Hindawi
Security and Communication Networks
Volume 2018, Article ID 2905730, 23 pages
https://doi.org/10.1155/2018/2905730

Research Article

Network Programming and Probabilistic Sketching for Securing the Data Plane

Maha Shamseddine,[1] Wassim Itani,[2] Ali Chehab,[1] and Ayman Kayssi[1]

[1]*Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon*
[2]*Department of Electrical and Computer Engineering, Beirut Arab University, Beirut, Lebanon*

Correspondence should be addressed to Wassim Itani; wassim.itani@bau.edu.lb

Received 6 February 2018; Revised 7 May 2018; Accepted 17 May 2018; Published 28 June 2018

Academic Editor: Roberto Di Pietro

This paper presents VISKA, a cloud security service for dynamically detecting malicious switching elements in software defined networking (SDN) infrastructures. The main contributions of VISKA lie in (1) utilizing network programming and secure probabilistic sketching in SDN environments to dynamically detect and isolate parts of the data plane that experience malicious behavior, (2) applying a set of focused packet probing and sketching mechanisms on isolated network partitions/views rather than focusing the security mechanisms on the whole physical network, (3) efficiently analyzing the network behavior of the resulting views by recursively partitioning them in a divide-and-conquer fashion to logarithmically reduce the problem size in order to localize abnormal/malicious switching units, and (4) providing an attack categorization module that analyzes live ingress/egress traffic of the maliciously detected switch(es) solely to identify the specific type of attack, rather than inspecting the whole network traffic as is done in traditional intrusion detection systems. This significantly enhances the performance of attack detection and reduces the load on the controller. A testbed prototype implementation is realized on the Mininet network emulator. The experimental analysis corroborated the algorithms' convergence property using the linear and FatTree topologies with network sizes of up to 250 switches. Moreover, an implementation of the attack categorization module is realized and achieved an accuracy rate of over 90% for the different attack types supported.

1. Introduction

The next generation networking model adopted is the SDN network architecture which is based on the separation of the network control and configuration logic from the network switching logic, with SDN controllers having a fine-grained control over network routing and reconfiguration [1]. SDN networks, as is the case with any packet switching network, experience a major security risk represented in the malicious operation of the network forwarding units. With the widely adopted network and infrastructure cloud services, which support network tenants with off-premise network topologies, a compelling demand is realized for dedicated security services at the data plane to ensure that the switching units are not executing or participating in any active attack on network traffic. This dedicated security service must provide, with high confidence, SDN tenants with sufficient guarantees that the network they are running their applications on is free of malicious activities on the data plane. Moreover, such a service should (1) trigger security alarms in real time, (2) be efficient in applying the network monitoring/probing operations using compact data structures, and most importantly (3) be specifically designed for securing SDN networks.

The flexibility and programmability features of the SDN network model provide appealing advantages for the advancement of network autonomous creation and configuration. The introduction of the concept of data plane/control plane separation significantly facilitates network programming and central control over the switching and routing mechanisms of the global network view [2].

In this work, we present VISKA, a cloud security service for SDN networks that tackles security breaches in the switching data plane by leveraging network programming and probabilistic sketching. The main focus in the literature

has been directed towards applying the security mechanisms at the whole network without taking advantage of applying these mechanisms at smaller subsets of the network to flexibly and dynamically localize misbehaving switching nodes. VISKA, on the other hand, provides an efficient probing mechanism on the network data plane and recursively partitions it into independent subnetworks to reach and isolate misbehaving activities at the switch granular level. The SDN control plane facilitates the isolation of the resulting network partitions/views by updating the necessary switches flow tables. The probing on each network view is carried out using efficient data summarization "sketches" that allow VISKA to detect with high accuracy and minimal memory requirements malicious deviation from the network forwarding behavior. When the VISKA probing and sketching algorithms are applied, the network view recursively gets divided by nearly half the size. The divide-and-conquer mechanism of the network parts continues until the malicious switches are localized. This process is of great significance to the VISKA security service since it results in an algorithmic complexity that is logarithmic in terms of the network size. After localizing the malicious elements of the network, a categorization mechanism is executed to detect the nature of the data plane malfunction. The malfunction could be a benign behavior such as an administrative misconfiguration in one or more switching units, an excessive communication delay resulting from congested switches, or even a malicious security attack. To provide exact attack categorization and mitigation, a security module scrutinizes live network traffic using data mining and analysis on the real ingress and egress flows of the malicious switch(es) solely rather than inspecting the entire network traffic flows as is the case in traditional intrusion detection systems. This significantly enhances the performance of attack detection and reduces the load on the controller. The output of this module identifies the type of attacks imposed by the maliciously detected switches. Accordingly, the control plane provides the necessary control plane mitigation mechanisms.

The proposed algorithms are implemented and analyzed on the Mininet [3] network emulation platform. The experimental analysis corroborated the algorithm's convergence property using the linear and *FatTree* topologies with network sizes of up to 250 switching units and comprising a defined set of malicious elements. A highly appealing application of the VISKA service is in the enforcement of net neutrality [4], a concept that forces network providers to treat all network traffic and services equally on their networks.

The rest of this paper is organized as follows: in Section 2, we provide a comprehensive literature review of the main SDN security models related to the work proposed in this paper. Section 3 presents the VISKA threat model and indicates the main security attacks that can be detected by the proposed security service. Section 4 discusses the security service design and presents the main algorithms for realizing the detection and localization of malicious switching behavior and the categorization of possible attacks. In Section 5, we present a testbed implementation of the proposed security service on the Mininet network emulator. Conclusions are presented in Section 6.

2. Related Work

The great promises proposed by the SDN networking architecture in terms of centralized network visibility and data plane programming have dramatically increased this architecture's adoption in both SDN-compliant hardware and software services. Security is one of the top of the list challenges facing SDN today, specifically when the network encompasses untrusted data planes whose switching components are configured and managed by several external providers. A lot of research works targeted the security aspects in modern SDN networks. The authors in [5] present a comprehensive survey on the topic focusing on the modification attacks that might be executed on the network data by the programmed forwarding units in the data plane. The paper stresses on the fact that the OpenFlow protocol specification [6] mentions the use of the Transport Layer Security (TLS) protocol [7] for enabling the mutual authentication between the SDN controller and the data plane switches and not among the switches themselves. Moreover, this controller-switch TLS authentication mechanism is optional in the specification, which renders most of the prominent SDN providers not adopting this authentication mechanism. This is the case in the majority of open source controllers and switches. Accordingly, this lack of TLS adoption can lead to successful man-in-the-middle (MITM) attacks that impersonate the controller and manipulate the control messages exchanged between the controller and the switches. OpenFlow does not consider any formal security mechanism for switch-to-switch communication, which aggravates the possibility of effective MITM attacks in the data plane. The survey in [5] concludes by a set of best practices that should be considered when deploying SDN networks to moderate the security risks imposed by logic centralization and data plane programmability.

A more focused survey on the security implications resulting from data plane programmability is presented in [8]. In this paper the authors focus on the security vulnerabilities that may arise due to the inclusion of state maintenance primitives in the forwarding units of the SDN data plane. The main challenges here are system security attacks on the switch's memory and CPU and MITM attacks due to the lack of authentication among the switches in the data plane. The paper presents detailed attack scenarios of the above-mentioned vulnerabilities but does not provide any attack mitigation mechanism.

In [1], the authors present a comprehensive security survey that summarizes the security threats of SDN frameworks and categorizes them based on the layers and SDN interface vulnerabilities. On the other hand, the survey discusses and categorizes the security solutions based on the SDN network programmability infrastructure. The centralization of network programming has introduced both security threats and at the same time new and dynamic security solutions. Most of the solutions involve middle boxes that enforce the network security policy and adjustment in the security monitoring and prevention capabilities. In [11], the various SDN threats and vulnerabilities are discussed with a thorough analysis. The work proposed a secure mechanism that targets each introduced SDN threat vector including network OS

replication, application level replication, and software and hardware solutions on the control plane to avoid common mode faults and bugs and increase the network tolerance to hardware and software accidents and malicious behaviors. Moreover, the authors introduced self-healing mechanisms, isolated security domains, fast and dynamic network recovery, and redundant switch-controller association mechanisms. This work represents a call for action to trigger further research in SDN security solutions. In [12], the authors tackle the problem of lack of trust in the network OS and applications running on top of them where redundant controllers were introduced to the SDN network and a new layer is created to compare the output of the controllers and ensure consistency among the controllers and the network state and policies. This paper lays the ground for designing a trust scheme for redundant controllers in SDN.

The work in [13] presents a formal verification methodology to ensure the safety, security, and reliability of SDN applications that have access to network monitoring APIs using the *OpenFlow* [6] semantics. However, it introduces some limitations in verifying the network reliability properties, which was justified due to the complexities and nonstandardized network topologies in SDN architectures. *FRESCO* in [14] introduced an *OpenFlow* security application framework, which facilitates rapid and dynamic creation and deployment of security functions for attack detection and mitigation at the *OpenFlow* layer.

In [15], the authors introduce a framework of multiple distributed controllers that coordinate SDN control to achieve high scalability and security measures. This is achieved via a cluster-based mechanism that allows the dynamic addition and removal of controllers to the network without network interruption and down time. Any type of *OpenFlow* controller can be used in the proposed framework where the switches and applications are unaware of the underlying reassignment of controllers. The JGroups tool, introduced in [16], is used to synchronize controllers and to ensure correct controller-switch mapping. This work recommends the deployment of multiple redundant controllers in the SDN infrastructure without any consideration to the performance and security implications.

In [17], the authors proposed a security framework to detect suspicious changes in network topology and the SDN data plane. The work uses the flow graphs abstraction to approximate the network operations and thereby detect any suspicious deviation that may be considered as an attack. The main limitation of the work is that the detection mechanism is nondeterministic and is dependent on the accuracy of the flow graph approximation mechanisms.

In [18], the authors presented a traffic monitoring system in SDN based on sketches. This model, named "Open Sketch", can support the detection of suspicious traffic surges that may spring as a result of a denial of service (DoS) attack on a particular network part. The main limitation in this work is mainly related to the following points:

(1) The monitoring operates on the physical network layer of the SDN model, which, as stated previously, renders it a traditional network security solution with no focus on the advantages of network softwarization and programming.

(2) Software engines running on SDN switches themselves carry out the sketch calculation and updates. Trusting the switches in calculating the sketches can falsify the resulting traffic monitoring measurements by compromised switches and accordingly can mislead any security decision related to the source of possible attacks.

(3) The work mainly focuses on detecting suspicious deviations in network traffic (similar to [19]) and does not address traffic dropping, augmenting, and modification attacks.

Several research works have proposed the application of machine learning (ML) techniques to provide intrusion detection services in the SDN network architecture [20]. These approaches mainly focused on Deep Learning (DL) and classification algorithms to enhance the accuracy of the intrusion detection system and maintain low false posities rates. In [20] the authors present a DL-based system for detecting distributed denial of service attacks in SDN. The system is implemented as a network application on top of the POX controller. DL is mainly employed for traffic classification and for reducing the large set of features extracted for the packet headers and needed for attack detection. The main limitation in [20] is the high processing resources it requires on the SDN controller in the packet collection and features extraction phases. In this DL model, every network packet across the whole network is collected for feature extraction which imposes a sizeable load on the SDN controller. This fact is aggravated in vast SDN networks composed of a large number of forwarding switches in the data plane which results in a serious bottleneck on the SDN controller. The VISKA SDN attack categorization model presented in this paper targets the detection of more attack types in addition to denial of service attacks such as interruption attacks, blocking attacks, and man-in-the-middle attacks. Moreover, VISKA imposes minimal overhead on the SDN controller by isolating the source of attack using a highly efficient probing mechanism before proceeding with attack categorization. As a result, the attack categorization module on the SDN controller needs to collect the ingress/egress network packets of a small set of switches that are detected malicious instead of collecting the entire network traffic.

A similar DL-based approach is presented in [21]. In this work Tang et al. propose a flow-based anomaly detection system based on deep neural networks for intrusion detection in SDN. This model uses a limited number of network features for attack detection for the purpose of enhancing the feature extraction process. The main limitation in [21] is represented in the accuracy of attack detection which reaches 75.75%. This renders it infeasible for competing with existing intrusion detection systems or for application in commercial products. In [22] the authors propose a framework for detecting and classifying anomalies in SDN using information theory and machine learning techniques. The network traffic profiles are collected using Sflow [23]. The process consumes high memory and processing power to analyze traffic information

and inspect packets. The resulting framework identifies flows as malicious, benign, or unknown to be further analyzed. In [24] the authors address DDoS attacks based on Support Vector Machine (SVM) classification algorithms in SDN environments. The design is based on the information collected from the switches flow table states. The flow table information is used to create six-tuple characteristic values based on which the SVM algorithm classifies traffic as normal or attacker abnormal traffic. The disadvantages of this work are that it only addresses the DDoS attack on one hand; on the other hand, the training phase has to be executed on real network data and on predetermined periods of time to ensure the correctness of the resulting classifier model. This necessitates more computing resources and processing power on the SDN controller. Similar ML-based intrusion detection approaches are presented in [25, 26].

Part of the work presented in this paper appeared in the proceedings of the IEEE International Conference on Communications (ICC'17) [27]. We comprehensively improved the article and added significant extensions and technical details to the protocol design and implementation addressing delay attacks, early attack detection, and categorization.

