Sketch-Based Attack Detection on Programmable Networks

João Romeiras Amado
joao.f.amado@gmail.com

Instituto Superior Técnico, Universidade de Lisboa, Portugal

November 2018

Abstract—**The implementation of an intrusion detection system deals with two problems. First, the need to obtain up-to-date statistics encompassing various metrics of interest that can range from network traffic information (*e.g.*, network load or latency) to security alerts. Second, the ability to extract relevant knowledge from the aforementioned data. The first problem is usually tackled through real-time network monitoring using low accuracy techniques such as packet sampling, requiring the placement of expensive hardware components in crucial network points in order to improve accuracy. In this thesis we plan to approach this challenge with programmable networking, a new approach to computer networks that separates the data plane from the control plane, enabling the centralization of network control and the execution of applications that direct the configuration of forwarding devices. This new paradigm includes the programmability of forwarding devices, such as switches, and enables the use of sketching algorithms directly in the data plane, that provide summary statistics about packet flows, allowing a more effective network monitoring. We tackle the second problem through unsupervised machine learning techniques that possess the ability to identify a specific behavior without any prior knowledge or training phase, serving as a powerful instrument to detect suspicious patterns. This work will, therefore, propose the design, implementation, and evaluation of a monitoring system using programmable switches that leverages machine learning algorithms to perform network attack detection.**

Keywords: Software-Defined Networks, Programmable Switches, Sketching, Machine Learning

I. Introduction

Network security, in particular, the capability to perform intrusion detection or to identify instances of malicious activity in an infrastructure, entails two fundamental problems. First, the need to possess statistics encompassing various metrics of interest that can range from network traffic management information (*e.g.*, network load or latency) to security alerts. Second, the ability to extract relevant knowledge from the aforementioned data – in this case, to detect anomalous activities. Such a task can be particularly difficult when an attack is new, and when depending only on the knowledge obtained from previous attacks is not enough.

To address the first problem, operators perform real-time monitoring using *sampling-based approaches* such as NetFlow [1]. This data collection process can be a daunting task on large-scale traditional computer networks, where for scalability reasons the sampling rate is typically low (1 in 1000 packets, or less, typically [2]), affecting the accuracy of network monitoring tasks. As a result, a hardware monitoring infrastructure has to be placed at key network points in order to perform a more precise data collection to effectively increase the detection accuracy, which tends to be a costly solution.

A recent approach to computer networks, *Software-Defined Networking* (SDN), improves on traditional networks by separating the network control plane from the data plane [3]. This separation allows the centralization of network control, enabling network-wide abstractions, with administration policies being defined through applications that run on the control plane, directing the configuration of forwarding devices. Moreover, network monitoring tasks are facilitated through this decoupling, since it avoids the need for an expensive monitoring infrastructure. However, the problem of low resolution in sample-based monitoring persists.

An alternative to packet sampling is the use of *sketching algorithms*, a class of streaming algorithms that provide summary statistics about packet flows [4]. This technique allows the processing of all packets, providing a more effective network monitoring and increased accuracy in threat detection mechanisms. This approach achieves an effective tradeoff between memory usage and accuracy, with variations depending on the specific sketch function used.

Despite the advantages of sketching algorithms, they introduce feasibility issues, since current switching hardware does not yet enable the sort of data plane computation required by sketching solutions. This situation is now changing, with the emergence of programmable switches. Recent switch designs [5] demonstrate the ability to reprogram hardware using a high-level language such as P4 [6] [7]. Production hardware, such as Barefoot Tofino [8], is already available; many compilers also exist for a variety of hardware (*e.g.*, P4FPGA [9], Tofino) and software targets (*e.g.*, Pisces [10]). The availability of programmable switch components enables sketching to be performed at line rate, as demonstrated in the UnivMon [11] and HashPipe [2] projects.

The second challenge consists of the extraction of relevant knowledge about security events from the collected monitoring data. Recent solutions have shown that intrusion detection systems can apply machine learning algorithms to extend their capabilities by analyzing the collected monitoring information [12]. More specifically, unsupervised machine learning techniques have the ability to identify a specific behavior without any prior knowledge or training steps, thereby reinforcing a monitoring system's capacity to detect common threats and also to identify new ones.

In this work, we propose a monitoring system using programmable switches that leverages machine learning algorithms to perform network attack detection. Our hypothesis is that it is possible to improve attack detection by improving the quality of the input measurement data, leveraging programmable hardware for this purpose.

Our solution runs on an SDN infrastructure, using the Open Network Operating System (ONOS) [13] controller. At the data plane level, a network of programmable switches collects basic flow statistics (based on a 5-tuple structure) and executes sketching algorithms developed in the P4 language. Specifically, we include two sketches in our solution. First, the countmin sketch, which relies on hash functions to map events to frequencies, is used for the detection of heavy-hitters, a class of flows that are larger than a specific fraction of the total packets observed. Second, the bitmap sketch, which uses an array of bits to count the number of specified unique elements and can be used to identify superspreaders, a type of network flow that contacts more than k distinct addresses during a specified time interval. The use of these two sketches together enables us to detect possible DDoS attacks and malware spreading. The employment of sketching algorithms in our solution allows real-time collection of flow statistics that process all packets, differing from a netflow-based approach, based on packet sampling. While the switches process the streaming of network packets, the SDN controller manages the active flows at any given moment, retrieves sketch data from the switch, and periodically exports all flow statistics.

