Traffic Flow Table 2, on page 63, illustrates a simple example of several hosts performing a DDoS against a web server. IP-address-based aggregation (illustrated in Figure 5.7) on page 62 cannot clearly detect this but aggregation using TCP port is more successful. Another scenario is, Traffic Flow Table 3 on page 64. In this case, a reduced group of hosts is targeting several applications; this cannot be caught by aggregation on TCP ports. However, the multidimensional tree depicted in

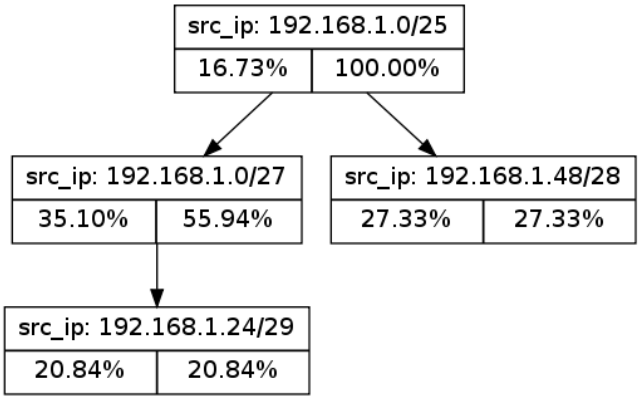


Figure 5.7: Single dimension tree (Source IP addresses) from Traffic Flow Table2

Figure 5.8 on page 63 is able to detect this behaviour. This highlights the fact that 20% of the HTTP traffic is due to web session initiations in the 192.168.0.0/20 subnet.

Multiple Dimensions

To extend the previous approach, we present an aggregation technique for monitoring network using multiple dimensions simultaneously (*e.g*: IP Address, Fully Qualified Domain Name FQDN, TCP ports, GPS Coordinates, etc). Multidimensional aggregation is performed by using a user-defined threshold α and an interval of η seconds to define the size of a time window. For each dimension, the data is assembled into a tree composed of nodes, including multiple dimensions, where only those having $acc\_vol > \alpha$ are kept.

In Figure 5.8, source and destination IP addresses and application from Traffic Flow Table 3 are aggregated.

Assuming a data instance that can be decomposed in many dimensions, a formal definition of a multidimensional tree of M dimensions and N nodes is as follows:

- A set of M associative arrays that model the M dimensions
- A set of N nodes, where T $= \{n_0 \dots n_N\}$ and $n_i = < \{f_{i_1} \dots f_{i_m}\}, vol_i >$, $f_{i_j} \in \{f_{i_1} \dots f_{i_m}\}$ is an associative array modelling the $j-th$ dimension according to the previous definition

  Thus, we can define $f_{i_1} = \{prefix : prefix_i , prefix\_length : prefix\_length_i\}$ for IP addresses. For the application/service level, $f_{i_3} = \{app : value_i\}$ where app is a fixed label and $value_i$ is the a string modelling a path in the taxonomic tree described in Figure 5.6.
- A parent-child relationship where a $child : T \rightarrow \mathcal{P}(T)$ returns a set of child nodes for a given node.

5.1.3 Aggregation Algorithms

To construct a single-dimensional tree, a leaf node is built after extracting relevant information (dimensions and values) to be inserted at the right place (or updating the node in the tree if it already

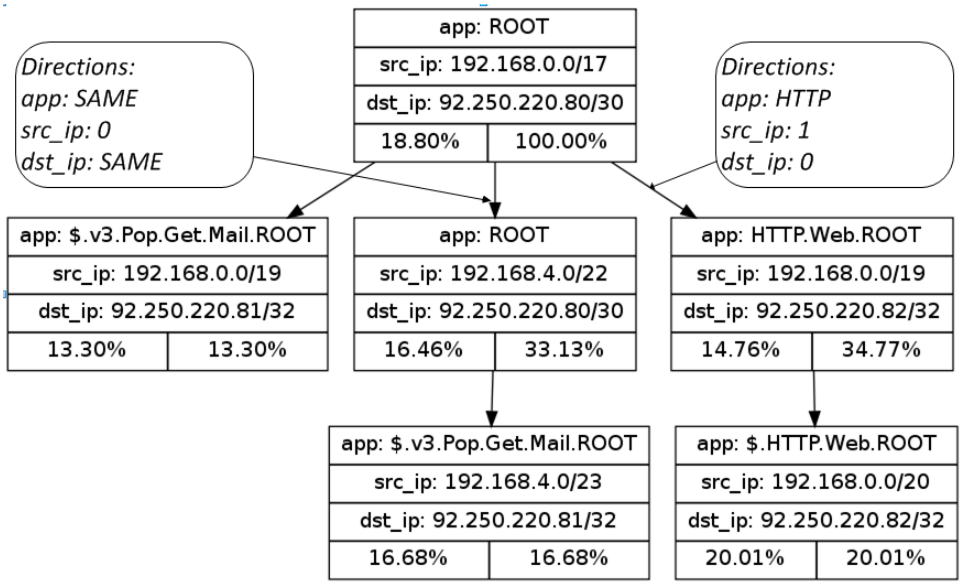


Figure 5.8: Multiple dimension aggregation based on Traffic Flow Table 3, $\alpha = 10\%$

Traffic Flow Table 2 Traffic Flow Table showing an example of a possible DDoS against a web server.

PORT	PROTO	KB	TIME	SOURCE	DEST
80	TCP	895	2010-02-24 02:20:59	192.168.1.17	92.250.220.82
80	TCP	47	2010-02-24 02:20:59	192.168.1.25	92.250.220.82
80	TCP	570	2010-02-24 02:20:59	192.168.1.45	92.250.220.82
80	TCP	952	2010-02-24 02:20:59	192.168.1.44	92.250.220.82
80	TCP	408	2010-02-24 02:20:59	192.168.1.61	92.250.220.82
80	TCP	609	2010-02-24 02:20:59	192.168.1.9	92.250.220.82
80	TCP	690	2010-02-24 02:20:59	192.168.1.15	92.250.220.82
80	TCP	88	2010-02-24 02:20:59	192.168.1.29	92.250.220.82
80	TCP	997	2010-02-24 02:20:59	192.168.1.27	92.250.220.82
80	TCP	650	2010-02-24 02:20:59	192.168.1.9	92.250.220.82
80	TCP	298	2010-02-24 02:20:59	192.168.1.46	92.250.220.82
80	TCP	502	2010-02-24 02:20:59	192.168.1.52	92.250.220.82

Traffic Flow Table 3 Traffic Flow Table example for a destination address being targeted by reduced group of hosts.

PORT	PROTO	KB	TIME	SOURCE	DEST
25	TCP	4660	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
443	TCP	2417	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
443	TCP	1945	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
21	TCP	4206	2010-02-24 02:20:59	192.168.1.1	92.250.220.82
80	TCP	4336	2010-02-24 02:20:59	192.168.1.3	92.250.220.82
110	TCP	2110	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
23	TCP	4257	2010-02-24 02:20:59	192.168.1.1	92.250.220.82
25	TCP	2005	2010-02-24 02:20:59	192.168.1.3	92.250.220.82
993	TCP	2434	2010-02-24 02:20:59	192.168.1.1	92.250.220.82
443	TCP	3270	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
993	TCP	4775	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
22	TCP	690	2010-02-24 02:20:59	192.168.1.3	92.250.220.82

exists). During then node insertion it may be necessary to create intermediate nodes, such as nodes representing intermediate IP subnets. If the node already exists, vol_i is updated accordingly, otherwise vol_i is initialized. To produce an outline subsequently, the tree is traversed post-order to aggregate nodes and to compute cumulative percentages (acc_vol). If the activity volume, vol_i, of a node n_i is less than the aggregation threshold, α, it is aggregated to its parent node, n_j. Thus, the node n_i is removed, vol_i is added to vol_j and all child nodes of n_i are attached to n_j. This allows the deletion of intermediate nodes which unlike their child nodes do not represent large activity volumes. Otherwise ($\text{vol}_i < \alpha$), the node is kept as it is.

During the post-order traversal, activity volumes, vol_i are computed as percentages. In fact, they are stored as absolute values during the tree construction since the total activity volume is not known. At the end, thanks a global counter, $\text{VOL} = \sum_i \text{vol}_i$, each vol_i is updated accordingly, `i.e.` vol_i/VOL.

Directions

Due to their hierarchical relationships, we suppose there is a strict order relation between values of the same dimension. To construct the multidimensional prefix tree structure, it is necessary to develop the concept of directions. Intuitively directions correspond to finding the correct path in the tree to attach a node or to navigate within the tree in order to access a given node.

Some dimensions are more likely to find or define a natural direction. For example, IP addresses have two directions, 0 or 1, modelling the next bit value. Taking the example of the subnet X.Y.Z.0/24, all IP addresses which the 25th bit is 1 (like X.Y.Z.128/25) will be placed on the left branch while every others , where this bit is 0, will be placed on the right branch.

Other dimensions such as UDP/TCP ports could simply be compared as integers but this is unlikely to be relevant, as services associated with a given category like mail do not necessarily use consecutive blocks of ports. In this case, a hierarchical classification, as shown in Figure 5.6, is required. In

our tool, the direction function may be customized by the user. This function is used to label the parent-child relationship in the multidimensional tree. To do this, we consider the longest common prefix which represents the most specific common ancestor of two nodes

Assuming $n_i = < \{f_{i_1} \dots f_{i_m}\}, \text{vol}_i >$ and $n_j = < \{f_{j_1} \dots f_{j_m}\}, \text{vol}_j >$, the longest common prefix is defined as:

$$\text{lcp}(n_i, n_j) =< \{f_{\text{lcp}_1} \cdots f_{\text{lcp}_m}\} \tag{5.1}$$

where f_{lcp_i} is the most specific common part for the ith dimension, which corresponds to the longest sequence of directions. Therefore, for IP address, this is a sequence of bits which is similar to the standard definition.

Considering T_M a M dimensional tree, defined in Section 5.1.2, a multi-dimensional direction is defined as a tuple of M directions (one for each dimension):

$$\begin{gathered} \forall n_i \in T_M, n_j \in T_M, n_k \in T_M, n_j \in \text{child}(n_i), n_k \in \text{child}(n_i) \\ n_k \neq n_j \iff \text{direction}(n_i, n_k) \neq \text{direction}(n_j, n_k) \iff \text{lcp}(n_i, n_k) \neq \text{lcp}(n_j, n_k) \end{gathered} \tag{5.2}$$

This corresponds to having only one child node per unique tuple of directions. Conceptually, a direction can be any set of tuples that allow child nodes to be distinguished. In practice, this usually matches some concrete value like the bit values for IP address or the application subclass for TCP ports. Directions using a 3-tuple (application, source and destination IP addresses) are illustrated in Figure 5.8 on page 63. The direction *SAME* was introduced to allow a child node to be different from its parent with a respect to a subset of dimension values while others remain the same. Preliminary tests show that this limits the number of internal nodes. However, at least one direction of the tuple must not be *SAME*.

Multidimensional Tree Construction

Intuitively, the tree is constructed by creating a root and then adding new nodes or updating existing nodes. Based on the directions, a pre-order traversal is made looking for a match to insert the node n_i (line 4 in Algorithm 1). Assuming that traversal stops at the node n_j, there are three different cases:

- if there is a perfect match (dimension values are the same), the activity volume is updated $vol_j \leftarrow vol_j + vol_i$. This is done in line 5 of the Algorithm 1.

- if n_i is a child n_j, a new child node is created by computing the directions tuple from n_j to n_i (there is not yet a node at this position, otherwise the traversal would have continued). This is done in line of the Algorithm 1. This situation is illustrated in Figure 5.9.

- otherwise, the traversal has followed the direction but ends at a node that is too specific. This happens, to avoid a scalability issue, because not all possible internal nodes are created: for instance X.Y.Z.0/24 may be a direct child of X.Y.0.0/16 on the IP address dimension. In this case, a new branching point (internal node) is created from $\text{lcp}(n_i, n_j)$. Thus n_i and n_j are the child nodes and two directions are then computed, $\text{direction}(n_i, \text{lcp}(n_i, n_j))$ and $direction(n_j, lcp(n_i, n_j))$. This is done at Line 15 of Algorithm 1.

Algorithm 1 Update Tree $insert\_node(tree, n_i)$

```
1: if tree is empty then
2:    tree.set_root(n_i)
3: else
4:    n_i ← tree.search_matching_node(n_i)
5:    if n_j matches all directions then
6:       update n_j volume {Perfect Match}
7:    else
8:       if n_j is n_i child then
9:          {Partial Match}
10:         for dim ∈ n_i do
11:            add n_i to n_j childs
12:         end for
13:         update tree set branching_point parent of n_j and n_i
14:      else
15:         branch ← empty node {New Branch case}
16:         for dim ∈ n_j do
17:            branch[dim] ← Direction_dim(n_j[dim], n_i[dim])
18:         end for
19:         update tree set branching_point parent of n_j and n_i
20:      end if
21:   end if
22: end if
```

Therefore, by construction, every node represents a subspace of its parent according to all dimensions.

Once the tree construction is finished at the end of the time window, aggregation takes place. Aggregation is done by traversing the tree in post order to find nodes having an activity volume $vol_i \leq \alpha$. These nodes are aggregated into their parents. While doing this, directions are discarded, since they are only needed during construction and because they may not satisfy equation (5.2) due to the deletion of internal nodes. Therefore, the k-dimensional space is not divided a priori and the space is not split at regular intervals. This allows an irregular granularity over the dimensions for efficient monitoring of the targeted events.

5.1.4 Online Tree Aggregation Strategies

Since memory consumption grows with the size of input data, and hence the size of the tree, we have developed strategies for maintaining the tree structure within a predefined size. Once the number of nodes is higher than MAX_NODES, one of the following strategies is triggered:

- Root aggregation: this strategy performs simple aggregation (as described in the previous section) from the root

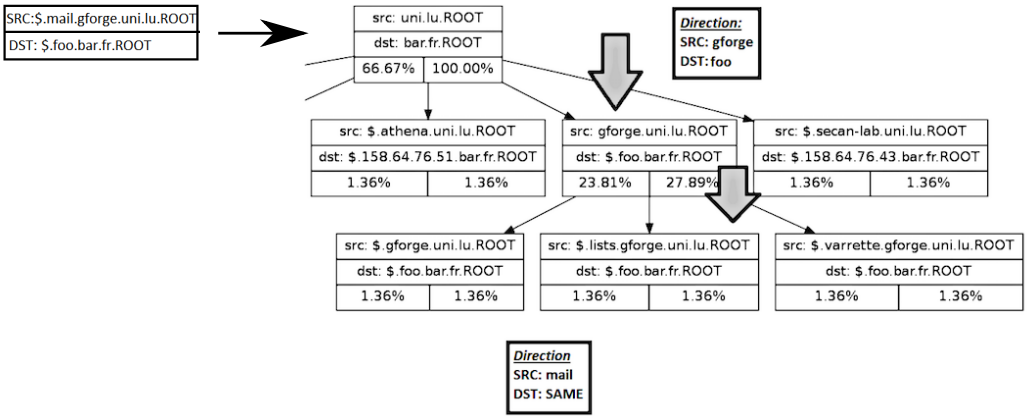


Figure 5.9: Example of insertion algorithm upon a partial match

- Least Recently Used (LRU): in this case the least recently used nodes are candidates for aggregation

These methods are described as online because they perform pre-aggregation before the end of the time window in order to save memory resources.

Root Aggregation

The root aggregation algorithm is shown in Algorithm 2. Every time the tree grows over the user defined threshold, the aggregation mechanism explained in Section 5.1.3 is triggered. While this is the simplest solution, it is not an efficient mechanism since all nodes below the threshold α are aggregated, even though the objective is just to reduce the number of nodes to MAX_NODES. This can be seen in line 5 of Algorithm 2. The cost of the aggregation mechanism triggered in line 6 is $O(n \times log(n))$ [18]. Its worst case complexity is $O(n^2 \times log(n))$ where n is the number of nodes. This is because worst case scenario represents triggering aggregation after every inserted node (For Loop in line 2).

Algorithm 2 Build Tree $T(dimensions, data)$

1: $tree \leftarrow empty\_tree$
2: **for** $d \in data$ **do**
3: $\quad node \leftarrow build\_node(d)$
4: $\quad tree.insert\_tree(node)$
5: $\quad$ **if** $tree.size > MAX\_NODES$ **then**
6: $\quad\quad tree.aggregate()$
7: $\quad$ **end if**
8: **end for**

LRU Aggregation

Algorithm 3 consists of triggering aggregation on the least recently used node only. The main idea behind this mechanism is to label every node with a timestamped tag that indicates the last time it was used. This is done in Algorithm 4 and used in line 4 of Algorithm 3 to update the timestamps of nodes which have been traversed when a node is inserted or updated (*i.e.*its parent and ancestor nodes).

Aggregation is performed only for the least recently used node. To achieve this, a min-max heap is employed [169] using the timestamped tag present in each node as a key. Algorithm 4 implements this mechanism extending Algorithm 1 functionality to label and maintain the heap structure used for retrieve the LRU node. This corresponds to line 14 of Algorithm 4. This operation is based on updating a min max heap, and has a complexity of $O(\log_2(n))$ [169]. Assuming n, the number of nodes, the average size of a tree branch is $\log(n)$. If every node in the branch has to be updated, an entry on the heap must be modified. Hence the subcost of that operation is $O(\log^2(n))$ and in the worst case $O(n \times \log(n))$ (a single branch of n nodes). To calculate this complexity, the Else branch in line 5 of Algorithm 4 is executed. During the last for-cycle (line 14), the list update_nodes will contain $\log(n)$ elements that corresponding to the path explored to place node in the tree.

After updating the last time used timestamp, the max heap is updated with a complexity of $O(\log_2(n))$. Hence the total complexity is $O(\log^2(n) + \log(n)) = O(\log^2(n)$.

Algorithm 3 Build Tree LRU $T(dimensions, data)$

```
1: tree ← empty_tree
2: for d ∈ data do
3:    node ← build_node(d)
4:    tree.update_lru_tree(node)
5:    if tree.size > MAX_NODES then
6:       candidate ← tree.lru_heap.top()
         Get the top element, candidate to be aggregated
7:       candidate.aggregate()
8:    end if
9: end for
```

In the *else* statement (line 5 of Algorithm 5), the timestamp is updated such that the leaf node (inserted or updated) has a timestamp older than its ancestors (reverse path is constructed during the insertion itself without any additional cost). This ensures that a leaf node is always retrieved and aggregated in line 7 of Algorithm 3. Otherwise, such an element could be an internal node and aggregation might lead to the removal of entire subtrees.

5.1.5 Summary

In this section we have introduced a multidimensional aggregation which is able to handle the multiple dimensions without requiring any predefined order. Moreover, the aggregation is guided by the nature of the events to being monitored. This leads to the space being split into partition of different sizes. This allows, distributed coordinated behaviours to be observed, unlike traditional approaches that rely

Algorithm 4 Update Tree $update\_lru\_tree(tree, node)$

```
1: if tree is empty then
2:    node.ltu ← now
3:    update_nodes ← [node]
4:    tree.set_root(node)
5: else
6:    path ← tree.insert_node(node)
7:    for n in reverse(path) do
8:       n.ltu ← now
9:       now ← now + 1
10:      update_nodes.append(n)
11:   end for
12: end if
13: for n in update_nodes do
14:   tree.ltu_heap.update(n)
15: end for
```

on a regular space division, which may not necessarily reflect the current distribution of the network activity. Based on formal definitions, this section highlights the data structure and algorithms used to perform aggregation using multiple dimensions related to network administration such as IP addresses, position-based applications, and DNS.

Since memory consumption grows with the size of input data, and hence the size of the tree, the root aggregation process cannot be applied in real time. Another limitation is its temporal complexity of $O(n^2 \times log(n))$ where n is the number of nodes. Therefore, we have introduced an improvement for this method based on an LRU cache algorithm to keep only recently updated nodes. This method keeps the data structure bounded to a certain number of nodes, limiting the memory consumption of the process, which has a temporal complexity of $O(log^2(n)$, where n is the total number of nodes.

In chapter 6 we explain in detail the proposed metric used in this thesis for data analytics using MAM as an enabler to enhance anomaly detection. In chapter 7 to validate this approach, we conduct experiments for detecting anomalies in three main scenarios: IP networks, DNS, and position-based applications. Since online aggregation involves sequentially reading data, a change in data ordering will produce different results. Inserting the same data instance at the end of the window will not have the same effect as at the beginning, since the tree may already have been aggregated many times before. The impact of the data ordering is evaluated in section 7.2.5.

5.2 Flow Management: Software Defined Networks

The scale and complexity of current compute and communication infrastructure is making them harder to manage for the network operators. The difficulty of managing communication infrastructure is a result of their size and the heterogeneous services they offer, as suggested in [7, 9]. One of the major challenges for the network operators is to meet the Quality of Service (QoS) requirements from a wide range of applications running on a shared infrastructure. Recent QoS-related functionality to add and

improve network design and management principals have gone through several iterations. A traditional way of meeting QoS requirements is by attributing classes to traffic using the IPv4 Type-of-Service (TOS) field or the IPv6 Traffic Class (TC) field, and then perform traffic engineering based on these classes. However ,such an approach lack flexibility, for example the need for dedicated routers, no single interface to update all router configurations, etc. and of limited use, as claimed in [170]. This lack of flexibility is one of the driving forces behind advanced protocols like GMPLS [171], but these have never been widely deployed despite a decade of development and standardization efforts. Additionally, as shown by the authors in [164], an application can have a poor performance in heterogeneous clusters, unless concurrent bandwidth usage is controlled and its consumption limited.

In recent times, Software Defined Networking offers the prospect of easing the network management tasks by decoupling the network control plane from the data forwarding plane. This separation of plane allows faster deployment and easier management of network functions [170,172]. OpenFlow [159], the de facto standard for SDN, has made the management of QoS related services easier by providing primitives to ensure flow-level QoS requirements. We leverage the features of SDN to propose an application-level flow management framework.

Problem Statement: How should application-based policies be applied to leverage network awareness in a SDN cloud-based environment?

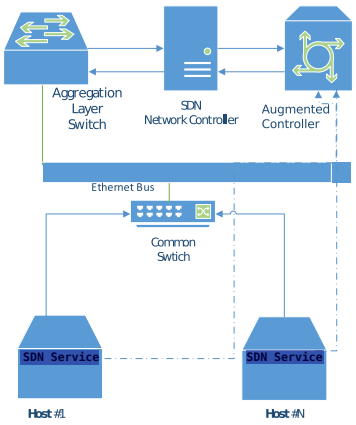


Figure 5.10: Example of the proposed network configuration

5.2.1 Overview

Our goal is to enable fine-grained traffic management policies based on application and the user requirements. Although current solutions aggregate traffic within static classes of service (VoIP, business data, etc.) with different priorities, our framework allows network-wide policy enforcement on users and applications. This is of a great importance in the current context of cloud data centres, which are slowly supplanting traditional single facilities. Nowadays, due to the advances in virtualization technologies, it is common to define the data centres as virtual computing centers with a high degree of flexibility. These cloud infrastructure, allow the execution of a user application at many locations and have the ability to migrate them to increase resource utilization and resilience. Hence, there is

a great need for flexibility in order to be able to monitor flows from these applications under such dynamic conditions [164]. This leads us to focus mainly on cloud infrastructure, although whereas our framework can be applied to any SDN-enabled environment.

We propose a framework to define and apply networking configuration rules while taking application-specific requirements into account. We present a technique that senses applications (such as HTTP browsers, MapReduce jobs, messaging clients, etc.) and implements a mechanism for translating application requirements to flow-based rules. We propose a SDN-based solution, which is able to aggregate flows associated with users and applications in near real time. While our framework can be used to enforce QoS policies, it can also be used to support any other type of policy, for example blocking a user. From a general point of view, our work enables the network to associate flows with the originating application's context. What follows, we use to phrase "application level" to refer to both users and applications on a host.