3. Adversary Model

The VISKA security service operates in a typical SDN network composed of a set of physical switching elements (data plane) configured and controlled by one or more controllers (control plane). The control plane is responsible of configuring the data plane with the necessary flow rules that form the basis of the switching units' flow tables. The communication between the control and data planes is governed by the rules of a protocol such as *OpenFlow*.

The adversary model we consider in this work is represented by a set of switching nodes within the SDN physical network. VISKA is capable of detecting, with high confidence levels, active attacks related to malicious or misbehaving switch operation and localizing the source(s) of the attacks. Active attacks mainly include packet modification, packet dropping, and packet injection, which induce a deviation from the normal network behavior. These attacks are analyzed and categorized in order to further secure the data plane and the control plane in the underlying SDN network. Examples of such active attacks consist of one or more switches colluding to

(1) inject malicious packets for the purpose of instigating DoS attacks on both layers, data and control, of the SDN network;

(2) drop network packets for the purpose of maliciously occluding particular network flows;

(3) augment network flows with padding packets to conceal the malicious effect of packet dropping;

(4) modify the contents of packets to cause traffic rerouting, to execute man-in-the-middle attacks, or to poison particular network flows;

(5) delay the forwarding of network traffic to disrupt the quality of service (QoS) of the SDN network.

VISKA assumes that the SDN controller is trusted and free of malicious security vulnerabilities. In other words, the controller is expected to be operated by legitimate administrative authorities and that it executes valid code that delivers authentic flow rules to the data plane switching units.

The VISKA service can be divided into two complementary modules: (1) packet probing-based security module for detecting malicious data plane elements; (2) real network data-based module for categorizing attacks and creating signatures for novel attacks within the SDN network.

The VISKA security algorithms are designed to operate in a highly malicious SDN environment and can tolerate relatively large number of misbehaving switches. This comes at the expense of the time complexity of the attack localization algorithms as will be demonstrated in Section 4.5. The accurate localization of the attack source highly facilitates the process of mitigating the attack. This is one of the great security advantages provided by the VISKA service. The mitigation strategy proceeds by firstly ceasing the malicious switch(es) forwarding activities and reporting this action, together with the details of the categorized attack, to the SDN service provider. The latter can administratively execute the necessary technical actions to inspect and possibly rectify the configuration and operational context of the malicious source(s) to resume its/their forwarding activities by leveraging the control plane global network view and the OpenFlow protocol.

4. System Design

The VISKA service architecture, as depicted in Figure 1, utilizes network programming for recursively partitioning the SDN data plane. The controller routing and forwarding achieved through OpenFlow messages on the data plane allow for the segregation of network partitions, consequently isolating parts of the SDN network referred to as views that would recursively map to the malicious switches, if any. To achieve the goal of localizing maliciously behaving switches, a graph-theoretic partitioning algorithm recursively divides the data plane network into two equal-degree network partitions that have minimal interconnecting edges. Each network partition is probed by a set of data packets dynamically generated by a probing module on top of the SDN controller. The controller probing module consists of two processes, a sender process (P_s) responsible of generating and pushing the probing packets into the data plane, and a receiver process (P_r) responsible of receiving the probing packets from the data plane. The routing of the probing packets from P_s to P_r is transparently updated by the SDN controller in the switches' flow table entries.

To achieve the goal of real-time detection of malicious activities in the network data plane, the Tug-of-war sketch data structure is employed on the probing streams. Sketches are probabilistic data structures that compactly represent the frequency of occurrences of items in data streams using a hashing function in sublinear space. Sketching algorithms are adopted in this work due to their efficient summarization of large data sets which allows VISKA to detect deviations in detect abnormal switch behavior in real time. For each

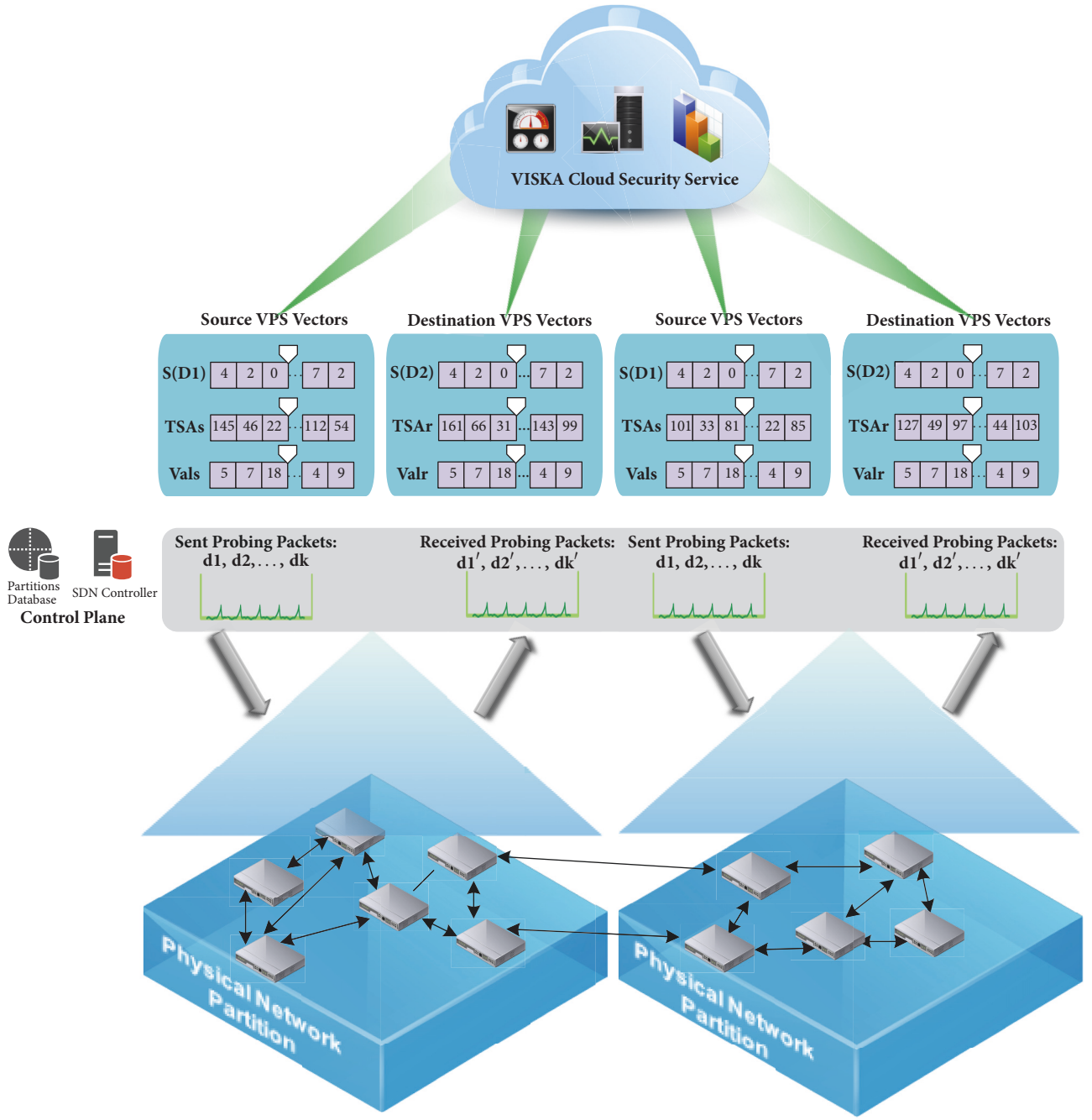

Figure 1: VISKA's conceptual SDN design. S(D1) and S(D2): the source and destination sketch data structures, TSAs and TSAr: the sender and receiver time stamp accumulator data structures, and Vals and Valr: the sender and receiver validity vectors.

probing time interval t, the computations of summarized sketches of the probing packets are generated at P_s and are appended with a timestamp accumulator indicating the packets' transmission time, t_s. The sketch data structure and the timestamps are sent to the active security service in the cloud for inspection. Analogously, the receiver process, P_r, computes and sends the sketch of the received probing packets and the corresponding packet receipt timestamps. The VISKA cloud security service algorithms compile the data structures received from P_s and P_r in order to

(1) recognize the levels of deviation between the data sketches of the sent and received probing packets,

(2) compute the average time delay on the probe path based on the sent and received timestamp accumulators.

The VISKA algorithms use these computations in order to decide on the probability of malicious switch behavior in the corresponding network partition and furthermore categorize the type of attack in the infected regions of the network by

inspecting the real traffic on the egress and ingress ports of exclusively the maliciously detected switch(es) and not of the whole network traffic.

The VISKA algorithms are thoroughly elaborated in the following subsections. It is worth mentioning here that the VISKA procedures utilize sizeable probing data streams and timestamps to ensure the accurate detection of misbehaving switches along the recursively generated network partitions. Sketching data structures in such setup lead to major reduction in computational complexity, better utilization of storage, and as a consequence, a performance-efficient real-time malicious detection.

4.1. The View Probing and Sketching Algorithm (VPS). The View Probing and Sketching (VPS) algorithm produces sketch summaries of the probing data at the source and destination controller processes by utilizing the Tug-of-war sketching algorithm [28]. The probing packet stream ($D : (d_0 \rightarrow d_v)$) is sent from P_s to P_r by traversing all the switches in a given network view. A probe-route module running on the SDN network controller pushes the necessary forwarding rules to ensure that the probing packets visit each switch in the corresponding network partition.

At P_s and P_r, the probing stream D is fed to a sketch engine to produce a compact sketch representation $s(D)$ that is sent to the cloud security service for analysis. For each probing packet d_i, a four-wise independent hashing function $\eta[d_i]$ is applied, which uniformly maps to a pair of values: an index j in the sketch vector and a value v in $\{-1, +1\}$; v is added to $s(D)$ at index j. The timestamp is appended by the controller probing processes to the sketch data structure corresponding to the sent and arrival times, respectively, using accumulator data structures, as will be explained later in this section.

The sketch representation of the corresponding probing data stream is evaluated as the summation of the dot product of the hashed values and the data stream as follows: $s(D) = \sum_{i=0}^{v} d_i \cdot \eta[d_i]$ which is a randomized linear projection of the input data stream. The resulting $s(D)$ vector at time t is sent to the cloud security service for analysis. The linearity property of Tug-of-war sketch indicates using the same family of pseudorandom hashing functions, η, on two sets of data streams, D_1 and D_2, then for any constants a and b, $aD_1 + bD_2 = as(D_1) + bs(D_2)$. This linearity property is essential for estimating the difference between the two probe data streams (sent and received) along a network partition.

As a result of the linearity of this sketch based on [28], the second norm difference between the two received sketches reflects any deviation between the sent and received data streams subject to an ϵ error and with minimum probability of $(1 - \delta)$; thus

$$\left|s(D_1) - s(D_2)\right|^2 = \Delta, \quad \text{where } \Delta = (1 \pm \epsilon) \times \left|D_1 - D_2\right|^2 \tag{1}$$

The sketch's second norm difference estimation results in a more accurate representation of the deviation between the corresponding data streams. Such deviation indicates a malicious activity if Δ is greater than a preset threshold, τ. This probing and sketching procedure is repeated every time interval t to detect abnormal switch behavior in real time.

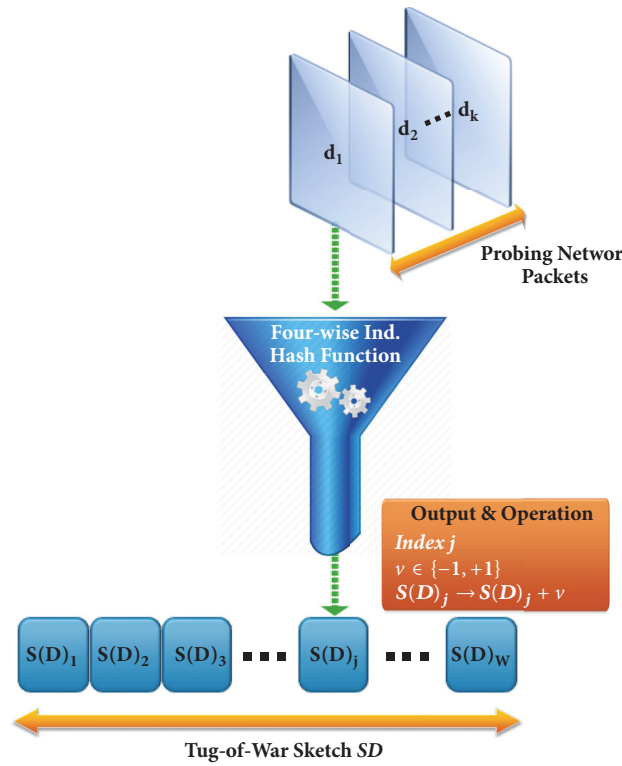

Figure 2: VISKA's sketch data structure on probing data.