The collected data is then processed through various normalization steps, as preparation for the subsequent unsupervised machine learning phase. We use the K-Means clustering algorithm to group similar entities according to their characteristics, using a predefined number k of clusters. Through various iterations using a combination of basic flow statistics and sketching algorithms, the resulting clusters group both regular traffic and the outliers, which represent possible network attacks. Subsequently, a manual analysis of all processed data obtains the final detection results.

II. Related Work

The following chapter describes previous contributions related to the subject of this work. As a starting point, Section II-A presents the concept of SDN, its distinction from traditional networks, and the concept of programmable networks using the P4 programming language. Section II-B provides an overview on network intrusion detection systems that use machine learning techniques, showing the more relevant machine learning methods and how they can be used to perform monitoring. In Section II-C, we present the concept of sketching, along with various network streaming techniques that rely on this structure to obtain data monitoring information.

A. Programmable Networks

Traditional computer networks have intrinsic properties that make them very complicated to manage on a large scale. In terms of infrastructure, the control plane (which manages network traffic) and the data plane (that executes traffic forwarding according to the control plane's instructions) are bundled together in each networking device. As a result, performing network policy modifications, such as changing security rules or load-balancing data, is highly complex due to the need for an administrator to configure each network device, without the ability to resort to global, network-wide abstractions. This legacy paradigm in network implementations has led to substantial inertia in the process of innovation.

Software-Defined Networking is an approach to computer networks in which the control logic is separated from the forwarding elements, such as routers and switches. By separating the control plane from the data plane, an SDN infrastructure provides increased flexibility and decentralization of the decision process. Network applications are programmed and executed in the controller, thus abstracting the lower-level interaction with the network routers and switches, which become simple forwarding devices. Hence, an SDN infrastructure is more apt to manage enterprise-grade networks that require both scalability and flexibility.

The OpenFlow protocol [14] was a crucial enabler of SDN, as it offered controllers a southbound path with direct access to the forwarding plane and allowed the definition of packet-handling rules from a remote, logically centralized location.

An SDN controller is a logically centralized element responsible for relaying the network configuration instructions from the network applications to the forwarding devices. In a sense, each controller functions as a traditional operating system, abstracting the lower-level details of each specific network implementation. The control plane infrastructure runs on commodity server technology and can either be a fully centralized system (*e.g.*, the NOX controller [15]) or be physically distributed, such as Onix [16], providing the abstraction of a centralized network view instead.

Some of the more relevant controller implementations include ONOS [13], which provides a set of high-level abstractions and modules (extensible at run-time) to be used by network applications, and the OpenDaylight project [17], a Linux Foundation initiative supported by a large community of developers and companies that promotes the development of an open SDN framework.

By leveraging open northbound APIs, an SDN infrastructure allows administrators to program applications that manage the control plane. Additionally, the forwarding devices in the data plane maintain a set of flow rules that define actions on incoming packets, based on controller specifications. The emergence of programmable switches gives the ability to reprogram network hardware, with languages such as P4 [6]

[7], thus increasing the expressiveness of the control flow abstraction, allowing the direct control of packet processing.

This high-level language, P4, is specifically tailored for reconfigurable switches, allowing the definition of new protocol headers and providing protocol-independent packet processing. It attempts to provide reconfigurability (redefinition of a packets parsing and processing), protocol independence (ability to specify packet parsers and match+action tables to process them), and target independence (the programmer does not need to consider the hardware details of the data plane; these are only relevant at compiler-level).

The P4 forwarding model is implemented using a programmable parser, followed by several match+action rules (in series and parallel). Its control is divided into two operations, configure and populate. The first programs the parser, defines the match+action rules and its specification, while the second implements these rules over the match+action tables, determining the policy applied to each packet.

B. Intrusion Detection Based on Machine Learning

The extraction of useful information from large datasets can be a time and resource consuming operation. Machine learning techniques, which allow knowledge extraction through the observation of patterns in specific environments, are useful to process, normalize and classify many different types of information sets. Machine learning schemes can be broadly classified in *unsupervised learning algorithms*, which receive unlabeled input data and identify correlations in order to classify the information, and *supervised learning algorithms*, which require a training phase in which labeled input data is used to train the system, as preparation to receive further sets of information.

Network intrusion detection systems can be implemented with machine learning methods that augment their capabilities. Using a combination of regular network data and various anomalies as a training set, they reinforce their ability to both detect common threats and to identify new ones. Traditional networks lack the centralized controller abstraction that SDN offers, so any monitoring task depends on specific hardware components deployed in strategic network points. However, an SDN-based IDS can build on such an abstraction to sidestep network infrastructure and distribution details, focusing all monitoring operations on the control plane.

An attempt to leverage traditional anomaly detection techniques and adapt them to an SDN scenario is explored by Mehdi *et al.*, using OpenFlow's monitoring capabilities and NOX as a controller [18]. This work focused on home/small office networks and managed to achieve a highly accurate line rate detection of anomalies, without incurring a relevant performance penalty on the regular network operations.

