Expert Systems With Applications 42 (2015) 9620–9633

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

STONE: A streaming DDoS defense framework☆

Vincenzo Gulisano[a,b,*], Mar Callau-Zori[a,1], Zhang Fu[a,2], Ricardo Jiménez-Peris[b], Marina Papatriantafilou[a], Marta Patiño-Martínez[b]

[a] Chalmers University of Technology, Gothenburg, Sweden
[b] Laboratorio de Sistemas Distribuidos, Universidad Politécnica de Madrid, Spain

ARTICLE INFO

Keywords:
DDoS detection
DDoS mitigation
Data streaming

ABSTRACT

Distributed Denial-of-Service (DDoS) attacks aim at rapidly exhausting the communication and computational power of a network target by flooding it with large volumes of malicious traffic. In order to be effective, a DDoS defense mechanism should detect and mitigate threats quickly, while allowing legitimate users access to the attack's target. Nevertheless, defense mechanisms proposed in the literature tend not to address detection and mitigation challenges jointly, but rather focus solely on the detection or the mitigation facet. At the same time, they usually overlook the limitations of centralized defense frameworks that, when deployed physically close to a possible target, become ineffective if DDoS attacks are able to saturate the target's incoming links.

This paper presents *STONE*, a framework with expert system functionality that provides effective and joint DDoS detection and mitigation. *STONE* characterizes regular network traffic of a service by aggregating it into common prefixes of IP addresses, and detecting attacks when the aggregated traffic deviates from the regular one. Upon detection of an attack, *STONE* allows traffic from known sources to access the service while discarding suspicious one. *STONE* relies on the data streaming processing paradigm in order to characterize and detect anomalies in real time. We implemented *STONE* on top of StreamCloud, an elastic and parallel-distributed stream processing engine. The evaluation, conducted on real network traces, shows that *STONE* detects DDoS attacks rapidly, provides minimal degradation of legitimate traffic while mitigating a threat, and also exhibits a processing throughput that scales linearly with the number of nodes used to deploy and run it.

1. Introduction

Internet security constitutes one of the most important challenges for a society where the demand of IT services increases every day. Among the plethora of possible existing threats, Distributed Denial-Of-Service (DDoS) attacks aim at rapidly exhausting the communication and computational power of a network target by flooding it with large volumes of malicious traffic. Despite the seek of comprehensive solutions against DDoS attacks pursued by researchers during decades, DDoS attacks are still a main threat since both their scale and typologies keep evolving and growing (Wang, Mohaisen, Chang, & Chen, 2015a).

DDoS defense frameworks are usually designed to react to an attack (e.g., by inspecting each packet and deciding whether to forward it or not) only after the latter has been detected (Dean, Franklin, & Stubblefield, 2002; Savage, Wetherall, Karlin, & Anderson, 2000; Song & Perrig, 2001). Reactive frameworks have two main benefits. On one hand, they do not degrade the user experience (e.g., in terms of processing latency) if no attack is in progress. On the other hand, by not distinguishing legitimate from illegitimate traffic by means of predefined, a priori rules, they can also prevent threats that, as for flash crowds, only involve legitimate traffic. A crucial aspect for an effective defense framework is to detect attacks as soon as possible and mitigate them before they successfully manage to disrupt the target's services. Anomaly-based detection methods have the advantage of being able to discover previously unseen attacks by means of discrepancies between observed and expected traffic behavior. Proposed methods are based on techniques such as principal component

☆ Some preliminar results have been published at the ACM Symposium on Applied Computing (SAC) in 2013 by Callau-Zori et al., 2013.

* Corresponding author at: Chalmers University of Technology, Gothenburg, Sweden. Tel.: +46-31-772 1045.
E-mail addresses: vincenzo.gulisano@chalmers.es (V. Gulisano), mar.callau-zori@irisa.fr (M. Callau-Zori), zhang.fu@ericsson.com (Z. Fu), rjimenez@fi.upm.es (R. Jiménez-Peris), ptrianta@chalmers.es (M. Papatriantafilou), mpatino@fi.upm.es (M. Patiño-Martínez).
[1] Present address: IRISA, Université de Rennes 1, France.
[2] Present address: Ericsson Research, Sweden.

http://dx.doi.org/10.1016/j.eswa.2015.07.027
0957-4174/

analysis (Lakhina et al., 2004; Lakhina, Crovella, & Diot, 2005), dominate states analysis (Xu, Zhang, & Bhattacharyya, 2005), multivariate correlation analysis (Tan, Jamdagni, He, Nanda, & Liu, 2014), clustering (Lee, Kim, Kwon, Han, & Kim, 2008), artificial neural networks (Wang, Hao, Ma, & Huang, 2010), genetic algorithms (Tsang, Kwong, & Wang, 2007), fuzzy logic (Shiaeles, Katos, Karakos, & Papadopoulos, 2012) and finite state machines (Su, 2010). However, these are usually studied as off-line analysis methods and can be hardly applied in online detection, due to their time and space complexities. It is important to notice that successful detection not only depends on the analysis defined by a defense framework, but also depends on its placement. Frameworks deployed close to the target can be useless if the attack achieves the congestion of the target's upstream routers. That is, DDoS defense mechanisms are more effective in detecting and mitigating an attack when deployed at the core of the network (e.g., backbone links, also referred to as vantage points; Gao, Li, and Chen, 2006). This placement, nonetheless, increases the overall volume of traffic to deal with and the spectrum of possible threats.

From an expert system point of view, a comprehensive DDoS defense framework should be able to detect possible threats in a timely manner and, subsequently, mitigate them effectively. A requirement for such an expert system is to define detection and mitigation schemes that share information useful both to spot a possible threat and to mitigate it (Zhang, Li, & Gu, 2004). Nonetheless, the leveraging of the information maintained to detect threats in existing expert systems such as the ones discussed by (Kumar & Selvakumar, 2012; Lee et al., 2008; Lin & Tseng, 2004; Wang et al., 2010) in order to support the mitigatation of possible threats (e.g., by distinguishing between legitimate and illegitimate traffic) is not straightforward. This is believed to be one of the causes for the lack of expert systems providing comprehensive DDoS attacks defense, together with placement limitations of frameworks that do not allow for parallel and distributed high-throughput and low-latency traffic analysis.

We present STONE, a framework with expert system functionality that provides reactive anomaly-based DDoS detection and mitigation. In a nutshell, STONE runs prefix-level aggregation (Kim, Lau, Chuah, & Chao, 2006b) to aggregate the traffic of individual sources into source clusters sharing a given prefix of their IP addresses. This aggregation saves space and is motivated by the observation that the aggregated behavior of a source cluster is more stable than the one observed for its individual flows (Kim et al., 2006b). Subsequently, STONE groups source clusters (depending on their traffic features) and detects threats by studying how such groups evolve over time. An anomaly is detected when an abrupt change is observed in the distribution of source clusters to groups. Differently from other defense frameworks, STONE has been designed to maintain information that does not only enable fast threat detection, but that is also leveraged during mitigation to specify which packets in the traffic should be filtered and which not.

In order to meet the high processing throughput and low processing latency needs of an effective DDoS defense framework, STONE has been designed and implemented to run its analysis in a streaming fashion. The data streaming processing paradigm has shown great potential in addressing the performance requirements of threat detection (Chin, Choudhury, Feo, & Holder, 2014; Ganguly, Garofalakis, Rastogi, & Sabnani, 2007; Gao, Li, & Chen, 2006). In data streaming, information is processed by incremental algorithms, producing results continuously. Existing distributed and parallel stream processing engines such as StreamCloud (Gulisano, Jiménez-Peris, Patiño-Martínez, & Valduriez, 2010) allow to process millions of records per second.

STONE makes the following contributions:

- *Joint detection and mitigation of DDoS attacks:* STONE is a configurable framework that provides expert system functionality in which the information maintained to detect threats is also leveraged during mitigation to effectively distinguishing legitimate from suspicious traffic. We provide an attack characterization and show that *STONE* is able to detect any attack conducted by a number of malicious sources that exceeds a certain threshold, the latter depending on *STONE*'s configuration parameters.
- *Online and scalable traffic analysis:* By relying on the data streaming processing paradigm, *STONE* allows for parallel and distributed traffic analysis that can be deployed at high-speed network links (e.g., back-bone links). At the same time, by employing data structures for approximated checks with low false positive/negative errors, it also reduces the space required to maintain the information leveraged for threat detection and mitigation.
- *Thorough experimental validation: STONE* is implemented on top of StreamCloud (Gulisano, Jiménez-Peris, Patiño-Martínez, Soriente, & Valduriez, 2012; Gulisano et al., 2010), an elastic and parallel-distributed stream processing engine. Its performance, measured in terms of detection accuracy, filtering efficiency, monitoring overhead and scalability, is evaluated using data sets derived from real network traffic collected by CAIDA (Hick, Aben, & claffy, 2007) and SUNET (Moradi, Almgren, John, Olovsson, & Philippas, 2011; The Swedish University Computer Network OptoSUNET, 2012). As we show, *STONE* is able to detect DDoS attacks quickly, to mitigate them by filtering out the majority of the suspicious packets while keeping a high percentage of the legitimate traffic unaffected. At the same time, it also exhibits a processing throughput that scales linearly with the number of nodes used to deploy it.

The rest of the paper is organized as follows: Section 2 discusses the system model, Section 3 presents an overview of the architecture of *STONE*, discussed in detail in Sections 4 and 5. Section 6 discusses the implementation of *STONE* on top of StreamCloud, Section 7 presents our evaluation, Section 8 discusses the related work while Section 9 concludes the paper.

2. System model and problem formulation

In this section, we introduce the network and stream models, we define the adversary model and state the problem that *STONE* aims to solve.

2.1. Network system model

The network is modeled as a system composed of four kinds of entities: (1) *protected entities*, network entities that can be potentially attacked and, therefore, protected by *STONE* (they can be servers, subnets, or network bottleneck links); (2) *legitimate hosts*, end-hosts that consume protected entities' services without malicious intentions; (3) STONE*machines*, a set of machines (can be either routers or common machines) used to run *STONE* and provide DDoS detection and mitigation and (4) *bots*, network hosts that are controlled by the adversary to launch DDoS attacks to the protected entities. The protected entities, legitimate hosts and the bots are connected via network links and routers, while the *STONE* machines form a separated private network that cannot be reached by them. Note that *STONE* can be used for protection of both a single host and multiple hosts. It can also be extended for deployment in traffic control frameworks such as CluB (Fu, Papatriantafilou, & Tsigas, 2011). For the ease of the explanation, and without loss of generality, we present in the following how *STONE* behaves considering only the traffic being sent to a specific protected entity.

STONE uses anomaly-based detection to detect flooding-based DDoS attacks aiming at depleting the victim's network bandwidth. To effectively detect the attacks, a baseline of the legitimate traffic is built and is constantly compared with real time traffic behavior to identify deviations that may be caused by an attack. When building and maintaining profiles, an important issue is the stability of the

baseline profile. If the normal traffic behavior is quite unpredictable, then anomaly-based detection is not feasible. Fortunately, the normal traffic behavior indeed has periodical patterns. Based on the observation from the traffic analysis literature (Aiello, Gilbert, Rexroad, & Sekar, 2005; Roughan et al., 2002; Sekar, Duffield, Spatscheck, van der Merwe, & Zhang, 2006) and also from the analysis of our own datasets from the network backbone traffic, *STONE* uses *time-of-week* profiles, assuming that during normal network situations (i.e., without DDoS attacks) traffic behavior has basic periodicity of one week and that traffic patterns are relatively stable week to week.