The Tug-of-war probabilistic sketching algorithm was adopted in the security model because of its light-weight processing requirements which typically consists of simple hash function calculations. Moreover, the relatively small sized sketch data structure representation relative to the number of probing packets it summarizes induces minimal overhead for network transmission and reception as well as storage. The sketches $s(D_1)$ and $s(D_2)$ represent a compact hash representation of the probing data streams with $O((1/\epsilon^2)\log(1/\delta))$ computations where ϵ is the error and $(1 - \delta)$ is the confidence level. Considering the network and storage overhead imposed by the sketch representation, each sketch data structure is comprised of W counters of z bits each where $z = 1 + (1/2)\log(4(k/W)\ln(200\,W/\delta))$. This is comprehensively described in Section 4.5.

The timestamp accumulator (TSA) data structures are created and computed at the sender and receiver processes concurrently with the sketch creation for the aim of detecting delay-causing attacks or otherwise network congestion. Each probe packet is further hashed to a value c which is utilized as an index in the TSA_r and TSA_s vectors. These vectors represent the summation of the timestamps of the probing packets that map to the hash value c at the controller sender and receiver processes, respectively.

Figures 2 and 3, respectively, describe the sketch and timestamp accumulator data structures on the probing data. Each outgoing probing packet is passed to the sketch engine

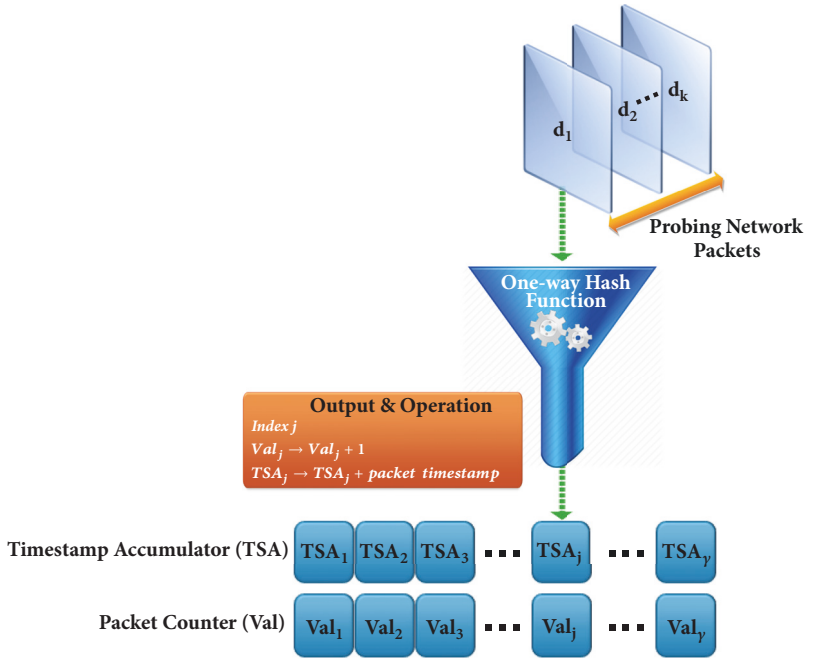

Figure 3: VISKA's timestamp accumulator data structure.

represented by the funnel symbol in Figure 2, to output an index i and a value v in $\{-1, 1\}$, which is appended to $s(D_1)$ at position i. The hashing function h is invoked on each packet to yield the index $\{-1, +1\}$ at which the timestamp of that packet is appended in the TSA_s and TSA_r vectors. When the probing stream is entirely transmitted, the sketch $s(D_1)$ comprises the corresponding sketch values and each TSA_s entry represents the sum of the time stamps of the packets according to the hash value mapping to TSA_s entry indices. In order to ensure the correctness of the packets timestamps, a validity vector (Val) is updated for each probing packet to count the number of packets according to their hashed value. Each time a timestamp is appended at index c, and the validity vector is incremented by one at that same index. Ultimately, the probing packets are sent to the controller process P_r while $s(D_1)$, TSA_s, and the validity vector Val_s are transferred to the VISKA cloud service for analysis.

Analogously, at the controller probing process P_r, the data structures $s(D_2)$, TSA_r, and Val_r are created and calculated. These data structures are sent to the VISKA service for comparison and malicious activity detection.

The VPS algorithm calculates the second norm difference Δ of the two sketches $s(D_1)$ and $s(D_2)$ received at the VISKA service. If Δ is greater than a preset threshold τ, the network is considered malicious and the attack categorization and summarization module MACM is invoked. On the other hand, the difference in the sent and received timestamps is calculated on the probing stream by the following:

(1) First, check that the validity vectors from sender and receiver at each index j are equal; this indicates that the corresponding packets are successfully received and the timestamp counters at that index j are valid.

(2) The timestamp difference $TSA_r[j] - TSA_s[j]$ is calculated and added to the total Δ_{TSA}, which is the total difference in timestamp of the current probing stream. The total time Δ_{TSA} is divided by the sum ValCount of the corresponding valid packet counts in vector Val_s and Val_r. This results in the average time delay of the correctly received probing stream as described in

$$\partial = \frac{(\text{TSAr} - \text{TSAs})}{\text{ValCount}} \tag{2}$$

If the value of ∂ exceeds a threshold Γ_d, the network is considered malicious. Otherwise, if the time delay is greater than the congestion threshold Γ_c, the corresponding data plane elements are considered congested.

The values of Γ_d and Γ_c depend on the overall network round trip time (RTT). Since RTT can change from one probing period to the other (even within an individual probing period), the values of Γ_d and Γ_c dynamically change based on this variation in RTT. To maintain a smooth variation in the values of Γ_d and Γ_c we followed an algorithm analogous to the retransmission timer calculation algorithm followed in TCP [29]. The details of the Γ_d and Γ_c calculations are presented in the following smoothing equations using the estimators mRTT and vRTT, respectively, representing the mean and variance of RTT in the i^{th} probing period.

$$\text{mRTT}_{i+1} = a \times RTT + (1 - a) \times \text{mRTT}_i \tag{3}$$

$$\mathrm{vRTT}_{i+1} = b \times \left(\left|RTT - \mathrm{mRTT}_i\right|\right) + (1-b) \times \mathrm{vRTT}_{i+1} \quad (4)$$

$$\Gamma_{c_{i+1}} = \mathrm{mRTT}_{i+1} + 4 \times \mathrm{vRTT}_{i+1} \quad (5)$$

$$\Gamma_{d_{i+1}} = 2 \times \Gamma_{c_{i+1}} \quad (6)$$

The gains a and b are set to 1/4 and 1/8, respectively.

In the first probing period the mean and variance estimators are set as follows:

$\mathrm{mRTT}_1 = RTT$;

$\mathrm{vRTT}_1 = RTT/2$.

(3) Finally, if the two values Δ and ∂ are within safe boundaries, the network is interpreted as normally operating.

After sending each sketch data and timestamp data structures, the probing hosts flush them to compute the following interval sketch.

The TLS protocol is implemented on the controller probing processes to ensure the integrity and authenticity of the source and destination sketches when transferred to the cloud service over the network links. When the VISKA service is to be adopted and executed in a real-world environment, it is very important to masquerade the patterns of the probing packets introduced in the network to prevent any malicious node ability to recognize VISKA functionality. This is addressed in the same sense wherein the software-based control on the probing processes and their parameters provide the VISKA probing module the control on randomizing certain fields in the probing packets IP header (e.g., Identification, TTL, Options and Padding, and ECN fields), the data sections are randomized to conceal any deterministic features that may reveal the probing nature of these packets.

It is worth noting here that the feasibility of using the timestamp accumulator data structure relies on the accurate time synchronization among the system clocks of the nodes in the network. To achieve this, we utilize the control plane centralization property of the SDN architecture by deploying a Network Time Protocol (NTP) [30] server on the SDN controller. NTP is the de facto standard in achieving high-quality time synchronization in modern Internet networking infrastructures. Relying on the local SDN controller in time synchronization instead of utilizing a remote NTP public server aids in a more precise time synchronization by avoiding the asymmetrical latency delays incurred by the NTP time packets exchanged between the probing module and the public time server. This results in a maximum of 0.5 to 1.5 msec time lag between any two network nodes. This is sufficient for correct operation of the VISKA timestamp accumulator realization.

4.2. The Network Views Partitioning Algorithm (NVP). To guarantee malicious-free switch behavior at the network data plane, the correct network functionality should be checked at the switch granularity level. In order to minimize the executions of the VPS algorithm, described in Section 4.1, in checking the physical switches, the SDN network is recursively partitioned into two semiuniform network partitions. Each resulting partition is probed for possible sources of deviation in the forwarding mechanism. Only partitions marked as malicious will be further subdivided using the NVP algorithm. In this sense, the NVP algorithm generates the network partitions and feeds them to the probing algorithm VPS to check any deviation in the corresponding current network partition. If the VPS algorithm indicates an above-threshold deviation in the sketch calculations and/or the timestamp analysis, the VISKA service recursively executes NVP to partition the respective network topology into two separate network views with minimum interconnecting edges. The two resulting network partitions are fed separately to the VPS algorithm at the controller for subsequent behavior check. This recursive partitioning method is applied until the algorithm isolates the source of maliciousness in the data plane, if present.

The efficient real-time detection and localization of malfunction were fully achieved in VISKA by ensuring the segregation of the network partitions owing to the programmable SDN network architecture. Two main approaches contributed to the feasibility of isolating the problem within separate network views:

(1) Karger's randomized algorithm for generating minimum graph cuts [31] was adopted in the NVP algorithm in order to partition the network graph into two separate sets (mapped to the corresponding network partitions) with minimum interconnecting edges.

(2) A probe-route module is executed on the SDN controller for the probing packets to traverse the switches incorporated within a network view. The controller pushes the necessary rules to the data plane switches to restrain the probing data stream to the corresponding network partition, thus ensuring network views segregation.

This recursive partitioning of the network views isolates the malfunctioning views, which results in a near logarithmic time complexity in the size of the network.

Karger's graph cut algorithm is based on the contraction of edges and merging the nodes in a connected unidirectional graph $G(n, e)$. This algorithm continues by randomly selecting nodes and merging them iteratively until the graph is reduced into two sets represented by two vertices. These two sets remain connected; to minimize the connecting edges between the two sets, Karger's algorithm repeats this contraction procedure a predefined number of times until a minimum graph cut is produced between the two partitions with a high probability. For a graph of n vertices and e edges, Karger's contraction method returns a minimum graph cut with a probability of success: $\mathrm{P_c} \geq \binom{n}{2}^{-1}$ and a probability of not attaining a minimum cut of $\mathrm{P_{nc}} \leq 1/n$

The algorithm randomly repeats the contraction procedure $R = \binom{n}{2} \ln(n)$ times in order to arrive at a minimum cut in time complexity of $O(R \cdot e) = O(n^2 e \log n)$.

In summary, the network topology graph is input to the NVP partitioning algorithm, which outputs two sets of switches constituting two separate network partitions with minimum interconnecting edges. Each partition is fed to the VPS algorithm to check if any malfunctioning is present. Depending on the output of the VPS algorithm, a network view/partition is either rendered correctly functioning and is thus discarded, or malfunctioning and thus, it is recursively partitioned by the NVP algorithm until the size of the switches in the respective partition is less than or equal to a minimum, m.

4.3. The Malfunction and Attack Categorization and Summarization Module (MACM). The first stage of the VISKA service operation is the VISKA malicious switch detection (Sections 4.1 and 4.2) where the VISKA VPS algorithm returns the switch(es) that was/were classified as malicious. The second stage of the VISKA service is the MACM module which is responsible of identifying and categorizing the attacks induced by the malicious network elements. The MACM module is invoked in Algorithm 1 in the VPS function when a malicious activity is detected. This stage is essential for securing the SDN network provider services. The SDN provider utilizes the VISKA service to detect malicious operation in its data plane. VISKA aids in guarding against an important set of attacks that initiate in these network infrastructures such as DoS, interruption, blocking, delay, and man-in-the-middle attacks. The second stage of the VISKA service, the MACM, primarily identifies two classes of data plane malfunction:

(1) The distorted traffic malfunction class (CatI), where the second norm difference of the sent and received sketches is beyond a preset threshold τ, which reflects a malicious deviation in the probing stream introduced by the data plane elements of the current network partition.

(2) The time delay malfunction class (CatII), where the packets are received by the probing host correctly (no significant sketch difference is recognized $\Delta < \tau$); however, the average time delay of the transmitted probe packet stream is beyond a normal network congestion value, $\partial \geq \Gamma_d$.

In the case of CatI malfunction detection, the maliciously categorized data plane switching elements are further investigated to summarize the attack in order to deduce and block probable network wide malfunction.

The egress and ingress traffic of the malicious detected switches are collected for certain time periods t_i in order to categorize and summarize the investigated attack. The SDN infrastructure is utilized in investigating the traffic ingress and egress of the malicious switches by forwarding traffic in and out of the malicious switch to the controller. The set of switches, mc, that are one-hop away from the malicious switch in the network, are primarily identified. A data collecting module running on the controller sends the mc switches the necessary action rules that dictate sending all packets having the malicious switches as their next hop to the controller. The controller monitoring module utilizes data mining features on the periodically collected data to categorize and summarize the attack or otherwise specify the malfunction as a benign behavior.

In order to achieve anomaly categorization and early stage attack detection, the VISKA cloud service primarily identifies the malicious switches in the VPS and NVP algorithms. Next, the VISKA MACM algorithm exploits the SDN controller centralized network programmability to capture the egress and ingress traffic of the malicious switches on the controller as depicted in Figure 4. These packets are first prepared and analyzed by grouping them according to source address, destination address, source port, and destination port. Important packet header fields are stored and grouped at the controller and are prepared for comparison and categorization at the VISKA cloud service to identify and specify potential network attacks. The MACM process consists of the following three phases.