A first use case for our approach, is to apply QoS policies to a certain group of applications to grant them a higher priority in the network. For example, prioritizing the tasks of a MapReduce-based application such as Hadoop, can help to reduce the time taken to assemble relevant data. In addition, our approach supports QoS policies for optimizing the usage of the network, which can be done by dedicating bandwidth to certain applications. This can be achieved by through a prior interaction between the application and the network controller, in which the application is able to state its network requirements. Finally, our approach can be used for access policies, as illustrated in Figure 5.11, where a particular application is blocked by the network due to its access rights .

5.2.2 Architecture

Our proposed network configuration uses a specific underlying network architecture, which in the context of an SDN based network. Thie allow us to assume that an SDN Controller will be available with a northbound API. As shown in Figure 5.10, we have introduced one additional component to the typical SDN architecture, the Augmented Controller (AC). The typical SDN architecture is: at least one Open Flow compatible switch, an SDN network controller, and several end hosts. The Augmented Controller is directly connected to the SDN Network Controller and servers as an additional logic unit. Its main function is to act as an external processing unit in order to aggregate the flows per application, and provides an interface for implementing policies and rules at the application level.

The proposed network configuration has in two main components: an SDN Service running on each host and an Augmented Controller, which is a logical extension to an SDN controller. In the following, we describe the components in detail:

- SDN **Service** The SDN Service (SDNS) is a background process running at each end host. The SDNS has both northbound and southbound API. The north bound API allows the applications running on that host to communicate parameters to the AC. For example, a MapReduce job can tell the AC about different parameters of its shuffle phase; or a FTP transfer can tell the AC about the size of the file being transferred. In addition to providing the northbound API, the SDNS collects system wide statistics and allows system administrators to specify user and application level access and QoS policies. As a part of translating the policies, the SDNS associates the users and processes in the host with the network flow signatures originating from that host. The SDNS's southbound API communicates these associations with the Augmented

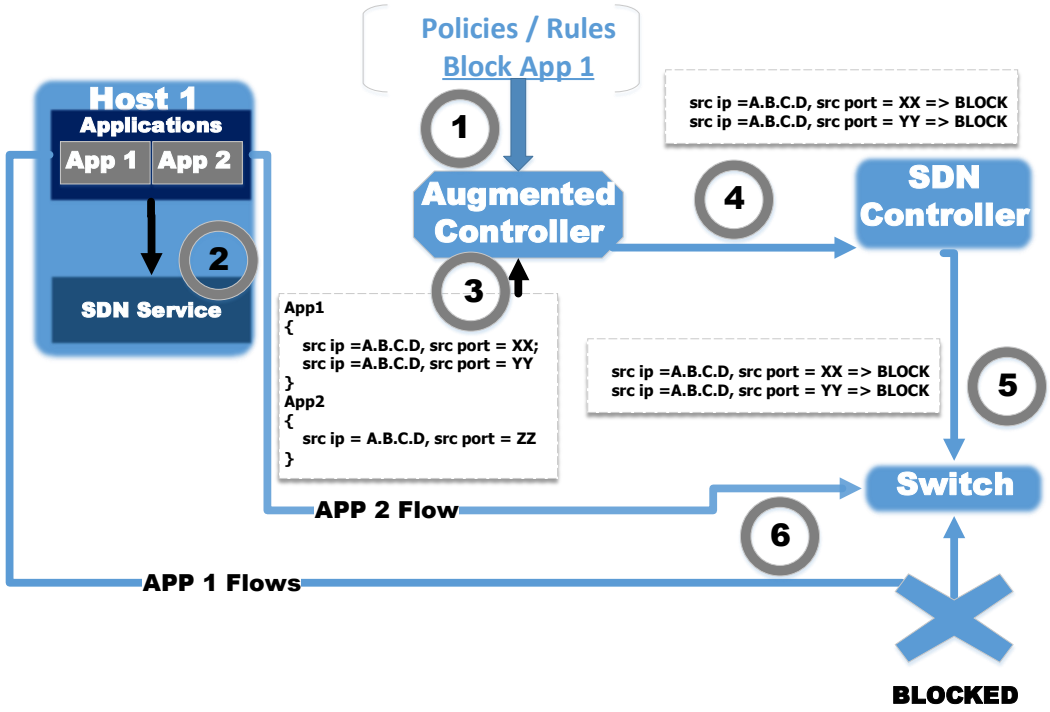


Figure 5.11: Example of Augmented Controller and SDN Service usage to enforce a blocking policy on a given application'sfFlows

Controller. The SDNS also communicates the different administrator set policies and application level requests to the AC as well. For example, a system administrator could tell the SDNS to limit the bandwidth usage of a group of users during some specific hours. In order to translate this policy, the SDNS identifies the processes run by the targeted user group and also identifies the flows originated from those processes. When any user from that group initiates a network transfer, e.g., an FTP file transfer, the SDNS exchanges parameters with the AC to ensure proper provisioning of resources in the network to enforce the administrator defined policies.

We have developed an in house protocol for the communication between the SDNS and AC. This protocol is based on HTTP and allows SDNS and AC to talk to each other in a RESTful manner.

- **Augmented Controller (AC)** The Augmented Controller is a logical extension to the SDN controller. The AC is informed by the SDNSs of different hosts about the association between users, processes and network flow signatures, and abount administrator/application specified policies. In order to handle the application specific requests, the AC can have custom application handler to handle request concerning a particular type of application. For example, the AC can have a batch processing job application that tries to optimize the network for data flow between the different stages of the batch jobs. The AC provisions appropriate resources in

the network to enforce these policies. In order to provision these resources, the AC translates the flow associations and the requests made by administrators/applications to OpenFlow rules with corresponding actions and time-outs. The SDN controller then installs these rules in the concerned OpenFlow switches to enforce the policies in the network.

5.2.3 Example

In this section, we illustrate the operation of AC and SDNS with an example. In Figure 5.11, *Host 1* is running two applications, *App 1* and *App 2*. A system administrator, *admin*, issues a command to the SDNS running in *Host 1* to block all traffic from *App 1* between midnight and noon. The SDNS will associate the network flow signatures originating at *Host 1* with the processes running in *Host 1*. The SDNS will also tell the AC about this association and also about the policy that all traffic from *App 1* should be blocked between midnight and noon. The AC uses this information to generate corresponding OpenFlow rules along with appropriate timeouts. On the request of the AC, the SDN controller installs these rules in the appropriate OpenFlow switch(es) to enforce the administrator-specified policy.

5.2.4 Operational Modes

The SDNS provides an internal interface that can be used by any application or by the OS to send and exchange custom messages with the Augmented Controller. The SDNS also collects system wide statistics and periodically sends them to the AC. This means that, even in the absence of application specific requirements, the AC gets an overview of the network usage requirements of the hosts is able to make some optimization decision based on that.

It should be noted that our design does not restrict how the AC is to be implemented. If the SDN controller has a northbound API to push rules, the AC can be implemented as a separate piece of software that uses the SDN controller's API. Otherwise, it can be implemented as a module within the SDN controller, thus providing a controller interface to interact with the SDNS.

To summarize, our framework has two modes of operation:

- **Passive mode:** In the passive mode, the applications do not interact directly with the SDN Service (SDNS). Which is responsible for associating flows with applications and users by monitoring the end-host systems. This corresponds to the example in Figure 5.11.

- **Active mode:** In the active mode, applications can send customized messages to the AC through the SDNS. Here, the SDNS acts as a middleware component and avoids the need for a tight coupling between the AC and the applications.

5.2.5 Implementation

In this section, we describe the functionalities of the AC and its association with the SDNS, together with implementation details (Section 5.2).

Augmented Controller REST API

Our work relies on the northbound API of Floodlight [173]. Therefore, actions are limited to the set of actions that floodlight can perform on the network configuration. The Augmented Controller is implemented as a service running on Linux, exposing a REST API offering the following functions:

- *Authentication & Authorization*: this function allows applications and users to authenticate themselves. This function also checks whether the authenticated applications and users have the proper privilege to set/request network resources on the AC. Normally, these permissions are set in advance by the network administrator, following the security policies and directives of a given network. For example, certain users or hosts might have a less constrained access to network resources, such as monitoring or network management applications. Authentication and authorization are a vital function to prevent malicious users from setting policies using the AC.

- *Flow Updates*: this function allows the SDNS to specify association between flows, applications and users. The AC maintains a database of these associations. We implemented the database using Redis[1], which provides high performance using in-memory storage, allowing speedy reactivity by the AC.

- *Set Policy*: this function allows to enable or disable a rule in the Augmented Controller. An example might be blocking or setting a higher priority for a given application running in a participating host.

- *Testing Capabilities*: this function allows checking what applications are supported by a custom application handler in the AC.

- *Push Message*: this function allows the SDNS to forward a custom message to the application handler in the AC at the request of a running application. To make use of this functionality, the application must be supported and have the appropriate rights granted.

SDN Service

The SDNS was implemented using *Python 2.7* and the standard Linux tool *Netstat*. It periodically polls the OS tables */proc/net/tcp*, */proc/net/udp*[2] and the *pid* to identify open sockets and processes. For every process, it is possible to correlate the inode from the system tables and the networking tables mentioned above. The SDNS only monitors processes which have subscribed. For instance, the administrator can configure the SDNS to monitor all processes or only applications that whish to obtain the benefit of the AC. For every process subscribed to the SDNS, the service communicates with the AC to update its flows. In addition, the service monitors every process termination and informs the AC that the associated flows can be discarded from the switch flow tables. This is an additional benefit of our approach which helps to limit the size of flow tables.

[1] http://redis.io/

[2] It can be easily extended to other protocols like ICMP but we consider that UDP and TCP are sufficiently representative for an initial implementation and testing

5.2.6 Summary and Considerations

In this section we have proposed an approach to leverage data analytic applications using an application level flow management method for SDN based cloud infrastructure. A major requirement of our proposal is to deploy the SDNS on as many end hosts as possible, so allowing the AC to make better decisions. However, as mentioned in Section 5.2, our focus in mainly on cloud environments. A cloud infrastructure is usually operated and administrated by a single authority. This allows us to make a rational assumption that the single authority will be able to easily configure the end-hosts to run the SDNS.

Another issue is type of service offered by the cloud. When a user deploys his own Virtual Machine (VM), the SDNS runs on the host system and cannot monitor processes and users within the VM. Therefore, the monitoring granularity of SDNS will be at the level of VM processes and their owners. This will allow the administrators to specify policies that are applied down only to the VM level. The cloud tenants could be encouraged to install the SDNS on their VMs in order to get the benefits out of a more optimized network. However, cloud providers may also offer other types of services (platform, software), which are executed in a completely controlled environment. Therefore, if a cloud provider is offering several big data analytics services like Hadoop or Storm[3], they can be monitored by the SDNS and so rules can be applied. For example, Hadoop flows (mainly batch jobs) would be placed on high throughput links while Storm flows (real-time processing) would be provided with low-delay links by configuring the network appropriately with SDN.

A final consideration is that, due to the interactions between the SDNS and the AC, flows may reach the switches before the corresponding rules have pushed to them. This is not a problem specific to our framework but is inherent in SDN.

[3]https://storm.incubator.apache.org/

Chapter 6

Data Analytics for Security

6.1 Overview

Monitoring is a fundamental part of network management. It is essential for checking the network activity and status, *e.g.* tracking abnormal occurrences or changes (attacks, configuration errors, failures, etc.). Several steps are required to do this. First, is collected from various sources and locations. Then, the data is stored before it can be directly visualized or analyzed by human expert to provide summarized information, such as alarms, to the network operational team. Aggregation has been widely adopted for network monitoring due to the recent large increase in data volumes (IP flow data, passive DNS database, etc.). It allows many interesting phenomena to be correlated, in particular in the security domain (distributed denial of service, botnet control, spam campaigns), which can only be monitored using the macroscopic view obtained from aggregation methods.

In Section 5.1 of the previous chapter we have introduced our multidimensional aggregation approach in Section 5.1. Each feature is associated with a dimensional space where data can be aggregated. For example, IP addresses can be aggregated by sub-networks and DNS names by subdomains. *MAM* is leveraged due to its two major advantages. Firstly, the data space is not divided a priori and the aggregation granularity varies over the space according to the events being monitored, *e.g.*reaching a user-defined volume-based threshold (number of bytes, alerts, etc), as shown in Chapter 5 on page 55. To illustrate, IP addresses are aggregated into subnets but the subnet size is not fixed manually and Multidimensional Aggregation Monitoring (MAM) can decide for example to aggregate part of data using a /16 prefix, while using /24 for another, or even keeping single host information (/32). The second advantage is that there are no order relationships between dimensions. When, for example, both domain names and IP addresses are monitored with a less flexible tool, the user may decide to aggregate data on addresses first and then on subdomains. With MAM, there is no need to make such a choice.

In this chapter we focus on the development of methodologies and metrics used to monitor and detect anomalies in three main scenarios:

- NetFlow Records: In Section 6.3, we study the volume of traffic aggregated by source and destination IP addresses simultaneously. In particular, we introduce a metric on NetFlow records and use it to leverage anomaly detection. Our detection method is based on the identification of anomalies in the volume of traffic in the network. TCP flood and Distributed Denial of Service

(DDoS) attacks can sometimes be detected by the large volumes of network activtiy that they produce.

- DNS Records: Malware and phishing websites have a relatively short life before they are reported and taken off-line [110]. In Section 6.4, we develop an approach to detect malicious websites and their domain name based on the time persistence of the match between IP addresses and names.
- Positioning-Based Records: Assuming a crowd-sourced application based on personal geographical location, a malicious user might report fake positions and inject them into the network to alter the estimated traffic conditions. To counter such attacks, Section 6.5 presents an approach for performing plausibility checks in an aggregated manner. In Section 6.5.1, we introduce our approach for the detection and recovery of data in location spoofing a scenario.

The temporal and spatial aggregation using MAM outputs a sequence of multidimensional trees, in which each node represents the activity volume of events during a certain interval of time. Our approach is to compare consecutive trees and define metrics that can select the relevant information from each event, leading to further evaluations. The comparison process is done pairwise between nodes, within a predefined window of consecutive trees representing aggregated events in time. In this chapter we develop a method for tree comparison inspired by the the notion of edit distance for strings. This has led us to create powerful metrics for comparing trees and their sub elements.

The subsection that follows comprehensively explores both the concept of distance and similarity between trees,and among instances of each dimension. In particular, metrics for evaluating similarity of DNS, NetFlow records and geospatial coordinates are explained in the detail. These metrics one of the most significant contributions of this work.

6.2 Distance Functions

6.2.1 Introduction

Trees are perhaps one of the most widely used, diverse and well-studied data structures in computer science. In this section, our goal is to introduce a formal notion of similarity or distance between multidimensional trees. The importance of having a distance function as a metric is crucial to the further use of trees for monitoring purposes. The Levenshtein distance [174, 175] is usually refereed to as the edit distance between two strings, and is defined as the minimum number of operations to transform one string into the other by deletion, insertion or substitution. As an example, the strings “sos” and “sbs” have a distance of 1 because one may be transformed into the other by performing one substitution of the character “o” into “b”. This notion of distance was initially defined for strings where every operation has a single cost of 1. For multidimensional trees we propose a distance function inspired in the Levenshtein distance. One simple approach is to encode the trees as strings using a traversal algorithm and compare them, but this is sometimes not accurate, since the string representation is not unique. Therefore, in our approach we calculate the required number of transformations to be performed on a multidimensional tree in the same manner as the Levenshtein distance. As a node is a more complex structure than a character, a single transformation step may require discrete operations on more than one of the node’s dimensions. Consequently, while each operation is still assigned a cost of 1, the cost of the whole step may be greater than 1.

6.2.2 Multidimensional Distance Function

In Section 5.1 on page 55 we introduced a tree based representation of network traffic for a given period of time. In this section construct a metric that reflects as generally as possible the changes in network traffic patterns. Additionally, since the multidimensional tree construction mechanism is sensitive to data order (as aggregation occurs on the fly), we introduce the distance metric as a tool to check that data order has no significant impact on our techniques and further experiments. In particular, we measure the degree of similarity for trees built from the same data but received in a different order.

According to the formal definition of a M-multidimensional tree, a node is defined as $n_i = < \{f_{i_1} \dots f_{i_m}\}, \text{vol}_i >$. Assuming a distance function for each dimension k as $\text{Distance}_k(f_{i_k}, f_{j_k})$, global metrics can be calculated such as:

- Average: $\frac{\sum_{k=1}^{M} \text{Distance}_k(f_{1_k}, f_{2_k})}{m}$ where m is the number of dimensions used.
- Sum: $\sum_{k=1}^{M} \text{Distance}_k(f_{1_k}, f_{2_k})$
- Min: $min_{k=1}^{M} \text{Distance}_k(f_{1_k}, f_{2_k})$
- Max: $max_{k=1}^{M} \text{Distance}_k(f_{1_k}, f_{2_k})$

We used the average as a global distance function to calculate the cost of an operation (insertion, removal or modification) since we seek to include the cost of every distance per dimension in a normalized manner. However, the remaining metrics can still be representative in different situations, for example, to calculate a minimal cost for operations that transform one tree into another.

Thus, the distance function must be defined for each dimension. Tree distance can be computed by calculating a distance matrix as the result of the pair wise comparison between the nodes of the trees being compared. We then calculate the average over the minimal distance of every row and column.

Given T_a having n nodes and T_b having m nodes with both $n, m \in \aleph$. The distance between T_a and T_b can be calculated comparing the trees pair-wise and looking for the most similar nodes at each step (using the min function). The following definition details one possible distance between trees:

$$DistanceMatrix(T_a, T_b) = \begin{pmatrix} Dist(T_{a_1},\ T_{b_1}) & Dist(T_{a_1},\ T_{b_2}) & \cdots & Dist(T_{a_1},\ T_{b_n}) \\ Dist(T_{a_2},\ T_{b_1}) & Dist(T_{a_2},\ T_{b_2}) & \cdots & Dist(T_{a_2},\ T_{b_n}) \\ \vdots & \vdots & \ddots & \vdots \\ Dist(T_{a_m},\ T_{b_1}) & Dist(T_{a_m},\ T_{b_2}) & \cdots & Dist(T_{a_m},\ T_{b_n}) \end{pmatrix}$$

$$Distance(T_a,\ T_b) = \frac{\frac{\sum_{j=1}^{m} \min\limits_{i=1\dots n} (Dist(T_{a_j}, T_{b_i}))}{m} + \frac{\sum_{j=1}^{n} \min\limits_{i=1\dots n} (Dist(T_{a_i}, T_{b_j}))}{n}}{}$$

6.3 NetFlow-based Trees

6.3.1 Problem Description

Network monitoring at Internet Service Provider (ISP) level is a critical task known to be helpful in network management and security context [176]. The data collected in network may consist of full packet captures on a network or NetFlow [75] records. An individual flow is an aggregated view of a stream of packets between a source and a destination characterized by IP address, protocol and port when necessary. Since nowadays, the volume of such information is growing rapidly, the scalability is a challenge. An ISP can expect to attend not less than 60,000 flows per second.

Aggregation allows the process of monitoring and analysis to be speed up, as pointed out by authors in [16, 18, 76]. Building on this finding, we propose an approach for modelling the network activity using more than one feature, including applications and IP Addresses, as illustrated in Figure 6.1.

In particular, the following problems are addressed in this section:

- Studying the IP traffic of a network through multidimensional aggregated data structures.
- Providing through aggregation an approach which increases scalability by compressing data and by creating summarized outputs which are smaller and easier to analyze.
- Detecting major events indicating changes in the volume of traffic in a network. Such events are often related to the occurrence of attacks such as DDoS and TCP Flood [177].

6.3.2 NetFlow-based Metric

In this section we describe a two dimensional representation of the NetFlow records. To do so, source and destination address are represented as: $IP_{\text{src}} = \{\text{prefix} : \text{prefix}_{\text{src}}, \text{prefix\_length} : \text{prefix\_length}_{\text{src}}\}$, and $IP_{dst} = \{\text{prefix} : \text{prefix}_{\text{dst}}, \text{prefix\_length} : \text{prefix\_length}_{\text{dst}}\}$.

Assuming a sequence of N trees $S = \{T_1, T_2, \ldots, T_N\}$, each node $n \in Node$ in a tree is defined by the following elements:

- $n^{\text{acc\_vol}}$ is the accumulated proportion of activity volume as defined in *MAM*,
- $n^{\text{prefix}_{\text{src}}}$ and $n^{\text{prefix\_len}_{\text{src}}}$ are respectively the IP_{src} prefix and its size,
- $n^{\text{prefix}_{\text{dst}}}$ and $n^{\text{prefix\_len}_{\text{dst}}}$ are respectively the IP_{dst} prefix and its size,

As is required by MAM, a distance function should be provided per dimension. In our case, we define it based on the longest common prefix between two IP addresses or on the longest common path in the taxonomy tree for ports. A notion of similarity based on the hierarchical structure of IP addressing (subnets), as suggested by authors in [14] as follows:

$$Dist(x, y) = \alpha \times IP\_sim(x_{src}, y_{src}) + \beta \times IP\_sim(x_{dst}, y_{dst}) + \gamma \times Vol(x_{vol}, y_{vol})$$

To avoid giving preference to any parameter, $\alpha = \beta = \gamma = \frac{1}{3}$ in the precedent equation.