2.2. Adversary and anomaly detection model

STONE focuses on mitigating network-layer flooding attacks where a powerful *adversary* can control numerous *zombie machines (bots)* to send large amounts of packets exhausting network resources such as link capacity, router buffers or server computation resources. *STONE* does not aim to mitigate *vulnerability attacks* (Mirkovic & Reiher, 2004) that exploit system vulnerabilities sending to the victim malformed packets that can cause excessive computation resources consumption or system reboot, such as Teardrop (CERT Coordination Center, 1997) or ping of death (CERT Coordination Center, 1997) attacks.

In particular, we focus on attacks that can be modeled as connection requests flooding (e.g., TCP SYN flooding) and bandwidth flooding (e.g., UDP or ICMP flooding). The potential targets can be a single server, a subnet or bottleneck network links. In a connection request flooding attack, the adversary initiates a massive number of seemingly legitimate connections to the victim server, holding most of its computation resources and thus denying services to legitimate clients. In a SYN-flood (Eddy, 2007), this is done by flooding a vast amount of SYN packets and thus forcing the victim to maintain many spurious sessions. During a bandwidth flooding attack, the adversary aggregates a big volume of traffic (possibly from zombie machines) to congest the target network link. The adversary can use different types of packets in the flooding attacks, such as TCP packets, UDP packets, ICMP packets, etc.

We assume that the adversary has no knowledge about characteristics of the victim's traffic such as the distributions of source addresses, the number of flows and the rates of flows. In other words, the adversary can hardly launch an attack without disproportional changes of the victim's traffic features. With respect to the reference information maintained by *STONE*, we assume that the attacker can neither modify nor pollute it. We stress that preventing the pollution of the reference information is orthogonal to the task of using it in order to detect attacks.

2.3. Stream processing model

In data streaming, incoming data is processed on-the-fly by *continuous* queries (simply referred to as queries in the remainder), defined as directed acyclic graphs of streams and operators. A stream is an unbounded sequence of tuples. All the tuples belonging to the same stream share the same schema, composed by attributes $A_1, \ldots, A_n$. For any schema, a field $ts \in A_1, \ldots, A_n$ represents the time when the tuple has been created. Given tuple t, $t.A_i$ (resp. $t.ts$) refers to the value of attribute A_i (resp. to its timestamp). Streams are consumed and produced by operators. Typical data streaming operators can be classified as *stateless* or *stateful*. *Stateless* operators do not maintain a state that evolves depending on the tuples beings processed and perform their computation based on individual tuples. Examples of *stateless* operators computations include tuple filtering or tuples' schema modifications. *Stateful* operators perform operations on sequences of tuples. Examples of *stateful* operators computations include aggregation of tuples or joining of tuples from different streams. Due to the unbounded nature of streams, stateful operators' results are computed over the most recent *sliding window* of tuples (e.g., tuples received in the last 5 min). Time-based sliding windows cover overlapping periods of *size* time, *advance* time units far from each other. For instance, a window of size 30 min and advance 10 min will cover periods [8:00:00–8:30:00), [8:10:00–8:40:00) and so on.

STONE defines two input data streams: *network stream, S*, and *aggregated network stream, S_a*. S represents the flow of packets sent to a protected entity; from *STONE*'s perspective, each packet can be seen as a tuple $\langle ts, srcIP, bytes\rangle$. Attributes *ts, srcIP* and *bytes* represent the timestamp, the source IP address and the size of the packet, respectively.

The stream S_a aggregates S packets on a per-*srcIP* basis over periods of time. A tuple in S_a is composed of attributes $\langle srcIP, ts_A, ts_B, packets, bytes\rangle$. As an example, given period 8:00:00–8:00:30, tuple $\langle A, 8{:}\ 00{:}\ 12, 8{:}\ 00{:}\ 25, 5, 250\rangle$ specifies that A sent 5 packets and a total of 250 bytes during the period of time starting at 8: 00: 12 and ending at 8: 00: 25. Stream S_a can be created from S using monitoring applications such as Cisco Netflow, which are widely supported by network devices or ISPs.

2.4. Problem formulation

Given a protected entity and its maximum bandwidth L (e.g., expressed in kbit/s), the goal of *STONE* is to monitor the traffic to (a) detect possible attacks and (b) to filter the traffic when it exceeds a *threshold bandwidth* αL, with $\alpha \in [0, 1]$ specified in advance. Whenever filtering is applied, *STONE* tries to maximize the percentage of legitimate traffic forwarded to the protected entity. The challenge lies in which criteria to use to discard or forward packets in S while ensuring that the traffic that reaches the protected entity does not exceed the entity's maximum bandwidth.

The reason why *STONE* activates the filtering mechanism only when the protected entity is near to be saturated is two-fold. On one hand, our solution is not intended to analyze the cause of the anomaly and does not distinguish between legitimate or illegitimate traffic spikes; therefore, we try to minimize the impact on the legitimate traffic by activating the filtering mechanism only when necessary. On the other hand, forwarding potentially malicious traffic when αL is not exceeded makes it harder for the adversary to adapt the attack depending on how the system is reacting to it.

STONE can be used to protect an entity from malicious activity as well as from legitimate peak loads (e.g., flash-crowds). For this reason, we use the term legitimate traffic in a global way to refer to sources that are either frequently communicating with the victim host or that were communicating with it before the attack or peak load started.

3. System overview

STONE analyzes and compares live and reference features of each protected entity's traffic in order to detect and mitigate attacks. This section overviews *STONE*'s components, described in detail later in Sections 4 and 5.

Fig. 1 presents *STONE*'s architecture. The *Detection Control Center* (DCC) is the subsystem in charge of detecting threats. It consumes the aggregated network stream S_a while comparing live and reference features, maintained at the *Historic Dataset* (HD). The *Mitigation Center* (MC) is placed between S and the protected entity and takes care of filtering the traffic if it exceeds the threshold bandwidth, αL. Its output stream S_m is equal to S whenever the incoming load is lower than αL, or a subset of S when filtering is applied. If the MC is not active, it simply forwards S packets, resulting therefore in a negligible overhead for the protected entity. Upon detection of a threat, the DCC shares with the MC the information needed to mitigate it. In the following, we provide an overview of the DCC and MC.

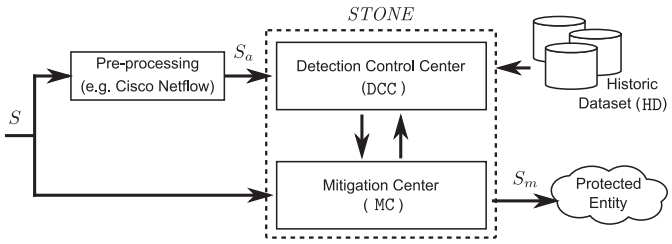


Fig. 1. *STONE* architecture.

Detection Control Center(DCC)*:* In order to detect suspicious traffic, the DCC needs to maintain information about the sources that communicate with the protected entity. Nevertheless, it might be impractical to maintain such information by means of individual per-IP profiles due to the large number of distinct sources that communicate with a protected entity. At the same time, predictions based on individual profiles could only be reliable for sources that contribute significantly to the protect ed entity's traffic (Kim, Won, & Hong, 2006a). For these reasons, the DCC initially aggregates the information carried by S_a tuples running IP prefix-level aggregation (Sekar et al., 2006). That is, it maintains traffic features for groups of source IPs, referred to as *source clusters* (*srcCL*), sharing the same prefix of *b* bits. Source clusters define the smallest reasoning unit for both threat detection and mitigation.

Tuples consumed by the DCC (stream S_a) carry information about the number of packets, bytes and the duration of each flow. Traffic information is consumed by the DCC, which maintains eight groups $\{G_0, \ldots, G_7\}$ of source clusters (each traffic feature is partitioned in 2). Groups' ratios (the fraction of source clusters falling into each group given the overall number of source clusters) characterize the traffic distribution (as histograms estimate probability density functions). Therefore, groups' ratios are continuously monitored and a threat is detected when the distance between its current and expected number exceeds a given threshold. In our model, the adversary has no knowledge about the characteristics of the protected entity's traffic. In other words, the adversary does not know which (and how many) of its source clusters stand out because of their contribution to the overall traffic in terms of packets, bytes or duration. Hence, he is not able to flood the system without modifying significantly the reference groups' ratios. More details are provided in Section 4.

In order to cope with the evolving nature of the protected entity's traffic, the DCC partitions source clusters to groups based on a portion of the most recent data. As introduced in Section 2, traffic is analyzed using *time-of-week* profiles. As presented in Fig. 2, each time a period starts (e.g., Monday, 11:00–12:00 a.m.), the information maintained at the HD is retrieved to compute the expected reference point group ratios. At the end of a monitoring period, the observed traffic is stored and merged with the information maintained at the HD.

Mitigation Center (MC)*:* The MC is responsible for mitigating threats while preventing the degradation of the protected entity's legitimate traffic. As discussed at the end of Section 2, we consider as legitimate the traffic generated by two types of sources: (1) sources communicating frequently with the protected entity and (2) sources have recently been communicating with the protected entity before an attack is detected. MC relies on two separate mechanisms to identify the traffic associated to these types of sources. Traffic generated by the sources that communicate frequently with the protected entity is identified relying on an "Acquaintance List" (*AL*), based on the information stored at the HD. At the same time, an enhanced Bloom Filter (Broder & Mitzenmacher, 2003) (*BF*) is used to maintain, a possibly large number of, source clusters that have been most recently communicating with the protected entity before the attack is detected.

To avoid bandwidth exhaustion by legitimate traffic and achieve *weighted fairness* among legitimate sources, legitimate traffic is forwarded through channels associated with different groups of source clusters. The channels are queues with bounded rate, implemented as a *Weighted Fair Queuing* mechanism (Zhang, 1990), where the weight of each queue depends on the average proportion of incoming traffic that belongs to the corresponding group. Traffic that is not recognized as illegitimate is forwarded through a dedicated channel (i.e. another queue in the weighted fair queuing mechanism) for suspicious traffic. However, in particular scenarios such as flash crowds, suspicious traffic might indeed be legitimate. For this reason, suspicious traffic is also forwarded relying on a Probabilistic Filtering (*PF*) mechanism that prevents the flooding of the protected entity. A detailed description of the MC filtering mechanism is given in Section 5.

4. Detection Control Center

In this section, we present in detail the operations performed by the DCC and the interaction with the HD. We also provide a characterization of the attacks that can be detected by the DCC (Section 4.3) and study its time complexity (Section 4.4).

The DCC consumes the aggregated stream S_a in order to detect threats. In Fig. 3, we show the four steps of its detection process:

Prefix-level aggregation and computation of traffic features: The DCC consumes tuples from S_a running IP prefix-level aggregation

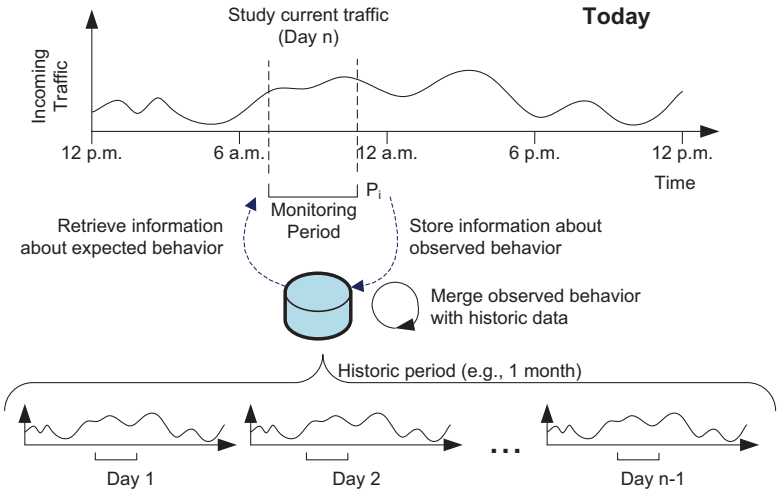


Fig. 2. Detection based on *time-of-week* monitoring.