Phase 1. The analysis of the egress and ingress traffic of the m switches is demonstrated in Figure 5. The packets are sent by the malicious switches neighbouring switches (mc) to the controller. The controller in turn captures these packets and passes them to the MACM module, which stores the necessary header information of the captured packets. The collected data is prepared and grouped according to specific header fields in information tables.

In Figure 5, two hashing functions are applied on specific fields of the packets' headers in order to find certain patterns and characteristics in the captured traffic for time intervals t_i. First, the packets' destination IP addresses are input to the hashing function hashd. Packets with the same hashed destination IP address are aggregated. The resulting E_Dest table includes the packet information categorized by their corresponding destination IP addresses. Analogously, another hash function hashs is applied on the source address of the captured packets to categorize and prepare the source address aggregation table (E_Src).

This procedure of hashing and aggregation is done on both egress and ingress traffic to prepare the data for analysis by the MACM algorithm. The prepared attributes that are stored for analysis in the aggregation tables for each packet include the source IP, destination IP, transport protocol, SYN packet, and ACK packet flags. This collected packet information is ready to be analyzed in phase 2.

Phase 2. The traffic information tables collected in phase 1 from the egress and ingress ports of the malicious data elements are analyzed, and certain features are extracted for the purpose of attack categorization. The following is a list of the parameters and features characterizing the collected packets:

(1) I_Sum(): the sum of the size of the ingress packet flow in bytes.

(2) E_Sum(): the sum of the size of the egress packet flow in bytes.

(3) I_count(destIP): the number of ingress packets with the same destination IP, destIP

VISKA Malicious Switch(es) Detection/Attack Categorization Algorithm
Let **G** be the graph representing the SDN network
Let **n** be number of switches in G
Let **e** be the number of edges in G
Let **V** be the granular network view/partition to be checked for maliciousness. Initially V = G
Let **m** be the maximum malicious partition size (m = 1 to reach single switch granularity)
Let $\boldsymbol{P_s}$, $\boldsymbol{P_r}$ be the controller probing processes allocated for probing the bootstrapping network view V
VISKA(V, $\boldsymbol{P_s}$, $\boldsymbol{P_r}$)
If (**VPS(V, $\boldsymbol{P_s}$, $\boldsymbol{P_r}$)** = malicious)
if ($n <= m$)
return V (containing malicious switch)
else
(V1, V2,$\boldsymbol{P_s}$, $\boldsymbol{P_r}$) = NVP (G, n, e)
VISKA(V1, $\boldsymbol{P_s}$, $\boldsymbol{P_r}$)
VISKA(V2, $\boldsymbol{P_s}$, $\boldsymbol{P_r}$)
else return (correct or congested partition behavior)
VPS: View Probing and Sketching function
VPS(V, $\boldsymbol{P_s}$, $\boldsymbol{P_r}$)
Randomly **Generate** probing data D1($d_1 \ldots d_k$) at $\boldsymbol{P_s}$
and send to $\boldsymbol{P_r}$
sketch: create sketch data vector $s = 0$, and TSAs = 0, counts=0 at $\boldsymbol{P_s}$
for each v packet d_v in D1
compute (i, value) = $\eta_j[d_v]$
insert at index i in sketch $s(D1)$ vector:
$s(D1)_i += value$
compute (j) = $h(d_v)$
$TSAs_j += sent\ timestamp$
$(V_s)_j += 1$
send: $s(D1), TSAs, Val_s$ to VISKA service for analysis
Compute: (steps **sketch** through **send**) on $\boldsymbol{P_r}$ to generate and send $vectors\ s(D2), TSAr, and\ Val_r$
At VISKA
count=0
$\Delta = |s(D1) - s(D2)|^2$
for i from 1 to k
if $Val_{s_i} = Val_{r_i}$
$\Delta_{TSA} += TSAr_i - TSAs_i$
ValCount+ = $(V_s)_i$
$\partial = \Delta_{TSA}$/ValCount
if $\Delta \geq$ threshold τ **or** $\partial \geq \Gamma_d$
return malicious, call MACM (Δ, ∂)
else if $\partial \geq \Gamma_c$
return correct, congested
else
return correct
NVP: Network Views Partitioning Function
NVP (G, n, e)
(G1, G2) = **Karger(G, n, e)**
Insert forwarding rules for the probing packets
on controller
At the SDN network controller
(i) Isolate network partitions V1, V2 corresponding
to the Karger output **(G1, G2)**
return(V1, V2, $\boldsymbol{P_s}$, $\boldsymbol{P_r}$)
MACM: Malfunction and Attack Categorization Module
MACM (Δ, ∂)
if $n \leq m$
if $\Delta \geq \tau$
$CatI = 1$ (attack $\in$ to Category I where an active attack is being initiated in the network)
else // $\partial \geq \Gamma_d$
$CatII = 1$(attack $\in$ to Category II where a time delay introducing attack is introduced in the network)
if $CatI = 1$
malicious switch(es) ingress and egress traffic is collected and mined:
if E_Sum( )/I_Sum( ) > $\Gamma\mathsf{E}d$

Algorithm 1: Continued.

```
for each destIP in table E_Dest
   if (E_Sum(destIP)-I_Sum(destIP)>  Γdos) AND
     (E_count(destIP) – I_count(destIP)> Γp)
   alarm: DoS on destIP (Flooding)
   if (E_count_ACK(destIP)-E_count_SYN(destIP) <  Γcon)
    AND (E_Avg(destIP)< Γsyn) AND ((E_count(destIP)
    – I_count(destIP)>  Γp)
   alarm: DoS on destIP (SYN attack)
else if E_Sum()/I_Sum() <  ΓId
   for each destIP in table E_Dest
      if (E_count(destIP) – I_count(destIP))<  Γbh
       alarm: interruption of traffic to destIP
else // E_Sum()/I_Sum() within boundaries)
   for each destIP in table I_Dest
      if (dest-IP is not in E_Dest) AND
(I_count(destIP)- E_count(destIP) <  Γbh)
      alarm: blocking on dest IP
   for each srcIP in table I_Src
       if (srcIP is not in E_Src) AND (I_count(srcIP)-
        E_count(srcIP) <  Γbh)
       alarm: blocking on srcIP
   for each destIP in E_Dest
      if destIP is not in I_Dest AND
E_count(destIP)>  Γmitm
      alarm: MITM attack at destIP
```

Algorithm 1: VISKA algorithms.

Table 1

E_Sum(destIP)	*E_count(srcIP)*	*E_count_srcIP(destIP)*
E_Avg(destIP)	*E_count(destIP)*	*E_count_SYN(destIP)*
E_SD(destIP)		*E_count_ACK(destIP)*

(4) I_count_srcIP(destIP): the count of the different source IPs for the ingress flows with the same destination IP, destIP.

(5) I_Sum(destIP), I_Avg(destIP), I_SD(destIP): the sum, average and standard deviation, respectively, of the size of the ingress packet flow with the same destination IP address in bytes.

(6) I_count_SYN(destIP), I_count_ACK(destIP): the number of ingress packets of type SYN and ACK, respectively, with the same destination IP address.

(7) I_count(srcIP): number of ingress packets with the same source IP, srcIP.

(8) I_count_SYN(srcIP), I_count_ACK(srcIP): the number of ingress packets of type SYN and ACK, respectively, with the same destination IP address.

Similarly, the parameters computed for the egress traffic are shown in Table 1.

Phase 3. The information tables, the features, and the characteristics of the ingress and egress data, which are collected and computed in phases 1 and 2, are subsequently utilized and analyzed to categorize the network attacks induced by the m switches. The following is a list of the MACM identified attacks.

(1) DoS on the Controller. The egress traffic is checked for packets where the controller is the destination address including packet-in messages from the malicious switch. If the number of these packets exceeds a certain permitted threshold for normal network operation Γ_c, then the switch is considered "DoSing" the controller.

(2) DoS on Network Node. This attack could be on a host or switching element in the network. This is identified by checking the count of the egress packets with the same destination address. When this count is larger than that of the ingress packets and at the same time it is beyond a threshold, Γ_{dos}, the corresponding destination is detected to be DoSed, and further analysis is done by the network administrator to identify the rival attack. The VISKA algorithms utilize the SDN centralization and programmability features for early detection of such DoS attacks on the network nodes in real-time and in early stages. As a result, the VISKA service secures the tenants SDN network against presumable DoS attacks.

(3) Data Blocking Attack. This intruder attack selectively blocks traffic to specific destinations in the network for the aim of inducing erroneous network operation. Such attacks decrease the tenants' trust in the corresponding SDN network provider. To avoid this, the VISKA MACM algorithms inspect the ingress and egress packets of the malicious switches (detected by the VPS VISKA algorithms)

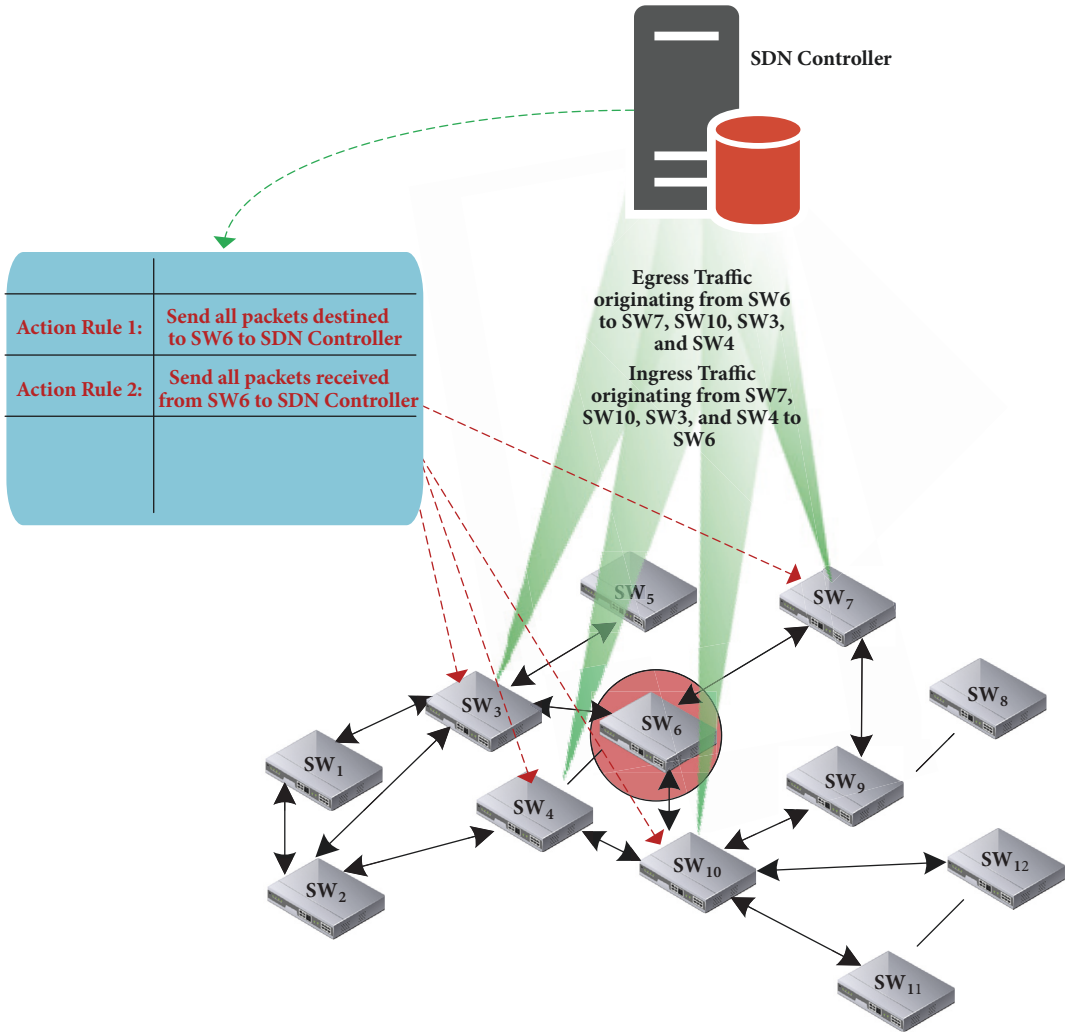

Figure 4: Ingress/egress traffic capture at the controller of the m switches neighbouring switches mc.

and if the egress traffic is found to be less by a certain network permissible threshold, Γ_{bh}, from the ingress one, the malicious switches are identified to be blocking network traffic. The blocking attack is therefore detected on the destination address or from the specific source address in the investigated network traffic. The IP addresses of the captured packets that were blocked are further analyzed and the involved network elements are inspected by the network administrator.

(4) Man-in-the-Middle (MITM) Attack. VISKA algorithms detect MITM attacks at very early stages in real-time. MITM attack prevention is of high significance to network tenants to ensure network confidentiality and integrity in the cloud. The MACM captures packets at the malicious switches and checks the homogeneity of all source-destination packet flows on the egress and ingress ports of the malicious switch. If the flows are not homogeneous within a certain permissible network limit, the packets are further investigated by checking the size of traffic to a single unique destination address with the same group of source addresses in the corresponding flows. This destination IP does not appear in the consequent flows on the ingress ports and concurrently, the count of the source addresses is the same as the investigated source address classified packet counts. Therefore, the same packets are being forwarded to another destination host, which indicates a MITM attack.