In order to implement a lightweight DDoS attack detection system, Braga *et al.* combined OpenFlow with NOX to collect flow table statistics and used the resulting dataset to train a neural network [19]. This method is easily adaptable, since the detection algorithms can be modified in order to detect new kinds of attacks. Additionally, the use of NOX as a controller platform allows new switches to be introduced and configured in the detection system in a simple, effortless process.

Another approach is followed by is Lee *et al.* in Athena [12], a framework for distributed anomaly detection in software-defined networks, with an emphasis on scalability and ease of implementation. No hardware modifications on the original network infrastructure are required, apart from the need for OpenFlow support. This project exports an API that allows developers to program new network anomaly detection applications while abstracting low-level details and data management. Additionally, the anomaly detection algorithms deployed are augmented by a machine learning library in order to speed up the runtime detection model generation.

Athena is implemented over ONOS, functioning as a subsystem that provides services to an application layer. Being a fully distributed framework, each instance is hosted on an SDN controller. All components perform monitoring functions, retrieving information associated with their hosted network controller and respective data plane, and share said information and state through the distributed database and clusters, connecting all instances and ensuring a scalable environment.

C. Network Monitoring Using Sketches

Recent monitoring algorithms leverage specific instances of streaming algorithms called *sketching* techniques [4] [11]. A *sketch* is a data structure that stores summary information about packet states captured through streaming, using a combination of hashing, counting and filtering techniques. Sketch-based algorithms observe *all* packets, instead of relying on sampling. Additionaly, sketches offer various improvements over generic flow-based counters, such as low memory usage and a strong balance between memory usage and accuracy.

Sketching algorithms are a powerful and efficient way to monitor a network, with recent advances starting to provide a more general monitoring approach, instead of focusing on a single feature. By streaming and processing all packets in a given network, instead of relying on packet sampling, threat detection mechanisms are able to function with a higher degree of precision and prevent attacks that could otherwise be unnoticed. The development of the P4 language made possible the execution of sketching algorithms on the data plane, by directly programming the hardware switches.

OpenSketch [4] is a sketch-based traffic measurement architecture designed to operate in commodity switch hardware that provides a more generic and efficient solution by separating the network measurement control and data plane functions. The measurement functions present in a generic data plane are the selection of which packet to measure, and storing/exporting the measurement data. In this particular instance, various combinations of hashing and classification rules are used in the selection process. For the data storage and export task, OpenSketch uses a table with complex indexing, which reduces the memory overhead and can be easily exported to the controller. A wide variety of measurement tasks are allowed on OpenSketch, many of which can be executed

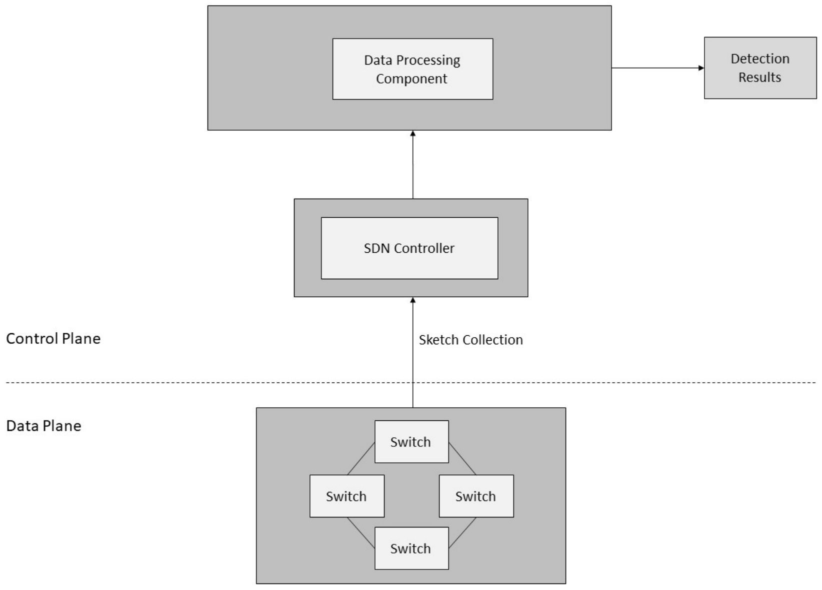

Fig. 1: General architecture of our solution.

using the sketches directly; in the case of more complex data analysis requirements, the system uses a combination of simple sketches with specialized software running on the controller.

Flow monitoring algorithms feed on network streaming data and can be particularly powerful when dealing with specific cases like heavy-hitter detection, or the detection of any flow that is larger than a fraction t of the total packets observed on a link (may also be referred to as one of the top k flows in terms of size). Identifying packets belonging to heavy flows is a difficult task, since the data plane has several technical constraints, *e.g.*, small processing time allocated to each packet, and limited memory in switching hardware.

HashPipe [2] is an algorithm implemented with the P4 programming language that tracks heavy flows within a network with high accuracy. By using a pipeline of hash tables, HashPipe retains the identifiers belonging to the heaviest flows while discarding the lighter ones. Each incoming packet hash is checked against the values already on the first stage of the pipeline. If there is a hit, then that counter is updated. In case there is a miss, the key with the smallest counter is sent down the pipeline, following the same process. This ensures that in the end, the identifiers retained in the hash tables are those with the larger counts, representing the heaviest flows.