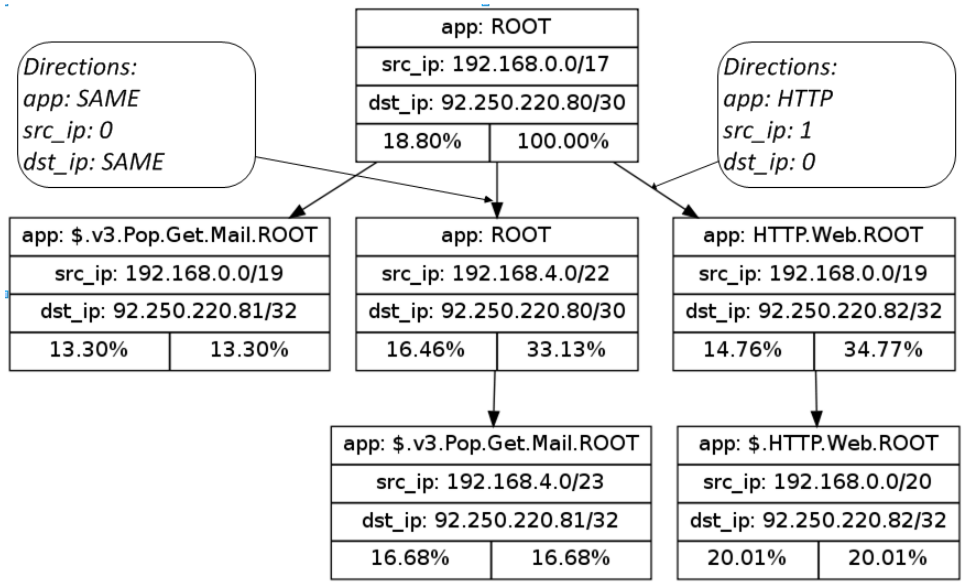


Figure 6.1: Multiple dimension aggregation based on Traffic Flow Table 6.2, $\alpha = 10\%$

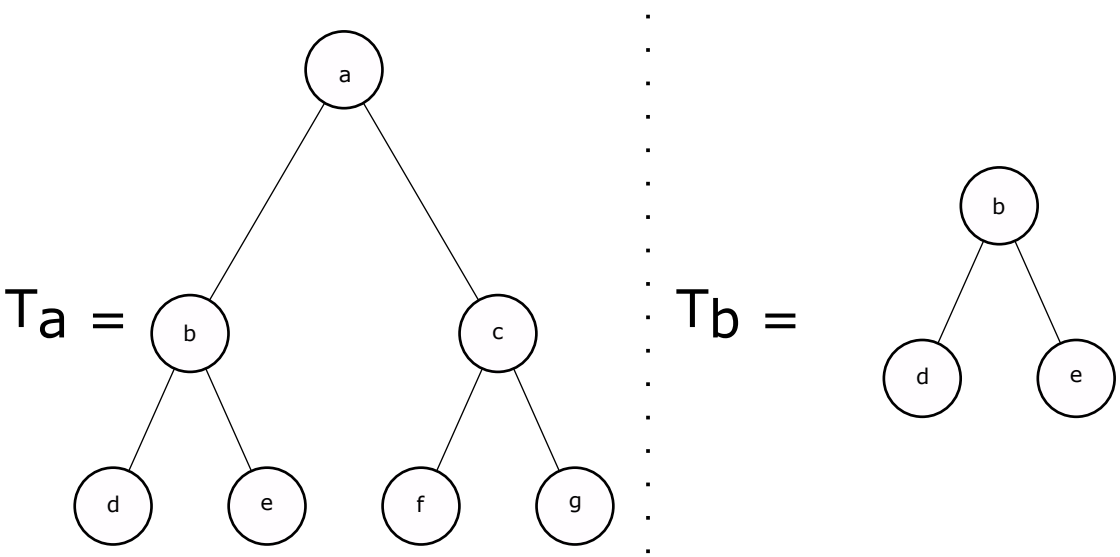
PORT	PROTO	KB	TIME	SOURCE	DEST
25	TCP	4660	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
443	TCP	2417	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
443	TCP	1945	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
21	TCP	4206	2010-02-24 02:20:59	192.168.1.1	92.250.220.82
80	TCP	4336	2010-02-24 02:20:59	192.168.1.3	92.250.220.82
110	TCP	2110	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
23	TCP	4257	2010-02-24 02:20:59	192.168.1.1	92.250.220.82
25	TCP	2005	2010-02-24 02:20:59	192.168.1.3	92.250.220.82
993	TCP	2434	2010-02-24 02:20:59	192.168.1.1	92.250.220.82
443	TCP	3270	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
993	TCP	4775	2010-02-24 02:20:59	192.168.1.2	92.250.220.82
22	TCP	690	2010-02-24 02:20:59	192.168.1.3	92.250.220.82

Figure 6.2: Traffic Flow Table example for a destination address being targeted by reduced group of host.

$$IP\_sim : Node \times Node \rightarrow \Re \in [0,1]]$$

$$IP\_sim(n1,n2) = \begin{cases} 1 - \dfrac{|n1^{\text{prefix\_len}} - n2^{\text{prefix\_len}}|}{32} & \text{if } n_1^{\text{address}} \subset n_2^{\text{address}} \\ 1 - \dfrac{|n2^{\text{prefix\_len}} - n1^{\text{prefix\_len}}|}{32} & \text{if } n_2^{\text{address}} \subset n_1^{\text{address}} \\ 0 & \text{else} \end{cases}$$

$$Vol : \Re \times \Re \rightarrow [0; 1]$$
$$Vol(n_1, n_2) = exp(-\frac{|n_1 - n_2|^2}{\phi})$$



Node	Source IP Address + Prefix Length	Destination IP Address + Prefix Length	Node Volume	Accumulated Volume
a	0.0.0.0/0	10.0.0.0/0	5%	95%
b	0.0.0.0/0	10.2.0.0/16	15%	60%
c	0.0.0.0/0	10.5.0.0/16	5%	35%
d	0.0.0.0/0	10.2.0.0/24	10%	20%
e	0.0.0.0/0	10.2.1.0/24	5%	15%
f	0.0.0.0/0	10.5.0.0/16	15%	15%
g	10.2.1.1/32	10.5.0.0/24	15%	15%

Figure 6.3: Example of an aggregate tree ($\alpha = 5\%$) comparison using Edit Distance, T_a and T_b. (Nodes are listed in the table above)

For example to calculate the distance for T_a and T_b as set out in Figure 6.3, the steps are:

$$\text{DistanceMatrix}(T_a, T_b) = \begin{pmatrix} Dist(a,\ b) & Dist(b,\ b) & Dist(c, b) \cdots & Dist(g, b) \\ Dist(a,\ d) & Dist(b,\ d) & Dist(c,\ d) \cdots & Dist(g,\ d) \\ Dist(a,\ c) & Dist(b,\ c) & Dist(c, c) \cdots & Dist(g, c) \end{pmatrix}$$

since T_b is contained on the left side of T_a. Dist(b, b) = Dist(d, d) = Dist(e, e) = 0. Dist(a, b) = Dist(a, d) = d

The remaining entries of the matrix account for:

$$\text{DistanceMatrix}(T_a, T_b) = \begin{pmatrix} d & 0 & 0.25 \cdots & 0.25 \\ d & 0.75 & 0.75 \cdots & 0 \\ 0.35 & 0.35 & 0 \cdots & d \end{pmatrix}$$

6.3.3 Summary and Conclusion

In this section we have summarized a part of the contribution made by [187]. Network monitoring is of paramount importance for enforcing security in networks, as it can lead to anomaly detection and the identification of attacks. Our approach in this section has been to define a multidimensional representation of source and destination addresses present in the NetFlow records. This definition can be extended to include other dimensions such as application type and port number. An evaluation of our approach is appears in Section 7.2, where we investigate the performance of the multidimensional IP-based aggregation, and leverage detection of anomalies methods by watching changes in distance. Also since, as mentioned in Section 5.1.5 on page 68, the order of data might impact the tree-based data structure, the impact of data order on aggregation is explored in Section 7.2.

6.4 DNS

6.4.1 Problem Description

The Domain Name Service (DNS) is a critical system for the Internet, which maps human-readable names such as *www.uni.lu* to IP addresses. It allows resource to be reallocated or distributed dynamically and transparently with little effort. However, these properties make it a good target and tool for attackers. For example, by disrupting a DNS server with a denial of service attack, DNS clients can be prevented from locating services. DNS spoofing attacks are even worse, because their objective is to falsify the IP address mapping of a domain name and so transparently redirect users to malicious machines [103]. DNS traffic monitoring can help to identify these two major threats, FastFlux and phishing, and can also provide insights into other malicious activities [24–26]. Firstly, intuition suggest that domains used in malicious activity are intuitively requested over short periods of time (during an attack campaign, for example, and/or before being disrupted by defensive measures). Secondly, the IP addresses associated with a malicious domain can be highly variable, FastFlux being the best example. These features are leveraged in [104, 105].

Aggregation helps to increase the visibility of the match between domain name and IP addresses, since one legitimate domain name (*e.g.* www.wikipedia.org) might be associated with multiple IP addresses. Therefore, at an aggregated view makes it possible to observe a domain name or subodmain associated to a group of IP addresses as sub networks. This type of views allow identifying changes in distributed events as required to monitor FastFlux attacks to be identified avoiding bias due to local phenomena. As both DNS and IP space follow ahierarchical organisation, we propose using MAM (Multidimensional Aggregated Monitoring) [187] to handle the IP and DNS spaces simultaneously.

In particular, the following problems are addressed in this section:

- Studying the DNS traffic behaviour using a Passive DNS Database, to identify the temporary behaviour of malicious domains in terms of the domain name and its IP address associations.

- Discovering malicious domains for detection purposes, based on macroscopic (long-term) analysis of the matches between IP addresses and domain names.
- Discovering through microscopic analysis (local view) which domains and IP addresses are responsible for these changes.
- Developing an efficient aggregation techniques which, given the large volume of data available in this scenario, helps out to reduce the scale of the problem minimizing information.

6.4.2 DNS representation using Aggregated Trees

We are interested in the mapping between domain names and IP addresses, the two monitored dimensions. To represent this data, we use a multidimensional tree, in which each node contains the following information:

1. IP address and subnet
2. domain name
3. percentage of aggregated activity (activity volume, defined as *vol*) for the current node
4. cumulative percentage of activity of the node and its sub-tree defined as *acc_vol*

An IP address is defined as prefix, a prefix length and suffix, similar to the definition in Section 6.3. As described in Section 3.3, the DNS follows a hierarchical structure, where the level domains represent intermediate nodes. We consider the root level (ROOT) and a final level denoted by $. This allows differentiation between a node corresponding to all `*.uni.lu` names like `www.uni.lu` or `snt.uni.lu` and a direct request to `uni.lu`, which is represented by `$.uni.lu`. Each unique mapping between a domain name and an IP address is considered to have a volume $vol = 1$. Hence, it only represents the existence of such a mapping and is independent of the number of times the domain has been requested.

For aggregation purposes, MAM creates leaf nodes representing unique data instances (/32 IP address and a domain like $.*.ROOT) and inserts them iteratively by creating intermediate nodes if necessary. Only nodes representing a minimum volume, $vol > \alpha$, are retained; otherwise they are aggregated into their parents by summing *vol* meanwhile.

Data is also aggregated over time by specifying η, a time window size in seconds. All events within a single window are aggregated using the space aggregation scheme described above. This is an essential feature of MAM since our goal is to study the dynamic of DNS mapping over time.

We analyze time series of the constructed by MAM. As illustrated in Figure 6.4, the trees are constructed using tuples $< timestamp, , IP\ address >$. Such tuples serve as input to MAM, with the IP address being the address matching a domain name in a DNS.

6.4.3 Steadiness metrics

DNS names are usually associated with a group of IPv4 Addresses, and this association tends to be stable over time for benign domains. However, inconstant mappings between domains and IP subnets are suspect [26, 106]. This concept is extended to the trees obtained from MAM.

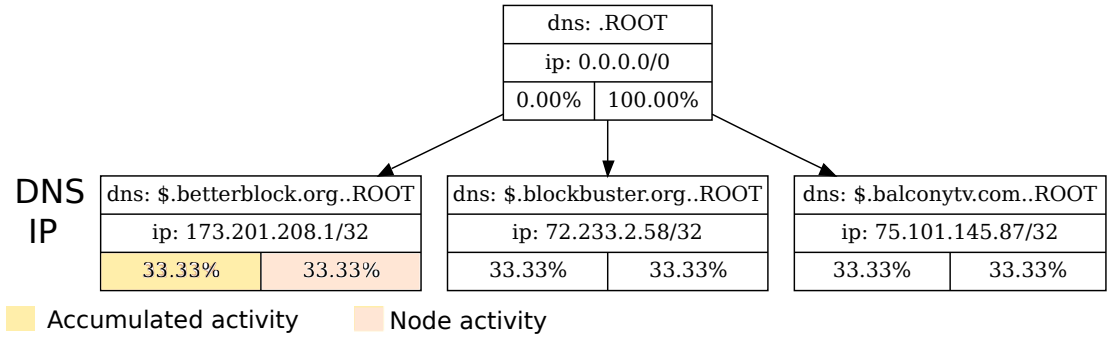


Figure 6.4: Sample tree for IP Address and FQDN

We propose a metric to characterize the persistence over time for a match between an IP address and a domain name (steadiness). While this can be taken as a measure of the local steadiness of a node in a tree, it is also possible to evaluate the global persistence of all nodes over a sequence of trees, and thus assess the general steadiness of the mapping between IP and DNS spaces.

Assuming a sequence of N trees $S = \{T_1, T_2, \ldots, T_N\}$, each node $n \in Node$ in a tree is defined by the following elements:

- n_{acc_vol} is the cumulated proportion of activity volume as defined in MAM,
- n_{prefix} and n_{prefix_len} are respectively the IP prefix and its size,
- n_{dns} is the set of level names of the Fully Qualified Domain Name (FQDN)(for example $\{www, uni, lu\}$ for `www.uni.lu`)

When evaluating the steadiness of a node $n1$ in a tree T_i, $stead(n1)$, the objective is to check for the presence of $n1$ in the previous tree T_{i-1}. However, while aggregation improves scalability, exact information is lost and precise correspondence between nodes is rare. Therefore it is necessary to find the most similar node to $n1$ in the tree T_{i-1}. The notion of similarity is based on the hierarchical structure of IP addressing (subnets) and DNS naming space (subdomains). Assuming two nodes, $n1$ and $n2$, the similarity is positive only if one of the following condition is verified:

- $n1$ and $n2$ represent the same domain and IP subnet
- $n1$ is a child of $n2$ for at least one dimension (DNS or IP) and is the same as $n2$ for the remaining ones
- $n2$ is a child of $n1$ for at least one dimension (DNS or IP) and is the same as $n1$ for the remaining ones

In brief, nodes with locations that are disjoint in at least one space will have a similarity of 0, i.e. none of them can be a child of the the other in a MAM tree. For example, 192.168.10.0/24 and 192.168.20.0/24 represent disjoint portions of the IP space; and `www.uni.lu` and `snt.uni.lu` are disjoint in the DNS space whereas `www.uni.lu` is a child (subdomain) of `uni.lu`.

Based on this definition, the built tree allows easy location of the most similar node to $n1$, denoted as $most\_sim(n1)$, since parent-child relationships are preserved. Assuming a current node $n0$ having a strictly positive similarity with $n1$, *i.e.* $sim(n1, n_0) > 0$, the algorithm considers each child node of n_0 in the tree to find the node $n0'$ such that $sim(n1, n_0') > sim(n1, n_0)$ and $sim(n1, n0') = max_{n2 \in child(n0)}\ sim(n1, n2)$. The goal is to find the child node with the maximum similarity. If no child guarantees $sim(n1, n0') > sim(n1, n0)$, the algorithm stops because this means that all child nodes represent divergent spaces relative to $n1$. The most similar node is thus $n0$, the current node. The algorithm is initialized by starting from the root.

The similarity between two nodes, $n1$ and $n2$, is calculated over the different features and the activity volumes as follows:

$$\begin{aligned} sim : Node \times Node \rightarrow [0;1] \\ sim(n1, n2) = \ & \alpha \times IP\_sim(n1, n2) \\ & +\beta \times DNS\_sim(n1, n2) \\ & +\gamma \times vol\_sim(n1, n2) \end{aligned} \tag{6.1}$$

subject to:

$$\begin{aligned} & IP\_sim : Node \times Node \rightarrow [0;1] \\ & IP\_sim(n1, n2) = 1 - \frac{|n1_{prefix\_len} - n2_{prefix\_len}|}{32} \end{aligned} \tag{6.2}$$

$$\begin{aligned} & DNS\_sim : Node \times Node \rightarrow [0;1] \\ & DNS\_sim(n1, n2) = \frac{|n1_{dns} \cap n2.dns|}{|n1_{dns} \cup n2_{dns}|} \end{aligned} \tag{6.3}$$

$$\begin{aligned} & vol\_sim : Node \times Node \rightarrow [0;1] \\ & vol\_sim(n1, n2) = 1 - |n1_{acc\_vol} - n2_{acc\_vol}| \end{aligned} \tag{6.4}$$

α, β and γ can be fine tuned to give preference to a certain feature based on the user knowledge ($\alpha + \beta + \gamma = 1$).

Being inspired by [178, 187], equation (6.2) considers the size difference between two IP subnets with respect to the number of IP addresses and normalized with respect to the total number, 2^{32}, of IPv4 addresses. This yields a value bounded between 0 and 1, which is subtracted from one to obtain a similarity value. Obviously, this is only meaningful only if one subnet is a subnet of the others, which, in our case, is guaranteed by searching for the most similar node. Therefore, the more similar the prefix size, higher the similarity.

The second part of the metric focuses on the variation of DNS names in Equation (6.3) (*e.g.*, `www.google.com` vs. `google.com`) by splitting the DNS name into a set of labels using the dot as a separator (`www`, `google` and `com` for `www.google.com`). Based on these sets, the Jaccard index is computed (the ratio between the number of elements in the intersection and the union) to evaluate the similarity between pairs of sets. Again, searching for the most similar node guarantees that compared domains are the same or, that one is a subdomain of the other. Thus, the similarity is proportional to the number of shared level names.

Therefore, the similarity is a value bounded between 0 and 1. The steadiness of the node $n1$ in T_i is evaluated regarding the similarity of $n1$ to the most similar node in the previous tree, T_{i-1}, and

the steadiness of the latter. This smoothing allows the steadiness of the node $n1$ to be dependent not only on the previous time window but also the earlier ones.

$$\begin{aligned} &n1 \in T_i, n2 \in T_{i-1}, n2 = most\_sim(n1) \\ &stead(n1) = sim(n1, n2) + \mu \times stead(n2) \end{aligned} \tag{6.5}$$

To avoid giving preference to any parameter, $\alpha = \beta = \gamma = \mu = 1/4$ in Equation 6.1 and Equation 6.5

Although the previous equation is helpful in calculating the individual steadiness of a node (local steadiness), the global persistence (macroscopic steadiness) of the tree T_i is the average steadiness of all nodes in T_i

$$pers(T_i) = \frac{\sum_{n \in T_i} stead(n)}{|\{n \in T_i\}|} \tag{6.6}$$

6.4.4 Conclusion

In this section we summarized the work which researches on the mapping between IP addresses and domain names. Thanks to the aggregation scheme, the volume of data to analyze is reduced and distributed behaviours can be identified more easily, which is helpful in the security context. In this section we have introduced a steadiness metric that can be applied to study the persistence over time of the association between domain names and IP addresses. This metrics are further employed by comparing consecutive time windows an observing the changes in the associations, to identify the behaviour of anomalous websites we propose a threshold-based detection system. It is useful in tracking entire harmful IP subnets or domain names while keeping the computational overhead low. We have defined a specific metric to study the steadiness of time series of trees produced by MAM. The evaluation on real data has proven its efficiency for detecting global abnormal changes (macroscopic steadiness) and for tracking specific domains and IP subnets which are responsible for them (local steadiness). In Section 7.3 we conduct a series of experiments to validate that our approach is helpful for detecting malicious domains. Also, the scalability has also been assessed, since our method leverages the analysis of aggregated data reducing the post processing time requirements.

6.5 Positioning-Based Applications: Steadiness and Recovery

6.5.1 Problem Description

In the context of a crowd-source position-based application used for vehicle navigation and traffic estimation there is no strong restriction on users, everybody is basically identified with pseudonyms and able to join the application[1]. Thus, an attacker can inject erroneous data, as shown in Figure 3.9, using multiple vehicle identities (Sibyl attack) using a standard computer which is considered as the most realistic and threatening scenario [123]. Even without a success rate of 100%, an attacker may cause unforeseen and catastrophic consequences to location based services (creating traffic congestion

[1] http://maps.google.com

and so indirectly rerouting people to specific locations, indicating false collisions which may provoke unnecessary braking, etc).

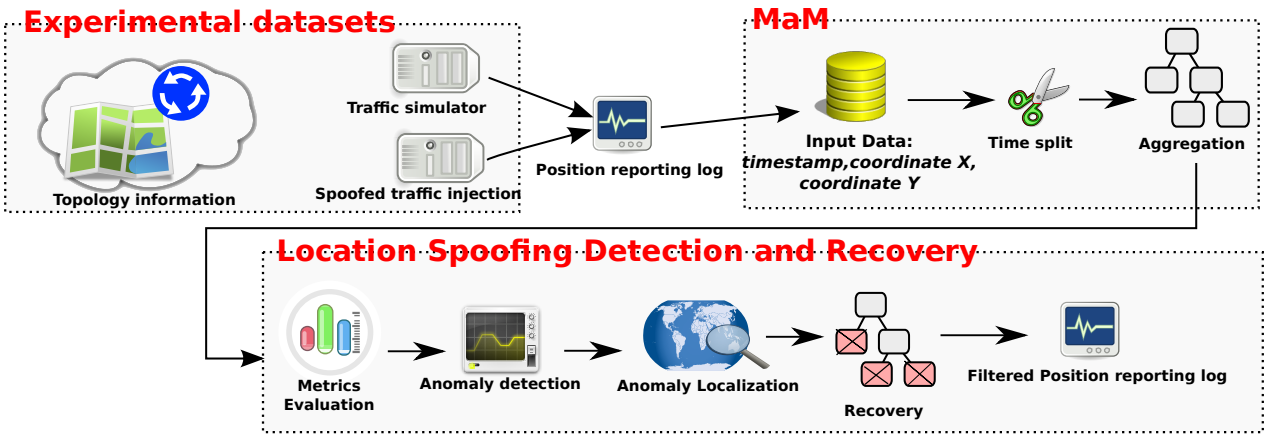


Figure 6.5: System overview

In this section, we aim to identify and localize an attacker who submits forged positions of numerous vehicles to a position-based service. Obviously, injection may be done continuously to be more stealthy and effective. Hence an attacker can mimic a driven vehicle by reporting multiple positions along a path in a coherent timely manner.

In particular, the following problems are addressed in this section:

- Location spoofing detection: the goal is to detect that an anomaly occurs in the reported locations.
- Spoofed position recovery: assuming that the location spoofing has been detected, the goal of the recovery consists into precisely identifying which positions are subject to the attack for discarding the corresponding records in order to keep the database of reported positions (and so the road traffic flows) in a safe state.

While the first objective is to detect when a spoofing attack occurs, the second one tries to localize where it occurs and remove the anomalous data.

In this work, we decided to model an area as bounded intervals. In fact, an area is defined as a bi dimensional space using Cartesian coordinates to represent the longitude and latitude (altitude is not considered). Hence, the areas are represented as rectangles defined by two points: $((X_1, Y_1) : (X_2, Y_2))$. The most specific representation is an interval of a single point: $X_1 = X_2$ and $Y_1 = Y_2$.

In the sample output shown in Figure 6.6, each node represents a specific rectangular area defined as:

1. X dimension range: (X_1, X_2) where $X_1 \in \Re$ and $X_2 \in \Re$
2. Y dimension range: (Y_1, Y_2) where $Y_1 \in \Re$ and $Y_2 \in \Re$
3. Percentage of vehicles in the corresponding area excluding its children: $vol \in \Re$

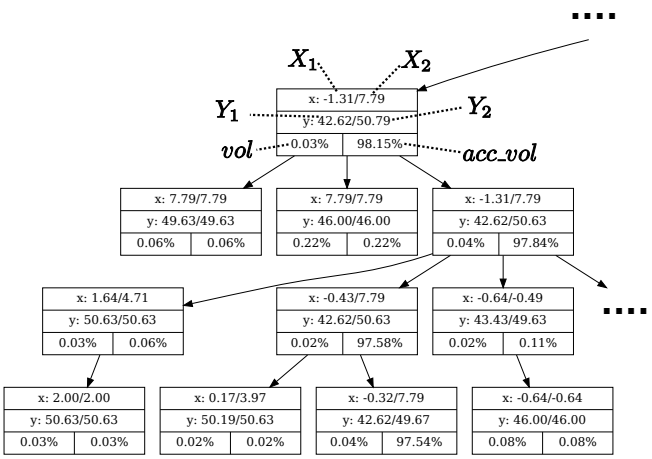


Figure 6.6: MAM sample output (Due to visualization, values are rounded with two decimal places)

4. Cumulated percentage of vehicles in the corresponding area (including its children): $acc\_vol \in \Re$

The corresponding area of a node is actually embedded into the area associated to its parent node.

As highlighted before, a hierarchical model has to be built over such dimensions. In order to keep the advantages of MAM, the areas are not fixed in advance as we could have done using a grid-based approach. Rectangle are built regarding the percentage of activity which, in this case, represents the percentage of vehicles. To do so, the areas are created on the fly by assembling points together. For example, assuming two individual points as $((X_1, Y_1) : (X_1, Y_1))$ and $((X_2, Y_2) : (X_2, Y_2))$, this will entails the creation of the area: $((mix(X_1, X_2), mix(Y_1, Y_2)) : (max(X_1, X_2), max(Y_1, Y_2)))$. When the third point has to be added, either it falls into this created area which implies the creation of an embedded area, or it falls outside creating a parent area. This process continues until no new data points have to be added. Hence, this automatically creates a hierarchy between areas which is not known a priori unlike IP subnetworks. In the meantime, each area maintains a counter about the number of vehicles it contains. Aggregation is then performed from the leaves to the root. For each node, if $vol > \alpha$, the node is kept, otherwise it is discarded and its vol value is added to the one of its parent node. This aggregation process is the standard one of MaM which is fully described in Section 5.1.3, page 66.

6.5.2 Metrics

We propose a method for analyzing series of aggregated trees based on stability over time. The intuitive idea behind is similar to plausibility checks by considering that vehicles movements from one area to another are bounded by physical properties but also depending on the traffic conditions. To achieve that, we defined a stability metric that captures the dynamic of the traffic, (*i.e.* the movement of the car) by comparing consecutive vehicle positions. Observing the variation of the stability when no attack occurs, we are then capable of detecting future abnormal variations. The aggregation helps in considering the global road traffic dynamic. In addition, this avoids to detect isolated deviant behaviours which are not necessary malicious (faulty ones) and, in fact, does not have an impact on

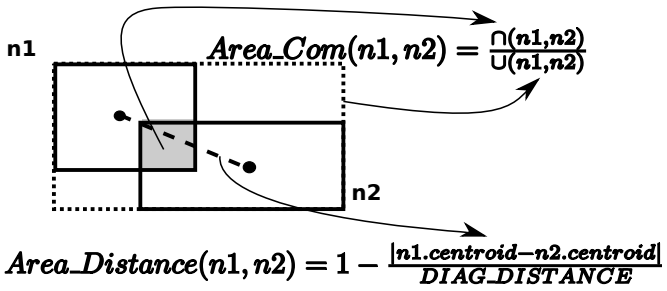


Figure 6.7: Area comparison

the large scale location-based service that relies on mass information. From a general point of view, stability based detection is equivalent to plausibility checks using aggregated views.