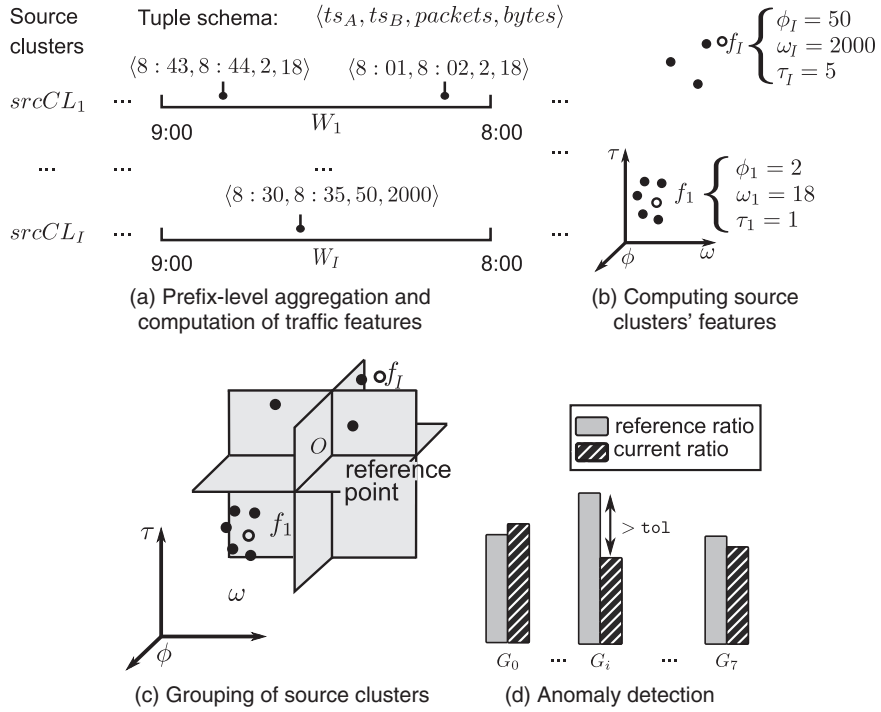


Fig. 3. Detection Control Center at a glance.

(Sekar et al., 2006). A time-based sliding window is used to maintain the traffic features of each *srcCL* observed over a period of one hour. In the example, two tuples referring to $srcCL_1$ are received in window W_1, covering period 8:00–9:00. Similarly, one tuple referring to $srcCL_I$ is received in window W_I (Fig. 3(a)).

Computing source clusters' features: Traffic belonging to each window W_i is aggregated into a feature 3D-point $f_i = (\phi_i, \omega_i, \tau_i)$, where ϕ_i, ω_i and τ_i represent the average number of packets, bytes and flow duration, respectively. In the example, the feature 3D-point of $srcCL_1$ is $f_1 = (2, 18, 1)$ (on average, flows in $srcCL_1$ involve 2 packets of 18 bytes and last 1 min) while the feature 3D-point of $srcCL_I$ is $f_I = (50, 2000, 5)$ (Fig. 3(b)). The concrete equations to compute the features in a sliding window are described in Section 4.1 and discuss the time complexity of the sec:dcc:complexity.

Grouping of source clusters: Source clusters are partitioned into eight groups $\{G_0, \ldots, G_7\}$ depending on the relative position of their features with respect to a reference point O. In Section 4.2 we discuss how the reference point O, crucial for the detection process, is computed. The reference point O is selected so that source clusters who contribute little to the overall traffic fall in G_0 while other source clusters (i.e., source clusters sending a significant amount of packets or bytes or showing long lasting flows) fall in groups $\{G_1, \ldots, G_7\}$ (Fig. 3(c)).

Anomaly detection: Being $\widehat{n}_i$ the number of source clusters in group G_i, the current ratio $\widehat{r}_i = \widehat{n}_i / \sum_j \widehat{n}_j$ and the reference ratio $r_i = n_i / \sum_j n_j$ (computed from the information maintained at the HD, as discussed in Section 4.2) are continuously compared, for $i = 0, \ldots, 7$. An anomaly is detected whenever $\max_i |r_i - \widehat{r}_i| \geq \texttt{tol}$ (Fig. 3(d)).

The above four steps describe how threats are detected based on the traffic features computed over a sliding window and based on the reference values (the reference point O and the reference ratios $\{r_i\}$) that are determinant for an effective threat detection. In the following, we describe in detail how source clusters' features are computed (Section 4.1) and how the DCC interacts with the HD to compute and maintain the reference values (Section 4.2). Finally, we discuss the possible impact of an attack given the information maintained by *STONE* (Section 4.3).

4.1. Computing source clusters' features over a time-based window

The DCC maintains four counters in order to compute the features $f_i = (\phi_i, \omega_i, \tau_i)$ of each cluster. Since *STONE* relies on the data streaming processing paradigm and its evolving *windows*, these counters refer only to the most recent information carried by S_a tuples. More concretely, given Eq. (1), the DCC maintains a separate counter for:

1. the overall number of packets in the current window, $\sum_{t \in W_i} t.packets$,
2. the overall number of bytes in the current window, $\sum_{t \in W_i} t.bytes$,
3. the overall elapsed time, $\sum_{t \in W_i} (t.ts_A - t.ts_B)$, and
4. the number of tuples observed in the current window, n_i,

The average number of packets, the average number of bytes and the average elapsed time are then computed by dividing each respective counter by the number of tuples observed in the window (n_i).

$$\begin{cases} \phi_i = \dfrac{1}{n_i} \displaystyle\sum_{t \in W_i} t.packets \\ \omega_i = \dfrac{1}{n_i} \displaystyle\sum_{t \in W_i} t.bytes \\ \tau_i = \dfrac{1}{n_i} \displaystyle\sum_{t \in W_i} (t.ts_A - t.ts_B) \end{cases} \tag{1}$$

4.2. Interaction with the Historical Dataset (HD)

The DCC uses historical information to compute the reference ratios $r_0, \ldots, r_7$ and the reference point O. The historical information is maintained at the Historical Dataset (HD).

The HD stores information from the past D days in intervals (e.g., Wednesday 10:00:00–11:00:00 during the last 10 weeks). Both the reference point and the reference ratios are calculated by weighting such information depending on its closeness to the present (i.e., giving more weight to recent information). Given D days $d_1, \ldots, d_D$ (being d_1 the earliest), the contribution of day d_j is w times bigger than

the contribution of day d_{j-1}. In our prototype, w is set to 2, but it can be changed as other parameters of *STONE*. The weight α_j of day d_j is computed according to Eq. (2).

$$\text{If } w \neq 1, \text{ then } \begin{cases} \alpha_1 = \dfrac{w-1}{w^D-1} \\ \alpha_j = w^{j-1}\alpha_1 \quad \forall j = 2,\ldots,D \end{cases} \qquad (2)$$

$$\text{If } w = 1, \text{ then } \quad \alpha_j = \frac{1}{D}, \forall j = 1,\ldots,D$$

Reference ratios $r_0,\ldots,r_7$ are computed using the number of source clusters belonging to all groups observed in the previous D days. The number of source clusters belonging to group G_g in day d_j, referred to as $n_g^{(j)}$ $\forall g = 0,\ldots,7$ and $j = 1,\ldots,D$, is maintained at the HD and updated by the DCC as traffic is analyzed. Hence, the weighted number of source clusters for group G_g during the previous D days is computed as $\sum_j \alpha_j n_g^{(j)}$. The reference ratio r_g of group G_g is computed as the ratio between the weighted number of source clusters in group G_g over groups $G_0,\ldots,G_7$ (Eq. (3)).

$$r_g = \frac{\sum_{j=1}^{D} \alpha_j n_g^{(j)}}{\sum_{j=1}^{D} \alpha_j \sum_{g=0}^{7} n_g^{(j)}} \qquad \forall g = 0,\ldots,7 \qquad (3)$$

Each source cluster's contribution to the reference point $O = (O_\phi, O_\omega, O_\tau)$ is weighted for each day $j = 1,\ldots,D$. However, the contribution is not weighted over all days D but rather on the number of days $\mathcal{D}_i$ during which each $srcCL_i$ contributed to the protected entity's traffic. Doing this, the contribution to the reference point O of the source clusters communicating frequently with the protected entity is greater than the one of source clusters communicating sporadically. The HD stores the features for each source cluster for each day $f_i^{(j)} = (\phi_i^{(j)}, \omega_i^{(j)}, \tau_i^{(j)})$, $\forall i = 0\ldots7$ and $j = 1,\ldots D$. The reference feature for each cluster is computed as presented in Eq. (4) (showing the computation of ϕ_i, feature ϕ of $srcCL_i$).

$$\phi_i = \sum_{j\in\mathcal{D}_i} \alpha_j \phi_i^{(j)} \Big/ \sum_{j\in\mathcal{D}_i} \alpha_j \qquad (4)$$

The reference point O is computed as the weighted 0.95-quantile of the reference features $\{f_i\}$ of each source cluster $srcCL_i$. The features of each source cluster $srcCL_i$ are weighted depending on $\mathcal{D}_i$ by $\gamma_i = \sum_{j\in\mathcal{D}_i} \alpha_j$. The 0.95 value is motivated by Kim et al. (2006a), stating that more than 90% of the traffic flows are small flows (in terms of their contribution to the overall traffic). Assuming that source clusters features are independent, group G_0 includes the $95\% \times 95\% \times 95\% \approx 86\%$ of the overall source clusters that send less packets, bytes and that exhibit the lower flow durations.

4.3. Attack characterization

This section formalizes the impact of an attack with respect to the information maintained by *STONE*. The characterization is helpful to determine DCC parameters such as the tolerance `tol` and the prefix length used to aggregate flows. Packet flooding attacks usually involve a large number of machines connected to a protected entity at the same time. We consider an adversary model where the amount of illegitimate traffic is generated uniformly from the source clusters to which illegitimate IPs belong to. Such a scenario takes place when the attacker chooses bots randomly and when each malicious source IP generates approximately the same amount of traffic. Since source clusters will behave similarly, most likely they will fall in the same group. This will lead to an abrupt change in the group frequencies that should be detected by *STONE*.

Given N possible source clusters, N_L source clusters are known by the DCC (i.e., their features are stored in the HD) while $N - N_L$ source clusters have not been observed before. We refer to N_I as the number of malicious source clusters selected to launch the attack. We first study which portion N_I^{new} of them overlaps with the previously unseen $N - N_L$ source clusters.

Proposition 4.1. *The number of malicious source clusters that overlaps with the previously unseen ones, N_I^{new} follows a Binomial $(N_I, 1-p)$ distribution, if N and $N - N_I^{new}$ are large enough and $p = N_L/N$ is close to 0.*

Proof. The number of malicious source clusters N_I^{new} follows a hypergeometric distribution $N_I^{new} \sim \mathcal{G}(N, m, n)$ with parameters N, $m = N - N_L$ and $n = N_I$. The proposition is proved using the probabilistic result by Rice (2001). □

Since the number of possible source clusters is $N = 2^b$, the above result gives an intuition about the mask length b that should be used to run prefix-level aggregation according to the protected entity traffic. Notice that the hypothesis is not too strict. The prefix length chosen to aggregate flows should define a large N in order to avoid a too coarse-grained partitioning of traffic to source clusters. In other words, N should be chosen to be much higher than N_L, the number of known source clusters. A tradeoff exists between the space complexity (that is, the information stored for N_L known source clusters) and the ability to discover threats.