Note that the attack detection mechanism is extendable to address additional attack categories based on tenants' demands and for specific network requirements. This is made feasibly by leveraging the controller's programmability features in SDN platforms and the modular VISKA attack detection module design.

The VISKA attack detection and categorization algorithms are summarized in Algorithm 1. A block diagram summarizing the architecture of the VISKA algorithmic modules and their interaction is illustrated in Figure 6.

The values of the thresholds in the MACM module are set empirically and are bounded in the ranges depicted in Table 2. The threshold values are dependent on the maximum throughput of the switching elements in the data plane, the average packet size, the time of ingress/egress flow collection,

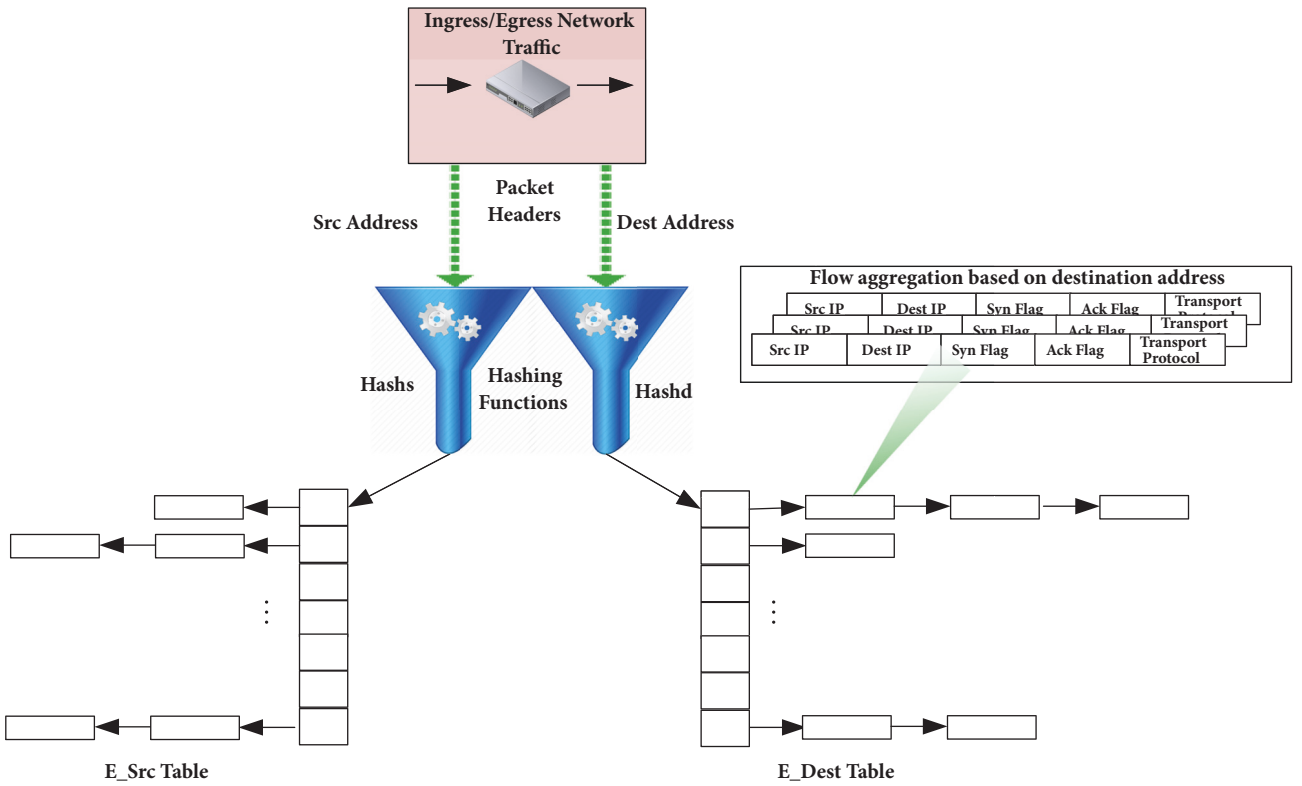

Figure 5: Egress flow summarization aggregated based on source and destination IPs (analogous hashing structures are applied on the Ingress traffic).

Table 2: Attack detection threshold ranges.

Attack type	Threshold(s) Range
Denial of Service Attack	$(1+\alpha_l) \le \Gamma Ed \le 3 \times (1+\alpha_l)$ $\lfloor P_{avg} \times X_{max} \times t_i \times \alpha_l \rfloor \le \Gamma dos \times 10^5 \le \lfloor 2 \times P_{avg} \times X_{max} \times t_i \times \alpha_h \rfloor$ $\left\lfloor \frac{\Gamma dos_{low}}{P_{avg}} \right\rfloor \le \Gamma p \le \left\lfloor \frac{\Gamma dos_{high}}{P_{avg}} \right\rfloor$
Interruption/Blocking Attack	$\lfloor X_{max} \times t_i \times \beta_l \rfloor \le \Gamma bh \times 10^5 \le \lfloor X_{max} \times t_i \times \beta_h \rfloor$
Man-in-the-Middle Attack	$\lfloor X_{max} \times t_i \times \gamma_l \rfloor \le \Gamma mitm \times 10^5 \le \lfloor X_{max} \times t_i \times \gamma_h \rfloor$

and the minimum and maximum percent of packets generated at the maximum switch throughput in the collection time period that are necessary to initiate a particular attack. This network-specific specification of the threshold ranges facilitates a more accurate threshold selection mechanism in real SDN network environments. The individual threshold range parameters for each attack type are presented in Table 2.

$\mathbf{P_{avg}}$ is the average size of a packet in the deployment network.

$\boldsymbol{X_{max}}$ is the maximum throughput of the switching elements in the deployment network.

$\boldsymbol{t_i}$ is the data collection time period used by the MACM module.

$\boldsymbol{\alpha_l}$, $\boldsymbol{\alpha_h}$ are, respectively, the minimum and maximum percent of packets generated by the malicious switching element at the maximum throughput X_{max} in the time period t_i (multiplied by a factor of 10^{-5}) necessary to initiate a DoS attack.

$\boldsymbol{\beta_l}$, $\boldsymbol{\beta_h}$ analogous to α_l and α_h, β_l, and β_h are, respectively, the minimum and maximum percent of packets generated by the malicious switching element at the maximum throughput X_{max} in the time period t_i (multiplied by a factor of 10^{-5}) necessary to initiate an Interruption/Blocking attack.

$\boldsymbol{\gamma_l}$, $\boldsymbol{\gamma_h}$ analogous to α_l and α_h, γ_l, and γ_h are, respectively, the minimum and maximum percent of packets generated by the malicious switching element at the maximum throughput X_{max} in the time period t_i (multiplied by a factor of 10^{-5}) necessary to initiate an Interruption/Blocking attack.

According to the nature of the detected attack in the network, the controller mitigates the attack by sending the necessary flow rules to isolate and suspend the operation of the maliciously detected forwarding element in the data plane.

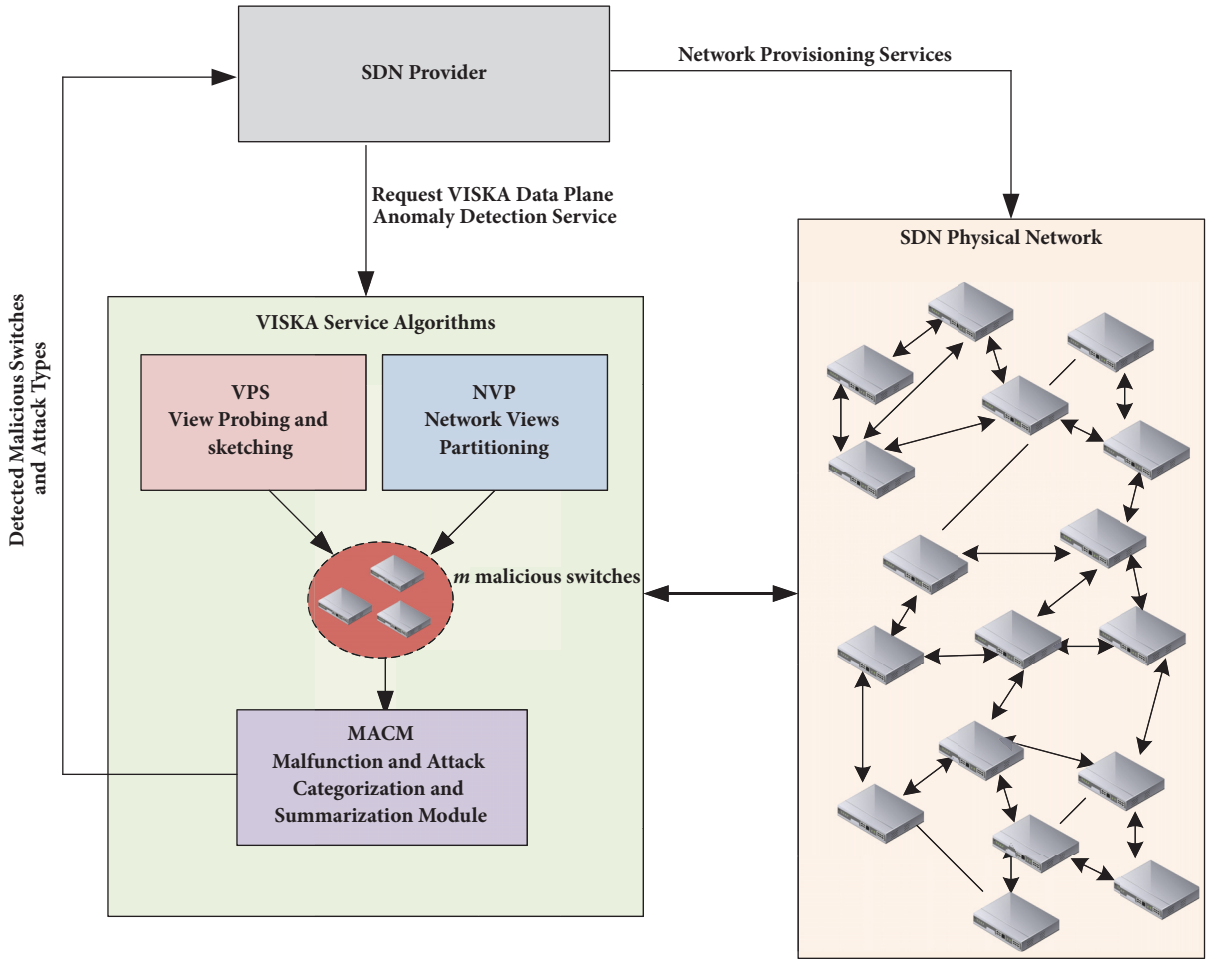

Figure 6: VISKA general architecture.

4.4. Algorithm Complexity and Convergence Analysis

4.4.1. Analysis of the Malicious Switch Detection Functions. In this section, we provide a mathematical complexity analysis of the worst case, average case, and best case running times of the VISKA malicious switch detection algorithms as a function of the network size. Moreover, we present the cost of the MACM malfunction and attack categorization algorithm invoked when a malicious activity is detected.

Worst Case Analysis. The worst case scenario arises when a malicious behavior is detected by the VPS function in every recursive network partition provided by the NVP partitioning function. The worst case runtime complexity $T(n)$ is given by

$$T_w(n) = 2T_w\left(\frac{n}{2}\right) + \text{Cost (NVP)} + \text{Cost (VPS)} \quad (7)$$

where n is the network size and Cost(NVP) and Cost(VPS) are the costs of running the NVP and VPS functions, respectively. The multiplicative factor of 2, in $2T_w(n/2)$, indicates that the recursive steps are applied on the two network partitions produced by the NVP function. This case arises when the VPS algorithm detects malicious activity in both network views. From the previous subsections

$$\begin{aligned} \text{Cost (NVP)} &= O\left(n^2 e \log n\right) \\ \text{and Cost (VPS)} &= O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right). \end{aligned} \quad (8)$$

Substituting in (3) we get

$$T_w(n) = 2T_w\left(\frac{n}{2}\right) + O\left(n^2 e \log n\right) + O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right) \quad (9)$$

Note that $O((1/\epsilon^2)\log(1/\delta))$ depends on the size of the sketch data structure allocated on the controller probing module. Since it does not depend on the network size n, it can be replaced by a constant C in (9) to get

$$T_w(n) = 2T_w\left(\frac{n}{2}\right) + O\left(n^2 e \log n\right) + C \quad (10)$$

Solving the recursive equation in (5), we get a closed form worst case runtime complexity of $O(n^2 e(\log n)^2)$.

Table 3: VISKA worst case complexity analysis summary.