As explained, the stability captures the dynamic of the cars. It is a bounded value between 0 and 1. Assuming cars which are blocked in a traffic jam, the stability will be very high. On the contrary, cars moving very fast entails a low stability. Therefore, the goal is to evaluate the stability of the traffic (and so vehicles positions) between consecutive time windows. To do so, it is firstly required to define the stability between two nodes, $n1$ and $n2$. It is higher if the distance between the associated areas ($Area\_Distance(n1, n2)$) are close, if the overlap between them is high ($Area\_Com(n1, n2)$) and if the number of vehicles is similar:

$$\begin{aligned} Stability(n1, n2) = \eta \times Area\_Com(n1, n2) + \\ \psi \times Area\_Distance(n1, n2) + \\ \gamma \times (1 - |n1.vol - n2.vol|) \end{aligned} \tag{6.7}$$

The common space is evaluated regarding the ratio between the intersected rectangular area and the merged rectangular area: $Area\_Com(n1, n2) = \dfrac{\cap(n1, n2)}{\cup(n1, n2)}$

The distance is evaluated regarding the centroid of the areas and normalized regarding the diagonal distance of the monitored map: $Area\_Distance(n1, n2) = 1 - \dfrac{|n1.centroid - n2.centroid|}{DIAG\_DISTANCE}$

The common area and centroids can be easily retrieved from the coordinates and Figure 6.7 illustrates them. Evidently, areas are not always intersected and in such a case, $Area\_Com(n1, n2) = 0$. Without specific knowledge, $\eta = \psi = \gamma = 1/3$ which weights equally each factor.

The objective is to compare the stability between t_i and a previous tree t_j. To achieve that, the individual stability of each node $n_i \in t_i$ is calculated. This is calculated by comparing with the most similar node in t_j which is $mostsim(n_i, t_j) \in t_j \; \forall_{n_j \in t_j} Stability(n_i, mostsim(n_i, t_j)) \geq Stability(n_i, n_j)$.

Finally, to globally assess the abnormality of a tree t_i, the average stability over all nodes is considered:

$$Avg\_stab(t_i, t_j) = \frac{\sum_{n_i \in t_i} Stability(n_i, mostsim(n_i, t_j))}{|t_i|} \tag{6.8}$$

6.5.3 Algorithms

In this section, the core algorithms involved in the location spoofing detection and recovery are presented. As mentioned before, the location spoofing attack is detected by evaluating the global stability in equation (6.8) and detecting abnormal variation. For sake of clarity, we use a single profile which corresponds to set a threshold value of an acceptable stability variation. It is also possible to imagine more complex models taking in account peak hours for example. However, the method remains the same by using the stability measure to establish multiple profiles depending on certain criteria. To not be biased by local events, our approach promotes the use of a sliding window by comparing a tree with its S predecessors.

Hence, the algorithm 5 computes the stability of all nodes of a tree assuming a list of predecessors. The idea is to compare pairwise nodes from the current tree (noted as parameter t) against the previous trees within the sliding window. This can be observed in line 6 where the most similar node (having the highest stability) of each node of t is searched in the predecessors, as defined in the previous section. Then, the average stability for every single node t is computed over all the predecessors (line 7). In line 3, the algorithm iterates over each node of the input tree which leads to $O(n)$ assuming n nodes in a tree. For each node a look up over all S predecessors is performed in line 5, and in line 6 the most similar node is searched for every element on the predecessors list, with so a final cost of $S \times n$. In line 7 scalar operation are performed with a constant cost. Thus, the average complexity of algorithm 5 is $O(n^2)$ where n is the number of nodes since S is fixed constant.

Algorithm 5 Calculate the stability for the nodes of a given tree and its predecessors

Input: $t : Tree$
$predecessors : List < Tree >$
Output: $StabMap(t, predecessors) : Map < Node >< Float >$
1: $nodes \leftarrow$ nodes from t
2: $res \leftarrow Map()$
3: **for** n in $nodes$ **do**
4: $st \leftarrow 0$
5: **for** $t_i \in predecessors$ **do**
6: $c \leftarrow mostsim(n, t_i)$
7: $st \leftarrow st + Stability(n, c)$
8: **end for**
9: $res(n) \leftarrow st/length(predecessors)$
10: **end for**
11: **return** res

The detection and recovery algorithm 6 works as follows. If the average stability variation equation (6.8) exceeds the threshold θ, then an attack is detected as shown in line 2 which requires to compute the stability of all nodes of the current tree based in line 1 which executes algorithm 5. The function *values* allows to extract only the values of the map data structure. In fact, the recovery is an iteration process which removes one by one the least stable leaves until the difference between the current and past average stability values does not exceed the threshold θ as shown in lines 4-6.

The computation of the stability of the nodes is done once in line 1 which has a complexity of

$O(n^2)$ where n is the number of nodes in a tree. The average stability is calculated but iterating over all values in $stabMap$ which corresponds to the number of nodes. In parallel, the least stable leaf can be found, which means that operations in lines 2 - 4 are executed meanwhile by iterating over all nodes. In the worst case (in case every node needs recovery), the while loop condition will be also evaluated n times. This finally leads to a complexity equals $O(n^2 + n^2) = O(n^2)$.

Algorithm 6 Detection and Recovery

Input: $current : Tree$
Input: $predecessors : List < Tree >$
Input: $previous\_stab : float$
Output: $Recovery(current) : Tree$
1: $stabMap \leftarrow StabMap(current,\ predecessors)$
2: **while** $|avg(values(stabMap)) - previous\_stab| > \theta$ **do**
3: $leafs \leftarrow$ leafs from $current$
4: $least\_stable \leftarrow l \in leafs, \forall l' \in leafs, stabMap(l') \geq stabMap(l)$
5: remove l from current
6: remove l from $stabMap$
7: **end while**
8: **return** $current$

Therefore, the general process takes as input position data with timestamps in order to create aggregated trees on the fly depending on the time window size β and the aggregation threshold α. When a time window ends, the corresponding tree is preprocessed using the Algorithm 5 before the detection and the recovery is triggered in Algorithm 6.

6.5.4 Conclusion

This section describes our proposed location spoofing detection and recovery mechanism [189] can be applied to a large scale database of position data such as those used in many vehicular applications. Our approach uses as a first step MAM for collection and data aggregation. On top of it, we defined dedicated metrics to asses the plausibility of moving vehicles in order to point out anomalies. To validate our approach, we conducted several experiments using realistic traffic simulations, in which we took under consideration two major types of urban grids, as introduced in Section 7.4.4. The main advantages of the aggregation are the global evaluation the traffic without considering too small events and the scalability improvement. In addition, a recovery mechanism is proposed which allows to safely removed most of the anomalous data while discarding very few benign data.

6.6 Summary

In this chapter we presented our approach to leverage security with data analytics based on multidimensional aggregation. During our approach we proposed several metrics and carried out their analysis, based on different contexts (NetFlow, DNS, and Position-based crowd-source applications). The main principle is to analyze and evaluate consecutive trees representing aggregated events in time.

We were able to apply and extend the initial methodology for multidimensional aggregation proposed in Chapter 5. Also, scalability was taken in account, in particular for position-based applications, where we developed an approach for recovering affected information from a position spoofing attack. These approaches are validated with experiments on the three mentioned fields in Chapter 7.

Part III

Evaluation

Chapter 7

Monitoring of distributed applications

7.1 Overview

This chapter outlines the evaluation of the methods and metrics proposed in Chapter 5 and Chapter 6 respectively. The first section of the chapter presents the empirical validation of the multidimensional aggregation methodology. To conduct the experiments we included diverse dimensions, and compared aggregated trees altering the data order to asses its impact to the aggregation process. As a case study, we performed the validation of the efficiency of aggregation techniques on a real world scenario, using NetFlow records offered by a major ISP from Luxembourg. Aditionally, we also conducted experiments to validate our performance estimations in terms of computing time and resources.

The remaining sections of this chapter introduce the validation of the metrics on aggregated tress for the detection of anomalies and malicious activities. As introduced in Chapter 6, we include three main case studies where we validated our approach: TCP/IP Networks using NetFlow records, Domain Name System studying the match between domain names and IP addresses, and postion-based crowd-source applications for vehicle routing and personal navigation.

7.2 IP Traffic

7.2.1 Introduction

In this Section we evaluate the following main aspects of the proposed method:

- Validate the aggregation process including multiple dimensions from NetFlow records
- Performance of the proposed strategies of online aggregation
- Order of data impact on aggregation accuracy

For sake of clarity, only partial trees are shown in Figures. Except when mentioned, LRU strategy is applied.

7.2.2 NetFlow-based Metrics

NetFlow [75] was developed by Cisco Systems and is supported by many device vendors. Thus, NetFlow or other flow-based approaches are now considered as a standard for IP monitoring. A flow record groups multiple packets sharing similar properties and in particular source and destination addresses, protocols and ports. Available information includes useful information like a time stamp, number of packets or bytes exchanged.

In Section 6.3 we introduced a metric for evaluating the similarity of NetFlow-based aggregated trees. This notion of similarity we used for the validation of the aggregation process is as follow:

$$Dist(x,y) = \alpha \times IP\_sim(x_{src}, y_{src}) + \beta \times IP\_sim(x_{dst}, y_{dst}) + \gamma \times Vol(x_{vol}, y_{vol})$$

To avoid giving preference to any parameter, $\alpha = \beta = \gamma = 1/3$ in the precedent equation.

$$IP\_sim : Node \times Node \rightarrow \Re \in [0,1]]$$

$$IP\_sim(n1,n2) = \begin{cases} 1 - \dfrac{|n1_{prefix\_len} - n2_{prefix\_len}|}{32} & \text{if } n_1.address \subset n_2.address \\ 1 - \dfrac{|n2_{prefix\_len} - n1_{prefix\_len}|}{32} & \text{if } n_2.address \subset n_1.address \\ 0 & \text{else} \end{cases}$$

$$Vol : \Re \times \Re \rightarrow [0;1]$$

$$Vol(n1,n2) = exp(-\frac{|n1-n2|^2}{\phi})$$

We compute the edit distance as defined in Section 6.2.2 between two trees to figure out the similarity and point out the cases where anomalies may be present, using a sliding window metric defined as follow:

$$z[n] = Dist(T_n, T_{n+1}) \text{ with } n \text{ in } 1 \ldots m-1 \text{ where } m \text{ is the last time window}$$

7.2.3 Data sets

Real NetFlow captures were provided by one major ISP in Luxembourg. As assumed to be free of attacks, we also inject a realist attack in the same manner as in [14] for assessing the ability of our approach to catch valuable information about anomalies.

The duration of the capture is 26 days from 01/30/2010 to 02/24/2010 with an average number of flows around 60,000/sec. A total of 279815 unique IP addresses using 64470 different UDP and TCP ports are represented.

The following information is extracted:

- Timestamp
- IPv4 Source Address
- IPv4 Destination Address

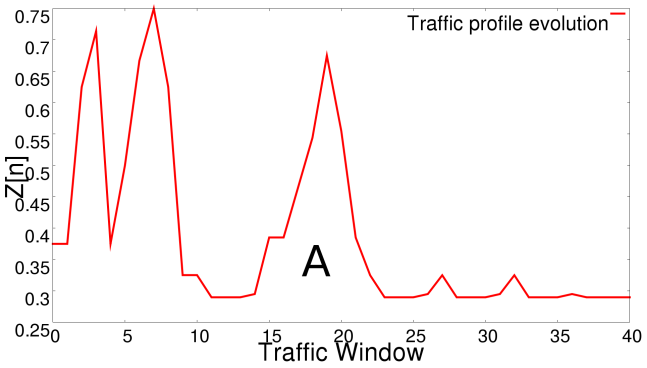


Figure 7.1: Distance Function Evaluation for aggregated trees generated from Flow capture of a TCP Flood attack

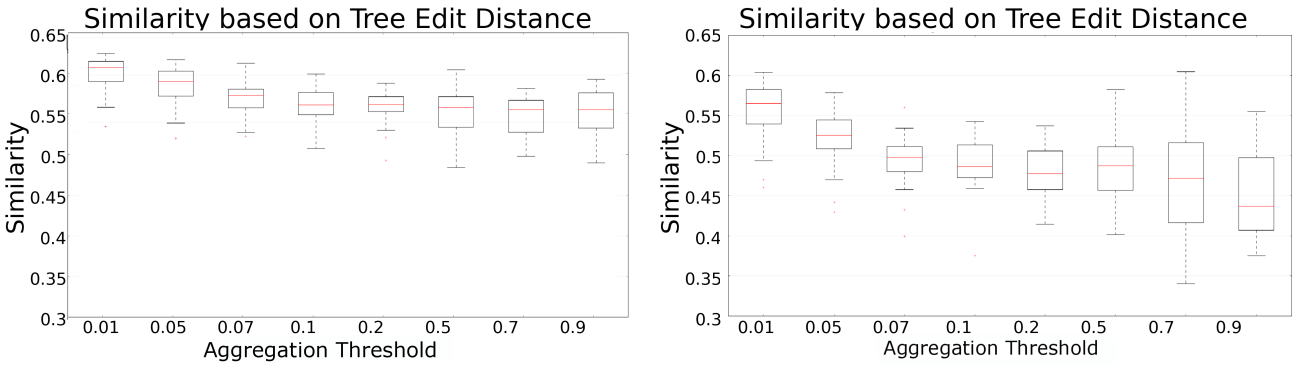
- TCP source port
- TCP destination port
- Traffic Volume in bytes is considered as the activity volume (vol)

7.2.4 Scenario

In order to evaluate the aggregation accuracy, several tests over the NetFlow dataset were conducted. The evaluation consists in comparing similar and different traffic flows by observing the similarity variation. From the dataset, 50 time windows ($\eta = 5min$) are randomly picked up from working days between 01/30/2010 and 02/24/2010. Once the tree is built, it is compared with the tree associated to data with a 24 hours or 36 hours offset. Activity at the same time between two working days (24hs offset) is usually considered as similar and so should exhibit a higher similarity than activity at a different times between two days (36hs offset). Figure 7.2 shows that the similarity with a 36hs offset decreases higher than with the 24h offset, leading to an easier differentiation. Even if the average similarity tends to be close between figures 7.2(a) and 7.2(b), the box plots clearly highlight a lower stability with a 36h offset which leads to the same conclusion. With respect to experiments on the data order (section 7.2.5), values are mainly lower than 0.6 for the 24h offset. Therefore, impact of the data order is lower than the impact of activity variation at the same time between two consecutive days. From this point of view, our approach is thus viable.

To evaluate the aggregation helpfulness in an malicious environment, the following experiment has been conducted:

- injection of DDoS attack directed against web servers running on TCP port 80 with a repeated sequence (3 packets and 128 bytes) sent by burst of 10 packets every 60-120 milliseconds. The attack was injected in two periods of 2 minutes each.
- 60 trees were generated, $T_0 \dots T_{59}$, to summarize network activity ($\eta = 25s, \alpha = 0.05$) by monitoring both the source and destination IP addresses meanwhile



(a) 24 hours offset

(b) 36 hours offset

Figure 7.2: Similarity - trees generated using NetFlow samples compared to trees generated after a NetFlow capture from a offset using a 5 minutes window

In figure 7.1, the two attack occurrences are clearly distinguishable by observing peaks due the high variation of the edit distance when the attacks start and end. Although more advance techniques, as for example classification methods, can be leveraged and evaluated on other kinds of attacks (on DNS and on position-based crowd-source applications), such a complete evaluation is explained further in this chapter, in sections 7.3 and 7.4.4

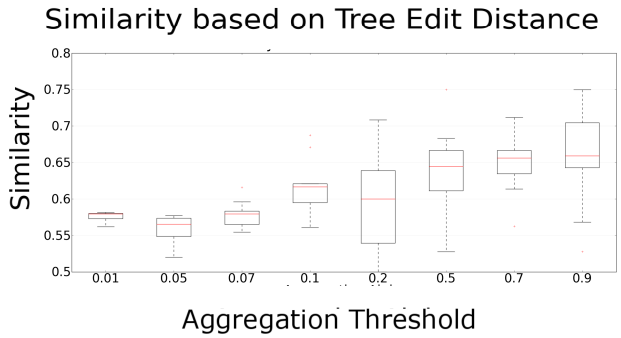
7.2.5 Order of data impact

Since data order may impact the aggregation process order, the focus is set at analyzing the distortion of the tree shape after altering the data order.

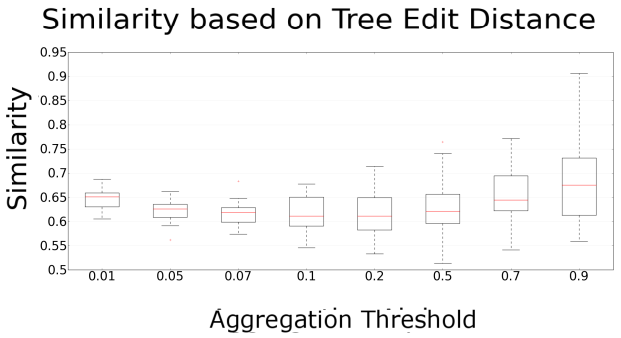
The evaluation was performed on 3 dimensions: source IPv4 addresses, destination IPv4 addresses and TCP ports. For strengthen the evaluation, experiments are executed 1000 times with different samples and percentiles (25th, 50th, 75th) as well as minimal and maximal values are then calculated and represented in a box plot graph like Figure 7.3, in which we show the similarity of trees compared to trees constructed with the same data but in different order. In these figures we presented the distribution of values for the similarity calculated 1000 times for range of aggregation thresholds (α) between 0 and 1.

- Normal Sample vs Reversed Sample: This test consists in comparing trees generated from a 5 minutes sample ($\eta = 5min$) of the NetFlow dataset and the same data, *i.e.* NetFlow records, in the reverse order. In Figure 7.3(a), the similarity grows with aggregation ratio α except for small values. This behaviour is due to the lower granularity, which keeps track of global phenomena which are usually present in the whole sample, `i.e.` not at a specific time unlike local events, and so less sensitive to the data order.

- Normal Sample vs Random Sample: this test is similar but data are not reversed but randomized. The same observations can be made in Figure 7.3(b) even if the impact of α is less visible.



(a) NetFlow samples compared to trees generated after a the same NetFlow capture reversed



(b) NetFlow samples compared to trees generated after the same randomized flows

Figure 7.3: Similarity - comparison of trees generated using NetFlow samples altering the data order for a 5 minutes window

It is important to remind, that the similarity is a number between 0 and 1, with 1 when the trees are identical. We can observe from our experiments that the data order has an impact on the aggregation process itself since the value never reaches 1. However, it is important to verify that the similarity value is lower when data differs. Therefore, we can assumed that in most of cases, a similarity higher than 0.6 may reflect similar data but which may have been process in a different order.

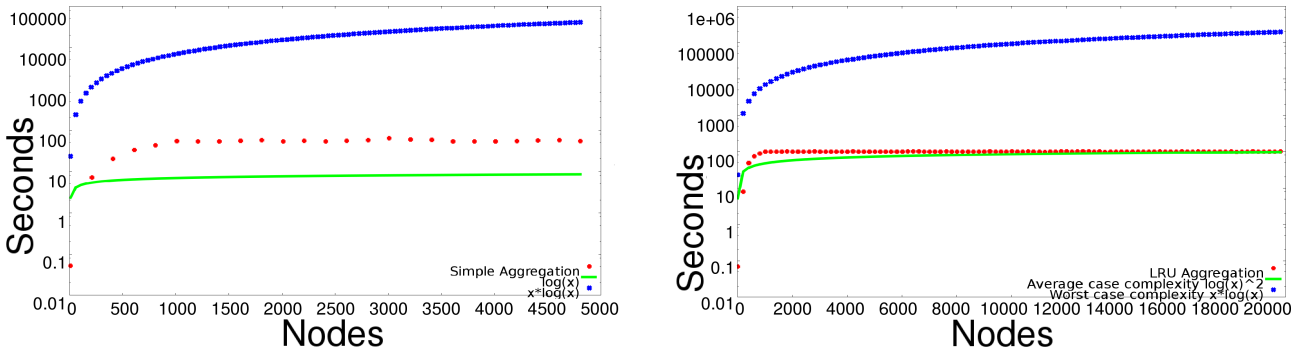
7.2.6 Performance

Several benchmarks have been done in a 8 core Intel(R) Xeon(R) CPU E5345@ 2.33GHz. They are based on the running times regarding the number of nodes in a tree. The Netflow dataset was used by considering source and destination addresses as well as destination ports. As introduced in Section 5.1.4, the worst case computational complexity of the LRU strategy is $n \times log(n)$. In figure 7.4(b), the empirical results are quite lower. This behavior can be explained because the worst case scenario is a pathological case that is not usually common found in the captures. Furthermore, this is coherent with the average computational complexity ($log^2(n)$) introduced in Section 5.1.4.

Figure 7.4(a) highlights similar results for the simple aggregation at the end of time windows. Therefore, algorithms are equivalent from a computational power side. Applying the LRU strategy does not entail any notable overhead. However, from a memory point of view, the LRU strategy saves a lot of memory space since the tree size is bounded by MAX_NODES.

7.2.7 Conclusion

In this section, we presented our experiments focused on sample scenarios where the aggregation reveals important changes in the network activity. In this section we validated two main aspects of our approach; the first related to the experimental validation of the ability to detect significant changes in the volume of the network activity, that in our case given the data set provided lead to



(a) Time performance for simple aggregation

(b) Time performance for LRU aggregation

Figure 7.4: Time performance for different strategies

the detection of DDoS attacks. The latter, related to the impact of data order on the aggregation process, in which we were able to observe that this phenomenon does not mask the events occurred in the network. Additionally, we were able to analyze a information in a scalable way, including an LRU aggregation approach for performing analysis with limited computing and memory resources.

Theoretical as well as practical complexities were studied, in particular to prove the efficiency of using a LRU strategy for improving the scalability of our approach. Furthermore, the data order problem due to such a process was evaluated and the results highlight a negligible impact.

7.3 Domain Name System

7.3.1 Introduction

In this section we outline the evaluation of the metrics used on our approach based on multidimensional aggregated trees in the context of DNS monitoring for detecting anomalies. In this section we present short and long term experiments to validate our hypothesis: the association between IP addresses and domain names related to malicious activities have a relatively unsteady persistence in time, compared to non-malicious related domain names and IP addresses.

The objectives of the experiments are:

- Check that steadiness is a feature that discriminates between malicious and normal domains
- To verify that it is possible to detect attacks, *i.e.* that the time series are highly steady for normal domains compared to malicious ones
- Show how to extract suspect data that has has an abnormal effect on the steadiness
- Assess the dependency between detection accuracy and aggregation granularity
- Evaluate the performance of aggregation in terms of computing resources and memory.

7.3.2 Metrics

The metrics validated in this section were introduced in Section 6.4.3 on page 84. The main idea behind the metrics is to reflect the similitude of IP addresses and domain names associations in aggregated data structures. This can be taken as a measure of the local steadiness of a node in a tree. It is also possible to evaluate the global persistence of all nodes over a sequence of trees, and thus assess the general steadiness of the mapping between IP and DNS space.