An ideal monitoring framework will offer a general-purpose approach while achieving high-fidelity. UnivMon [11] is a P4 framework implementation that proposes such an approach using universal streaming, a single universal sketch abstraction which allows network-wide monitoring without focusing only on a specific feature. The system relies on a single universal switch abstraction that guarantees the successful measuring of different dimensions of traffic (*e.g.*, source IPs, destination ports), making it seem like the traffic entering the network is monitored by a unique switch.

Sketching algorithms are a powerful mechanism we will include in our solution, due to their versatility in collecting network monitoring information.

III. Design

In this chapter, we describe a network monitoring system using programmable switches that leverages machine learning techniques to perform attack detection. We start with a general overview of the architecture, followed by a more detailed analysis of its various components.

Our work intends to address two fundamental problems related to network intrusion detection:

- The need to possess statistics encompassing various metrics of interest that can range from network traffic management information (*e.g.*, network load or latency) to security alerts;
- The ability to extract the relevant knowledge from the aforementioned data – in this case, to detect anomalous activities.

To address the first problem, we use sketching algorithms running on programmable switches. The second challenge is approached with unsupervised machine learning techniques, in order to analyze the collected monitoring information.

A. Overview

The scheme in Figure 1 presents an overview of the architecture of our solution. The data plane is composed of a

TABLE I: Flow Statistics Features

Features
Number of packets
Number of bytes
Source and destination IP address
IP protocol
UDP/TCP source and destination port
Count-min sketch
Bitmap sketch

set of programmable switches managed by an SDN controller. Periodically, the controller collects basic flow statistics and sketch values from the data plane. These statistics are then sent to the data processing component, where we apply data normalization and machine learning algorithms. Following this process, we perform a human analysis on the processed data, in order to obtain the final detection results.

On the data plane, a set of programmable switches apply the active forwarding model at any given time with the SDN controller periodically collecting all flow statistics from the data plane switches. The streaming network data collected in real time includes traditional flow metrics, *e.g.*, the number of packets/bytes and source/destination addresses and ports. Additionally, we implement two sketch algorithms, count-min and bitmap, thus augmenting the collected flow statistics compared to traditional approaches.

In our solution, due to resource constraints, the maximum number of active flows at any given time is limited by the controller. When this number is exceeded, the flow deemed less relevant is removed and a new one takes its place.

The collected flow statistics are then sent through pre-processing steps which remove unnecessary flow information and perform data normalization, mapping every feature to a numerical value and guaranteeing one common scale between all elements.

A clustering algorithm is used to analyze the preprocessed data, in order to detect network attacks. The use of machine learning techniques groups potential network anomalies and, thus, simplifies the subsequent human analysis of the obtained results.

B. Network Flow Statistics Collection

The proposed monitoring system relies on the retrieval of network statistics, which are collected by the SDN controller, periodically, from programmable switches. The SDN paradigm allows the configuration of diverse statistics collection strategies by introducing programmable forwarding devices.

The SDN controller manages the data plane composed of programmable switches, that execute the active forwarding model at any given moment, and several network applications that provide services to end hosts and neighboring networks. Additionally, these programmable switches perform the sketch algorithm calculations in real-time. The collected flow statistics are presented in Table I.

Traditionally, active flows in the control plane are discarded after a specific time period of inactivity: any flow that is not updated within a certain time interval (*e.g.*, the controller has not received any packet matching that flow) is removed from the active flow table. Since we aim to collect flow statistics spanning a large number of packets, we do not set this time limit. In this way, we remove the need to keep a record of past flows and their respective statistics that would have to be constantly compared to the active flow table, which allows us to track all statistics pertaining to a specific flow with a higher degree of accuracy.

However, maintaining a limitless flow table is impractical due to memory constraints. For that reason, we developed an adaptation of the *space-saving algorithm* [2], that uses $O(k)$ counters to count k flows: when a packet arrives, if it does not match an active flow and if the counter table is not full, the algorithm inserts the new flow with a count of 1. If the packet matches an active flow, the algorithm increases the corresponding flow counter. However, if the packet does not match any active flow and if the counter table is full, the algorithm replaces the flow entry that has the minimum counter value v in the table with the packets respective flow and increments the counter.

C. Count-Min and Bitmap Sketches

The two sketching algorithms included in our solution, the count-min sketch and the bitmap sketch, are presented next. These particular sketches were chosen due to their ability to be used in the detection of several network anomalies, particularly DDoS attacks and malware spreading.

The count-min sketch is a probabilistic data structure that serves as a frequency table of events in a stream of data. It uses hash functions to map elements, such as specific network flows, to frequencies, making it useful in packet monitoring scenarios for heavy-hitter detection: the identification of large flows that occupy more than a fraction k of the network link capacity during a time interval. This sketch's structure consists of an array of counters, each accessed through a specific hash function. When an event i arrives, the sketch is updated by calculating all hash values of i and subsequently incrementing each respective counter on the array. When the current count for a certain event j is requested, the estimated value is obtained by comparing all counters in order to find the minimum stored value.