Term	Meaning	Value
$T_w(n)$	The worst case runtime complexity of the VISKA malicious switch detection algorithms. n designates the network size.	$2T_w(n/2)$ + Cost (NVP) + Cost(VPS)
Cost (NVP)	The worst case runtime complexity of the Network Views Partitioning function. This is typically the cost of the Karger algorithm.	$O(n^2 e \log n)$
Cost (VPS)	The worst case runtime complexity of the View Probing and Sketching function. This cost depends on the size of the sketch data structure and not on the network size. Accordingly it can be replaced by an constant C.	$O((1/\epsilon^2)\log(1/\delta))$
$T_w(n)$	After solving the recurrence relation in Equation (5).	$O(n^2 e(\log n)^2)$

Table 4: VISKA average case complexity analysis summary.

Term	Meaning	Value
$T_a(n)$	The average case runtime complexity of the VISKA malicious switch detection algorithms. n designates the network size. Note that $T_a(n/2)$ is not multiplied by a factor of 2 since the recursion is only applied on one network partition.	$T_w(n/2)$ + Cost (NVP) + Cost(VPS)
Cost (NVP)	The worst case runtime complexity of the Network Views Partitioning function.	$O(n^2 e \log n)$
Cost (VPS)	The worst case runtime complexity of the View Probing and Sketching function. This cost depends on the size of the sketch data structure and not on the network size. Accordingly it can be replaced by an constant C.	$O((1/\epsilon^2)\log(1/\delta))$
$T_w(n)$	After solving the recurrence relation in Equation (5).	$O(n^2 e(\log n))$

A summary of the worst case complexity equations and their meaning is presented in Table 3.

Average Case Analysis. The average case scenario arises when a malicious behavior is detected by the VPS function in one of the two recursive network views provided by the NVP partitioning function. The average case runtime complexity $T_a(n)$ is given by

$$T_a(n) = T_a\left(\frac{n}{2}\right) + \text{Cost (NVP)} + \text{Cost (VPS)} \quad (11)$$

Note that $T_a(n/2)$ is not multiplied by a factor of 2 since the recursion is only applied on one network partition and not on both partitions as is the case in the worst case scenario.

Replacing the values of Cost(NVP) and Cost(VPS) in (11), we get

$$T_a(n) = T_a\left(\frac{n}{2}\right) + O\left(n^2 e \log n\right) + C \quad (12)$$

Solving the recursive equation in (12) we get a closed form average case runtime complexity of $O(n^2 e(\log n))$.

A summary of the average case complexity equations and their meaning is presented in Table 4.

Best Case Analysis. The best case analysis scenario obviously occurs when the whole network does not contain any malicious activity. In this case, no recursive partitioning is going to be carried out on the network view and thus, the base case will be reached by first applying the VPS function on the network view. As such, the best case runtime is simply a constant C.

The convergence of the VISKA malicious switch detection algorithm is evidently guaranteed by having a deterministic base case step in Algorithm 1 that ends the recursion on a particular network view when either (1) a set of source malicious switches of size m is isolated by the VPS function, or (2) a network view is found to be nonmalicious by the VPS function.

4.4.2. Analysis of the Attack Categorization Module. The MACM module is executed once the malicious switches are detected and identified. The main complexity in this module is related to the transmission of the ingress/egress traffic to the SDN controllers by the *mc* switches and the analysis of such traffic by the controller for the purpose of attack categorization. The complexity mainly depends on the number of *mc* switches and the flow size crossing them per unit time. We faithfully believe that an SDN controller can tolerate such traffic load and categorization processing due to the following reasons:

(1) The MACM module analyses the ingress/egress traffic flowing solely into/from the maliciously detected switch(es) rather than inspecting the whole network

traffic as is the case in traditional intrusion detection systems in the literature.

(2) The number of mc switches in modern data center topologies is minimal compared to the total number of network switches. For instance, in the k-ary FatTree topology, which is widely deployed in today's data centers, the total number of switch nodes is $O(k^2 + k)$. Each edge switch is connected to $O(k/2)$ neighbours and each aggregation or core switch to $O(k)$ neighbours.

(3) The MACM algorithm operates merely on the packet headers, thus eliminating the need to transmit the entire flow of packets to the controller. This drastically reduces the size of the ingress/egress flows transmitted to the SDN controller and the resources needed to process them.

(4) Most modern SDN architectures are relying on replicated controller instances, which aid in balancing and distributing the processing load of the collected network data for analysis and mining.

(5) A feasible approach that can be easily employed in the VISKA implementation to reduce the load on the SDN controller is to employ a central dedicated host for traffic collection and analysis. In this sense the MACM attack categorization module can be efficiently deployed on an external host connected to the controller to further enhance the VISKA efficiency and performance.

MACM Malfunction and Attack Categorization Cost. This module is invoked when the malicious switches, with the predetermined granularity size m, are detected. As demonstrated in Section 4.3 (MACM phases 1 and 2), the algorithm captures packets at the malicious switch(es) ingress and egress ports and stores necessary header information of the real network traffic captured packets (this collection process of phase 1 has a constant runtime complexity C). The collected information tables from the egress and ingress ports of the malicious data element(s) are analyzed, and certain features are extracted for the attack categorization. The basic operation in the MACM phase 2 complexity is the hashing operation (refer to Figure 5) which is executed on the total number of packets in the ingress and egress traffic flowing into the m maliciously detected switch(es). Let f_{em} and f_{im}, respectively, represent the number of egress and ingress packets in the collected traffic. This renders the complexity of phase 1 as follows:

$$\text{Cost of MACM}_{\text{phase 1,2}} = \theta\left(\text{C} + 2\left(f_{\text{em}} + f_{\text{im}}\right)\right) \tag{13}$$

The factor 2 in (13) is the result of applying the hashing operations on the source as well as on the destination IP packet addresses in the collected flows as demonstrated in Figure 5.

The MACM phase 3 (Malfunction and Attack Categorization Module), presented in Algorithm 1, analyzes the traffic information tables collected and stored in phases 1 and 2. The complexity of this phase depends on the specific attack type which is summarized in (14) below:

$$\begin{aligned}
&\text{cost of MACM}_{\text{phase 3}} \\
&\quad = \text{cost (DoS detection)} \\
&\qquad + \text{cost (Interruption detection)} \\
&\qquad + \text{cost (blocking on dest IP)} \\
&\qquad + \text{cost (blocking on srcIP)} \\
&\qquad + \text{cost (MITM attack at destIP)}
\end{aligned} \tag{14}$$

Analyzing (14) based on the MACM phase 2 code in Algorithm 1, we get (15) which depends on the number of egress and ingress packets in the collected traffic, f_{em} and f_{im}, respectively. This is formulated as follows:

$$\begin{aligned}
\text{Cost_MACM}_{\text{phase 3}} &= O\left(f_{\text{em}}\right) + O\left(f_{\text{em}}\right) + O\left(f_{\text{im}}\right) \\
&\quad + O\left(f_{\text{im}}\right) + O\left(f_{\text{em}}\right) \\
&= O\left(f_{\text{em}} + f_{\text{im}}\right)
\end{aligned} \tag{15}$$

Based on (9) and (11), the complexity cost of the MACM module is designated as follows:

$$\begin{aligned}
T_{\text{MACM}}\left(f_{\text{em}}, f_{\text{im}}\right) &= \text{Cost of MACM}_{\text{phase 1,2}} \\
&\quad + \text{Cost of MACM}_{\text{phase 3}} \\
T_{\text{MACM}}\left(f_{\text{em}}, f_{\text{im}}\right) &= \theta\left(\text{C} + 2\left(f_{\text{em}} + f_{\text{im}}\right)\right) \\
&\quad + O\left(f_{\text{em}} + f_{\text{im}}\right)
\end{aligned} \tag{16}$$

The MACM module is therefore effective order of $O(f_{\text{em}} + f_{\text{im}})$.

A summary of the complexity analysis of the MACM module is presented in Table 5.

4.5. Sketch Size Analysis. The sketch data structures are created on the controller probing module and are incrementally updated based on the probing data packets. The size of the sketch allocated counters has a great influence on the error ϵ and the confidence level δ of the sketch computations.

Sketch Number of Counters W. Based on the analysis in [32, 33], using two four-wise independent hashing functions for the index $\{0, ..W-1\}$ and the value $\{-1, +1\}$ computations, the second normal difference $|s(D_1) - s(D_2)|^2 = \Delta$ will correctly estimate $|D_1 - D_2|^2$ with error ϵ and confidence level $(1 - \delta)$ given that the depth of the sketch W (number of counters) is directly proportional to the number of computations required by the four-wise independent hashing function in the worst case complexity analysis: $W \in O((1/\epsilon^2)\log(1/\delta))$. This is described in Section 4.1. Therefore, W for each sketch is dependent on the error ϵ and the confidence level δ and not on the number of the probing data packets k. Having W independent of the size of the input data is of

Table 5: VISKA complexity analysis summary of the MACM attack categorization module.

Term	Meaning	Value
$T_{\text{MACM}}(f_{\text{em}}, f_{\text{im}})$	The cost of the MACM attack categorization module as a function of the number of egress packets f_{em} and the number of ingress packets f_{im}.	Cost of $\text{MACM}_{\text{phase 1,2}}$ + Cost of $\text{MACM}_{\text{phase 3}}$
Cost of $\text{MACM}_{\text{phase 1,2}}$	Represents the cost of data collection and applying the hash operation on the destination IP and source IP addresses of each egress and ingress packet.	$\theta(C + 2(f_{\text{em}} + f_{\text{im}}))$
$\text{Cost\_MACM}_{\text{phase 3}}$	The worst case runtime cost of detecting the different types of DoS, Interruption/Blocking, and MITM attacks.	$O(f_{\text{em}} + f_{\text{im}})$
$T_{\text{MACM}}(f_{\text{em}}, f_{\text{im}})$	Adding the costs of phases 1, 2, and 3 and simplifying the complexity terms, we get the worst case runtime cost of the MACM module to be in the order of $f_{\text{em}} + f_{\text{im}}$.	$\theta(C + 2(f_{\text{em}} + f_{\text{im}})) + O(f_{\text{em}} + f_{\text{im}}) = O(f_{\text{em}} + f_{\text{im}})$

Table 6: Simulation environment technical details.

SDN controller	FloodLight 0.91 [9]
Network topology	Linear and FatTree
Network size	For each topology, we implemented five network sizes comprising 10, 50, 100, 150, 200, and 250 SDN switching elements. These parameters are chosen to cover a wide range of data center sizes for viably testing the scalability of the algorithms on real SDN networks.
Remote cloud service	The cloud algorithms are developed using the Java platform and deployed on an Amazon EC2 [10] VM.
Cloud VM instance	the t2.micro with 1 vCPU and 1 GB RAM
Physical machine running Mininet	MacBook Pro Mid 2015 laptop running OSX10.12 and supported with 2.5 GHz Intel Core i7 and 16 GB of DDR3 RAM

great significance in the VISKA algorithm implying that the probing size can be dynamically increased with limited space requirements on the controller.

Sketch Counter Size z. Each counter in the sketch data structure can hold values between $[-b, +b]$ as a result of the increments/decrements of the hash function on the probing data packets. Range b depends on the size of memory allocated for each counter. Evidently, b is dependent on the size of the probing input k, the depth of the sketch, and the computations' confidence level. Based on the following equation, b is $O(\log k)$:

$$z = \log(2b) = 1 + \frac{1}{2}\log\left(4\frac{k}{W}\ln\left(\frac{200\,W}{\delta}\right)\right) \quad (17)$$

Equation (13) is based on the following analysis and computations [32]. Based on the union bound [2], no counter overflows in the counter array with a maximum probability of $(1 - \delta/100)$ since the confidence level is $(1 - \delta)$ for the correct functioning of the sketch with error ϵ. This suffices that a counter would overflow with a maximum probability of $((1/W)(\delta/100))$. Consider variable X_i to represent the sketch resulting values in $\{-1, +1\}$ to be stored in the sketch counters at index i. X_i will be equal to 1 with probability $W/2$ and -1 with probability $W/2$ and 0 otherwise. Variable $X = \sum_{i=1}^{k} X_i$ is the result count in each bin after applying all the input probing packets. Applying the Chernoff bound [34] for the size of each counter $|X|$ to exceed the allocated size b, we get the following:

$$P(|X| \geq b) \leq 2e^{-b^2/2k\text{var}[X_i]} < \frac{\delta}{100W} \quad (18)$$

Knowing that $\text{var}[X_i] = 1/W$ and solving (18) result in (17).

5. System Implementation

The VISKA system design is implemented on top of the Mininet network emulator. We created two main testbed network topologies in Mininet represented in the theoretical linear topology and the popular data center FatTree [35] network topology. This choice is adopted to test the VISKA algorithm behavior on diverse network topologies in order to empirically analyze the performance efficiency of the algorithms on linearly and hierarchically connected network switch infrastructures. The technical specification of the simulation environment is described in Table 6.

A probing module is developed on the FloodLight SDN controller for the purpose of (1) exchanging a set of k probing packets, (2) calculating the corresponding sketches and timestamp accumulators, and (3) sending their sketch and timestamp data structure results to the VISKA cloud service.

The VISKA cloud service calculates the sketch's second norm difference estimation as well as the valid timestamp differences of the probing packets and recursively executes

Table 7: Attack types supported by the MACM module.