7.3.3 Data set and methodology

We built a dataset using DNS Records from a Passive DNS database provided by two major Internet providers in Luxembourg in collaboration with the University of Luxembourg. The records give the associations between domain names and IPv4 addresses. DNS records were collected from 2011-04-23 to 2012-06-30. Due to a non-disclosure agreement, the exact probe location cannot be revealed. To establish the ground truth, two subsets were extracted from A records (IPv4 addresses):

- a malicious set by extracting DNS records related to domains obtained in the following malware databases: MalwareDomains [1], Exposure [2] and FIRE [3].
- a normal set excluding domains from the previous set

	Domains	IP Address
Normal	661968	164559
Malicious	173066	174619
Total	835034	339178

(a) General information

Malware	45%
Phishing	30%
Harmful	13%
Malvertising	4%
Trojan	3%
Others	5%

(b) Malicious domains

Table 7.1: Datasets

As highlighted in Table 7.1(a), we considered that 662k normal domains would be representative. In comparison, 173k malicious domains were represented. However, the number of distinct IP addresses is similar, justifying steadiness of the mapping between domains and IP addresses as a relevant metric.

The blacklists constitute the ground truth in our experiments and correspond to malicious domains (details in Table 7.1(b)). As the blacklists may contain errors, a bias may be introduced. Another solution, largely leveraged in related work like [26, 105], is whitelisting using Alexa [4] which ranks the top 1,000,000 websites by access count. However, this inevitably results in a strong bias as it represents popular sites, which are highly constant by nature. To keep our evaluation valid, such an approach was rejected.

[1] `http://www.malwaredomains.com/`, accessed on 15/03/2015
[2] `http://exposure.iseclab.org/`, accessed on 15/03/2015
[3] `www.maliciousnetworks.org/`, accessed on 15/03/2015
[4] `http://www.alexa.com/topsites`

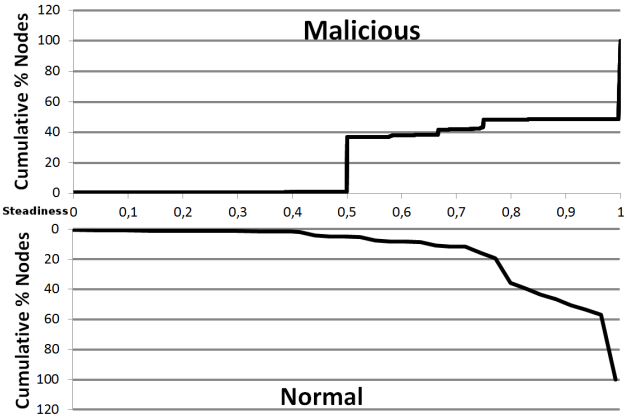


Figure 7.5: Cumulative distribution functions of nodes according to steadiness values

7.3.4 Steadiness Distribution

In this experiment, the validity of the proposed metric for evaluating the steadiness in Section 6.4.3 was verified.

For the complete dataset, the local steadiness of the nodes was computed using equation (6.5) over the two disjoint datasets: malicious and normal domains gathered from the passive DNS database. For each dataset, the aggregation relies on a time window (η) of one week and an aggregation threshold of 2% (α). This tuning is the same for following subsections, unless otherwise mentioned, and was determined at by running preliminary experiments over a small sample of data.

By considering every tree (and so every week), Figure 7.5 shows the cumulative distribution of nodes regarding their steadiness (buckets of $[i; i + 0.1]$ have been constructed). In both distributions, there is a significant increase on the highest values of steadiness. This phenomenon occurs because some generic nodes qualified at the higher levels exhibit a high steadiness independently of the dataset. For example, the `.com` domain and `0.0.0.0/0` are always present.

For normal domains, the curve continuously remains increasing beyond 0.75 unlike malicious domains. This means that there is a significant proportion of nodes having a steadiness higher than 0.75 for normal domains.

Nevertheless, there is one intermediate high increase, at 0.5 for the malicious dataset and at 0.75 for the normal dataset. Therefore, the steadiness metric is able to reveal the differing impact of malicious and normal domains in constructed trees.

7.3.5 Steadiness Evolution

While the previous experiment assessed the steadiness of nodes independently of time, the next experiment considered macroscopic steadiness (equation (6.6)) of the trees over time. To conduct this experiment, 10 consecutive weeks, from 2011-07-09 to 2011-09-24, were chosen, since they represent the greatest number of records. The total size of the dataset for 10 weeks is 81806 DNS Records (both malicious and normal domains).

For this research, we were interested in observing how steadiness varies when different percentages of malicious domains are present. Thus data sets were generated by injecting records of malicious domains into the normal dataset. In this way, it was possible to create datasets with various proportions (0.1%, 1%, 10%) of malicious domains, so strengthening the validation. It is important to note that this was not a random injection since we mixed datasets by respecting the time scale, *i.e.* only data of from the same week i from both malicious and normal datasets was mixed. Consequently, it was not always possible to reach the desired proportion of malicious domains. However, this different proportion of malicious does not variate significantly to cause bias. This is equivalent to evaluating our approach in worst case conditions. In order to compare results to a baseline and contrast the variation in steadiness, the steadiness evolutions for purely malicious and for normal domains were calculated.

In Figure 7.6, the average steadiness is smoothed using a sliding window: the value at week i is the average from week $i-2$ to week $i+1$. The chart represents the average steadiness of the ten consecutive trees except for the first, since its steadiness cannot be calculated with respect to a previous week (according to equation (6.5)). The results for pure datasets (normal and malicious) are easy to distinguish. Although the normal dataset shows a fairly steady value around 0.85, the malicious one reveals variations between 0.65 and 0.83. Generally, as expected, the steadiness of trees constructed from malicious domains is lower and varies considerably, meaning that the steadiness of malicious domains differs from one to another more that is observed for normal ones.

When both malicious and normal domains are mixed together, steadiness is impacted and the values are logically lower than when only the normal dataset is used. Since data from malicious and normal domains can be aggregated into shared nodes, there is however no clear ordering between the curves, i.e. the curves may cross. But even with only 0.1% of malicious domains injected into the normal database, steadiness is lower than for the normal domains. Therefore, this experiment confirms the viability of steadiness, as we have defined it, as a discriminative feature for tracking malicious domains.

7.3.6 Macroscopic Steadiness during attacks

In the previous experiment, malicious domains clearly impacted steadiness, but the differences could be very low (see weeks 8,9 in Figure 7.6). However, a monitoring system should detect an anomaly when it occurs, whereas the previous experiment focused on the continuous presence of malicious data. Assuming a clean database to initialise the system (as we did with blacklists), this experiment shows that steadiness drops in the presence of malicious data.

The dataset used in this experiment was made by first aggregating 40 consecutive weeks, then every 5 weeks, malicious domains related DNS records were injected for the next 5 consecutive weeks. The periods where malicious domains are included are annotated as *Attack* in Figure 7.7.

It can be seen that there is a drop in average steadiness correlated when malicious domains are injected. At the right of Figure 7.7 the tail of the chart appears to remain stable, which seems to be due to a steadiness that has remained low even without malicious data (weeks 30 - 34). This might be interpreted as a false positive in the context of anomaly detection. It could be due to limited knowledge provided by blacklists, which cannot guarantee 100% coverage of all malicious domains in our passive DNS database.

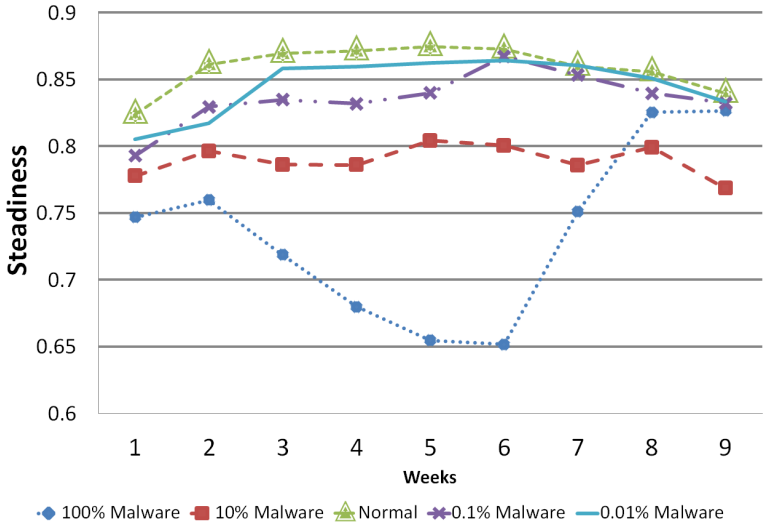


Figure 7.6: Comparision of average steadiness for 10 weeks period

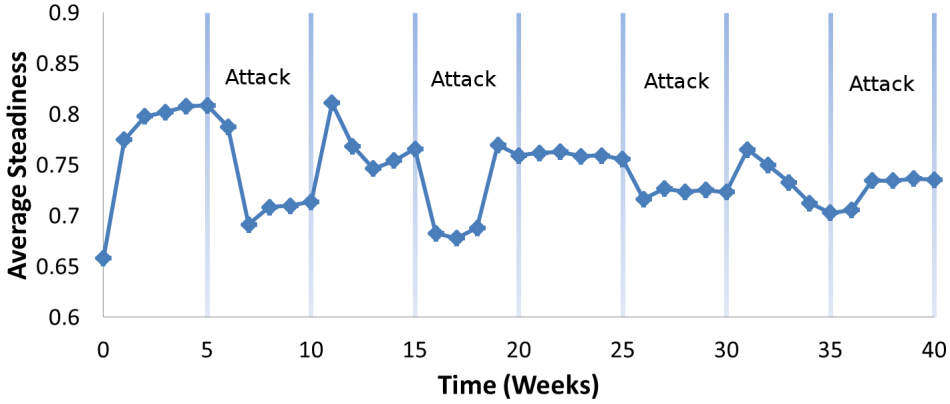


Figure 7.7: Macroscopic evaluation of average steadiness for 40 weeks alternating DNS Records containing malicious domains every 5 weeks

7.3.7 Detection

Although the previous experiment allowed the detection of windows containing anomalies (malicious domains), detecting the exact anomaly is more interesting. The goal of the next experiment is to determinate an experimental threshold such that nodes having a steadiness lower than this threshold can be identified as malicious, since Figure 7.5 shows that most of nodes related to malicious domains are less steady than normal ones.

Due to the aggregation scheme, both abnormal domains and IP addresses can be detected. However, obtaining a representative blacklist of IP addresses is quite unfeasible as most of services offer limited online checking [5] or simply block entire countries or regions. Therefore, our evaluation relies on the blacklists we described in Section 7.3.3, which contain only DNS names.

As the blacklists only include the FQDNs of final hosts, there is no ground truth for entire domains. Hence, any assessment of our detection scheme can only be based on the *final domain* nodes, *i.e.* the nodes in the tree representing final names like `$.*.ROOT`. Such nodes mostly correspond to leaf nodes, which are usually collapsed into their ancestors during aggregation. Hence, α is set to zero to preserve entire trees and so track suspect domains at a specific time. Nevertheless, the steadiness of corresponding *final domain* nodes still takes the tree of the previous time window into account, possibly leading to computing the similarity in equation (6.1) with a non *final domain* node. Thus, when evaluating steadiness at time i, the previous trees are still aggregated ($\alpha > 0$). Therefore, the steadiness metrics retain the advantage of aggregation and provide a richer comparison than with either DNS names or IPv4 Addresses (as for example with pair-wise comparisons).

Figure 7.8 shows the cumulative distributions of final domain nodes according to the local steadiness. Supposing a steadiness threshold, the curves representing malicious and normal domains can be considered respectively as the True Positive Rate and False Positive Rate.

As shown in Figure 7.5, a steadiness of 0.5 seems to characterize malicious domains. Setting the threshold to 0.5 is equivalent in Figure 7.8 to detecting 73% of malicious domains malicious domains with fewer than 1.6% of false positives. While 1.6% may represent too high a number of false positives, our approach can be considered as a means of selection suspicious domains for further checking with a more in-depth investigation like to MalwareDomains [6]. Furthermore, 0.45 is a better value according to Figure 7.8 as the same true positive rate (73%) can be achieved with only 0.3% of false positives. This represents a highly acceptable result.

7.3.8 Performance

To compute the steadiness of a node in a tree, the worst case may need to iterate over all nodes of the previous tree following the process described in Section 6.4.3 (linear complexity), leading to a quadratic complexity if the steadiness of every node is evaluated (equivalent to pair-wise comparisons). Hence, the number of nodes is the main parameter which impacts on scalability and this depends on the aggregation threshold α.

We can observe in Table 7.2 that the tree size decreases as α grows. Since the steadiness evaluation is performed after aggregation and depends on the size of the tree (number of nodes), this helps in improving scalability. If the scalability and memory consumption are critical factors, the tool is still

[5] As an example, the Spamhaus Project, `www.spamhaus.org/`

[6] `http://www.malwaredomains.com/`

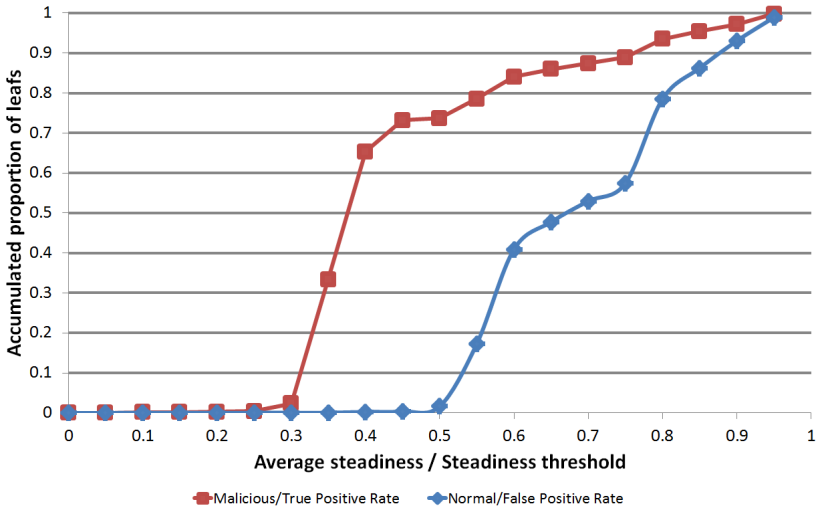


Figure 7.8: Cumulative distribution of local steadiness of final domain nodes equivalent to the true positive and false positive rates assuming the x-axis as the threshold for considering a domain as normal

Table 7.2: Average tree size

α	0%	2%	5%	10%
Average #nodes	84651	2238	1755	1525

usable, since the number of nodes after aggregation at the end of a time window is drastically reduced. The average tree size is around 2200 nodes with the default parameter used in previous experiments ($\alpha = 2\%$) which thus greatly limits the overhead when computing the steadiness, even though the complexity is quadratic in the worst case. For reference, the average number of distinct IP addresses and domains per week are respectively around 13000 and 5300.

Assuming now the previous case for detecting malicious domains (Section 7.3.7), the number of comparisons is given by the number of leaf nodes (*final domain* names) multiplied by the number of nodes in the previous aggregated tree, which corresponds to an average of around 14400×2200 comparisons considerably fewer than a pair-wise comparison of leaf nodes (14000^2).

7.3.9 Conclusion

In this section, we validated the efficiency of our approach based on aggregated multidimensional trees to study the mapping between IP addresses and domain names. Thanks to the aggregation scheme, the volume of data to analyze is reduced and distributed behaviours can be identified more easily, which is helpful in the security context. It is useful in tracking entire harmful IP subnets or domain names while keeping the computational overhead low. We have validated through short and long term experiments the capability to outline anomalies by using a specific metric to study the steadiness of time series of trees produced by *MAM*. The evaluation on real data has proven its efficiency for detecting global abnormal changes (macroscopic steadiness) and for tracking specific domains and

IP subnets which are responsible for them (local steadiness). We have observed that evaluating the steadiness of domain names may become a major asset for characterizing malicious domain names. In order to calibrate the threshold for our approach, we studied the steadiness distribution of both malicious and normal domains. Simultaneously, we observed its variation while changing the malware proportion in the dataset. We were able to classify malicious domains with a True Positive Rate of around 0.75 while keeping a low False Positive Rate of 0.03.

Scalability has also been assessed, we have validated that aggregation reduces significantly the volume of data, with a low information loose in terms of specificity. The size of aggregated trees was in average an order of magnitude smaller than the size of non-aggregated trees. In particular, as the aggregation threshold variates. Therefore, we validated that our methodology is efficient for analysis and reducing the post processing time requirements.

7.4 Crowd-sourced location-based applications for vehicular routing

7.4.1 Introduction

In this section, we introduce the validation of our methodology for monitoring and analysis of crowd-sourced position-based applications. Assuming the context of an application for vehicular routing and personal navigation, we present experimental results for anomaly detection in reported geographical positions. As introduced in Section 6.5.1, our methodology consists in monitoring the geographical positions reported by users from a global scope. Aggregation allows us to keep track of global traffic events such as congestion, or high vehicle density areas. We perform plausibility checks on the series of aggregated trees generated by MAM to look after incoherent changes in traffic patterns, as for example, sudden appearance or disappearance of clusters of vehicles. In our approach, the focus is set on significant traffic changes caused by large vehicles groups, and not in checking individual vehicles at a microscopic level.

The metrics used in our approach evaluate how stable or coherent is a given traffic state (aggregated tree) within a window of time. Therefore, we use a metric based on the similarity of trees, which is calculated pair-wise. We use a sliding window to compare consecutive trees, in order to detect unlikely (unstable) changes.

For the experimental part, we first generated traces of normal traffic using simulated urban mobility patterns. Then, we developed an attack model allowing to inject spoofed positions with parameters such as frequency, radius or time length. We conducted a series of experiments to study the impact of the attacks on the average stability metric on aggregated trees. Additionally, we validated our approach to mitigate the impact of the attack, by removing the flowed positions which are identified by the sudden changes of stability in the nodes of aggregated trees.

7.4.2 Scenario

Traffic Simulator

The simulator SUMO (Simulation of Urban MObility) [179] was used in order to generate vehicle traffic flows in an urban environment. SUMO is an Open Source traffic simulator used in many relevant research in the field [180–182].

Urban Scenarios

Our evaluation is based on two scenarios to represent different possible urban topologies. A Manhattan mobility model relies on a grid road topology and is representative of modern cities especially in North America. In order to have a good trade-off between SUMO performances and representativeness, the grid road topology is bounded to a 5 km $\times$ 5 km square containing regular 100 meters length blocks. Because such a topology is not representative of old European cities. We included the city of Luxembourg as a complementary approach for simulating traffic. For fairness, the Luxembourg scenario is also bounded to a 5 km $\times$ 5 km which is centred in the city centre. This is obtained using OpenStreetMap[7].

7.4.3 Traffic Simulation

For simulating traffic in both scenarios, a random mobility pattern is used. It consists in defining, for each normal vehicle, a random departure and an arrival street point. In addition, when the car arrives at its destination, its location information is not provided any longer to mimic a real vehicle. For both scenarios we generated traces for 10 consecutive hours. A total of normal 220,000 vehicles were injected along the simulation. The number of misbehaved vehicles (spoofed locations) is then dependent on the simulations as explained in next sections. For being realistic, the cars are not injected simultaneously but also at a random number, every seconds. Around 20833 new cars are injected every hour as highlighted in Table 7.3, which gives also some other characteristics of the datasets. Moreover, each vehicle reports its position every second.

	Manhattan	Luxembourg
Total Cars	220,000	220,000
New cars per hour	20833.33	20833.33
Average Car per Block	8	11
Average Speed	15	17
Average Car Trip Time	12 min	8 min

Table 7.3: Simulation Performance in numbers

7.4.4 Malicious Traffic Data Generation & Injection

Malicious traffic was injected in order to extend each original dataset during our experiments. The main principle consists into injecting faked reported positions of vehicles in a predefined area which thus mimics spoofed vehicles in this area. Therefore, the following parameters are used:

- Time: the time interval to perform the data generation.
- Center: the center of the area where faked vehicles are injected
- Radius: the radius size of the area where faked vehicles are injected

[7] `http://www.openstreetmap.org/`

- Volume of cars: overall amount of injected vehicles.
- Frequency: proportion of vehicles per time unit to be injected regarding the number of normal cars.

The injection takes in consideration the map topology for creating vehicles at valid locations. Besides, the traffic injection models may generate spoofed location attacks against multiple areas. To achieve that, several locations (center and radius) can be defined. In such a case, the volume of cars and the frequency of data generation are global to all targeted locations. In our experiments, the radius is fixed to 500 meters the number of targeted locations is 2 for the Luxembourg scenarios and 3 for the Manhattan scenario. We deliberately choose small values in order to strengthen our approach by considering targeted attacks unlike global attacks that could impact all the map and so be highly visible.

As highlighted in Section 6.5.1 in page 87, a unique identification of every vehicle would be helpful but does not sound plausible as we are focusing on open crowd-sourced applications relying on a non dedicated infrastructure. This eases the task of the attacker to create faked identity as many as he desires. In addition, drivers and so vehicle identification is a major privacy issue. We implemented the most general model without identifying vehicles but only targeted areas which thus conforms to our detection approach.

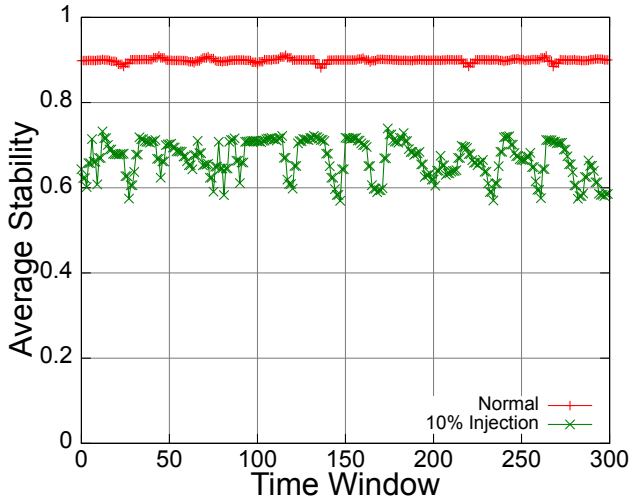
7.4.5 Experimental Results

In this section, experimental results are presented. The first experiment was conducted to evaluate the impact of the attacks on the stability and so to show that a threshold-based technique (θ) is viable. The second experiment assesses the impact of injection within different topologies. The third experiment is dedicated to the localization of the attack and its recovery. Then, this section also highlights the gain of using the aggregated trees compared to individual vehicle positions.

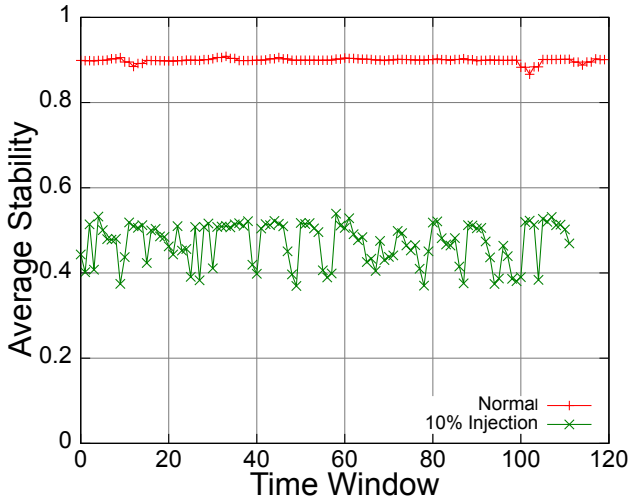
Preliminary experiments help in identifying good parameter values for the aggregation threshold $\alpha = 2\%$ and the sliding window size $S = 5$ for computing stability.