The bitmap sketch maintains an array of bits to count the number of unique elements (*e.g.*, IP destination addresses). Each item in a stream of elements is hashed to one of the b bits in the array, with the respective bit changed from 0 to 1. The bitmap sketch value is obtained by the sum of all values from the bitmap array. We can use this sketch, for instance, to easily count the number of accesses to distinct destinations by a single source address. As the packets arrive, the first step consists of calculating a hash for every destination address. Subsequently, a filtering process selects all the packets matching the intended source address. If a certain packet matches, then the bitmap array is updated at the index given by the hash value of the destination address, with its value modified from 0 to 1. By summing all values in the bitmap array, we obtain the number of unique destinations.

D. Data Normalization

The collected flow statistics are composed of various features, of which many contain numerical values. As such, we need to perform data normalization, in order to prepare the statistics for analysis.

Since our flow statistics contain non-numerical features, the first step is the conversion of any of these, such as IP addresses, to numerical values. The second step consists of mapping all values to a specific range, in this case, the interval [0,1], with 0 being the minimum and 1 the maximum.

E. Clustering and Classification

For this project, we rely on an unsupervised clustering algorithm that allows us to group similar network behaviors. In this way, we do not require any training phases to provide our system with prior knowledge about similar situations, enabling us to detect new attacks.

Clustering algorithms group related entities according to their characteristics. The input data is fed to the algorithm, after which, by adjusting specific parameters, similar data points are grouped together; data entities containing different and more uncommon features are considered outliers and also identified.

In this project, we use K-Means, a widely used clustering algorithm that provides reliable results in a reasonable execution time. Requiring a predefined number *k* of clusters as input, K-Means starts by randomly choosing a centroid value for each cluster, after which it iteratively performs three steps:

1) Find the Euclidean distance given by $d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$, between each data entity and each clusters centroid.
2) According to the previous results, assign each data entity to its closest cluster.
3) From the mean value of all entities belonging to a cluster, update the respective centroid values.

The iterations continue until the centroid values no longer change significantly.

By applying a clustering algorithm such as K-Means on the obtained flow statistics, we expect the regular traffic representing the bulk of the dataset to be grouped in several large clusters, and the scarcer network anomalies to be clustered together as outliers and subject to a manual analysis in search of network attacks.

IV. Implementation

This chapter describes the implementation of our solution. We used the P4 language to program the network switches, ONOS as the SDN controller, and the Python library Scikit-Learn to implement the data normalization and machine learning algorithms. Next, we detail the P4 programs compiled on the switches, how ONOS retrieves network statistics from the switches, and finally the data preprocessing and clustering algorithms responsible for the analysis of the measurement data.

A. P4 Sketches

In order to implement new flow statistics, including sketches, we rely on P4 programs running on the data plane topology, i.e., the switches. Our implementation uses Mininet, a virtual network emulator, to emulate the desired host and switch configuration. This virtual data plane is controlled by ONOS, effectively functioning as an SDN infrastructure.

The P4 language provides direct control over data plane behavior, allowing the definition of custom packet parsers and match+action tables. As such, we defined a table matching the following type of flow:

- Ethernet source and destination address;
- Ethernet type protocol;
- IPv4 source and destination address;
- IPv4 type protocol;
- UDP/TCP source and destination port.

This procedure allows the extraction of flows with a fine granularity. In our solution, all packets are sent to a specific switch in the emulated network. When a new packet arrives, it matches the table, and its respective flow is added to the ONOS flow table. Follow-up packets from this flow just increment the required counters.

The deployed P4 program we developed includes the implementation of the two sketches used in our solution, the count-min sketch and the bitmap sketch, that perform their respective functions after a packet is matched with the table referred above. Both sketches rely extensively on hashing functions, P4 metadata, and register arrays. Metadata is data associated with a single packet, whereas registers allow the storage of persistent data across packets.

The count-min sketch performs three different hash function calculations, which are used to access their respective registers position and increment its value by 1. A comparison is then performed between the three values, in order to determine the minimum. The final count-min value is stored in a fourth register for later retrieval by the SDN controller.

The bitmap sketch implementation in P4 relies on two registers: The first is accessed using a hash composed of the packet's source and destination IP address, and determines if that pair is new (the register's value is 0 in case of a new combination, and 1 otherwise). The second register is accessed using a hash composed of the packet's source address, only if the previous pair was new. In this case, the respective value is incremented by 1. Since the objective of the bitmap sketch is to count the number of unique elements for a given set of features, we use this implementation to store the number of unique destination accesses for a given source address. The SDN controller accesses the second register for the bitmap sketch values.

B. ONOS

In the ONOS controller, a specific type of application referred to as *pipeconf* allows the storage and deployment of

P4-defined data plane packet forwarding pipelines, defined as sets of match-action stages that execute a corresponding action in case a particular entry is matched by incoming data.

The use of pipeconf applications effectively allows ONOS to understand and control P4 switches. A pipeconf implementation is composed of:

- Pipeline model description that allows ONOS to understand a P4 program/pipeline;
- Implementation of pipeline-specific driver behaviors, giving ONOS the capacity to control a pipeline;
- Target-specific artifacts produced, *e.g.*, by the P4 compiler, to deploy a P4 program to a device.