If we consider the number of unknown source clusters that must appear in order to detect a traffic anomaly, we have that this number a is related to the information archived in the DCC and the reference parameters.

Proposition 4.2. *In the worse case, the number of new source clusters which must appear to detect a traffic anomaly is $a = \texttt{tol}\frac{N_L}{1-(\max_j r_{j+\texttt{tol}})}$.*

Proof. Lets G_{j_0} be the group raising the anomaly, that is, $|r_{j_0} - \widehat{r}_{j_0}| \geq \texttt{tol}$. The current ratio is given by $\widehat{r}_{j_0} = (r_{j_0}N_L + N_I^{new})/(N_L + N_I^{new})$; according to the number of legitimate source clusters $r_{j_0}N_L$, the number of new source clusters N_I^{new} and the total number of source cluster $N_L + N_I^{new}$. We have $\texttt{tol} \leq |r_{j_0} - \frac{r_{j_0}N_L+N_I^{new}}{N_L+N_I^{new}}| = \frac{N_I^{new}}{N_L+N_I^{new}}(1-r_{j_0})$, then $N_I^{new} \geq \texttt{tol}\frac{N_L}{1-(r_{j_0}+\texttt{tol})}$. In the worse case, the value is achieved when $r_{j_0} = \max_j r_j$, proving the result. □

The result helps in determining the tolerance parameter `tol` according to the host traffic (N_L and $\max_j r_j$). Merging both results, we get a general view of the attack characterization that helps in choosing the DCC parameters given the traffic behavior.

Theorem 4.3. *STONE* detects an attack with high probability, $1-\delta$, if the number of source clusters affected by the attacker N_I is at least $\alpha = a + \frac{\log(1-\delta) - a\log(1-p)}{\log p}$, where the value of a is given by Proposition 4.2.

Proof. If $N_I \geq \alpha$, then $(N_I - a)\log p + a\log(1-p) \geq \log(1-\delta)$. Obtaining $Pr\{N_I^{new} = a\} = \binom{N_I}{a}(1-p)^a p^{N_I-a} \geq 1-\delta$ due to Proposition 4.1. As a result of Proposition 4.2 an anomaly is detected if $N_I^{new} = a$. □

4.4. Time complexity analysis

In this section, we show that our detection method has a $O(1)$ time complexity. As discussed in Section 4, the DCC module performs four main steps for each incoming aggregated packet. Firstly, the computation of traffic features finds the *srcCL* to which the aggregated packet belongs to (searching through all of them in a worst case scenario in which source clusters are maintained using a list or, as in our case, in a constant average time by relying on a hash table). Thus, the first step has a time complexity $O(m \times p)$ since one profile is maintained for each *srcCL*. Secondly, in order to compute source clusters' features, the aggregation of each feature results in a time complexity $O(m)$, independently of the window size. Thirdly, to determine the source cluster's group it is enough to check the relative position of

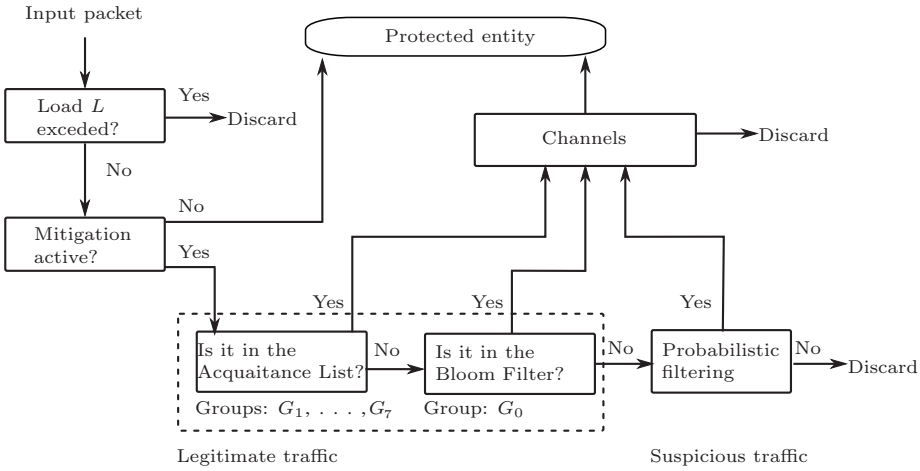


Fig. 4. Filtering protocol.

the feature with respect to the reference point. Thus, the third step has $O(m)$ time complexity. Finally, whenever a group changes, only two ratios must be updated (old and new group ratio). Hence, the time complexity of the DCC is $O(m \times p)$. Because the number of features, m, is a fixed number (3 in our case) and because the maximum number of profiles, p, is determined by prefix aggregation in advance, the overall time complexity for the DCC module is $O(1)$.

5. Mitigation Center

The Mitigation Center (MC) is responsible for discarding packets during an attack while not exceeding the protected entity maximum load L and while minimizing the degradation of the legitimate traffic.

Fig. 4 presents the MC filtering protocol. The mitigation mechanism is reactive, concretely, the mitigation it is activated (resp. deactivated) when the load forwarded to the protected entity exceeds αL (respectively falls below αL); and the information used to filter packets is passed to the MC by the DCC. As introduced in Section 3, the MC relies on the Acquaintance List (*AL*) to forward the traffic generated by the sources that communicate frequently with the protected entity (i.e., traffic generated by source clusters falling in groups $G_1, \ldots, G_7$). At the same time, it relies on a Bloom Filter (*BF*) to forward the traffic of the sources communicating with the protected entity before an attack is detected. Since traffic generated by source clusters falling in groups $G_1, \ldots, G_7$ is already forwarded based on the *AL*, the *BF* is used to maintain only the source clusters falling in group G_0. A portion of the protected entity's bandwidth, referred to as the *allowed suspicious bandwidth* L_A, is dedicated to suspicious traffic (which might be legitimate in scenarios such as flash crowds). A naïve approach to forward suspicious traffic might be to forward all suspicious packets as long as L_A is not exhausted (e.g., given $L_A = 100$ B/s, the first 100 bytes of suspicious traffic would be forwarded during each period of 1 s). Such approach would not provide the same chance to reach the protected entity to all the suspicious packets belonging to the same period. This limitation is overcome by *STONE*'s non-blocking Probabilistic Filtering (*PF*) mechanism. As introduced in Section 3, together with the *AL*, the *BF* and the *PF* mechanism, channels are used to distribute the available bandwidth to groups resembling the distribution observed in the baseline profile.

In the rest of the section, we discuss the filtering based on the Bloom Filter *BF* (Section 5.1), the probabilistic filtering of suspicious packets *PF* (Section 5.2) and the channel mechanism (Section 5.3).

5.1. Filtering based on the Bloom Filter

STONE relies on a *BF* to maintain source clusters belonging to group G_0. Bloom Filters are designed to maintain sets of elements; new elements can be added to the set but not removed, which is a shortcoming under the sliding window model. In order to overcome this limitation, we define a *time-based* Bloom Filter to check the membership of elements in a time-based window. The timestamp of the last inserted element is associated with each entry in the BF main array. An element does not belong to a given time interval if any of its positions in the array is set to 0 or if the time distance between any of the timestamps associated to its positions and the timestamp of the last element inserted exceeds the window size. The presented time-based Bloom Filter has the same accuracy and complexity that the one discussed by Broder and Mitzenmacher (2003).

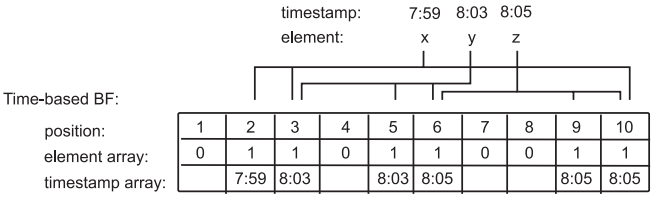


Fig. 5. Time-based Bloom Filter.

Fig. 5 presents an example where three elements *x, y* and *z* (with timestamps 7:59, 8:03 and 8:05) are inserted in a time-based BF whose window size is of 5 min. In the example, the BF array size is 10 and 3 hash functions are used to add elements. First, element *x* is added to the BF setting 1 and timestamp 7:59 to positions 2, 3 and 10. Subsequently, element *y* is added, setting 1 and timestamp 8:03 to positions 3, 5 and 6 (elements *x* and *y* overlap in position 3). Finally, element *z* is added setting 1 and timestamp 8:05 to positions 6, 9 and 10 (elements *x* and *z* overlap in position 10 while elements *y* and *z* overlap in position 6). Suppose that, when adding element *z*, we are interested in checking whether element *x* has been added to the BF in the last 5 minutes. We can conclude that element *x* was added more than 5 min ago because, even if all its positions are set to 1, the distance from the timestamp at position 2 (7:59) and the timestamp of *z* (8:05) is greater than 5 min.

5.2. Probabilistic filtering of suspicious traffic

With probabilistic filtering, each suspicious packet p is forwarded to the protected entity according to a Bernoulli trial (this way, each packet gets the same chance of being forwarded to the protected entity). The trial probability is related to the allowed suspicious bandwidth L_A, representing the fraction of the protected entity bandwidth devoted to suspicious traffic. The probability to forward a packet p is given in Eq. (5). $1 - \mu$ represents the remaining time of the current

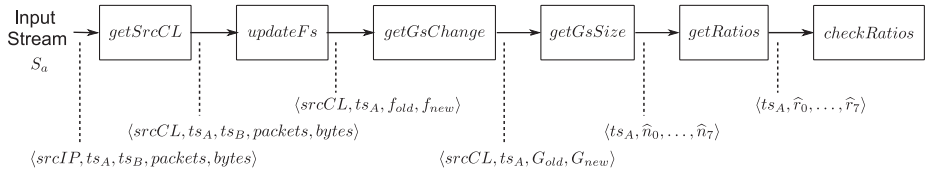


Fig. 6. Data streaming operators composing the Detection Control Center.

Table 1
Example of probabilistic filtering.

Packet: ⟨time (sec), bytes⟩	Available	L_A next restart	$Pr\{forward\}$	Action
$p_1 = (1.01, 60)$	100	0.99	1	Forward
$p_2 = (1.05, 55)$	40	0.95	0.42	Discard
$p_3 = (1.7, 20)$	40	0.3	1	Forward
$p_4 = (1.75, 10)$	20	0.35	0.57	Forward
$p_5 = (2.15, 10)$	100	0.85	1	Forward

period while parameter β is used to bound the probability.

$$\Pr\{p \text{ is forwarded}\} = \max\left\{\beta \frac{L_A}{1-\mu}, 1\right\} \tag{5}$$

In the following, we present a sample execution of the probabilistic filtering mechanism (Table 1). We consider a sequence of 5 suspicious packets $\{p_1, p_2, p_3, p_4, p_5\}$. In each row, the table presents the available suspicious bandwidth L_A (set to 100 bytes), the remaining time of each period, the probability with which the packet is forwarded and the resulting action in the experiment.

5.3. Channel mechanism

The filtering mechanisms presented so far are intended to prioritize legitimate traffic over suspicious one. Nevertheless, their effectiveness might be impaired if source IPs employed by the attacker and legitimate ones overlap. That is, if there is no upper bound for the bandwidth that can be consumed by each source cluster and if part of the malicious IPs happens to belong to a legitimate source cluster, the attacker could potentially generate enough traffic to exceed L. We stress that, given our adversary model, this scenario does not happen intentionally.