Window size impact

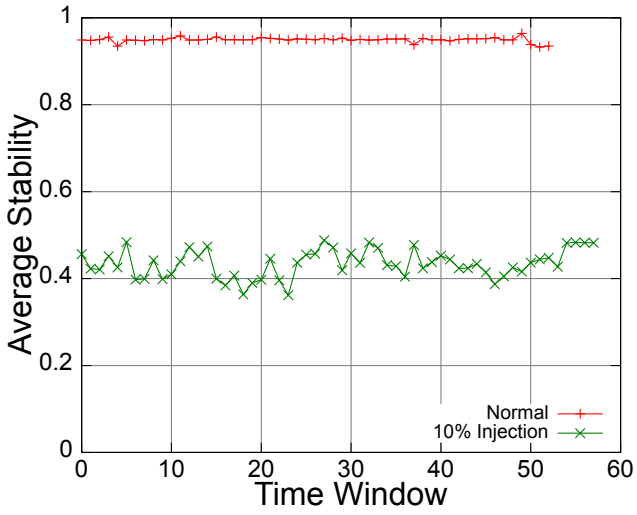
In order to characterize regular traffic stability, Figure 7.9 compares the average stability when there is no attack and when 10% of faked vehicles are injected. As mentioned before in this experiment we aim to asses the impact of β (time aggregation window). This experiment considers a Manhattan grid topology and the window size is successively fixed to 120, 300 or 600 seconds. The average stability shown in Figure 7.9 is computed over ten independent simulations. Regardless the window size, the average stability when an attack occurs is always different and so an attack could be easily detected. Next section considers more stealthy attack with less faked vehicles. However, as illustrated in the Figure 7.9 and especially in Figure 7.9(c), increasing the window size improves the separability of the curves which thus will help in detecting the attacks. In fact, this discards local bias by merging much position data. However, this will delay the attack detection until the time window ends. Assuming a standard context of road traffic monitoring, 600 or 300 seconds seems reasonable and provides good results in Figures 7.9(b) and Figure 7.9(c). Therefore, such values for β are used in the other experiments.



(a) $\beta = 120$ seconds

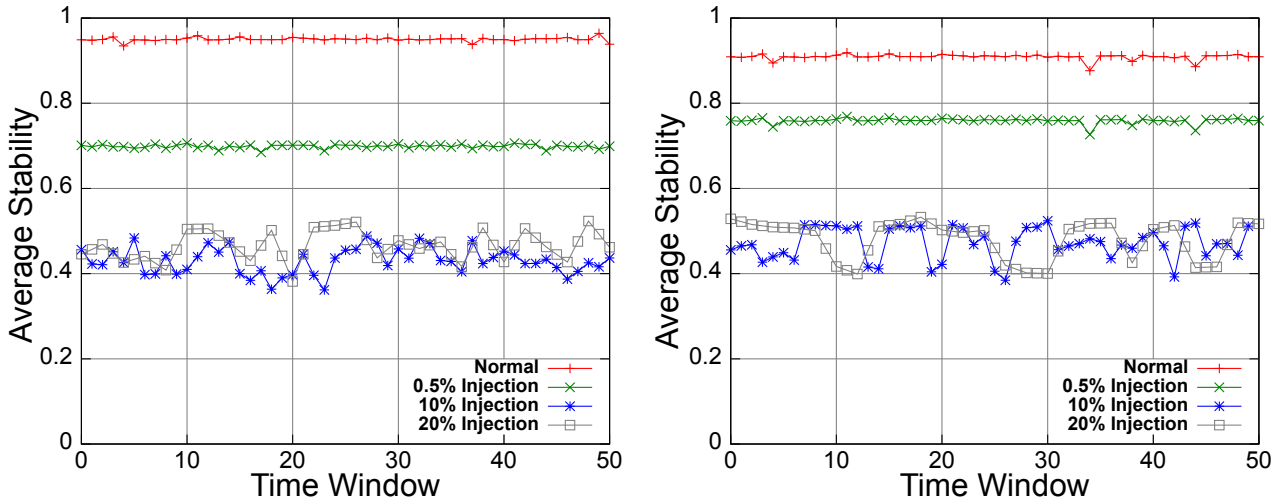


(b) $\beta = 300$ seconds



(c) $\beta = 600$ seconds

Figure 7.9: Average Stability for 10 hours simulation on a Manhattan Grid without attack or with 10% of injected data



(a) Manhattan topology

(b) Luxembourg topology

Figure 7.10: Average Stability with various attack level, $\beta = 600$ seconds

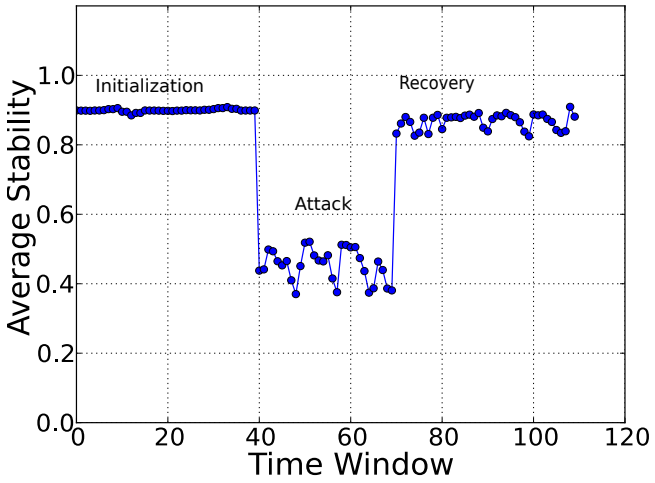


Figure 7.11: Attack injection with recovery phase, $\beta = 300$ seconds, Manhattan topology

7.4.6 Attack Identification

In this experiment, $\beta = 600$ seconds and the objective is to assess the impact of the attack injection level, between 0.5% and 20% of normal vehicles. In Figure 7.10, both the Luxembourg and the Manhattan scenarios are considered. The curve on top of each graph represents the average stability as defined in equation (6.8) when no attack occurs. In such a case values are between 0.85 and 0.95. Logically, when the attack aggressiveness increases, the average stability drops in higher proportion. However, considering 10% of spoofed vehicles, reaching a limit of disturbance for this experiments. On a tiny scale with only 0.5% injection (the number of vehicles injected correspond to 0.5% of total volume of cars), the average stability is lowered around 0.7 and so clearly distinguishable from normal ones. Therefore, setting $\theta = 0.15$ in algorithm 6 is enough to detect stealthy attacks independently of the topology.

7.4.7 Recovery

Previous experiments show that establishing a threshold experimentally is viable. It can be easily determined through learning and past observations of the stability when no attack occurs. Assuming that an attack is detected thanks to this method, this section assesses the ability of our system to recover to a safe state by sanitizing the data from spoofed reported locations. In algorithm 6, this corresponds to remove unstable leaf nodes. Using $\theta = 0.15$ as previously established, Figures 7.11 and 7.12 depict the average stability before the attack, during the attack and when the recovery is performed. When the attack is launched starting from time window 40. Until the time window 75 the attack causes a massive drop of average stability and also produces a great oscillation. In the last phase of the sequence, recovery phase is launched. Despite the attack continues, by removing nodes responsible for the average stability decay, the stability is restored to acceptable values. In reality, recovery phase should start earlier (when the attack is detected) and we voluntary delayed it in this experiment to highlight the changes in the stability value. This experiment was performed using a simulation for Manhattan Grid during 5 hours. α was set to 0.2 and the time window size to 300 seconds.

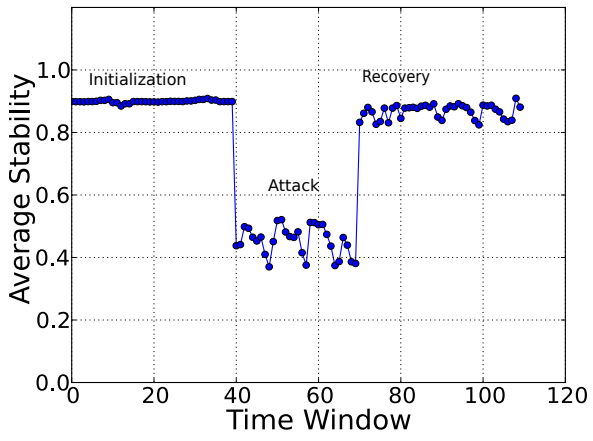


Figure 7.12: Attack injection with recovery phase, $\beta = 300$ seconds, Manhattan topology

While restoring an acceptable stability is simple by removing leaves until the difference between the past observed stability is under the variation threshold θ, it is important to evaluate if the discarded data properly belongs to the attack (true positives) or not (false positives). This is evaluated in Figure 7.13 by calculating the True Positive Rate (TPR) and False Positive Rate (FPR). FP are due to random normal traffic which, at some high attack rates, can be interpreted as an alert.However, most of them occur with excessively high attack rates.

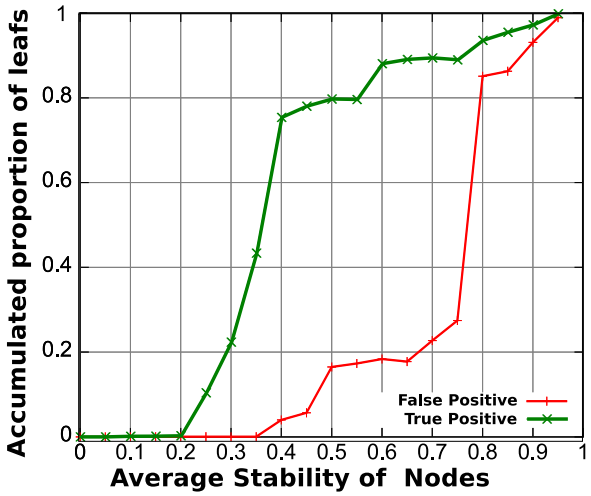


Figure 7.13: Threshold detection for simulation containing attacks with up to 10% of increased traffic

The FP represents the real road traffic that will be discarded and so can have an hight impact on monitoring, undetected traffic jam for example. Such examples can be more discussed to show how FP may impact applications. Where we assess a "single" attack, it corresponds to test every individual one separately, and then compute an average, but all are tested. The FPR and TPR are calculated based on the stability value of the leaf nodes. For instance, assuming $\theta = 0.15$ and a normal stability around 0.9 as highlighted in the first phase of Figure 7.12, this means that we are considering an absolute threshold of 0.75. In such a case, the TPR reaches 89% with less than 27% of false positives. FPR can be drastically reduced using a smaller threshold like 0.4 which leads to a TPR equal 75% and FPR equal 4%.

7.4.8 Performances

As discussed in Section 3.4, page 37, a simple approach could verify if two vehicles are at the same position but this needs further refinement to be viable. However, considering that such a case is possible, this entails to verify pair-wise vehicles. Hence, with V vehicles, $\frac{V(V-1)}{2}$ verifications have to be done. For computing the stability of all nodes of a tree, detecting anomalies and recovering, it has been shown that the complexity is $O(n^2)$ in section 6.5.3, assuming n as the number of nodes in a tree. From a theoretical complexity point of view, this is similar to consider complexity assuming individual vehicles (quadratic complexity in both cases) but, in practice, the tree size is highly lower than the number of vehicles due to the aggregation process. Especially, Figure 7.14 highlights the

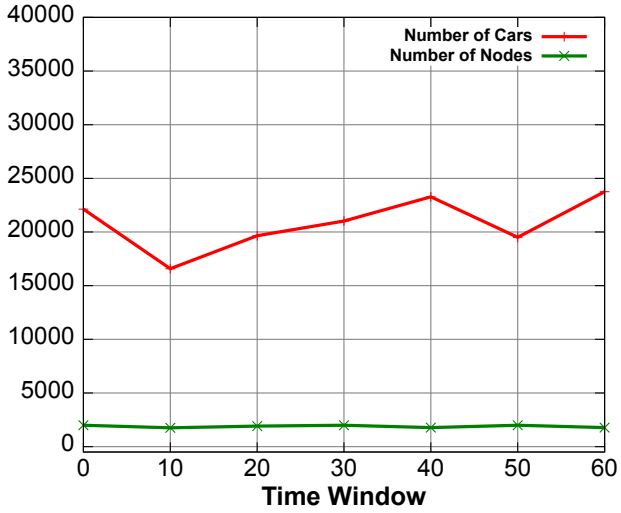


Figure 7.14: Number of cars and size of the trees, 10 hours simulation, $\beta = 600$ seconds

number of individual vehicles and the size of the trees. The latter is at least 1760 and varies around 2000. In contrast, the number of cars is always around 20,000 which leads to around 200 millions comparisons. This is ten times more than using our tree-based approach when $S = 5$ (5×2000^2). In addition, our process also includes the recovery mechanism and so is not only limited to the detection.

7.4.9 Conclusion

This section described the validation of our approach for monitoring position-based services, in particular targeting location spoofing attacks. We validated also a recovery mechanism that can be applied in the context of services as those used in many vehicular applications. We first validated the efficiency of our proposed metrics on multidimensional trees to discriminate between normal traffic and position spoofing attacks. In our first experiments we observed that, the attacks have a significant impact on the average stability of aggregated trees representing the traffic state. Also, we validated that using a threshold-based technique was viable for detecting position spoofing attacks. Further experiments were carried out removing the unstable elements from the trees as recovery technique to mitigate the consequences of an attack.

Additionally, we validated our aggregation approach using various threshold values and window sizes in order to observe and characterize changes in regular traffic stability. We observed that regardless the window size, the average stability always drops when an injection is carried out. In particular, the changes in the average stability become more notorious as the window size increases. Also, we observed that the stability drops, even with a small proportion of injected vehicles. We were able to localize the attack by identifying the nodes in trees causing the stability to drop. Therefore, we validated our approach to safely remove most of the anomalous data while discarding very few benign data.

The scalability of our approach was also addressed in our experiments. Using several urban topologies for simulation and analysis, we were able to observe that the volume of data to analyze was greatly

reduced by using aggregation.

Chapter 8

Flow Management

8.1 Introduction

In this section we show and discuss the results of the evaluation of our approach to leverage network awareness in a SDN cloud-based environment to apply flexible applications-based policies for security requests, availability and performance. To evaluate our approach we first defined a Test bed to host virtual machines and its applications, and evaluate the impact that different policies may have on the performance of flows. We used system and network metrics to study performance changes in terms of computing and networking in several scenarios. For the measurement of system (computing and memory) performance we based our metrics on the native Linux benchmarking tools. For the network performance measurement, we based our metrics in the real throughput obtained during the experiments.

Our experimental validation was conducted in two main phases. Firstly, we have conducted a proof-of-concept experiment to evaluate the feasibility our approach and therefore validate the viability of application-level network awareness using the Augmented controller and the SNDS service. Secondly, we studied its efficiency and the global impact on performance (in terms of CPU and memory , and in terms of network). We set up a scenario for testing and benchmarking the implementation of application-level policies. To avoid bias caused by any specific type of application, we performed our evaluation with several types of applications.

8.2 Test Bed

In this section we describe the Test bed used for the experimental part. This test bed was built to mimic the behaviour of a larger network, with a larger user base and applications running, however the cost of building such a network either virtual or physical is very high. Despite its size and restrictions, our test bed is able reflect how priority and access policies take effect and impact the flow performance. It's important to mention that our choice on software was based on the requirements of the SDN Service. We developed a scalable virtual network using the following network configuration for evaluation and benchmarking, as illustrated in Figure 8.1:

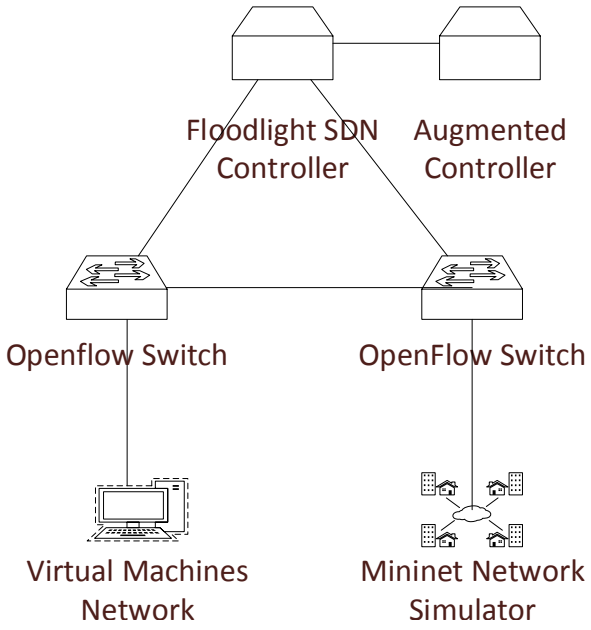


Figure 8.1: Example of a configuration for our Test bed

- SDN Controller: We choose Floodlight Network Controller [173][1] as SDN Controller because of its flexibility implementing rules and its external API.
- OpenFlow Switches: OVFSwitch [183] is an open source virtual switching software, to emulate OpenFlow capable devices for packet forwarding.
- Virtual Machines Network: Based on Virtual Box[2] Virtual Networking, where we plugged the virtual hosts running the the SDN Service and the benchmark applications.
- Network Simulator: implemented using Mininet [184], we used Mininet Network Simulator to emulate a large network of similar hosts running a reduced Linux version. We used this part of the test bed to perform performance and overhead experiments.
- Augmented Controller: it was deployed on a virtual machine running *Ubuntu 14.04 LTS (GNU/Linux 3.13.0-32-generic x86_64)*

8.3 Methodology

The evaluation of our methodology is split into two main categories:

- Networking Performance: We collected data to evidence the network impact of the Augmented Controller in terms of networking metrics, such as Bandwidth, TX Packets and Lost Packets.

[1] http://www.projectfloodlight.org/floodlight/
[2] http://www.oracle.com

- Host's Computing Performance: We collected CPU and RAM consumptions, mainly to evaluate the impact of the SDN Service in term of computing resources.

For evaluating the networking performance of our methodology, we decided to use diverse applications, to represent a heterogeneous network usage. Among them: Iperf [185], Apache2 HTTP Server, Pidgin and Links. The reason for choosing such applications, is to propose a set of applications to evaluate and asses different and diverse network traffic patterns. However, the vast majority of the results presented in this work correspond to those obtained using Iperf. Thus, the usage of Iperf allows to quickly retrieve standard metrics. Therefore, our results are comparable with other results from other existing work mentioned in Section 4.4.1.

For evaluating the Host's system performance and the impact caused by the SDN Service (ie: Linux Daemon), we decided to use the native operative system resources. We recorded the consumption of CPU and RAM during the total length of the experiments. The tool used is the command *ps*.

In order to meter proposed work performance, we recorded CPU and memory consumption per processes as a standard practice. Additionally, most of the results of CPU and memory that are showed in this work are normalized. During this work, all the applications were run on a multi-core architecture but without using a concurrent model. The reason behind of this decision is to keep the results as representative as possible.

8.3.1 Scenarios

The following scenarios were used to evaluate our approach for application-level network awareness. Since we are interested in showing results of the impact on for both system and network performance, we have included scenarios recreating different situations. To evaluate the impact on system performance we focused on recreating an heterogeneous use of the system, by running the following multiple purposes applications

- a web browser and server, to recreate flows where multiple resources are downloaded initially.
- an Internet messenger to recreate flows of small packages in a stream-like pattern.
- Iperf application to recreate bandwidth intensive applications, to emulate large data transfers.

In addition, our scenario targeted at recreating a situation where multiple users make use of limited resources of the network, in particular the available bandwidth and available links. We designed a scenario with two users (*User A* and *User B*) and, a priority policy in favour of one of them. Despite the scenario is succinct, it is representative enough. In particular to show how changing the priority of a user application may affect its network performance in terms of bandwidth available, throughput and packet loss. In the scenario, both users have the same hardware capabilities, therefore there is no differences in terms of networking infrastructure. Both users also share the virtual network infrastructure as peers, without any difference in terms of link capacity.

8.3.2 Metrics

Network Metrics

- Requested Bandwidth: We used as a performance metric the requested bandwidth, by increasing the size of the data to transfer over time, implemented already in OVSwitch and Floodlight. In

our experiments, we tested the requested bandwidth against the real bandwidth. This was observed from the OVSwitch.

- Packet Loss: As a measure of network performance, but also of service degradation, we included the percentage of packet loss for a given application. This includes all the flows that during the execution of an application existed in the network.

- System's Transmitted packets: As an hybrid metric for system and network performance, we collected the number of transmitted packets per unit of time. This metric is highly versatile and it can be collected either from the host system or the networking infrastructure.

System Metrics

- CPU Usage: We recorded the usage of CPU as a percentage from the native tools for benchmarking. Since all the hosts used in our experiments were single core this percentage represents the global usage of the CPU. Additionally, we computed the CPU usage per application and at system level.

- Memory: We recorded the usage of RAM memory per application and its global usage while using the SDNS service and while the service was deactivated.

8.4 Impact on Performance

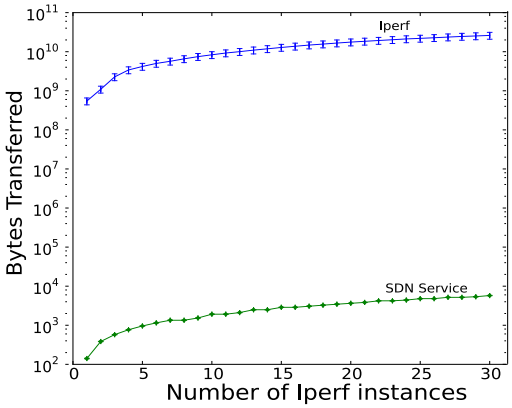


Figure 8.2: Number of bytes transmitted using Iperf

During this section, we will present a set of results for both metric categories described in Section 8.3.2 (system and network performance). Those results come from running several applications, on a variable network configuration environment using the metrics described in the section above.

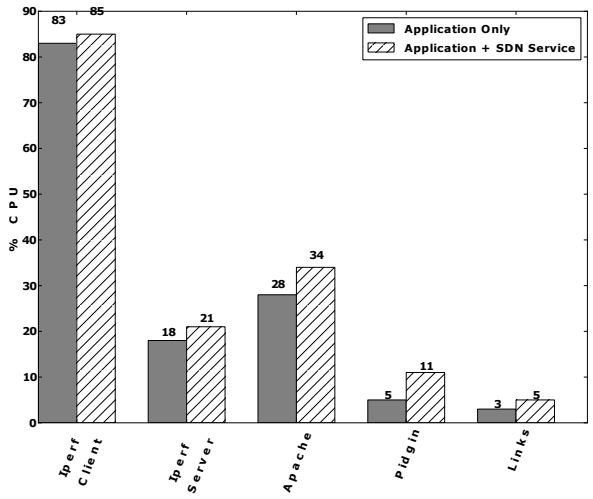


Figure 8.3: Example of the impact on performance upon SDN Service usage

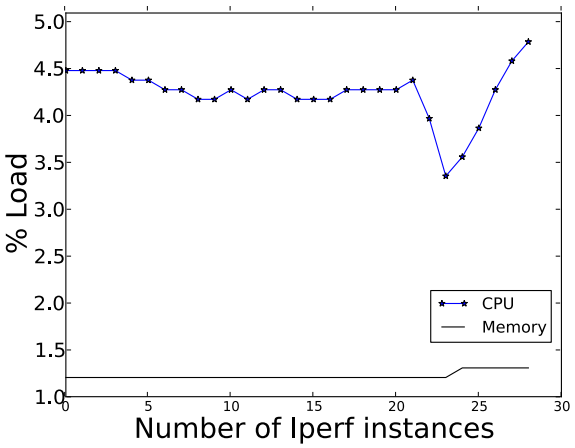


Figure 8.4: SDN Service resource consumption

8.4.1 SDN Service Overhead

In this section we target identifying the overall system's performance impact (in terms of CPU and RAM) of running the SDN Service in a end-user host associated to several applications. The aim, is to identify any significant downgrade of system's performance. This benchmark was performed in the following method:

- Baseline benchmark: We chose several applications among the test set listed in Section 8.3.1. We monitored the CPU and RAM consumptions of the applications running in an undisturbed environment, with no other processes rather than basic system tasks running in background.
- SDN Service running attached to applications benchmark: In this case, we monitored and measured the CPU and RAM consumption of the SDN Service (running in the host). Additionally, we conducted an experiment of running an increasing number of processes of the same application (in this case iperf), an monitored the CPU and Memory consumption of the SDN Service. The results are explained later in this Section and illustrated in Figure 8.4 and Figure 8.2.