The P4 program referred in Section IV-A is controlled by an ONOS application we developed, named Flowstats, that implements a pipeconf, sharing with the controller its specific header fields and packet manipulation actions.

The main function of this application is the extraction of flow statistics at runtime. Every 5 seconds, a flow listener method generates an event in case ONOS detects a new flow rule or an existing flow rule update. After checking if the respective flow rule matches our desired characteristics (IPv4 address, with defined IP protocol and UDP/TCP ports), its various components are stored.

While ONOS provides multiple features allowing the control of P4-enabled devices, such as device connection, match-action table operations, or counter reads, the direct access to P4 registers is still being developed and is not yet supported. Since the count-min and bitmap sketch values are being saved in P4 registers, we implemented a bash script that receives a register index and returns the desired value. This script is executed once every minute, retrieving the sketch statistics corresponding to all active flows. The registers use hashes as index values, so ONOS calculates the corresponding hash for each register (for instance, the final bitmap registers hash is calculated from the packets source IP address) in order to pass it as input for the bash script and receive the flows sketch values.

The Flowstats application also manages the number of active flows present in the controllers flow table. Despite the flows not having a time limit before removal, due to hardware limitations we enforce a limit of 1000 on the number of active flows at any given time. After collecting all the desired statistics from a flow, the application uses its implementation of the space-saving algorithm to determine if, in case of a full flow table, an older flow must be removed.

The periodically collected flow statistics from all active flows are stored internally in ONOS and saved to a *csv* file every minute.

C. Unsupervised Learning

After the flow statistics collection phase, the data is processed through various data normalization steps, before being fed to the unsupervised machine learning mechanism we implemented.

All the following procedures were developed using the Python language and its popular external package scikit-learn, that provides numerous data processing and machine learning tools.

The specific clustering algorithm used, K-Means, requires the prior specification of the exact number k of clusters. Different values of k can affect the algorithms output in drastic ways, so a technique known as the *Elbow Method* was used. By executing K-Means clustering on the dataset for a range of values of k (for instance, from 1 to 20), the *Elbow Method* computes the sum of squared errors (SSE) for each value. This error function is used as a stopping criterion by K-Means. The resulting values of k are then plotted, resulting in a graphic showing a descending curve. The value of k where the SSE is 0 is unusable, since it represents a data point being its own cluster. The ideal value is a small k that also has a small SSE, represented by the elbow in the arm-like graph.

The *Elbow Method* is an heuristic technique which gives only an approximation of the ideal value of k, depending on each specific dataset. Its output serves as a starting point, around which we can try various values of k for our clustering tests.

As the flow statistics used as input for K-Means contain 7 features, an effective cluster visualization can be hindered, since a representation of a 7-dimensional cluster would be highly impractical. To achieve a proper visualization, we use the *Principal Component Analysis (PCA)* statistical technique. PCA uses the set of features in an input dataset and projects them onto a feature space that has a lower dimensionality. The resulting features are referred to as the *principal components*, and contain the highest amount of variance in the dataset, thus summarizing most information from the original data set. Since these components are the eigenvectors of a covariance matrix, they are orthogonal and mutually uncorrelated. As such, we obtain a representation of the cluster set in 2-dimensions, and are able to visually analyze the various clusters formations and identify potential network anomaly groups.

V. Evaluation

This chapter describes the evaluation method used to validate the performance of our implementation. In the following sections we analyze:

- Our system's ability to collect reliable and accurate flow sketches from a P4 switch infrastructure.;
- Network anomaly detection using clustering algorithms receiving as input a combination of regular flow statistics collected through ONOS, and the P4 sketches, with a comparison between both netflow-based statistics and our solution;
- The performance of our solution, in terms of processing time and resource consumption.

The CAIDA dataset [20], collected by the Center for Applied Internet Data Analysis, contains anonymized passive bidirectional traffic traces from various monitors on high-speed internet backbone links. The traces provided are available as *pcap* files containing timestamped packet data. 30 samples

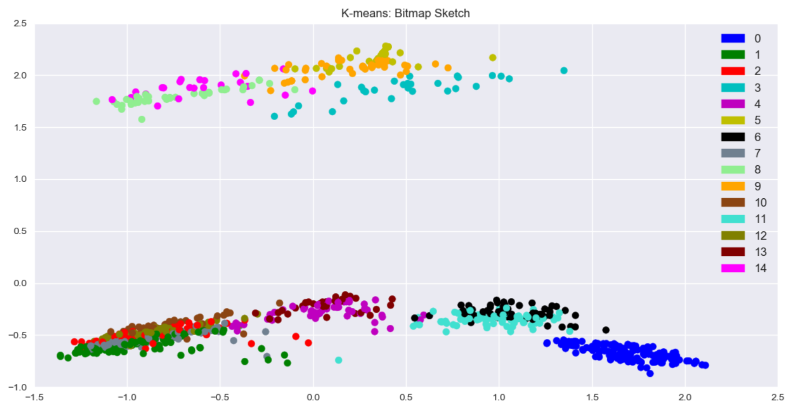

Fig. 2: Bitmap sketch clustering.

from this dataset were used as input for our evaluation tests, specifically from CAIDAs 2014 Chicago monitor.