The channel mechanism is responsible for adjusting the portion of the overall bandwidth that can be used by each source cluster. More concretely, channels are used to distribute L to groups resembling the traffic distribution observed in the baseline profile, thus avoiding individual source clusters traffic from exceeding L. *STONE* maintains 9 channels: one channel for each group plus a dedicated channel for suspicious traffic. Each channel is associated to a bandwidth weight (weight ξ_g for each group G_g and weight ξ_S for the suspicious traffic). To resemble the baseline traffic distribution, groups use the information stored in the HD, while suspicious traffic is based on the maximum probability to forward a suspicious packet β. That is, without channel mechanism, the bandwidth used by suspicious traffic is (in mean) βL. The channels are implemented as a *Weighted Fair Queuing* mechanism (Zhang, 1990), where the concrete weight of each queue is computed in Eq. (6).

$$\begin{cases} \xi_g = \dfrac{s_g}{(1+\beta)\sum_i s_i}, & \forall g = 0 \ldots 7 \\ \xi_S = \dfrac{\beta \sum_i s_i}{(1+\beta)\sum_i s_i} = \dfrac{\beta}{1+\beta} \end{cases} \tag{6}$$

where $s_g^{(j)}$ is the number of bytes received by group G_g in the jth day, $\forall g = 0, \ldots, 7$ and $j = 1 \ldots, D$, $s_g = \sum_j \alpha_j s_g^{(j)}$ represents its weighted average over the past D days (according to the day weights presented in Eq. (2)).

6. Implementation on top of the StreamCloud stream processing engine

This section presents how *STONE* has been implemented on top of *StreamCloud* (Gulisano et al., 2012), a state-of-the-art parallel-distributed SPE. We first discuss how the DCC and the MC have been implemented by means of data streaming operators. Subsequently, we give a brief overview about how *StreamCloud* parallelizes data streaming operators. Finally, we present how both the DCC and the MC are run in parallel by *StreamCloud*.

As presented in Fig. 6, the DCC is composed of six data streaming operators. Operator *getSrcCL* processes S_a tuples and modifies their schema $\langle srcIP, ts_A, ts_B, packets, bytes\rangle$ replacing the *srcIP* with its respective source cluster *srcCL*. The operator *updateFs* maintains the features of each source cluster. For each incoming tuple t, an output tuple composed by attributes $\langle srcCL, ts_A, f_{old}, f_{new}\rangle$ carries the features of the source cluster *srcCL* before (f_{old}) and after (f_{new}) processing t. Features f_{old} and f_{new} are used by the *getGsChange* operator to compute groups G_{old} and G_{new}. Whenever a source cluster appears, disappears or changes the group it belongs to (i.e., whenever $G_{old} \neq G_{new}$), a tuple composed by attributes $\langle srcCL, ts_A, G_{old}, G_{new}\rangle$ is forwarded to the operator *getGsSize*. For each incoming tuple, the operator *getGsSize* updates G_{old} and G_{new} counters and produces a tuple composed by attributes $\langle ts_A, \widehat{n}_0, \ldots, \widehat{n}_7\rangle$. These tuples are consumed by the operator *getRatios*, which updates the current ratio of each group, producing tuples composed by attributes $\langle ts_A, \widehat{r}_0, \ldots, \widehat{r}_7\rangle$. Updated ratios are forwarded to operator *checkRatios* and compared with the reference ratios in order to spot anomalies.

Operator *updateFs* maintains a sliding window of tuples, W_i, in order to update the features f_i of each source cluster (i.e., it is a *stateful operator*). As introduced in Section 4, the operator relies on four EHs (Exponential Histograms), one for each counter, to maintain the features of each source cluster. All the other operators do not maintain tuple windows (i.e., they are stateless) and have constant order space complexity in the window size. A naïve approach to compute a sum over a window would be to maintain all its tuples (updating the sum for each tuple entering or leaving the window). As discussed by Datar, Gionis, Indyk, and Motwani (2002), EHs reduce the space complexity required to compute sums from linear to poly-logarithmic in the window size aggregating the information of consecutive tuples.

The MC is composed of a single stateless data streaming operator. More concretely, a filter is used to forward or discard each packet of stream S.

By relying on *StreamCloud*, *STONE*'s operators can be run in parallel at an arbitrary number of SPE instances. Operators are parallelized by deploying multiple copies of them at different SPE instances and by partitioning their input streams.

Fig. 7 presents how *STONE*'s operators *getSrcCL* and *updateFs* are deployed in parallel at M and N SPE instances, respectively. The term *subcluster* refers to the set of SPE instances running a given parallel operator. At each SPE instance, a *Load Balancer* (LB) routes output tuples to downstream instances while an *Input Merger* (IM) merges input tuples forwarded by upstream instances. How tuples are routed

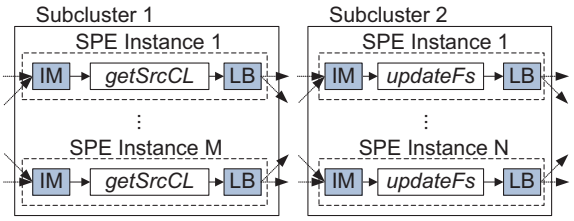


Fig. 7. Sample parallel-distributed deployment of operators *getSrcCL* and *updateFs*.

between two generic subclusters SC_1 and SC_2 depends on the first operator deployed at SC_2. Round-robin routing is employed if the operator deployed at SC_2 is stateless while hash-based routing is employed if the operator is stateful. In this second case, the stream connecting SC_1 and SC_2 is partitioned into B buckets depending on the semantics of the stateful operator and each bucket is assigned to exactly one of the SC_2's instances. In the example, the stream connecting *getSrcCL* and *updateFs* will be partitioned on the *srcCL* attribute (*updateFs* is a stateful operator). Doing this, tuples referring to the same *srcCL* will always be processed by the *updateFs* operator running at the same SPE instance. We refer the reader to Gulisano et al. (2010) for a detailed discussion about how queries are partitioned into subclusters in order to maximize the overall query throughput.

Fig. 8 presents the parallel-distributed deployment of *STONE*'s query in *StreamCloud*. Following *StreamCloud* parallelization rules, the DCC has been partitioned into three subclusters. A first subcluster is defined for *getSrcCL*. A separate subcluster has been defined for *updateFs* and the following stateless operators *getGsChange* and *getGsSize*. Finally, a centralized subcluster has been defined for *getRatios* and *checkRatios*. These two operators are centralized since they need to gather all the group counters maintained by *getGsSize* in order to compute the observed ratios and compare them with the expected ones. It should be noticed that the centralized execution of *getRatios* and *checkRatios* does not constitute a bottleneck. As an example, gathering the counter periodically every 0.05 s from 100 SPE instances running *getGsSize* results in a rate of 2000 tuples/s, which is lower than the maximum throughput of *getRatios* and *checkRatios*, estimated at approximately 10000 tuples/s.

The MC (*filter* in Fig. 8) is run in parallel in a dedicated subcluster. The instances of operator *filter* take decisions about which packets to discard depending on the available bandwidth of the protected entity. Such information must be derived from the bandwidth consumed by the packets forwarded by each operator instance. For this reason, *filter* instances are interconnected in order to share the amount of packets being forwarded by each one of them.

7. Evaluation

In this section, we evaluate *STONE*'s detection and mitigation capabilities together with its scalability. An effective DDoS defense mechanism should detect anomalies quickly and mitigate them without affecting legitimate users' traffic. For this reason, we focus in this evaluation on the (1) detection time, the time elapsed between the beginning of the anomaly and its detection, the (2) mitigation precision, which quantifies the degradation of the legitimate traffic and the (3) traffic volume shaping, which quantifies the portion of overall traffic discarded during an anomaly. With respect to the scalability evaluation, we study how the throughput and the CPU consumption of STONE*machines* evolve for increasing loads and different parallel-distributed setups of 1, 10, and 20 nodes.

7.1. Evaluation setup

The legitimate traffic is composed by anonymized data traces from an OC-192 (10 Gbits/s) backbone link of OptoSUNET (Swedish University Computer Network) (The Swedish University Computer Network OptoSUNET, 2012). The link connects OptoSUNET to NORDUnet (The NORDUnet IP & MPLS network, 2012), via which the former can reach the rest of the world, and carries around one third (3 backbone links exist) of its inbound traffic. The data traces are excerpts of the traffic happening on 9 consecutive Thursdays (during the year 2010) between 11:00 a.m. and 12:00 a.m. During the experiments the data traces' packets are fed to *STONE* at their original rate while tuples belonging to the aggregated network stream S_a are generated for consecutive intervals of 5 min. We select as the protected entity one of the destination hosts appearing in the data traces showing continuous incoming TCP connections. The HD is populated using the traffic's features observed during the first 8 Thursdays while the last day trace is used as *legitimate data trace*.

The *suspicious data trace* is taken from CAIDA (Hick et al., 2007) and contains anonymized packets from a DDoS attack. According to the description from CAIDA, this attack attempted to block access to the targeted server by consuming both the computing resources and the incoming link's bandwidth of the server. Since *STONE* does not analyze the cause of an anomaly and cannot state whether a single

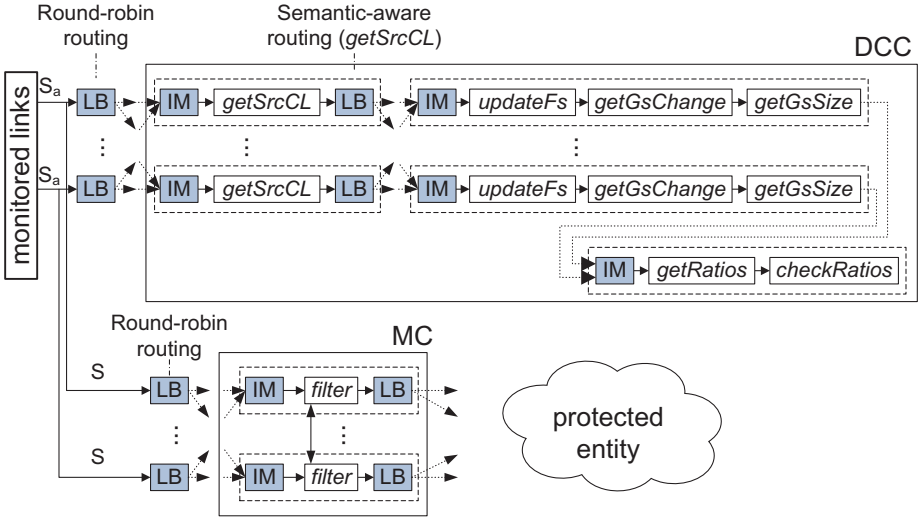


Fig. 8. Complete parallel and distributed deployment of *STONE* in *StreamCloud*.

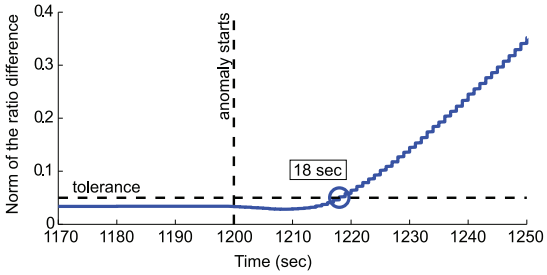


Fig. 9. Detection time. The attack is detected after 18 s.

packet is illegitimate or not, we refer in the following to the attack data as suspicious to put ourselves in *STONE*'s perspective. The number of packets in the *suspicious data trace* exceeds by two orders of magnitude the one in the *legitimate data trace*. During the evaluation, the *suspicious data trace* is mixed with the *legitimate data trace*. The *legitimate data trace* covers a period of one hour, the *suspicious data trace* is injected starting from minute 20 (second 1200). The destination host in the *suspicious data trace*'s packets has been changed to the protected entity appearing in the *legitimate data trace*. In our evaluation we use a cluster of quad-core nodes equipped with Xeon X3220@2.40 GHz, 8 GB of RAM and 1 Gbit Ethernet.