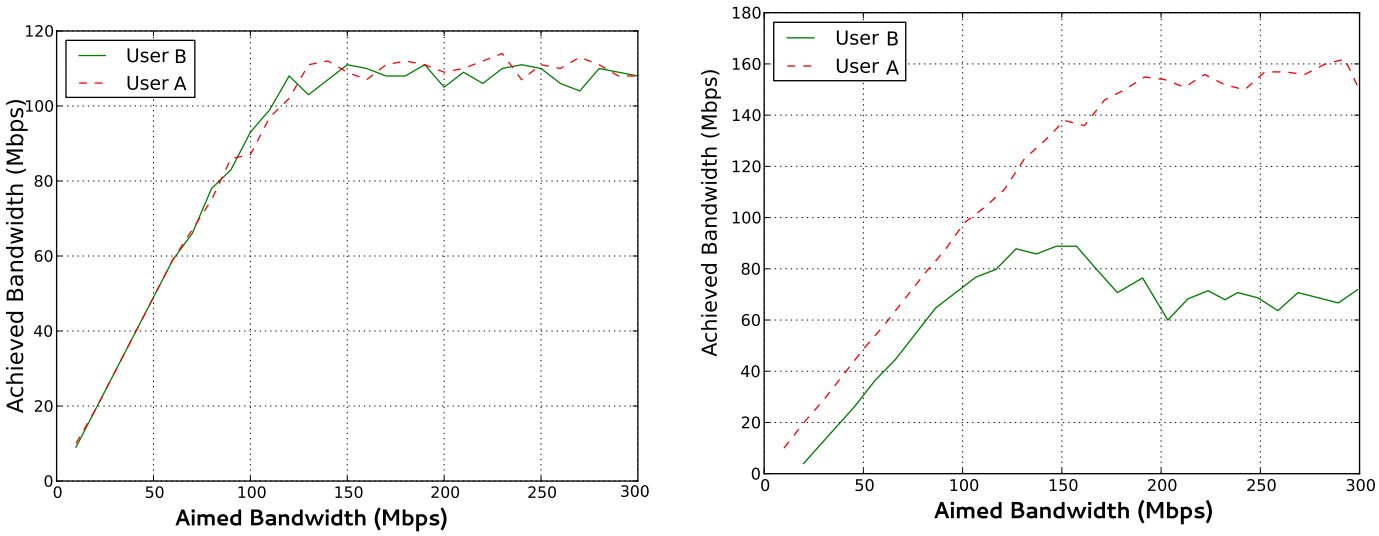
In Figure 8.3, several applications were run with either the SDN Service enabled or disabled. In case the SDN Service is enabled, these application's flows are notified to the Augmented Controller in order to raise network awareness to the SDN Controller. Thus, we monitored the overall impact on CPU consumption as an aggregated value combining the CPU resources used by the application and the SDN Service. As a result from this experiment, we observed a steady trend in CPU consumption between the cases when the SDN Service is enabled and disabled. This steady trend is around an increase of 2% in CPU consumption.

It is possible to observe in Figure 8.4, the impact of the SDN Service is almost neglected in terms of the overall resource consumption. This result, allow us to claim that the SDN Service might not cause a host system performance decay, even in the cases there is a bigger number of processes attached to it and reporting its flows.

8.4.2 Network Performance Impact

For evaluating the impact on the network traffic, we run a set of incremental tests using *Iperf*

Allocated Bandwidth vs Requested



(a) SDN Service Inactive

(b) SDN Service Activated

Figure 8.5: Bandwidth Aimed vs Bandwidth Achieved

To recreate a context where different performance policies can be applied to evaluate the impact on the flows throughput, we designed an scenario with two users (User A and User B), having exactly the same hardware and software, and in connection with the Augmented Controller. A possible approach is to establish two different user's policies to grant a larger use of network resources to one of the users. To evaluate the efficiency of this approach we first evaluated the throughput of the network in normal conditions, and then we introduced the policy and meter the network performance using metrics mentioned in Section 8.3.2. Our scenario, is still representative because is possible to observer the impact of the policies on network performance from a flow perspective. Additionally, overcrowding the scenario with more users may not show clearer neither better interpretable results.

The results presented in this section correspond to the metrics obtained by the following experiment:

- In a similar topology as the one illustrated in Figure 8.1, two users (User A and User B) launch IPerf establishing a bidirectional TCP flow between the host and a IPerf Server running in the network.
- Initially, User A and User B are both using the network without any QoS, or priority system. The network available bandwidth will be shared among both of them. This can be illustrated

in Figure 8.5(a).

- In a next step, User A activates the SDN Service, and it makes a bandwidth allocation request for the given flow. After this action, the Augmented Controller will interface with the Network Controller, attempting to allocate the bandwidth resources requested by User A. The results of this experiment can be seen in Figure 8.5(b).

As it can be observed in Figure 8.5(a) the network capacity is shared equally between *User A* and *User B*. As long as both users increase their bandwidth requirements, the network capacity would reach its limit and would not match the individual user's bandwidth expectation. As it illustrated, the maximum bandwidth achieved per user per flow is limited at 130 MB/s. The applications used in these experiments are greedy in terms of throughput, an they will saturate the capacity besides the Test bed capabilities. In part, this is explained by the virtualization technology used (since all the network interfaces share the internal bus of the virtual host computer). In a bare metal installation the users should be able to reach the theoretical maximum of the medium. Which, in case of a Gigabit network will be significantly higher than in our test bed.

In Figure 8.5(b), is possible to observe the allocated bandwidth for the flows belonging to *User A* Iperf. Since, for this experiment we introduced a rule to inform the controller to prioritize this instances Iperf, this is implemented with a QoS queue in the SDN Floodlight controller, allowing *User A* to achieve a better network performance than *User B*. As illustrated in Figure 8.5(b) *User A* is still limited by the virtual network capacity but achieved a better network performance in terms of throughput than *User B*, which only reaches to transfer at rates of 60 to 80 MB/s. Another phenomenon observable in the results, is the stability of the transfer once the rules have been made effective at the SDN Controller.

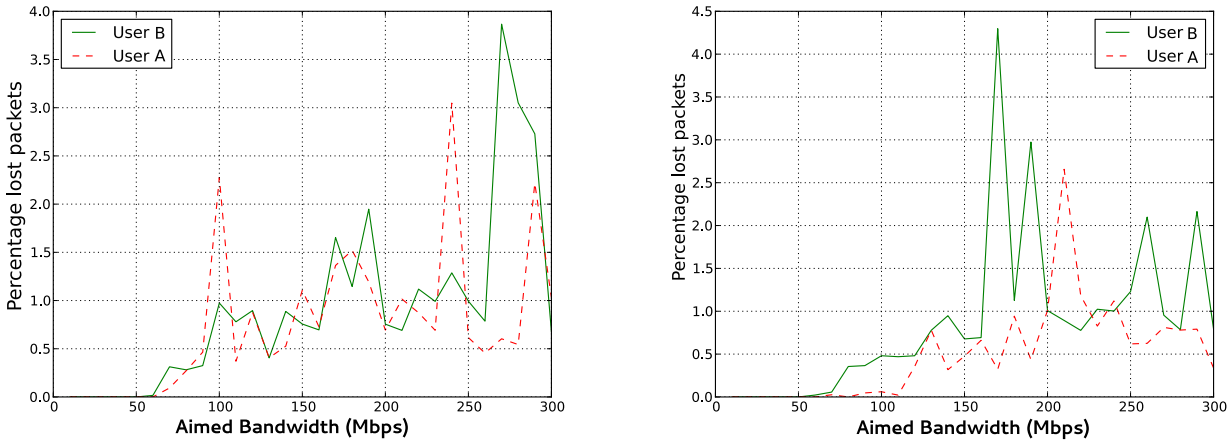
The implementation of QoS rules can be ported by implementing the rules at other SDN controllers (such as NOX or POX [186]). Is is important to notice, that once a flow is out in the network there are not efficient mechanisms for pausing it or halt the responsible applications. However, using the SDNS Service is possible to synchronize the application (prior modification) in order to make a better usage of the network resources.

Packet loss

Analogously, to observe how the Augmented Controller in addition with the SDN Service could affect the Packet loss rate, we conducted a similar experiment as in Section 8.4.2. The topology used for this experience is similar as the one showed in Figure 8.1. Also, in the same way the Iperf instances belonging to *User A*, receive a higher priority, having as a result a lower rate of packet loss.

8.5 Conclusion

In this chapter we have validated of our approach to leverage network awareness in a SDN cloud-based environment to apply flexible applications-based policies. We first conducted a proof-of-concept to validate the feasibility of our approach. To validate our methodology and conduct the experiments we developed a Test bed representing multiple scenarios, including representative applications such as *IPerf*, Web Browsers and an Internet messenger.



(a) SDN Service Inactive

(b) SDN Service Activated

Figure 8.6: Bandwidth Aimed vs Packet loss

We were able to observe that the SDN Service and Augmented Controller do not cause a large impact in terms of performance according to our measurement in network and system benchmarks. Subsequently, we were able to validate that large performance gain in data transfer rate be achieved by implementing QoS application-based policies available in our approach. This last result was observed by measurements of requested throughput, achieved bandwidth and package loose rate. Our approach consists in an architecture that can be developed into a data centre for optimizing data analytics and Cloud-based applications. However, it is yet to be adopted by large industry players to see its full potential.

Conclusion

Chapter 9

Conclusions and Perspectives

Conclusion

Network Security The initial parts of this thesis described several methods and models to aggregate multidimensional data. The importance of aggregation is highlighted by the growing volumes of data needed to be analyzed in large networks such as at ISPs level. At such levels, diverse sources of information are available to report the different types of events that can occur in a network. Hence, analyzing and correlating the events observed at different levels of the infrastructure become of a major importance. However, the scale of the volume of information and its diversity can bring strong limitations to the existing approaches.

The first contribution made of this thesis consisted in methods for monitoring and analysis of distributed applications combining multiple sources of information in a highly flexible manner, *i.e.* the dynamic granularity and multidimensionality of data are deal automatically with few human exper tuning. For example, to aggregate network traffic from using IP address, defining the size of the subnet could be required. However, our approach allows the data to be aggregated with a flexible granularity without pre partitioning the space a priori. A second advantage of our approach consists in not considering any specific order among the features to aggregate. This is beneficial to network administrator and data specialists because it eases the data collection process. As for example, a network administrator could potentially use our approach to analyze traffic patterns without having to specify if he wants to analyze first source and then destination IP addresses. A third aspect of our approach is the ability to reduce the scale of data reducing the data loss, this becomes critical for post processing, where scalability is crucial. With MAM we reduced the scale of large data sets for traffic analysis and its further evaluation. In particular, we focused on three scenarios (TCP/IP networks, DNS and crowd-sourced position-based applications for vehicular traffic).

As a first case study, we applied the spatial and temporal aggregation methods to analyze TCP/IP network traffic in order to leverage anomaly detection. At ISP level, NetFlow-based approaches for accounting and monitoring are widely used, this is explained by the volume of traffic present at an ISP. For this case, we introduced a model for representing NetFlow records using several fields such as IP addresses, port, and application-type, which we analyzed using a multidimensional approach based on spatial and temporal aggregated tree distance. For this case study, we extended the notion of edit distance of strings to multidimensional aggregated trees. By studying the dynamics

of traffic patterns in TCP/IP networks using the tree distance, we developed our methodology for anomaly detection, which we later validated during experiments using traces from a Luxembourg-based ISP. The experiments results suggested that our methodology is efficient for detecting anomalies in aggregated network traffic corresponding to DDoS attacks and TCP Flood. Additionally, during this case study we validated that the order in which data is used for the tree construction and subsequent aggregation process does not have a significant impact in the overall process. Our approach can leverage detection techniques with multidimensional aggregation-based analysis on other domains.

As a second use case, we analyzed and evaluated the match between DNS and IP subnetworks to find out potential malicious websites. Phishing is a prominent type of Internet-based attack targeting individuals and used to carry out financial fraud. Simultaneously, FastFlux attacks are also of a major security concern, since they are carried out to hide the control centre of botnets. Theses types of attacks operate using a rapid changing set of IP addresses, which are associated one or more domain names. Several limitations arise while monitoring these types of attacks. Initially, it is difficult to manage the granularity for monitoring the rapid changing set of IP addresses associated to a suspected domain name. Additionally, the volume of data to be analyzed can reach large proportions, therefore, efficiency is of a major concern for detecting these attacks on time. In this scenario, we proposed to apply multidimensional aggregation, which can cope efficiently with a large volume of data and flexible granularity to leverage the analysis of DNS records and IP subnetworks. To carry out our approach, we developed a set of metrics for assessing the persistence over time (steadiness) of the match between domain names and subnetworks. Therefore, after conducting short-term and year-long field experiments we were able to validate the efficiency of our metrics. The results of our experiments suggested that our approach is valid for detecting malicious websites with an acceptable rate of false positives for a given threshold of steadiness.

Last but not least, a third case study is to use multidimensional aggregation to leverage the analysis and detection of anomalies on crowd-sourced position-based applications for vehicular traffic. Nowadays, the high adoption rate of smart phones and GPS-enabled mobile devices has led to a vast spectrum of mobile applications for vehicle routing and personal navigation. A large portion of these applications has loose access restrictions; almost any user can subscribe to these applications without a high cost neither strict identity control. However, in this scenario malicious users may create multiple fake accounts (Sybil), and forge the theirs positions to alter the traffic estimations. Stricter access policy can detract users from joining these applications and in some cases it can affect the privacy of the users. Also, enforcing a plausibility model at the infrastructure level is not trivial, since usually it is controlled by network operators and not by the application server-side. Additionally, performing location verification at a mobile device level can be very costly in terms of computing and network resources. Therefore, assuming a central repository of geo-locations our approach can be used to analyze the distribution of traffic in an aggregated manner and leverage the detection of anomalies including position spoofing attacks. To carry our detection approach, we developed and validated experimentally metrics based on multidimensional aggregated trees. Additionally, we were able to identify the anomalies and we proposed a method for recovering a portion of the data prior to be injected with spoofed positions.

Our contributions to network monitoring and security of distributed applications gain relevance in a context of exceptional increase of network requirements in terms of bandwidth and throughput, as it was highlighted by many forecasts done recently. This increase also affects the security requirements,

since more volume of data has to be analyzed and attacks become more complex. In this scenario, efficient data analytic approaches are needed to leverage monitoring, analysis and evaluation of events, as for example at an Intrusion Detection System level for forensics and detection. Recent advances in distributed computing and cloud-based approaches have been proposed in the field of data analytics for network security and monitoring. Therefore, our approach is to leverage cloud-based applications (in particular, for data analytics) enabling application-level awareness for Software Defined Networking.

Network Management In recent years Cloud computing has emerged as a solution for centralized computing and storage of private and personal information. Many cloud-based applications are hosted in dedicated computing facilities (data centres), which have specific network topologies to match certain high bandwidth requirements. Simultaneously, distributed computing paradigms as for example MapReduce have been widely used to maximize the performance of data analytic applications in data centres. Software Defined Networks play an important role in this scenario by decoupling the control plane from the data plane, allowing a greater flexibility for defining and applying flow-based policies. However, an Software Defined Networks (SDN) controller is not able to take fully optimized decision without knowing about the context of the flows. In this scenario our approach consists in a method and its framework to leverage application-based policies with network awareness in a SDN-based cloud. Our framework allows the SDN controller to learn about the flows context and take more optimized decision. Implementing the framework has validated the feasibility of our approach. The context of flows is related to the application or the users, which has been its initiator. Hence, we provide an interface in the network control plane to distinguish between flows based on users and applications. Our evaluation results also reveal that our proposed framework has very little footprint, while enabling the network to take much optimized decision to meet QoS requirements of applications. We have also shown through experimentation that our framework leaves very low compute, memory, and network footprint.

Perspectives

This section outlines some possible research perspectives open by our contributions. In this thesis we have introduced a methodology for multidimensional spatial and temporal aggregation. This technique can be used in many fields for the analysis of events, as for example: accounting, provisioning, security, forensics or other uses. Our aggregation method is based on a threshold regarding the activity volume (aggregation α), thus the automatic estimation of the aggregation threshold is a challenging problem. Additionally, since the multidimensional aggregation process combines multiple sources of information, setting the threshold based on rules per activity source is also a possible direction for research.

Another research problem could be to carry out an aggregation process using a variable window, which could be defined by rules on the events, as for example, until certain number of events is reached. Another possible rule could be until a number of events of a given type or having a given attribute occurs. For example, until certain number of network packets of a given protocol or host are on the medium. A variable time window could put in evidence the rate of activity at which events take place, which would be used to estimate consumption trends in various fields as social networks, distributed Vehicular Ad Hoc Networks (VANETs) simulations or other computing process with a large number of communication events.

A fundamental part of this thesis was the research of data analytic techniques for detection of anomalies and potentially malicious data. Research problems can be investigated towards more robust metrics to lower the rate of false positives while performing detection. As for example, in the context of spoofing position detection in crowd-sourced based applications, a possible improvement is to consider also the speed. This can reinforce the plausibility checks.

In this thesis we validated our multidimensional aggregation approach in three case studies, which in the short term they can be extended to not yet fully adopted protocols, such as IPv6 and DNSSEC. Additionally, a legitimate research problem is to include as additional sources of information text structure or human language, because it is extremely abundant in the Internet and reflects many patterns in human behaviours (such as global events as civil events, natural catastrophes and cultural activities). Additionally, human language could be also interpreted as hierarchical data. This dimension can be used together with network activity to study the propagation of content and viralization in social media. Another possibility is to include as a dimension, traces from the applications' execution call stacks, which are also hierarchical and sometimes can reach very large volumes of data.

From the computational perspective, some of the algorithms require long run times to yield their results. A possible optimization is to implement the multidimensional aggregation process in a distributed fashion. As for example using the framework Hadoop[1] to implement it as a MapReduce tasks, or using Storm[2] in a queue-based solution for streaming data. Additionally, this implementation could be hosted at a data centre using an SDN-based cloud, in which our proposed framework for application awareness can be also deployed.

From a network management perspective, it is yet to be explored how to leverage application-awareness in a multi-tenant virtualized data centre. One possible approach can be to establish a virtual machine QoS policy using our proposed framework to implement it. Another challenging topic is how to perform resource management to maximize the usage of the resources in compliance with the application requirements. Using the contributions of this thesis, a promising field is to include at the application-level the state of the network to orchestrate the launch of MapReduce-based tasks.

[1] http://www.hadoop.apache.org/
[2] http://www.storm.apache.org/

List of Tables

List of Figures

Acronyms

DDoS Distributed Denial of Service. 9, 30DNS Domain Name Service. 8, 10, 25–27, 31, 33, 34, 36, 38, 92, 115
buffer overflow flaw in a binary executable file that allows a malicious user to write beyond the application m
disseminating node Host participating in a VANET and actively sharing its position. 41
exabyte 10^{18} bytes or one million terabytes. The symbol for the exabyte is EB. 7
Linux is a generic term referring to the family of Unix-like computer operating systems that use the Linux ker
Man in the middle An attacker is positioned as an intermediary between the communicating parties. 30, 41
password cracking The process of revealing a password by either brute forcing or cryptanalysis. 31**phishing**
session forging The process of gaining access privileges by stealing an legitimate active session from a user. 3
XSS Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into oth
zettabyte 10^{21} bytes or one thousand exabytes. The symbol for the zetabyte is ZB. 8

Bibliography

[1] Cisco Inc, "Cisco visual networking index: Forecast and methodology, 20132018," 2013. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf

[2] ——, "Cisco visual networking index: Global mobile data traffic forecast update 20142019 white paper," 2014. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

[3] A. Networks, "Annual security report - worldwide infrastructure security report," 2014. [Online]. Available: http://www.arbornetworks.com/resources/infrastructure-security-report

[4] K. LAB, "It security risks survey 2014: A business approach to managing data security threats," 2014. [Online]. Available: http://media.kaspersky.com/en/IT_Security_Risks_Survey_2014_Global_report.pdf

[5] A.-P. W. Group, "Phishing activity trends report 1st quarter 2014," 2014. [Online]. Available: http://docs.apwg.org/reports/apwg_trends_report_q1_2014.pdf

[6] K. LAB, "Spam and phishing statistics report q1-2014," 2014. [Online]. Available: http://usa.kaspersky.com/internet-security-center/threats/spam-statistics-report-q1-2014#.VQXMyRDF_pA

[7] Big Data Working Group, "Big data analytics for security intelligence," 2013. [Online]. Available: https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Analytics_for_Security_Intelligence.pdf

[8] A. Cole, "Network infrastructure concerns for big data," 2014. [Online]. Available: http://www.enterprisenetworkingplanet.com/datacenter/network-infrastructure-concerns-for-big-data.html

[9] Cisco Inc, "Cisco global cloud index:forecast and methodology, 20132018," 2013. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html

[10] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 67–72. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342455

[11] G. Wang, T. E. Ng, and A. Shaikh, "Programming your network at run-time for big data applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 103–108. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342462

[12] U. Wu, J. Tian and A. K., "Forecast: x86 server virtualization, worldwide, 3q12 update," 2013. [Online]. Available: https://www.gartner.com/doc/2210316/forecast-x-server-virtualization-worldwide

[13] Lee Doyle, "SDN use cases emerge across the LAN, WAN and data center," 2012. [Online]. Available: http://searchsdn.techtarget.com/essentialguide/SDN-use-cases-emerge-across-the-LAN-WAN-and-data-center

[14] C. Wagner, J. Francois, T. Engel *et al.*, "DANAK: Finding the odd!" in *Network and System Security (NSS), 2011 5th International Conference on*. IEEE, 2011, pp. 161–168.

[15] J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, "Botcloud: detecting botnets using mapreduce," in *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*. IEEE, 2011, pp. 1–6.

[16] Y. Hu, D.-M. Chiu, and J. C. S. Lui, "Entropy based adaptive flow aggregation," *IEEE/ACM Trans. Netw.*, vol. 17, no. 3, pp. 698–711, 2009.

[17] G. Moreira Moura, R. Sadre, A. Sperotto, and A. Pras, "Internet bad neighborhoods aggregation," 2012.

[18] K. Cho, R. Kaizaki, and A. Kato, "Aguri: An aggregation-based traffic profiler," in *Quality of Future Internet Services*. Springer, 2001, pp. 222–242.

[19] Cisco Inc, "Introduction to cisco ios NetFlow," 2012. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.pdf

[20] ——, "Cisco 2014 annual security report," 2014. [Online]. Available: http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf

[21] S. Kumar, "Smurf-based distributed denial of service (DDoS) attack amplification in internet," in *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on*. IEEE, 2007, pp. 25–25.

[22] H. Choi and H. Lee, "Identifying botnets by capturing group activities in DNS traffic," *Comput. Netw.*, vol. 56, no. 1, pp. 20–33, Jan. 2012.

[23] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in dns traffic," in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*. IEEE, 2007, pp. 715–720.

[24] S. Marchal, J. François, R. State, and T. Engel, "Proactive discovery of phishing related domain names," in *Recent Advances in Intrusion Detection*, ser. LNCS. Springer, 2012. [Online]. Available: http://lorre.uni.lu/~jerome/files/raid12.pdf

[25] S. Hao, N. Feamster, and R. Pandrangi, "Monitoring the initial DNS behavior of malicious domains," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. New York, NY, USA: ACM, 2011.

[26] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis," in *Network and Distributed System Security Symposium - NDSS*, 2011.

[27] F. Weimer, *Passive DNS replication*, 2005.

[28] J. Levine, "DNS Blacklists and Whitelists," RFC 5782 (Informational), Internet Engineering Task Force, Feb. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5782.txt

[29] B. Intelligence, "The connected car report: Forecasts, competing technologies, and leading manufacturers," 2015. [Online]. Available: http://uk.businessinsider.com/connected-car-forecasts-top-manufacturers-leading-car-makers-2015-3?r=US

[30] N. H. T. S. A. U.S. Department of Transportation, "Vehicle-to-vehicle communications: Readiness of v2v technology for application," 2014. [Online]. Available: http://www.nhtsa.gov/staticfiles/rulemaking/pdf/V2V/Readiness-of-V2V-Technology-for-Application-812014.pdf

[31] T. Li, S. Hazra, and W. Seah, "A position-based routing protocol for metropolitan bus networks," in *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, 2005.

[32] I. Sumra, H. Hasbullah, and J. Manan, "Vanet security research and development ecosystem," in *National Postgraduate Conference (NPC), 2011*, sept. 2011, pp. 1 –4.

[33] T. H.-J. Kim, A. Studer, R. Dubey, X. Zhang, A. Perrig, F. Bai, B. Bellur, and A. Iyer, "VANET alert endorsement using multi-source filters," in *International workshop on VehiculAr InterNETworking*, ser. VANET. ACM, 2010.