The SDN components, such as the ONOS controller and the P4-programmed switched running on mininet, were implemented and tested on an Ubuntu 16.04 virtual machine with 4 virtual processor cores and 6 GB of RAM. The data normalization and clustering phases were implemented using Python libraries and tested on a Windows 10 system with a dual-core i7 processor and 16 GB of RAM.

A. *P4 Sketch Collection*

In this section, we evaluate the ability of our P4-sketching approach to provide accurate network flow statistics in real-time. To perform an accurate comparison, the obtained *pcap* files from the CAIDA dataset were analyzed, in order to determine a ground truth corresponding to the original captured network traffic.

In our implementation, the count-min sketch maintains statistics about flows with the same source-destination pair of IP addresses, as a means for heavy-hitter detection. Hence, the ground truth was obtained by analyzing the *pcap* files and counting the occurrence number of all source-destination address pairs.

The bitmap sketch is used to determine the existence of superspreader sources, by counting the number of distinct destinations for every source address. The same logic was applied to obtain the ground truth: All source addresses in the *pcap* file were sorted, and each distinct destination was counted.

As previously referred, the space-saving algorithm is used to limit the number of active flows. This approach reduces the accuracy of the result, as an old flow with relevant sketch statistics might be replaced if the algorithm deems it dispensable, a not uncommon case in superspreader sources. Bitmap values are commonly obtained through a combination of many single accesses from a particular source to many destinations, resulting in multiple flows that may have a high bitmap value, but are considered less relevant by the space-saving algorithm and removed.

TABLE II: Source Addresses with Highest Bitmap Values and Respective Cluster.

	Src IP	**BM**	**Cluster**
1	186.75.209.126	63	0
2	186.75.214.167	59	0
3	186.75.212.122	59	0
4	186.75.212.123	53	0
5	186.75.214.164	49	0

After performing multiple tests, we consider that the obtained flow sketch statistics are accurate, displaying very similar values when compared with the baseline. This factor is essential so that the subsequent clustering and analysis process can accurately reveal network anomalies and potential attacks.

B. *Clustering Evaluation using the CAIDA dataset*

In this section, we evaluate various clustering experiments using the obtained flow statistics and compare a traditional netflow-based monitoring approach with our solution using sketching algorithms.

All network flow statistics collected from the SDN infrastructure are normalized, in order to serve as input for the K-Means clustering algorithm. Many values of k were tested as input for K-Means, so that, depending on the specific feature being observed, an optimal result could be achieved. Furthermore, the centroid values representing the mean value of each feature were analyzed for all clusters, as a means to identify the more relevant components of each, with special consideration to sketching centroid values.

An example run of the K-Means algorithm with 15 clusters is presented in Figure 2. By analyzing both the image and the cluster distribution for each flow, the results are revealed to be positive, as the flows containing the highest bitmap values are concentrated on cluster 0, displayed in the lower-right corner.

By looking at the cluster column in table II, that shows the top 5 flows ordered by bitmap sketch value, we see that all values are indeed in cluster 0, as expected. Since this sample of 11943 flows contains 6222 distinct destinations, we observe that these source IP addresses can be considered superspreaders, each accessing a considerably high number of distinct destinations. Additionally, the first 9 source IP addresses are part of the same subnet, which might indicate a malware infection.

Since the CAIDA dataset used in our tests did not contain any identified network anomalies, such as heavy hitters, we modified sets of samples in order to test a situation where this would be present. The modification consisted of cloning 7 specific packets in a sample *pcap*, so that its corresponding flows would be identified as possible heavy hitters. Additionally, we also tested the bitmap sketch, so that the performance of both sketches could be analyzed on the same cluster set. In our tests, we observed that using the count-min sketch to generate clusters is less effective than with the bitmap sketch, since only the most prominent network anomalies, shown as outliers, are identified. We observed that the bitmap sketch was not affected by the modifications, as expected, with the higher bitmap value flows being grouped in the same cluster.

By combining both sketch statistics in the same clustering process, we determine that the results are effective, with different clusters forming around both heavy hitters and superspreaders. In this way, we are able to use the sketch statistics to detect both network anomalies at the same time.

The last experiment asks how would a traditional, netflow-based solution, using packet sampling, would behave. For this purpose, we performed sampling of 1 in 100 packets in samples of the CAIDA dataset data, after which we performed a regular execution of our system, collecting instead the netflow statistics mentioned above. Subsequently, the results were compared against the sketch-based solution.

In order to verify if traditional netflow statistics influence the K-Means clustering process, we analyzed multiple samples and observed the mean values of all the features for each cluster. Thus, we concluded that the netflow features, the number of packets and bytes, have negligible influence in each cluster, as evidenced by values that never go above 0.1. In other words, it is not possible to identify any cluster with anomalous behavior.