Attack Type	Description
DoS attack on host with IP address 10.0.0.4	To simulate this attack, we crafted a static flow action rule that commands the malicious switch SW6 to forward all traffic received on any of its ingress ports to the IP address 10.0.0.4. The following is an example of a StaticEntryPusher command that simulates this attack: $curl - X\ POST - d\ '\{"switch" :"00:00:00:00:00:00:00:06", "name": "DOS\_on\_10.0.0.4", "priority": "32768", "in\_port": "any", "active": "true", "eth\_type": "0x0800", "actions": "set\_field = eth\_dst \rightarrow a6:b8:34:23:8e:42, set\_field = ipv4\_dst \rightarrow 10.0.0.4, output = all"\}'\ http://localhost:8080/wm/staticentrypusher/json$
DoS attack on controller	Analogous to the DoS attack on a specific IP, we command SW6 to send all traffic received on any of its ingress ports to the controller.
Interruption of traffic to host with IP address 10.0.0.4	To simulate this attack scenario, we craft a static flow action rule that commands the switch SW6 to drop a portion of the packets destined to host 10.0.0.4. For instance, switch SW6 only drops the packets with destination IP 10.0.0.4 and received on ingress port 1. Packets received on ingress ports 2 and 3 and destined to 10.0.0.4 are forwarded normally.
Blocking traffic destined to host: IP address 10.0.0.4	To simulate this attack, we crafted a rule similar to the one above but which commands the malicious switch SW6 to drop all packets destined to host 10.0.0.4. That is SW6 drops packets destined to 10.0.0.4 received on any of its ingress ports. This is achieved using the StaticEntryPusher commands.
Blocking traffic from host with IP address 10.0.0.3	In this attack scenario the malicious switch SW6 is commanded to drop all packets with source IP 10.0.0.3 received on any of its ingress ports.
Man-in-the-Middle attack via host 10.0.0.2	In this attack scenario, the malicious switch SW6 modifies the destination IP and Ethernet addresses to those of the host 10.0.0.2.

the NVP graph partitioning algorithm based on the detection of malicious operation in the tested partitions.

To simulate the Category II attack (refer to Section 4.3), we used the Mininet delay property on all the links connecting the malicious switch SW6 to the neighbouring switches SW3, SW4, SW7, and SW10 (refer to Figure 4). The implementation value set for the delay property to initiate a delay attack is 40 ms.

To simulate the various Category I attacks presented in Section 4.3, we used the *Floodlight* REST APIs. Table 7 describes the list of attacks supported.

We simulated the attacks described in Table 7 using the Floodlight REST APIs. This resulted in generating a dataset for testing the performance (accuracy) of the attack categorization algorithms in the testbed implementation as well as in further real-world deployments on target SDN networks. The generated dataset is comprised of TCPDump raw network packets collected in the SDN network topology presented in Figure 4 and generated using the IPerf tool [36], web browsing, email messaging, and video streaming over a time period of 4 hours. The dataset consists of 130547 packets with a total size of 978 MBs simulating 30 DoS attacks, 30 interruption/Blocking attacks, and 30 MITM attacks. The main purpose of the generated data set is to tune the attack detection thresholds employed in the MACM algorithm to optimal values based on the ROC curves described later in this section.

The attacks introduced in Table 7 change in the probing streams that resulted in a second norm difference in the calculated sketches $\Delta \geq \tau$ and in a timestamp difference $\partial > \Gamma_d$ in the delay attack simulation. Subsequently, the attacks were successfully detected to the granularity of m switches, which was set to as low as one switch in the experiments.

Table 8: Summary of terms used in the simulation.

Network Size n	10, 50, 100, 150, 200, 250
$\tau\ (\geq \epsilon)$	0.1
$1 - \delta$	99%
Δ	≥ 0.1
k	10^8 packets
z	18 bits
$W > ((1/\epsilon^2)\log(1/\delta))$, size_of (TSA), size_of (Val)	700 counters
Maximum malicious partition size m	3
Percent malicious switches Ms (resulting in $\Delta \geq \tau$)	5%, 10%, and 15% of network size

A highly significant parameter that is considered in the experiments is the percent of malicious switches introduced (Ms). Therefore, we tested a malicious switch number consisting of 5%, 10%, and 15% of the network size for the two topologies. The parameters used in the experiments are summarized in Table 8. It is worth mentioning here that we employed a sketch data structure of 1575 bytes ($W = 700$ and $z = 18$ bits), which is typically the size of a single IP packet, to summarize a network traffic flow consisting of 10^8 probing packets.

In this work we present the analysis of the DoS, interruption/Blocking, and MITM attacks for each of the previously mentioned topologies and sizes. In the analysis of the VISKA attack detection and localization part, for each tested configuration, we calculate the average number of recursive steps and the total average convergence time needed by the VISKA VPS and NVP algorithms to localize all malicious nodes in the

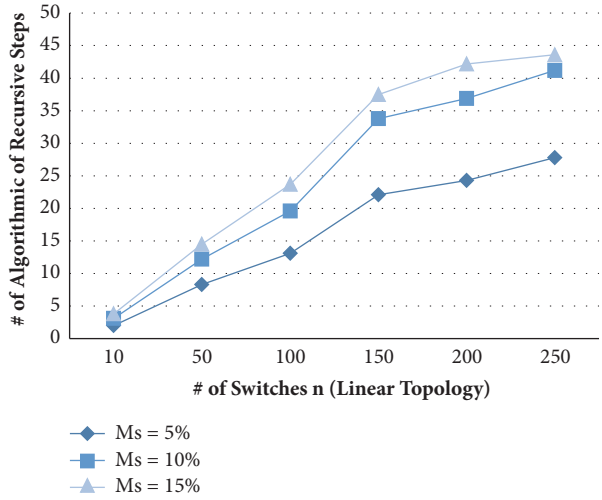

Figure 7: Average number of recursive steps required to detect the malicious switches in the linear network topology.

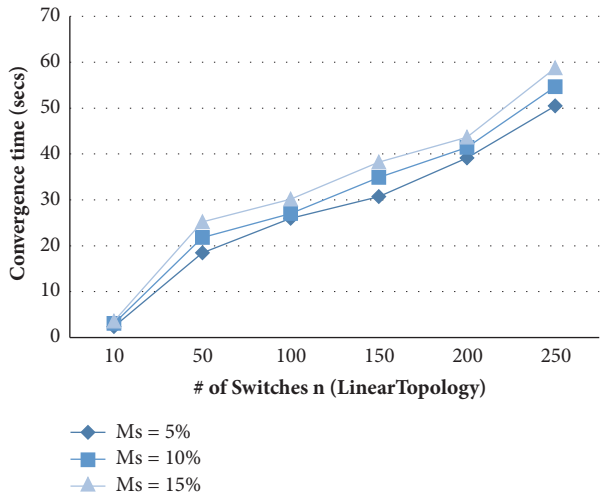

Figure 8: Convergence time required by VISKA to detect the malicious switches in the linear network topology.

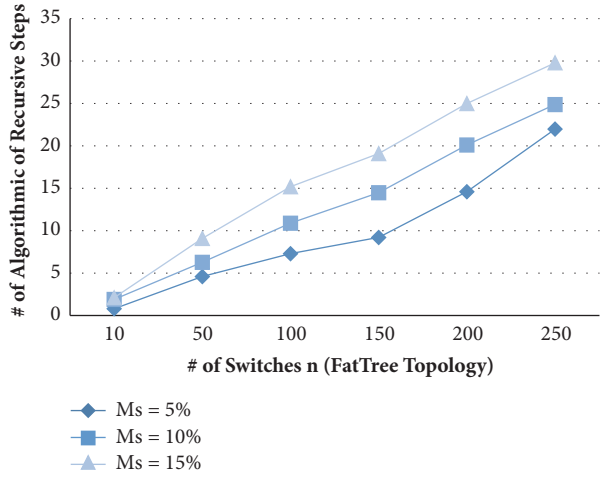

Figure 9: Average number of recursive steps required to detect the malicious switches in the FatTree network topology.

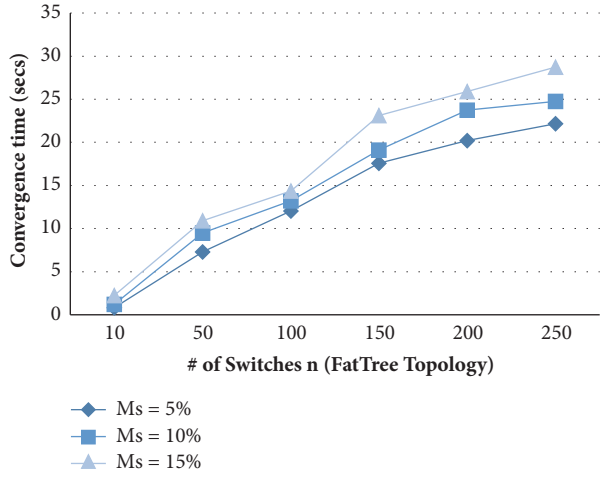

Figure 10: Convergence time required by VISKA to detect the malicious switches in the FatTree network topology.

different SDN network topologies. The experiments on each configuration are replicated 5 times. The average number of recursive steps and the convergence time results are plotted in Figures 7 and 8, respectively, for the linear topology, and Figures 9 and 10 for the FatTree topology.

The VISKA algorithms successfully converged to detecting the sources of malicious forwarding in the SDN data plane in all the executed experiments. The presented results demonstrate relatively better performance of VISKA on the FatTree topology compared to the linear one. The improvement reached an average of 42% in the number of recursive steps and 49.6% in the convergence time over the different network sizes and degrees of maliciousness tested. This renders the VISKA algorithms better suited for operation on a real hierarchical data center topology represented by the FatTree topology. Evidently, the convergence time of VISKA algorithms is proportional to the SDN network size and the degree of switch maliciousness. The proposed VISKA service is thus demonstrated to provide the network with a scalable network security solution with the flexibility and dynamism of software. The convergence time results support the scalability of the algorithms on linear and hierarchical data center topologies.

In the linear topology, the increase in network size from 10 to 50 resulted in a convergence time increase of 16 sec (Ms=5%), 18.75 sec (Ms=10%), and 21.66 sec (Ms=15%), while the increase from 50 to 250 resulted in an increase of 32 sec (Ms=5%), 32.84 sec (Ms=10%), and 33.5 sec (Ms=15%).

Similar to the *FatTree* topology, the increase in netwok size from 10 to 50 resulted in a convergence time increase of 6.5 sec (Ms=5%), 8.3 sec (Ms=10%), and 8.7 sec (Ms=15%), while the increase from 50 to 250 resulted an increase of 14.85 sec (Ms=5%), 15.3 sec (Ms=10%), and 17.8 sec (Ms=15%).

The uniform variations in the convergence time as the network size and degree of maliciousness increase show that the algorithms scale well as the network size and

degree of maliciousness increase because of the recursive polylogarithmic nature of the VISKA partitioning algorithm, which focuses the security probing, solely, on isolated parts of the network.

The analysis of the MACM attack categorization algorithm is realized on the topology presented in Figure 4, for simplifying the analysis. This starts by implementing a mechanism to collect the ingress and egress traffic flowing into/from the malicious switches, respectively. To achieve this, we use the *Floodlight* REST APIs to push static flow action rules into the *mc* switches that are sending and receiving traffic from/to the malicious switch(es). The static flows pushed by the controller will transparently result in sending all traffic destined to and received from a maliciously identified switch into the controller. Crafting such flows in *OpenFlow* is pretty simple: first, we create a JSON message indicating (1) the name of the flow, (2) the DatapathID (DPID) of the switch we want to insert this flow on, (3) the set of criteria matching the traffic to be transparently redirected, and (4) the actions to be executed by the switch on the traffic matching the flow criteria. For instance, transparently sending a copy of all traffic flowing from switch SW6 to switch SW10 in the topology demonstrated in Figure 4, we leverage the curl tool [37] to push the corresponding action rule as follows:

$$curl - X\ POST - d\ '\{"switch": "00:00:00:00:00:00:00:0a", "name": "flow - SW6 - SW10", "priority": "32768", "in\_port": "4", "active": "true", "eth\_type": "0x0800", "actions": "set\_field = output = normal, controller"\}'\ http://localhost:8080/wm/staticentrypusher/json$$

where

(i) "*switch*": "*00:00:00:00:00:00:00:0a*" is the DPID of switch SW10,

(ii) "*in_port*": "*4*" designates the SW10 interface connected SW10 to SW6,

(iii) "*actions*": "*set_field = output = normal, controller*" commands SW10 to send all traffic matching the specified criteria to the controller and to the normal interface specified by the switch's L2 pipeline.

Analogous static flow rules are injected into switches SW3, SW4, and SW7. Moreover, similar JSON messages are crafted to configure the *Floodlight* controller to send action rules to switches SW3, SW4, SW7, and SW10 enforcing the transfer of a copy of the ingress traffic destined to SW6 into the controller.

After collecting the ingress/egress traffic, we applied the *hashs* and *hashd* aggregation functions to categorize the packets based on the source and destination IPs, respectively. The list of MACM phase 2 features specified in Section 4.3 are extracted from the aggregated data and analyzed for the purpose of attack categorization. The hashing and analysis procedures are implemented in Python in the Floodlight controller's address space.

The most significant step in the accurate detection of the various attack types in the MACM attack categorization module is the tuning of the threshold parameters to achieve an optimal true positives/false positives attack detection rate. The MACM attack categorization module relies on

(1) the ΓEd, Γdos, and Γp threshold parameters for detection DoS attacks,

(2) the Γbh threshold parameter for detecting Interruption/Blocking attacks,

(3) the Γmitm threshold parameter for detecting MITM attacks.