7.2. Detection time

Fast detection of DDoS attacks is crucial for an effective mitigation. As discussed in Section 3, an anomaly is detected if the maximum difference between the reference and current ratio ($\max_i |r_i - \widehat{r}_i|$) for any group i exceeds a given tolerance `tol`. In our evaluation, tolerance `tol` has been set to 0.05 (we discuss how to appropriately chose `tol` in Section 4.3). Fig. 9 shows the maximum difference between the reference and current ratios before and after the anomaly. To better appreciate how the difference increases after the beginning of the anomaly, the x-axis shows a time frame of 80 s around second 1200 (the second at which the anomaly starts). The difference starts increasing quickly a few seconds after the beginning of the anomaly. In this experiment, the anomaly is detected 18 seconds after it starts. It can be noticed that higher tolerance values will result in longer detection times (for instance, the detection time would be of approximately 30 s given $\texttt{tol} = 0.1$).

7.3. Mitigation precision

These experiments study how effective the MC is in mitigating DDoS threats. We first study its effectiveness if filtering is applied only based on the AL and the BF (that is, if no suspicious packet is forwarded at all). We then repeat the experiment allowing suspicious packets to be forwarded applying probabilistic filtering. As discussed in Section 5, probabilistic filtering may be used to forward suspicious packets that are actually legitimate (e.g., during a flash crowd). The portion of bandwidth assigned to the suspicious traffic has been set to 10^3 kbit/s (i.e., $\beta L = 10^3$ kbits).

Acquaintance List and Bloom Filter filtering: Fig. 10(a) presents the fractions of legitimate and suspicious traffic that are forwarded to the protected entity. The x-axis considers a time frame of approximately 15 min before the beginning of the anomaly and after its completion. Before the anomaly starts, all the legitimate traffic is forwarded to the protected entity. Starting from second 1200, both legitimate and suspicious traffic are forwarded to the protected entity. Upon detection, most of the suspicious traffic (approximately 99%) is discarded while approximately 90% of the legitimate traffic is still forwarded to the protected entity. When the anomaly ends, all the legitimate traffic is forwarded again to the protected entity.

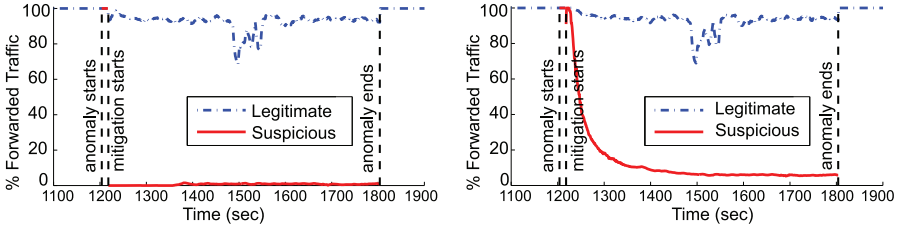
Probabilistic filtering: Fig. 10(b) presents the fractions of legitimate and suspicious traffic forwarded to the protected entity when applying probabilistic filtering. It can be noticed that the portion of legitimate traffic forwarded to the protected entity is comparable with the one in Fig. 10. This is because legitimate packets that are not forwarded by the Acquaintance List or the Bloom Filter have a much smaller chance of being forwarded applying probabilistic filtering than suspicious ones (suspicious traffic exceeds the legitimate one by 2 orders of magnitude). The suspicious traffic forwarded to the protected decreases as the overall injected suspicious traffic increases and never exceeds βL (as also discussed in the following experiment).

7.4. Traffic volume shaping

Together with the mitigation precision evaluation, we evaluate how effective the MC is in shaping the protected entity's traffic volume. As for the mitigation precision, we first consider the traffic volume shaping when mitigation is only based on the AL and the BF. Subsequently, we present how traffic volume is shaped when probabilistic filtering is also applied to the suspicious data.

Acquaintance List and Bloom Filter filtering: Fig. 11(a) presents the traffic volume shaping when filtering is only based on the BF and the AL. The load is expressed in KBit/sec, using a logarithmic scale for the y axis. The solid line represents the overall traffic load injected during the anomaly while the dashed line represents the one forwarded to the protected entity. While the mitigation is not active, all the input traffic is forwarded to the protected entity. Once the anomaly is detected, most of the overall traffic ($\approx$ 97%) is discarded. That is, the BF and the AL together are able to reduce drastically the amount of suspicious traffic.

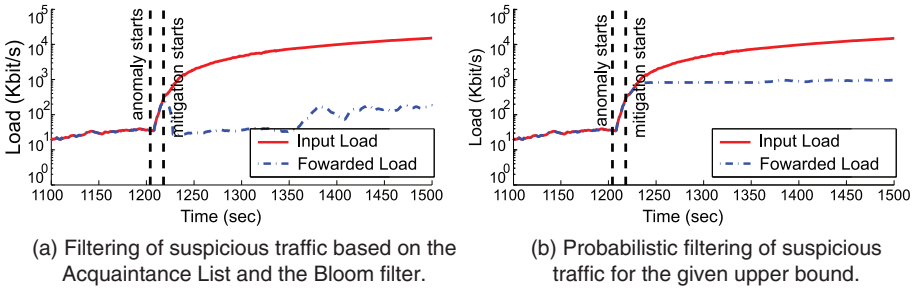
Probabilistic filtering: Fig. 11(b) presents the traffic volume shaping when probabilistic filtering is applied. Once the anomaly is detected, the forwarded traffic increases to βL, set to 10^3 kbits in the example, before being discarded. Differently from the previous experiment, all the legitimate and suspicious traffic are forwarded as long as they do not saturate the protected entity.



(a) Filtering of suspicious traffic based on the Acquaintance List and the Bloom filter.

(b) Probabilistic filtering of suspicious traffic for the given upper bound.

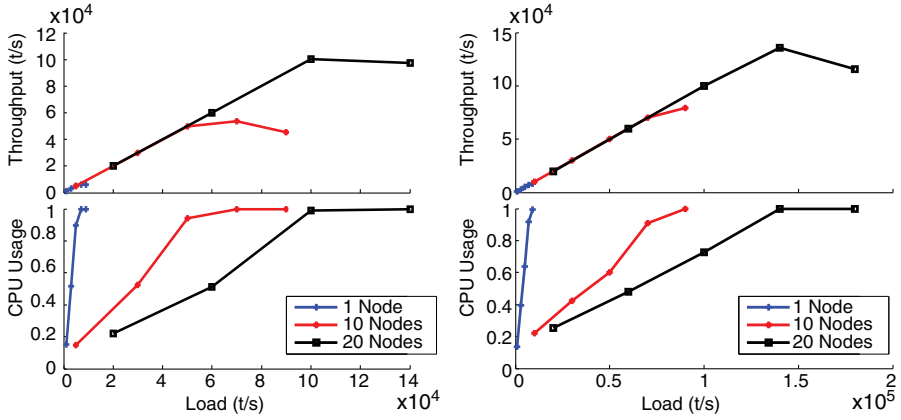
Fig. 10. Mitigation precision. Low degradation of legitimate traffic is observed during mitigation.



(a) Filtering of suspicious traffic based on the Acquaintance List and the Bloom filter.

(b) Probabilistic filtering of suspicious traffic for the given upper bound.

Fig. 11. Filtering of suspicious traffic observed during mitigation.



(a) DCC's stateful subcluster

(b) Mitigation Center's subcluster

Fig. 12. Scalability evaluation.

7.5. Scalability evaluation

In this experiment, we evaluate the scalability of *STONE* implementation on top of *StreamCloud*. We study how its throughput and CPU consumption evolve for different setups of 1, 10 and 20 nodes and a linearly increasing load. In all experiments, we inject the mixed *legitimate data trace* and *suspicious data trace* simulating a linearly increasing load. The timestamp of the injected packets is modified accordingly with the injection rate. We focus on the scalability of the stateful subcluster in the DCC and the subcluster composing the MC.

Detection Control Center scalability: Fig. 12(a) presents the evolution of the throughput (upper part of the figure) and CPU consumption (bottom part of the figure) for the DCC's stateful subcluster. With respect to *STONE*'s throughput, the subcluster is able to process approximately 6000 tuples/s (t/s) when running at a single node (centralized execution) . Using a setup of 10 nodes, the maximum throughput increases to approximately 55,000 t/s (9.2 times the throughput achieved by the centralized execution). Finally, when running at 20 nodes, the maximum throughput increases to approximately 98,000 t/s (16.3 times the throughput of the centralized execution). Focusing on the CPU consumption curves, it can be noticed that the higher the number of SPE instances running the parallel subcluster, the milder the slope of the curves. This result is expected, since the injected load is increasing linearly.

Mitigation Center scalability: Fig. 12(b) presents the throughput and CPU evolution for the MC's subcluster. Results for the MC are similar to the ones observed for the DCC. The centralized execution of the subcluster achieves a throughput of approximately 8000 t/s. The throughput increases to approximately 79,000 t/s when running the subcluster at 10 nodes (9.9 times the centralized execution). The maximum throughput increases to approximately 135,000 t/s (16.9 times the centralized execution) when running the subcluster at 20 nodes. As for the DCC scalability evaluation, the CPU consumption grows with a milder slope as more instances are deployed for the parallel subcluster.

8. Related work

In this section, we present DDoS attacks defense solutions discussed in the literature and compare them with *STONE*. We first discuss DDoS attacks detection methods. Subsequently, we focus on DDoS attacks mitigation methods. Finally, we present a dedicated comparison with other expert systems designed for DDoS attacks defense.

8.1. Network anomaly detection methods

Network anomaly detection is a rich research area. As we discussed in Section 1, detection on its own (especially if defined as an offline method) is not enough to protect a target if not coupled with an effective mitigation scheme.

At the same time, as Peng, Leckie, and Ramamohanarao (2007) discuss, the deployment of a DDoS detection method is also of crucial importance. Nonetheless, as evidenced also by recent studies running extensive comparisons between existing anomaly detection schemes (Bhuyan, Bhattacharyya, & Kalita, 2014), distributed analysis solutions are not usually provided. Differently from *STONE*, none among the 71 publications summarized in the article tables provides a completely distributed solution. At the same time, none of the 17 network intrusion detection systems (NIDS) taken into account provided online and real time detection of DDoS attacks (one of the motivating challenges behind *STONE*).

Tan et al. (2014) proposed a DoS attack detection method based on multivariate correlation analysis. Although the method results in a high detection rate, its detection time is not evaluated, differently from *STONE*. The proposed method has time complexity $O(m^4)$, where

m is the number of features. This complexity is higher that *STONE*'s one, which is $O(m \times p)$, where p is the number of profiles. Also in this case, authors do not discuss how the method can be run in a distributed and parallel fashion.

Bhuyan, Bhattacharyya, and Kalita (2015) presented an empirical evaluation for DDoS attack detection based on two metrics: the Renyi's entropy and the information divergence. Both metrics are based on the empirical probability distribution over three features (source IP, destination IP and protocol) in a window of 10 s. Two empirical distributions are computed for current traffic and legitimate one. The entropy-based approach relies on the difference in entropy of such distributions to detect attacks. The divergence-based approach, on the other hand, detects attacks by directly comparing the two empirical distributions. However, at vantage points, the number of source IP is large and maintaining an empirical probability distribution over each source IP is not feasible in terms of space and computational resources. Evaluation over CAIDA traces shows that these techniques are able to detect high-rate and low-rate DDoS attacks, but, also in this case, do not evaluate the time required to detect such attacks.