[34] G. Yan, S. Olariu, and M. C. Weigle, "Providing VANET security through active position detection," *Comput. Commun.*, vol. 31, no. 12, 2008.

[35] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful gps spoofing attacks," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 75–86. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046719

[36] G. Samara, W. Al-Salihy, and R. Sures, "Security issues and challenges of vehicular ad hoc networks (vanet)," in *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on*, may 2010, pp. 393 –398.

[37] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, Barcelona, Spain, October 2011, pp. 63–68.

[38] M. Armbrust, A. Fox, Griffith *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[39] P. Fenwick, "A new data structure for cumulative frequency tables," *Software: Practice and Experience*, vol. 24, no. 3, pp. 327–336, 1994.

[40] D. Morrison, "PATRICIA - practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.

[41] R. Finkel and J. Bentley, "Quad trees: a data structure for retrieval on composite keys," *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[42] A. V. Aho, J. E. Hopcroft, and J. Ullman, *Data Structures and Algorithms*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983.

[43] W. Bowman, "System and method for detecting fraudulent network usage patterns using real-time network monitoring," May 6 1997, uS Patent 5,627,886. [Online]. Available: https://www.google.com/patents/US5627886

[44] H. Lim, M. Y. Kang, and C. Yim, "Two-dimensional packet classification algorithm using a quad-tree," *Comput. Commun.*, vol. 30, no. 6, pp. 1396–1405, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2007.01.004

[45] E. Tanin, A. Harwood, and H. Samet, "Using a distributed quadtree index in peer-to-peer networks," *The VLDB Journal*, vol. 16, no. 2, pp. 165–178, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1007/s00778-005-0001-y

[46] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[47] J. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[48] K. Sklower, "A Tree-Based Packet Routing Table for Berkeley Unix," in *USENIX Technical Conference*, 1991, pp. 93–104.

[49] R. De La Briandais, "File searching using variable length keys," in *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, ser. IRE-AIEE-ACM '59 (Western). New York, NY, USA: ACM, 1959, pp. 295–298. [Online]. Available: http://doi.acm.org/10.1145/1457838.1457895

[50] G. C. Moreira Moura, R. Sadre, A. Sperotto, and A. Pras, "Internet bad neighborhoods aggregation," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2012)*, April 2012.

[51] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, p. 37, 1996.

[52] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, "CRISP-DM 1.0 step-by-step data mining guide," 2000.

[53] L. A. KURGAN and P. MUSILEK, "A survey of knowledge discovery and data mining process models," *The Knowledge Engineering Review*, vol. 21, pp. 1–24, 3 2006. [Online]. Available: http://journals.cambridge.org/article_S0269888906000737

[54] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Advances in knowledge discovery and data mining," 1996.

[55] D. Michie, D. Spiegelhalter, and C. Taylor, *Machine Learning, Neural and Statistical Classification*. Prentice Hall, 1994.

[56] C. M. Bishop *et al.*, *Pattern recognition and machine learning*. springer New York, 2006, vol. 1.

[57] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[58] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011. [Online]. Available: http://dx.doi.org/10.1002/widm.30

[59] P. Zikopoulos, C. Eaton *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[60] C. Snijders, U. Matzat, and U.-D. Reips, "Big data: Big gaps of knowledge in the field of internet science," *International Journal of Internet Science*, vol. 7, no. 1, pp. 1–5, 2012.

[61] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, A. H. Byers, and M. G. Institute, "Big data: The next frontier for innovation, competition, and productivity," 2011.

[62] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, May 2010, pp. 94–103.

[63] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[64] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: A survey," *SIGMOD Rec.*, vol. 40, no. 4, pp. 11–20, Jan. 2012.

[65] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, p. 21, 2007.

[66] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in *SIGMOD International Conference on Management of Data*. ACM, 2014.

[67] P. Mockapetris, "Domain names - implementation and specification," RFC 1035 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[68] ——, "Domain names - concepts and facilities," RFC 1034 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt

[69] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, and M. Karir, "Atlas internet observatory 2009 annual report," *Arbor Networks, the University of Michigan and Merit Network, Tech. Rep*, 2009.

[70] G. Blank and D. Groselj, "Dimensions of internet use: amount, variety, and types," *Information, Communication & Society*, vol. 17, no. 4, pp. 417–435, 2014. [Online]. Available: http://dx.doi.org/10.1080/1369118X.2014.889189

[71] G. Blank and W. H. Dutton, "Next generation users: Changing access to the internet," *Selected Papers of Internet Research*, vol. 3, 2013.

[72] The World Bank, "World development indicators," 2014. [Online]. Available: http://data.worldbank.org/indicator/IT.NET.USER.P2/countries/1W?display=graph

[73] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101 (Proposed Standard), Internet Engineering Task Force, Jan. 2008, obsoleted by RFC 7011. [Online]. Available: http://www.ietf.org/rfc/rfc5101.txt

[74] ——, "Cisco Systems NetFlow Services Export Version 9," RFC 3954 (Informational), Internet Engineering Task Force, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[75] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Rfc 3954: Cisco systems NetFlow services export version 9," Tech. Rep., 2004.

[76] C. Wagner, "Security and network monitoring based on internet flow measurements," Ph.D. dissertation, University of Luxembourg, Luxembourg, Luxembourg, 2012, http://hdl.handle.net/10993/15492.

[77] J. W. Jorgensen, "IP-flow classification in a wireless point to multi-point (PTMP) transmission system," Sep. 17 2002, uS Patent 6,452,915.

[78] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *Communications Surveys & Tutorials, IEEE*, vol. 12, no. 3, pp. 343–356, 2010.

[79] C. Morariu, T. Kramis, and B. Stiller, "DIPStorage: Distributed storage of IP flow records," in *Local and Metropolitan Area Networks, 2008. LANMAN 2008. 16th IEEE Workshop on.* IEEE, 2008, pp. 108–113.

[80] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better NetFlow," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 245–256, 2004.

[81] I. Paredes-Oliva, P. Barlet-Ros, and J. Solé-Pareta, "Portscan detection with sampled netflow," *Traffic Monitoring and Analysis*, pp. 26–33, 2009.

[82] P. Giura and N. Memon, "Netstore: An efficient storage infrastructure for network forensics and monitoring," in *Recent Advances in Intrusion Detection*. Springer, 2010, pp. 277–296.

[83] C. Simmons, C. Ellis, S. Shiva, D. Dasgupta, and Q. Wu, "AVOIDIT: A cyber attack taxonomy," 2009.

[84] M. Wood and M. Erlinger, "Intrusion Detection Message Exchange Requirements," RFC 4766 (Informational), Internet Engineering Task Force, Mar. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4766.txt

[85] H. Debar, D. Curry, and B. Feinstein, "The Intrusion Detection Message Exchange Format (IDMEF)," RFC 4765 (Experimental), Internet Engineering Task Force, Mar. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4765.txt

[86] N. Brownlee and E. Guttman, "Expectations for Computer Security Incident Response," RFC 2350 (Best Current Practice), Internet Engineering Task Force, Jun. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2350.txt

[87] J. P. Anderson, "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.

[88] A. Simmonds, P. Sandilands, and L. Van Ekert, "An ontology for network security attacks," in *Applied Computing*. Springer, 2004, pp. 317–323.

[89] E. H. Spafford, "The internet worm program: An analysis," *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 1, pp. 17–57, 1989.

[90] D. Moore, C. Shannon, and k. claffy, "Code-red: A case study on the spread and victims of an internet worm," in *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, ser. IMW '02. New York, NY, USA: ACM, 2002, pp. 273–284. [Online]. Available: http://doi.acm.org/10.1145/637201.637244

[91] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Computing Surveys (CSUR)*, vol. 39, no. 1, p. 3, 2007.

[92] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *Network and Distributed System Security Symposium (NDSS)*, 2008.

[93] H.-T. Lin, Y.-Y. Lin, and J.-W. Chiang, "Genetic-based real-time fast-flux service networks detection," *Computer Networks*, vol. 57, no. 2, pp. 501 – 513, 2013.

[94] P. Sroufe, S. Phithakkitnukoon, R. Dantu, and J. Cangussu, "Email shape analysis for spam botnet detection," in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE.* IEEE, 2009, pp. 1–2.

[95] E. Rescorla and A. Schiffman, "Security Extensions For HTML," RFC 2659 (Experimental), Internet Engineering Task Force, Aug. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2659.txt

[96] D. Atkins and R. Austein, "Threat Analysis of the Domain Name System (DNS)," RFC 3833 (Informational), Internet Engineering Task Force, Aug. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3833.txt

[97] C. Wagner, J. François, R. State, T. Engel, A. Dulaunoy, and G. Wagener, "SDBF: Smart DNS Brute-Forcer," in *IEEE/IFIP Network Operations and Management Symposium - NOMS*, 2012.

[98] J. François, S. Wang, R. State, and T. Engel, "BotTrack: Tracking botnets using NetFlow and PageRank," in *IFIP/TC6 NETWORKING 2011*, Springer, Ed., Valencia, Spain, May 2011. [Online]. Available: http://hal.inria.fr/inria-00613597/en/

[99] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining in a data-flow environment: Experience in network intrusion detection," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '99. New York, NY, USA: ACM, 1999, pp. 114–124. [Online]. Available: http://doi.acm.org.proxy.bnl.lu/10.1145/312129.312212

[100] R. Kaizaki, K. Cho, and O. Nakamura, "Detection of denial of service attacks using AGURI," in *International Conference Telecommunications, Beijing China*, 2002.

[101] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web." 1999.

[102] R. Bush, D. Karrenberg, M. Kosters, and R. Plzak, "Root Name Server Operational Requirements," RFC 2870 (Best Current Practice), Internet Engineering Task Force, Jun. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2870.txt

[103] N. Alexiou, S. Basagiannis, P. Katsaros, T. Dashpande, and S. A. Smolka, "Formal analysis of the kaminsky DNS cache-poisoning attack using probabilistic model checking," in *International Symposium on High-Assurance Systems Engineering (HASE).* IEEE, 2010.

[104] S. Marchal, J. François, C. Wagner, R. State, A. Dulaunoy, T. Engel, and O. Festor, "DNSSM: A large-scale Passive DNS Security Monitoring Framework," in *IEEE/IFIP Network Operations and Management Symposium*, 2012.

[105] R. Perdisci, I. Corona, and G. Giacinto, "Early detection of malicious flux networks via large-scale passive DNS traffic analysis," *Transactions on Dependable and Secure Computing*, pp. 714–726, 2012.

[106] A. Berger and E. Natale, "Assessing the real-world dynamics of DNS," in *International conference on Traffic Monitoring and Analysis (TMA).* Springer, 2012.

[107] A. Caglayan, M. Toothaker, D. Drapeau, D. Burke, and G. Eaton, "Real-time detection of fast flux service networks," in *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*. IEEE, 2009, pp. 285–292.

[108] H. Orman and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys," RFC 3766 (Best Current Practice), Internet Engineering Task Force, Apr. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3766.txt

[109] Anti-Phishing Working Group and others, "Phishing Activity Trends Report - 1H2011," 2011.

[110] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, II, and D. Dagon, "Detecting malware domains at the upper dns hierarchy," in *Proceedings of the 20th USENIX conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 27–27.

[111] I. Security and S. A. Committee, "SAC 025 - SSAC advisory on fast flux hosting and dns," 2008. [Online]. Available: https://www.icann.org/en/system/files/files/sac-025-en.pdf

[112] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[113] L. Deri, L. L. Trombacchi, M. Martinelli, and D. Vannozzi, "Towards a passive DNS monitoring system," in *Symposium on Applied Computing (SAC)*. ACM, 2012.

[114] S. Marchal, J. François, C. Wagner, and T. Engel, "Semantic exploration of DNS," in *IFIP/TC6 Networking 2012*, Prague - Czech Republic, 2012.

[115] S. Marchal, J. François, R. State, and T. Engel, "Semantic based DNS Forensics," in *Workshop on Information Forensics and Security - WIFS*, IEEE, Ed., Tenerife, Spain, 2012. [Online]. Available: http://lorre.uni.lu/~jerome/files/wifs12_semantic1.pdf

[116] S. Garera, N. Provos, M. Chew, and A. Rubin, "A framework for detection and measurement of phishing attacks," in *Workshop on Recurring malcode*. ACM, 2007.

[117] A. Blum, B. Wardman, T. Solorio, and G. Warner, "Lexical feature based phishing url detection using online learning," in *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, ser. AISec '10. New York, NY, USA: ACM, 2010, pp. 54–60. [Online]. Available: http://doi.acm.org/10.1145/1866423.1866434

[118] B. Gyawali, T. Solorio, M. Montes-y Gómez, B. Wardman, and G. Warner, "Evaluating a semisupervised approach to phishing URL identification in a realistic scenario," in *Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*, ser. CEAS '11. New York, NY, USA: ACM, 2011, pp. 176–183. [Online]. Available: http://doi.acm.org/10.1145/2030376.2030397

[119] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," in *USENIX Security*, 2010.

[120] B. Zdrnja, N. Brownlee, and D. Wessels, "Passive monitoring of DNS anomalies," in *International conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 2007.

[121] P. Prakash, M. Kumar, R. Kompella, and M. Gupta, "Phishnet: predictive blacklisting to detect phishing attacks," in *INFOCOM*. IEEE, 2010.

[122] S. Hao, M. Thomas, V. Paxson, N. Feamster, C. Kreibich, C. Grier, and S. Hollenbeck, "Understanding the domain registration behavior of spammers," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 63–76. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504753

[123] T. Leinmuller, R. Schmidt, E. Schoch, A. Held, and G. Schafer, "Modeling roadside attacker behavior in VANETs," in *GLOBECOM Workshops, 2008 IEEE*, 2008.

[124] S. Čapkun, L. Buttyán, and J.-P. Hubaux, "SECTOR: secure tracking of node encounters in multi-hop wireless networks," in *Workshop on Security of ad hoc and sensor networks*, ser. SASN '03. ACM, 2003.

[125] N. Sastry, U. Shankar, and D. Wagner, "Secure verification of location claims," in *Workshop on Wireless security - WiSe*. ACM, 2003.

[126] B. Xiao, B. Yu, and C. Gao, "Detection and localization of sybil nodes in VANETs," in *Workshop on Dependability issues in wireless ad hoc networks and sensor networks – DIWANS*. ACM, 2006.

[127] C. Chen, X. Wang, W. Han, and B. Zang, "A robust detection of the sybil attack in urban VANETs," in *International Conference on Distributed Computing Systems Workshops*. IEEE Computer Society, 2009.

[128] T. Leinmuller, E. Schoch, and F. Kargl, "Position verification approaches for vehicular ad hoc networks," *Wireless Communications, IEEE*, vol. 13, no. 5, pp. 16–21, 2006.

[129] J.-H. Song, V. Wong, and V. C. M. Leung, "Secure location verification for vehicular ad-hoc networks," in *Global Telecommunications Conference - GLOBECOM IEEE*, 2008.

[130] H.-C. Hsiao, A. Studer, C. Chen, A. Perrig, F. Bai, B. Bellur, and A. Iyer, "Flooding-resilient broadcast authentication for VANETs," in *Annual international conference on Mobile computing and networking*, ser. MobiCom. ACM, 2011.

[131] N. Bißmeyer, J. Njeukam, J. Petit, and K. M. Bayarou, "Central misbehavior evaluation for VANETs based on mobility data plausibility," in *International workshop on Vehicular inter-networking, systems, and applications*, ser. VANET. ACM, 2012.

[132] P. Zhang, Z. Zhang, and A. Boukerche, "Cooperative location verification for vehicular ad-hoc networks," in *International Conference on Communications (ICC)*. IEEE, 2012, pp. 37–41.

[133] J. Wang and N. Jiang, "A simple and efficient security scheme for vehicular ad hoc networks," in *Network Infrastructure and Digital Content, 2009. IC-NIDC 2009. IEEE International Conference on*, nov. 2009, pp. 591 –595.

[134] M. Raya, P. Papadimitratos, and J.-P. Hubaux, "Securing vehicular communications," *Wireless Communications, IEEE*, vol. 13, no. 5, pp. 8 –15, october 2006.

[135] "Standard for wireless access in vehicular environments - security services for applications and management messages," *IEEE Std 1609.2*, 2006.

[136] A. Studer, M. Luk, and A. Perrig, "Efficient mechanisms to provide convoy member and vehicle sequence authentication in VANETs." in *SecureComm*. IEEE, 2007.

[137] T. Leinmüller, C. Maihöfer, E. Schoch, and F. Kargl, "Improved security in geographic ad hoc routing through autonomous position verification," in *International workshop on Vehicular ad hoc networks - VANET*. ACM, 2006.

[138] P. Golle, D. Greene, and J. Staddon, "Detecting and correcting malicious data in VANETs," in *International workshop on Vehicular ad hoc networks*, ser. VANET. ACM, 2004, pp. 29–37.

[139] C. Laurendeau and M. Barbeau, "Probabilistic localization and tracking of malicious insiders using hyperbolic position bounding in vehicular networks," *EURASIP J. Wirel. Commun. Netw.*, 2009.

[140] L.-D. Chou, J.-Y. Yang, Y.-C. Hsieh, D.-C. Chang, and C.-F. Tung, "Intersection-based routing protocol for VANETs," *Wirel. Pers. Commun.*, vol. 60, no. 1, Sep. 2011.

[141] R. Kumar and M. Dave, "A comparative study of various routing protocols in VANET," *arXiv preprint arXiv:1108.2094*, 2011.

[142] S. Ma, O. Wolfson, and J. Lin, "A survey on trust management for intelligent transportation system," in *International Workshop on Computational Transportation Science*, ser. CTS. ACM, 2011.

[143] *Cisco Data Center Infrastructure 2.5 Design Guide*, 2008. [Online]. Available: http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5_book.html

[144] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Conference on Data Communication - SIGCOMM*, 2009.

[145] C. E. Leiserson., "Fat-trees: Universal networks for hardware-efficient supercomputing." vol. 1985. IEEE, 1985, pp. 892–901.

[146] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 75–86. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402968

[147] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," *Computer Communication Review*, vol. 40, no. 4, pp. 51–62, Aug. 2010.

[148] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Conference on Networked Systems Design and Implementation*, ser. NSDI. USENIX Association, 2012.

[149] A. Iselt, A. Kirstadter, A. Pardigon, and T. Schwabe, "Resilient routing using mpls and ecmp," in *High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on.* IEEE, 2004, pp. 345–349.

[150] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 339–350. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851223

[151] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "Serverswitch: A programmable and high performance platform for data center networks." in *NSDI*, vol. 11, 2011, pp. 2–2.

[152] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "Dac: Generic and automatic address configuration for data center networks," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 84–99, Feb 2012.

[153] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "RFC 2475: An Architecture for Differentiated Service," Internet Engineering Task Force, 1998.

[154] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "RFC 2205: Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," Internet Engineering Task Force, September 1997.

[155] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," 2013.

[156] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks," in *Network Operations and Management Symposium*, ser. NOMS. IEEE/IFIP, 2014.

[157] S. Roschke, F. Cheng, and C. Meinel, "Intrusion detection in the cloud," in *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on.* IEEE, 2009, pp. 729–734.

[158] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[159] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[160] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[161] K. C. Webb, A. C. Snoeren, and K. Yocum, "Topology switching for data center networks," in *Hot-ICE Workshop*, 2011.

[162] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[163] A. D. Ferguson, A. Guha, Liang *et al.*, "Participatory networking: An api for application control of sdns," in *ACM SIGCOMM 2013*. ACM, 2013, pp. 327–338.

[164] Z. Li, Y. Shen, B. Yao, and M. Guo, "Ofscheduler: a dynamic network optimizer for mapreduce in heterogeneous cluster." Springer, 2013, pp. 1–17.

[165] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu, "Transparent and flexible network management for big data processing in the cloud," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*. Berkeley, CA: USENIX, 2013. [Online]. Available: https://www.usenix.org/conference/hotcloud13/workshop-program/presentations/Das

[166] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 31–36.

[167] A. Dixit, P. Prakash, and R. R. Kompella, "On the efficacy of fine-grained traffic splitting protocolsin data center networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 430–431. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018504

[168] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Rfc1519: Classless inter-domain routing (cidr): an address assignment and aggregation strategy," 1993.

[169] M. Atkinson, J. Sack, N. Santoro, and T. Strothotte, "Min-max heaps and generalized priority queues," *Communications of the ACM*, vol. 29, no. 10, pp. 996–1000, 1986.

[170] S. Das, Y. Yiakoumis, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and P. Desai, "Application-aware aggregation and traffic engineering in a converged packet-circuit network," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC)*, 2011, pp. 1–3.

[171] E. Mannie, "RFC3945: Generalized Multi-Protocol Label Switching (GMPLS) Architecture," Internet Engineering Task Force, 2004.

[172] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[173] "Floodlight, SDN Controller," http://Floodlight.openflowhub.org/, accessed: 2014-08-14.

[174] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[175] P. Bille, "A survey on tree edit distance and related problems," *Theoretical computer science*, vol. 337, no. 1, pp. 217–239, 2005.

[176] I. Drago, R. Sadre, and A. Pras, "Report of the third workshop on the usage of NetFlow/ipfix in network management," *J. Netw. Syst. Manage.*, vol. 19, no. 4, pp. 529–535, 2011.

[177] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," RFC 4987 (Informational), Internet Engineering Task Force, Aug. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4987.txt

[178] C. Wagner, J. François, T. Engel *et al.*, "DANAK: Finding the odd!" in *International Conference on Network and System Security (NSS)*. IEEE, 2011.

[179] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo-simulation of urban mobility-an overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011, pp. 55–60.

[180] F. J. Martinez, C. K. Toh, J.-C. Cano, C. T. Calafate, and P. Manzoni, "A survey and comparative study of simulators for vehicular ad hoc networks (vanets)," *Wireless Communications and Mobile Computing*, vol. 11, no. 7, pp. 813–828, 2011.

[181] J. Harri, F. Filali, and C. Bonnet, "Mobility models for vehicular ad hoc networks: a survey and taxonomy," *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 4, pp. 19–41, 2009.

[182] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo–simulation of urban mobility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3 and 4, pp. 128–138, 2012.

[183] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "e.a.: Extending networking into the virtualization layer," in *In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009.*

[184] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[185] M. Gates, A. Tirumala, J. Dugan, and K. Gibbs, *Iperf version 2.0.0*, Part of Iperf's source code distribution, NLANR applications support, University of Illinois at Urbana-Champaign, Urbana, IL, USA, May 2004. [Online]. Available: http://iperf.sf.net

[186] "NOX, SDN Controller," www.noxrepo.org/nox/about-nox/, accessed: 2014-08-14.

List of Publications

[187] L. Dolberg, J. François, and T. Engel, "Efficient multidimensional aggregation for large scale monitoring," in *Proceedings of the 26th international conference on Large Installation System Administration: strategies, tools, and techniques*. USENIX Association, 2012.

[188] L. Dolberg, J. François, and T. Engel, "Multi-dimensional aggregation for DNS monitoring," in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, Oct 2013, pp. 390–398.

[189] ——, "Tracking spoofed locations in crowdsourced vehicular applications," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[190] ——, "Assessing artificially caused congestion on urban scenarios: A case study on luxembourg sumo traffic," in *3rd GI/ITG KuVS Fachgesprach Inter-Vehicle Communication, Technical Report, 2015, Ulm University*, 2015.

[191] J. François, L. Dolberg, O. Festor, and T. Engel, "Network Security through Software Defined Networking: a Survey," in *IIT Real-Time Communications (RTC) Conference - Principles, Systems and Applications of IP Telecommunications (IPTComm)*. Chicago, United States: ACM, Sep. 2014. [Online]. Available: https://hal.inria.fr/hal-01087248

[192] L. Dolberg, Q. Jerome, J. François, R. State, and T. Engel, "Ramses: Revealing android malware through string extraction and selection," in *Proceedings of 10th International Conference on Security and Privacy in Communication Networks*. EAI, 2014.

[193] L. Dolberg, J. François, S. Chowdhury Rahman, R. Ahmed, R. Boutaba, and T. Engel, "Network configuration and flow scheduling for big data applications," in *Networking for Big Data*, W. Ashley, Ed. CRC Press - Taylor & Francis Group, USA, 2015.