Subsequently, we performed comparison runs between our sketch-based implementation and the traditional netflow-based approach using the same samples. By comparing the traditional approach with the bitmap sketch statistics, it is possible to determine that, unlike the accurate results of the bitmap clustering, the netflow sampling data was unable to group the top potential superspreaders, spreading them instead among 4 different clusters. Additionally, a similar comparison was made between netflow and the count-min sketch statistics. Since the samples used did not contain any modifications, the count-min values are low, and we observed the subsequent clustering results to be less effective when compared with the previous section. In this comparison, the netflow-based approach was able to group the two heavier flows, compared to three flows from the count-min statistics.

Our results show that a network monitoring approach with improved capabilities in flow statistics collection will return higher quality metrics that can assist intrusion detection systems based on machine learning in improving the anomaly detection rate in real-time.

C. Performance

This section presents a performance assessment for the various components developed in this thesis. The first part discusses the resource usage of all the SDN infrastructure, specifically ONOS and the mininet-virtualized switches, while the second part analyzes the machine learning components, from the data normalization steps to the K-Means algorithm.

The SDN components proved to be extremely resource-intensive, a fact accentuated by the use of a virtual machine for all the execution tests. We identified the main bottleneck as the I/O virtual disk access by ONOS. A higher number of packets-per-second seemed initially a potential way to reduce the processing time, since the bottleneck was not in the program simulating the packets, TcpReplay, which placed all packets in RAM before sending them, nor on the simulated switches, that displayed a consistently low I/O usage. Our subsequent tests showed that, due to low disk I/O capacity, ONOS created a bottleneck by maintaining packets not yet processed in a waiting list, drastically augmenting the processing time. This situation grew worse the higher the number of packets arriving each second. Hence, a reduction on the number of packets-per-second proved to be the solution for the I/O bottleneck in the network simulations performed.

The second part of our implementation, consisting of data normalization procedures and various runs of the K-Means algorithm, proved to be lightweight in its resource usage, particularly when compared to the network monitoring stage. We observed the various steps as considerably fast, with the clustering algorithm runs occupying much of the processing time.

VI. Conclusion

In this thesis, we presented a detection system for attacks in software-defined networking infrastructures that relies on the collection of basic packet flow statistics and network sketches from programmable switches. Additionally, our solution leverages unsupervised machine learning algorithms to process the obtained data and identify potential anomalies. After the various data collection and processing stages, a subsequent manual classification phase verifies the obtained results and determines the existence of network attacks.

As use cases of our solution, we studied heavy-hitter flows and superspreaders, since these occurrences are commonly linked to DDoS attacks and malware spreading, respectively. To accurately detect these particular traffic behaviors, we implemented two sketching algorithms in the P4 language, count-min and bitmap. The combination of regular, 5-tuple flow statistics, and flow sketch values proved effective in the latter processing stages. Following several data normalization steps, the collected flow statistics were clustered using the K-Means clustering algorithm. In order to obtain reliable evaluation results concerning network traffic behavior, we used a real dataset. For this purpose, samples from the CAIDA dataset were used for our evaluation tests, with the obtained flow statistics primarily analyzed for network anomalies. The resulting cluster separation showed a division between regular network traffic and specific outlier cases, as initially expected. Subsequent analysis between flow statistics and cluster division results revealed that flows with higher sketch values, representing possible network attacks, tended to be grouped in the same cluster. Furthermore, an important factor is the similarity between our obtained flow sketch statistics and the corresponding ground-truth data. Various comparisons between both datasets revealed the high accuracy of the implemented flow statistics collection process.

Additional comparisons were also performed between the proposed approach and a more traditional netflow-based collection that relies on packet sampling. An analysis of both techniques proved that our sketch-based implementation is more effective in identifying network anomalies.

Our results show both the effectiveness of sketching algorithms for network anomaly detection approaches and its potential when working in tandem with unsupervised learning approaches.

References

[1] B. Claise, "Cisco systems netflow services export version 9," 2004. [Online]. Available: https://tools.ietf.org/html/rfc3954

[2] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*. ACM, 2017, pp. 164–176.

[3] F. M. Ramos, D. Kreutz, and P. Verissimo, "Software-defined networks: On the road to the softwarization of networking," *Cutter IT journal*, 2015.

[4] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch." in *NSDI*, vol. 13, 2013, pp. 29–42.

[5] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 99–110.

[6] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[7] "P4 website," 2017. [Online]. Available: https://p4.org/

[8] "Barefoot tofino," 2017. [Online]. Available: https://barefootnetworks.com/products/brief-tofino/

[9] "P4fpga website," 2017. [Online]. Available: https://p4fpga.github.io/

[10] "Pisces website," 2017. [Online]. Available: http://pisces.cs.princeton.edu/

[11] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 101–114.

[12] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran, "Athena: A framework for scalable anomaly detection in software-defined networks," in *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*. IEEE, 2017, pp. 249–260.

[13] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[14] 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.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[16] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *Operating Systems Design and Implementation*, vol. 10, 2010, pp. 1–6.

[17] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. IEEE, 2014, pp. 1–6.

[18] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2011, pp. 161–180.

[19] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE, 2010, pp. 408–415.

[20] "The caida ucsd anonymized internet traces - 2014," 2018. [Online]. Available: http://www.caida.org/data/passive/passive$_{dataset.xml}$