When focusing on streaming-based DDoS detection methods, the main approach to detect threats is to track destination IP addresses having a large number of half-open connected source IPs. Ganguly et al. (2007) proposed a distinct-count sketch to estimate the top-k distinct-source frequencies. A sketch is a probabilistic summary which relies on projection along random vectors to achieve space efficiency with guaranteed probabilistic reconstruction accuracy. Differently from *STONE*, sketch-based solutions do not support continuous monitoring with sliding windows, since the random vectors used for maintaining the sketches are reset when some anomalies are detected or some predefined period expires. Other proposed stream-based solution based on sketches (Gao et al., 2006; Krishnamurthy, Sen, Zhang, & Chen, 2003) or hash buckets (Kompella, Singh, & Varghese, 2007; Zhao, Xu, & Kumar, 2006) exhibit the same limitations.

Sketches are also used by Anceaume and Busnel (2014) to estimate the KL-divergence between two streams. The KL-divergence metric can be employed to detect threats by comparing current traffic with known legitimate traffic. The evaluation conducted by the authors shows an accurate approximation of the proposed sketch-based KL-divergence value to the exact divergence value. Moreover, authors present a distributed version of the algorithm to compute the global KL-divergence among a set of l distributed data streams with respect to the expected one. The authors define global KL-divergence as the sum of the stream-to-stream KL-divergence. However, in the presence of a potential DDoS attack in which the malicious traffic is distributed along the l possible streams, a low stream-to-stream KL-divergence would result in undetected threats. In addition, the proposed algorithm is incremental, not allowing for deletions but rather considering all the traffic history despite the major shift in the sources behavior that could happen during time (the latter being taken into account by *STONE*).

8.2. DDoS mitigation methods

Differently from detection, less solutions have been proposed about mitigation of DDoS attacks. The mitigation technique presented by Chen, Tang, and Du (2007) proposes filtering rules that can be deployed at firewalls. Nevertheless, it assumes that a given detection scheme exists and is able to specify which portion of the traffic should be filtered out. By assuming this, it overlooks a fundamental motivation and challenge behind *STONE*, the joint detection and mitigation of DDoS attacks. At the same time, the proposed solution differs from *STONE* since it does not discuss whether it can be leveraged for protection of threats such as flash crowds and also because its evaluation is mainly based on simulations.

A joint detection and mitigation framework for DDoS attacks is presented by Zhou, Jia, Wen, Xiang, and Zhou (2014). Nonetheless, the proposed scheme is only valid for application layer attacks. *STONE*, on the other hand, can protect from other attacks (such as SYN flood attacks) too. At the same time, Zhou et al. (2014) provide distributed but not parallel analysis. That is, it can be deployed at vantage points such as backbone links but treat the traffic of each link independently. Differently from this, *STONE* can maintain statistics about a target independently of the number of links through which its data travels to.

Detection schemes tailored to particular scenarios are also discussed in the literature. Despite the different focus between such schemes and *STONE*, it is still possible to note how *STONE* is able to overcome some of their limitations, as discussed in the following. Sahu and Pandey (2015) discuss a defense scheme for wireless sensor networks. In this case, the proposed detection and mitigation methods are based on monitoring of the frequency with which clients connect to a given entity (the possible target of a DDoS attack). As we showed in Section 4, frequency is an important feature but it is not enough by itself to detect different types of DDoS attacks.

Wang, Zheng, Lou, and Hou (2015b) and Yu, Tian, Guo, and Wu (2014) focus on DDoS defense systems in the context of Cloud Infrastructures. Wang et al. (2015b) discuss how to match DDoS defense systems and cloud infrastructures. In this direction, an interesting point of view about leveraging of Cloud infrastructures for DDoS defense is discussed by Yu et al. (2014). The authors claim that the key to success from the target perspective is to have access to computational power that exceeds the adversary's one. This mitigation method is legitimate for big Cloud infrastructures that have considerable resources to provision in order to exceed the ones of an attacker. Nevertheless, it is of limited utility for small Cloud infrastructures. At the same time, by increasing the computational power in response to a possible threat, such a mitigation scheme would also incur in extra costs. The appeal of Cloud infrastructures stems for their reduced maintenance and deployment costs for the users. In this sense, detection and mitigation schemes as the ones proposed by *STONE* could still be leveraged to discard suspicious traffic and thus reduce the computational power needed to communicate with legitimate clients.

8.3. DDoS defense based on expert systems

As we have already mentioned, the focus of expert systems has been mostly applied in anomaly detection rather than attack mitigation. Wang et al. (2010) present an intrusion detection system relying on two analysis steps. In the first step, a fuzzy algorithm is used to find groups of similar traffic. Then, an artificial neural network is build over each cluster to profile the traffic. The algorithm achieves good results in terms of precision and recall when compared with other methods such as decision trees, naive Bayes and back-propagation neural networks. However, the complexity of the algorithm might not allow for meeting near real-time constraints as needed; for example in the case study shown in the paper, the algorithm needs more than 35 minutes to build the profiles. Although the authors discuss directions to parallelize the algorithm, there is no evaluation focusing on how such system would address the high throughput and low latency analysis challenges addressed by *STONE*.

Choras, Saganowski, Renk, and Holubowicz (2012) present an anomaly detection framework based on signal-based features extraction and consider 15 features decomposing feature-signal with a matching pursuit. Signal decomposition is used to create profiles that are later used to detect anomalies. Although the proposed method focuses on anomaly detection in a broad sense, rather than on DDoS attacks concretely, the fifteen features taken into account seem to be useful for detection of threats such as TPC flood and UDP flood ones. Results show their effectiveness in terms of detection rate and false positive rate. However, authors do not present any results about

the time needed to process incoming traffic (to maintain profiles updated) or the time spent to detect and attack.

Kumar and Selvakumar (2012) present an anomaly detection system focused in high rate flooding DDoS attacks. Authors consider 6 metrics associated to the three flooding attacks (SYN, UPD and HTTP). Then, for each type of flooding attack, they extract profiles using a mixture of five machine learning algorithms (two based on decision trees, one based on a multi-layer perceptron, one based on k-nearest neighbor and one based on naive Bayes). As usual, profiles are the base for detecting anomalies. The evaluation shows detection rates and false positive rates over DDoS traces (as CAIDA dataset). Authors argue than the low number of features make the algorithm suitable for real time detection systems. However, when an attack happens, the volume of data increases quickly and the machine learning algorithms can become a bottleneck not only because of the number of features, but also because of the large volume of samples to analyze (as we discussed, this is one of the motivations behind *STONE*'s adoption of the data streaming processing paradigm).

9. Conclusions and future work

We presented *STONE*, a framework with expert system functionality that provides effective online defense from DDoS attacks. *STONE* is composed by two main modules, the Detection Control Center, in charge of detecting traffic anomalies, and the Mitigation Center, in charge of filtering the network traffic and preventing an attack from saturating the communication and computational resources of a target. The novelty of *STONE* lies in its joint online detection and mitigation schemes. This is enabled by the information shared by *STONE*'s modules, leveraged to both detect and mitigate threats. It is also enabled by its parallel-distributed streaming analysis, which allows for fast detection and effective mitigation of traffic volumes that can be observed at vantage points such as backbone links.

STONE differs from other expert systems proposed in the literature since the latter usually focus on the detection facet of DDoS attacks, but overlook their mitigation, as discussed in Sections 1 and 8 . To the best of our knowledge, only Zhang et al. (2004) discussed the features of an expert system focusing both on DDoS detection and mitigation. Nevertheless, the authors focused on the interaction between the detection and mitigation phases, but did not propose any specific implementation. Differently, *STONE* introduces a complete framework, designed and developed to process large volumes of data in an online fashion.

As future work, the detection and mitigation techniques proposed by *STONE* could be tested once deployed in real-word vantage points. This would give useful insights on the latency overhead introduced by the geographically distributed analysis run by *STONE*. At the same time, it would also allow for a deeper study, complementary of work such as (Wang et al., 2015a), of the types of DDoS attacks observed on a daily basis and the ratio between true and false positive alarms. We believe it could also be of interest to study how the information maintained and used by other expert systems to detect DDoS attacks could be leveraged to complement *STONE*'s mitigation schemes. In such a scenario, it would be challenging to study how to maintain such information in an online fashion and how to share it with *STONE*'s mitigation center.

Based on our experience with *STONE*, we believe future directions can be also delineated for novel expert systems focusing on other data analysis scenarios. The stream processing paradigm has been successfully leveraged in scenarios of interest for expert systems, such as financial markets (Tripathi & Pavaskar, Nov. 2012), cyber-physical systems (Gulisano, Almgren, & Papatriantafilou, 2014a; 2014b), vehicular systems (Gulisano, Nikolakopoulos, Walulya, Papatriantafilou, & Tsigas, 2015) and health care and patient monitoring (Andrade, Morgan, & Turaga, 2014). In this sense, it would be interesting to explore how to leverage the data streaming paradigm in expert systems focusing on the mentioned domains. Orthogonality, but still complementary to this direction, it would also be of interest to study the adoption and tailoring of streaming-based expert systems in conjunction with cloud infrastructures and lambda architectures. Cloud infrastructures are of particular interest for real-world use cases (Wang et al., 2015b; Yu et al., 2014) because of their reduced deployment and maintenance costs. Nonetheless, as discussed by Gulisano et al. (2012), proper leveraging of Cloud infrastructures depends on the design and implementation of features such as load balancing and elasticity, which would depend on the specific expert system taken into account. Lambda architectures such as Zaharia, Chowdhury, Franklin, Shenker, and Stoica (2010), which allow for store-then-process and streaming paradigms to co-exist, would also be of interest from an expert system perspective. Such architectures would allow for offline expert systems (or systems leveraging the traditional store-then-process paradigm) to be leveraged in conjunction with streaming based ones. At the same time, they could also ease the transition of certain functionality from the former to the latter. This transition delineates a future direction itself, since it would require new effective ways for porting the functionality and for sharing the information maintained by existing store-then-process expert systems (e.g., when coupling the detection method of a store-then-process based expert system with a novel streaming-based mitigation scheme).

Acknowledgments

This work has been partially funded by the SysSec European Union Seventh Framework Programme (FP7/2007-2013) under Grant agreement number 257007, the Research grant for project "Robust and secure communication", by the Swedish Civil Contingencies Agency (MSB), by the Spanish Research Council (MICCIN) under project BigDataPaaS (TIN2013-46883), and by the Regional Government of Madrid (CM) under project Cloud4BigData (S2013/ICE-2894) cofunded by FSE and FEDER and CoherentPaaS project funded by the European Commission under cContract FP7-611068.

Ricardo Jiménez-Peris and Marta Patiño-Martínez filed a patent at USPTO with number US13/112,628 related to this paper. We thank Farnaz Moradi and Wolfgang John for their support with the data traces collection.

References

Aiello, W., Gilbert, A., Rexroad, B., & Sekar, V. (2005). Sparse approximations for high fidelity compression of network traffic data. In *Proceedings of the 5th ACM SIGCOMM workshop on internet measurment (IMW'05)* (p. 22).

Anceaume, E., & Busnel, Y. (2014). A distributed information divergence estimation over data streams. *IEEE Transactions on Parallel and Distributed Systems, 25*(2), 478–487.

Andrade, H. C. M., Morgan, J. P., & Turaga, D. S. (2014). *Fundamentals of stream processing. application design, systems, and analytics.* Cambridge University Press.