In the testbed MACM implementation, we utilized 21 threshold values for each attack type in the ranges specified in Table 2. The following threshold range parameters are used: $X_{\max} = 1.72$ Mpps, $P_{avg} = 576$ bytes, $t_i = 60$ secs, $\alpha_l = 10\%$, $\alpha_h = 90\%$, $\beta_l = 5\%$, $\beta_h = 70\%$, $\gamma_l = 5\%$, and $\gamma_h = 50\%$. The 21 threshold values are presented in Table 9 and are generated in increments of 5% between the low threshold range and the high threshold range. The attack scenarios included in the dataset described earlier in this section are adopted on each of the 21 threshold values presented in Table 9 in order to empirically find the optimum value for the DoS thresholds ΓEd, Γdos, Γp, the interruption/Blocking threshold Γbh, and the MITM threshold Γmitm. Based on the true positives and false positives rates of the attack detection system observed, we generate the ROC curve for each attack type (refer to Figures 11, 12, and 13). The resulting ROC curves for each attack type shows that at higher values of thresholds, the system does not detect the attack. As we gradually increment the threshold values, an optimum point is reached representing the best true positives/false positives detection rate. This is represented in point P12 in the DoS ROC (refer to Figure 11) where ΓEd = 1.98, Γdos = 463.6×10^3, and Γp = 804.6 at which the system resulted in almost 90% true positives rate and around 8% false positives rate. After this point, for lower threshold values, the attack is detected; however the false positives rate increases rapidly. The optimum threshold value for the interruption/Blocking attack detection is located at point P12 (refer to Figure 12) with Γbh = 319.4 resulting in a 93% true positives rate and an 8.6% false positives rate. The optimum Γmitm (213.75) for the MITM attack detection is located at point P13 (refer to Figure 13) with a true positives rate of 92% and a false positives rate of 8.4%.

6. Conclusion

The paper presented VISKA, a novel approach in localizing malicious nodes in the SDN data plane and categorizing any present attacks by utilizing network programming and probabilistic sketching. The VISKA security algorithms are designed to run in real time with minimal convergence time for isolating malicious forwarding elements in the data plane. This is the main contribution of the work where malicious switch detection is achieved by an efficient logarithmic divide-and-conquer approach that divides the network view in half in each recursive iteration. The network programming functions in SDN allow the system to autonomously isolate

Table 9: Threshold values used in the MACM implementation.

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ΓEd	3.3	3.19	3.08	2.97	2.86	2.75	2.64	2.53	2.42	2.31	2.2
Γdos	1069977	1019450.3	968923.6	918396.9	867870.2	817343.5	766816.8	716290.1	665763.4	615236.7	564710
Γp	1857	1769.3	1681.6	1593.9	1506.2	1418.5	1330.8	1243.1	1155.4	1067.7	980
Γbh	722	688.45	654.9	621.35	587.8	554.25	520.7	487.15	453.6	420.05	386.5
Γmitm	516	492.75	469.5	446.25	423	399.75	376.5	353.25	330	306.75	283.5
	P11	**P12**	**P13**	**P14**	**P15**	**P16**	**P17**	**P18**	**P19**	**P20**	
ΓEd	2.09	1.98	1.87	1.76	1.65	1.54	1.43	1.32	1.21	1.1	
Γdos	514183.3	463656.6	413129.9	362603.2	312076.5	261549.8	211023.1	160496.4	109969.7	59443	
ΓP	892.3	804.6	716.9	629.2	541.5	453.8	366.1	278.4	190.7	103	
Γbh	352.95	319.4	285.85	252.3	218.75	185.2	151.65	118.1	84.55	51	
Γmitm	260.25	237	213.75	190.5	167.25	144	120.75	97.5	74.25	51	

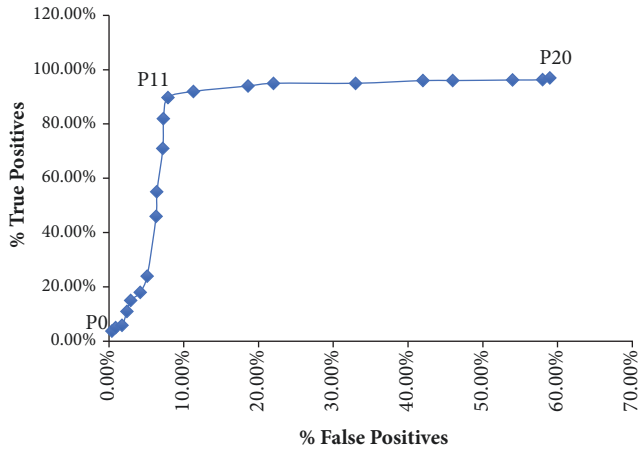

Figure 11: ROC of the true positives versus the false positives rates for the different DoS attack threshold parameter triplets ΓEd, Γdos, and Γp.

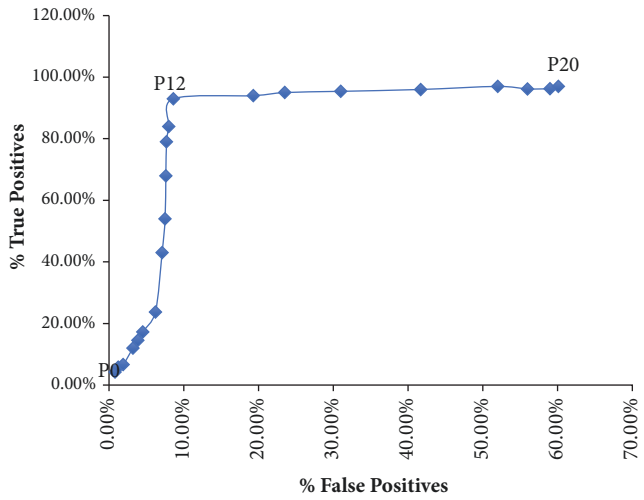

Figure 12: ROC of the true positives versus the false positives rates for the different values for the interruption/Blocking attack threshold parameter Γb*h*.

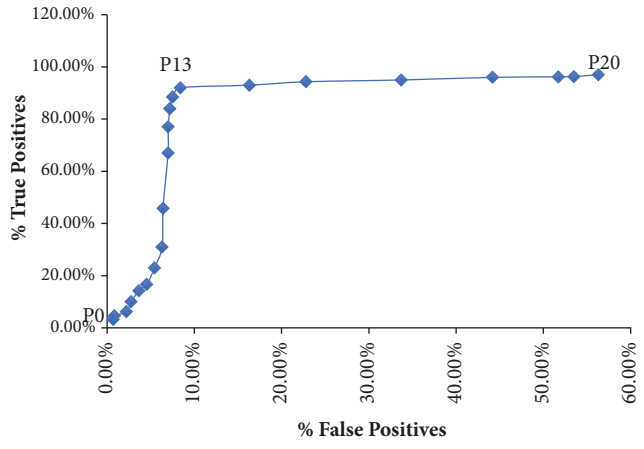

Figure 13: ROC of the true positives versus the false positives rates for the different values for the interruption/Blocking attack threshold parameter Γmitm.

network partitions that may be experiencing malicious activity. This is done flexibly with pure software operations. The attacks detected include (1) network time delay insertion, (2) MITM, (3) DoS on a certain server, (4) block on a certain source IP, (5) block on a certain destination, (6) miscellaneous blocks to induce network malfunction, and (7) DoS on the controller. The algorithms were tested for convergence using a variety of SDN network sizes and number of malicious switching elements. The various attacks were experimented and the detection thresholds were identified. The system was capable of achieving over 90% detection accuracy. It is worth mentioning here that a very appealing application to the VISKA model is in supporting net neutrality in modern SDN-based NaaS provider networks. VISKA attack categorization mechanisms can provide a valuable feedback on probable breaches that violate net neutrality exertion in an SDN-based network. This is demonstrated by the following points:

(1) VISKA detects malicious traffic shaping violations that induce delay attacks on network packets by leveraging the timestamp accumulator data structure presented in Section 4.1.

(2) VISKA detects DoS attacks that interfere with the "freedom of speech" approach pushed by the Open Internet [38] standards. The Open Internet approach indicates that the full network resources should be accessible by clients transparently and easily.

(3) VISKA prevents any discrimination by IP address by detecting blocking attacks on a certain destination or source network address.

(4) VISKA aids in preventing malicious over provisioning of network bandwidth by detecting delay attacks resulting from unfair bandwidth distribution.

Data Availability

All the data necessary to execute the testbed experiments are available to interested readers upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," *Journal of Network and Computer Applications*, vol. 67, pp. 1–25, 2016.

[2] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: a survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[3] Mininet homepage, http://www.mininet.org.

[4] H. K. Cheng, S. Bandyopadhyay, and H. Guo, "The debate on net neutrality: A policy perspective," *Information Systems Research*, vol. 22, no. 1, pp. 60–82, 2011.

[5] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[6] W. Li, W. Meng, and L. F. Kwok, "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures," *Journal of Network and Computer Applications*, vol. 68, pp. 126–139, 2016.

[7] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," Tech. Rep. RFC 4346, 2006.

[8] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A Survey on the Security of Stateful SDN Data Planes," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.

[9] FloodLight Controller, http://www.projectfloodlight.org/floodlight/.

[10] Amazon EC2 home page, https://aws.amazon.com/ec2.

[11] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 55–60, ACM, Hong Kong, August 2013.

[12] S. Betge-Brezetz, G.-B. Kamga, and M. Tazi, "Trust support for SDN controllers and virtualized network applications," in *Proceedings of the 1st IEEE Conference on Network Softwarization, NETSOFT 2015*, gbr, April 2015.

[13] R. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, "A verification platform for SDN-enabled applications," in *Proceedings of the 2nd IEEE International Conference on Cloud Engineering, IC2E 2014*, pp. 337–342, usa, March 2014.

[14] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in *Proceedings of NDSS*, 2013.

[15] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a software-defined network via distributed controllers," https://arxiv.org/abs/1401.7651.

[16] JGroups project homepage, http://www.jgroups.org/.

[17] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: detecting security attacks in software-defined networks," in *Proceedings of the Network and Distributed System Security Symposium*, 2015.

[18] Y. Minlan, J. Lavanya, and M. Rui, "Software Defined Traffic Measurement with OpenSketch," in *Proceedings of the NSDI 13*, 2013.

[19] G. Gheorghe, T. Avanesov, M.-R. Palattella, T. Engel, and C. Popoviciu, "SDN-RADAR: Network troubleshooting combining user experience and SDN capabilities," in *Proceedings of the 1st IEEE Conference on Network Softwarization, NETSOFT 2015*, gbr, April 2015.

[20] Q. Niyaz, W. Sun, and A. Y. Javaid, "A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)," *ICST Transactions on Security and Safety*, vol. 4, no. 12, Article ID 153515, 2017.

[21] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," in *Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications, WINCOM 2016*, pp. 258–263, mar, October 2016.

[22] A. Santos Da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," in *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016*, pp. 27–35, tur, April 2016.

[23] P. Phaal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," Tech. Rep. RFC3176, 2001.

[24] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A DDoS Attack Detection Method Based on SVM in Software Defined Network," *Security and Communication Networks*, vol. 2018, Article ID 9804061, 8 pages, 2018.

[25] D. Jankowski and M. Amanowicz, "On Efficiency of Selected Machine Learning Algorithms for Intrusion Detection in Software Defined Networks," *International Journal of Electronics and Telecommunications*, vol. 62, no. 3, pp. 247–252, 2016.

[26] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN '10)*, pp. 408–415, Denver, Colo, USA, October 2010.

[27] M. Shamseddine, W. Itani, A. Kayssi, and A. Chehab, "Virtualized network views for localizing misbehaving sources in SDN data planes," in *Proceedings of the 2017 IEEE International Conference on Communications, ICC 2017*, fra, May 2017.

[28] G. Cormode and M. Garofalakis, "Sketching streams through the net: Distributed approximate query tracking," in *Proceedings of the VLDB 2005 - 31st International Conference on Very Large Data Bases*, pp. 13–24, nor, September 2005.

[29] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," Tech. Rep. RFC6298, 2011.

[30] D. L. Mills, "Network Time Protocol (NTP)," Tech. Rep. RFC0958, 1985.

[31] D. R. Karger, "Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm," in *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 21–30, ACM, 1993.

[32] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford, "Path-quality monitoring in the presence of adversaries," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, p. 193, 2008.

[33] M. Thorup and Y. Zhang, "Tabulation based 4-universal hashing with applications to second moment estimation," pp. 615–624, ACM, New York.

[34] K. Levchenko, "Chernoff bound," 2008, http://cseweb.ucsd.edu/~klevchen/techniques/chernoff.pdf.

[35] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM '08)*, pp. 63–74, Seattle, Wash, USA, August 2008.

[36] Iperf tool, https://iperf.fr/en/.

[37] *Conquering The Command Line Unix And Linux Commands For Developers*, chapter 3, 2014, CURL.

[38] R. S. R. Ku, "Open Internet Access and Freedom of Speech: A First Amendment Catch-22," *Tulane Law Review*, vol. 75, p. 87, 2000.