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

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2015). An empirical evaluation of information metrics for low-rate and high-rate DDoS attack detection. *Pattern Recognition Letters, 51*, 1–7.

Broder, A. Z., & Mitzenmacher, M. (2003). Network applications of bloom filters: a survey. *Internet Mathematics, 1*(4), 485–509.

Callau-Zori, M., Jiménez-Peris, R., Gulisano, V., Papatriantafilou, M., Fu, Z., & Patiño-Martínez, M. (2013). STONE: a stream-based DDoS defense framework. In *Proceedings of the 28th annual ACM symposium on applied computing (SAC'13)* (pp. 807–812).

CERT Coordination Center (1997). CERT Advisory CA-1996-26 Denial-of-Service attack via ping. http://www.cert.org/advisories/CA-1996-26.html. Accessed 11.07.15.

CERT Coordination Center (1997). CERT Advisory CA-1997-28 IP Denial-of-Service attacks. http://www.cert.org/advisories/CA-1997-28.html. Accessed 11.07.15.

Chen, S., Tang, Y., & Du, W. (2007). Stateful DDoS attacks and targeted filtering. *Journal Network and Computer Applications, 30*(3), 823–840.

Chin, G., Jr., Choudhury, S., Feo, J., & Holder, L. (2014). Predicting and detecting emerging cyberattack patterns using StreamWorks. In *Proceedings of the 9th annual cyber and information security research conference* (pp. 93–96).

Choras, M., Saganowski, L., Renk, R., & Holubowicz, W. (2012). Statistical and signal-based network traffic recognition for anomaly detection. *Expert Systems, 29*(3), 232–245.

Datar, M., Gionis, A., Indyk, P., & Motwani, R. (2002). Maintaining stream statistics over sliding windows (extended abstract). In *Proceedings of the 13th annual ACM–SIAM symposium on discrete algorithms (SODA'02)* (pp. 635–644).

Dean, D., Franklin, M., & Stubblefield, A. (2002). An algebraic approach to IP traceback. *ACM Transactions on Information and System Security, 5*(2), 119–137.

Eddy, W. M. (2007). RFC 4987. TCP SYN flooding attacks and common mitigations. http://tools.ietf.org/html/rfc4987. Accessed 11.07.15.

Fu, Z., Papatriantafilou, M., & Tsigas, P. (2011). CluB: a cluster based framework for mitigating distributed denial of service attacks. In *Proceedings of the 26th ACM symposium on applied computing (SAC, 2011)* (pp. 520–527).

Ganguly, S., Garofalakis, M. N., Rastogi, R., & Sabnani, K. K. (2007). Streaming algorithms for robust, real-time detection of DDoS attacks. In *Proceedings of the 27th IEEE international conference on distributed computing systems (ICDCS'07)* (p. 4).

Gao, Y., Li, Z., & Chen, Y. (2006). A DoS resilient flow-level intrusion detection approach for high-speed networks. In *Proceedings of the 26th IEEE international conference on distributed computing systems (ICDCS'06)* (p. 39).

Gulisano, V., Almgren, M., & Papatriantafilou, M. (2014a). METIS: A Two-tier Intrusion Detection System for Advanced Metering Infrastructures. In *Proceedings of the 5th international conference on future energy systems (e-Energy'14)* (pp. 211–212). ACM.

Gulisano, V., Almgren, M., & Papatriantafilou, M. (2014b). Online and scalable data validation in advanced metering infrastructures. In *Proceedings of the IEEE PES Innovative smart grid technologies conference Europe (ISGT-Europe)* (pp. 1–6).

Gulisano, V., Jiménez-Peris, R., Patiño-Martínez, M., Soriente, C., & Valduriez, P. (2012). StreamCloud: an elastic and scalable data streaming system. *IEEE Transactions on Parallel and Distributed Systems, 23*(12).

Gulisano, V., Jiménez-Peris, R., Patiño-Martínez, M., & Valduriez, P. (2010). StreamCloud: a large scale data streaming system. In *Proceedings of the 30th IEEE international conference on distributed computing systems (ICDCS'10)* (pp. 126–137).

Gulisano, V., Nikolakopoulos, Y., Walulya, I., Papatriantafilou, M., & Tsigas, P. (2015). DEBS grand challenge: deterministic real-time analytics of geospatial data streams through scalegate objects. In *Proceedings of the 9th ACM international conference on distributed event-based systems (DEBS 2015)* (pp. 316–317).

Hick, P., Aben, E. claffy, k. (2007). The CAIDA "DDoS attack 2007" dataset. http://www.caida.org/data/passive/ddos-20070804_dataset.xml. Accessed 11.07.15.

Kim, M.-S., Won, Y. J., & Hong, J. W. (2006a). Characteristic analysis of Internet traffic from the perspective of flows. *Computer Communications, 29*(10), 1639–1652.

Kim, Y., Lau, W. C., Chuah, M. C., & Chao, H. J. (2006b). PacketScore: a statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE Transactions on Dependable and Secure Computing, 3*(2), 141–155.

Kompella, R. R., Singh, S., & Varghese, G. (2007). On scalable attack detection in the network. *IEEE/ACM Transactions on Networking, 15*(1), 14–25.

Krishnamurthy, B., Sen, S., Zhang, Y., & Chen, Y. (2003). Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3th ACM SIGCOMM workshop on internet measurment (IMW'03)* (pp. 234–247).

Kumar, P. A. R., & Selvakumar, S. (2012). M2KMIX: identifying the type of high rate flooding attacks using a mixture of expert systems. *International Journal of Computer Network and Information Security, 4*(1), 1–16.

Lakhina, A., Crovella, M., & Diot, C. (2005). Mining anomalies using traffic feature distributions. In *Proceedings of the ACM SIGCOMM 2005 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM'05)* (pp. 217–228).

Lakhina, A., Papagiannaki, K., Crovella, M., Diot, C., Kolaczyk, E. D., & Taft, N. (2004). Structural analysis of network traffic flows. In *Proceedings of the international conference on measurements and modeling of computer systems (SIGMETRICS'04)* (pp. 61–72).

Lee, K., Kim, J., Kwon, K. H., Han, Y., & Kim, S. (2008). DDoS attack detection method using cluster analysis. *Expert Systems With Applications, 34*(3), 1659–1665.

Lin, S., & Tseng, S. (2004). Constructing detection knowledge for DDoS intrusion tolerance. *Expert Systems With Applications, 27*(3), 379–390.

Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review, 34*(2), 39–53.

Moradi, F., Almgren, M., John, W., Olovsson, T., & Philippas, T. (2011). On collection of large-scale multi-purpose datasets on internet backbone links. In *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security* (pp. 62–69).

Peng, T., Leckie, C., & Ramamohanarao, K. (2007). Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computer Survey, 39*(1).

Rice, J. A. (2001). *Mathematical statistics and data analysis*. Duxbury Press.

Roughan, M., Greenberg, A., Kalmanek, C., Rumsewicz, M., Yates, J., & Zhang, Y. (2002). Experience in measuring backbone traffic variability: models, metrics, measurements and meaning. In *Proceedings of the 2nd ACM SIGCOMM workshop on internet measurment (IMW'02)* (pp. 91–92).

Sahu, S. S., & Pandey, M. (2015). A probabilistic packet filtering-based approach for distributed denial of service attack in wireless sensor network. In L. C. Jain, S. Patnaik, & I. Nikhil (Eds.). In *Intelligent computing, communication and devices: 309* (pp. 65–70). Springer India. doi:10.1007/978-81-322-2009-1_8.

Savage, S., Wetherall, D., Karlin, A., & Anderson, T. (2000). Practical network support for IP traceback. In *Proceedings of the ACM SIGCOMM 2000 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM '00): Vol. 30* (pp. 295–306).

Sekar, V., Duffield, N., Spatscheck, O., van der Merwe, J., & Zhang, H. (2006). LADS: large-scale automated DDOS detection system. In *Proceedings of the annual conference on USENIX'06 annual technical conference* (p. 16).

Shiaeles, S. N., Katos, V., Karakos, A. S., & Papadopoulos, B. K. (2012). Real time DDoS detection using fuzzy estimators. *Computers & Security, 31*(6), 782–790.

Song, D. X., & Perrig, A. (2001). Advanced and authenticated marking schemes for IP traceback. In *Proceedings IEEE INFOCOM 2001, the conference on computer communications, twentieth annual joint conference of the IEEE computer and communications societies: vol. 2* (pp. 878–886).

Su, M. (2010). Discovery and prevention of attack episodes by frequent episodes mining and finite state machines. *Journal Network and Computer Applications, 33*(2), 156–167.

Tan, Z., Jamdagni, A., He, X., Nanda, P., & Liu, R. P. (2014). A system for denial-of-service attack detection based on multivariate correlation analysis. *IEEE Transactions on Parallel and Distributed Systems, 25*(2), 447–456.

The NORDUnet IP and MPLS network (2012). https://www.nordu.net/content/nordunet-ip-and-mpls-network-0. Accessed 11.07.15.

The Swedish University Computer Network OptoSUNET (2012). http://basun.sunet.se/engelska.html. Accessed 11.07.15.

Tripathi, K. K., & Pavaskar, M. A. (2012). Survey on credit card fraud detection methods. *International Journal of Emerging Technology and Advanced Engineering, 2*, 721–726.

Tsang, C., Kwong, S., & Wang, H. (2007). Genetic-fuzzy rule mining approach and evaluation of feature selection techniques for anomaly intrusion detection. *Pattern Recognition, 40*(9), 2373–2391.

Wang, A., Mohaisen, A., Chang, W., & Chen, S. (2015a). Capturing DDoS attack dynamics behind the scenes. In *Proceedings of the 12th international conference on detection of intrusions and malware, and vulnerability assessment (DIMVA, 2015)* (pp. 1–20).

Wang, B., Zheng, Y., Lou, W., & Hou, Y. T. (2015b). DDoS attack protection in the era of cloud computing and software-defined networking. *Computer Networks, 81*, 308–319.

Wang, G., Hao, J., Ma, J., & Huang, L. (2010). A new approach to intrusion detection using artificial neural networks and fuzzy clustering. *Expert Systems With Applications, 37*(9), 6225–6232.

Xu, K., Zhang, Z.-L., & Bhattacharyya, S. (2005). Profiling internet backbone traffic: behavior models and applications. In *Proceedings of the ACM SIGCOMM 2005 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM'05)* (pp. 169–180).

Yu, S., Tian, Y., Guo, S., & Wu, D. O. (2014). Can we bbeat DDoS attacks in clouds? *IEEE Transactions on Parallel and Distributed Systems, 25*(9), 2245–2254.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: cluster computing with working sets. In *Proceedings of the 2nd usenix conference on hot topics in cloud computing (HOTCLOUD'10)* (p. 10). https://spark.apache.org/. Accessed 11.07.15

Zhang, G., Li, J., & Gu, G. (2004). Research on defending DDoS attack – an expert system approach. In *Proceedings of the IEEE international conference on systems, man & cybernetics* (pp. 3554–3558).

Zhang, L. (1990). Virtual clock: a new traffic control algorithm for packet switching networks. In *Proceedings of the ACM symposium on communications architectures & protocols (SIGCOMM '90)* (pp. 19–29).

Zhao, Q., Xu, J., & Kumar, A. (2006). Detection of super sources and destinations in high-speed networks: algorithms, analysis and evaluation. *IEEE Journal on Selected Areas in Communications, 24*(10), 1840–1852.

Zhou, W., Jia, W., Wen, S., Xiang, Y., & Zhou, W. (2014). Detection and defense of application-layer DDoS attacks in backbone web traffic. *Future Generation Computer Systems, 38*, 36–46.