An Unsupervised Approach for Online Detection and Mitigation of High-Rate DDoS Attacks Based on an In-Memory Distributed Graph Using Streaming Data and Analytics

J. J. Villalobos, Ivan Rodero, Manish Parashar
Rutgers Discovery Informatics Institute
Rutgers University, The State University of New Jersey
Piscataway, New Jersey, USA
{jj.villalobos,irodero,parashar}@rutgers.edu

ABSTRACT

A Distributed Denial of Service (DDoS) attack is an attempt to make an online service, a network, or even an entire organization, unavailable by saturating it with traffic from multiple sources. DDoS attacks are among the most common and most devastating threats that network defenders have to watch out for. DDoS attacks are becoming bigger, more frequent, and more sophisticated. Volumetric attacks are the most common types of DDoS attacks. A DDoS attack is considered volumetric, or high-rate, when within a short period of time it generates a large amount of packets or a high volume of traffic. High-rate attacks are well-known and have received much attention in the past decade; however, despite several detection and mitigation strategies have been designed and implemented, high-rate attacks are still halting the normal operation of information technology infrastructures across the Internet when the protection mechanisms are not able to cope with the aggregated capacity that the perpetrators have put together. With this in mind, the present paper aims to propose and test a distributed and collaborative architecture for online high-rate DDoS attack detection and mitigation based on an in-memory distributed graph data structure and unsupervised machine learning algorithms that leverage real-time streaming data and analytics. We have successfully tested our proposed mechanism using a real-world DDoS attack dataset at its original rate in pursuance of reproducing the conditions of an actual large scale attack.

CCS CONCEPTS

• **Security and privacy** → **Denial-of-service attacks**; • **Computing methodologies** → **Machine learning**; • **Information systems** → *Information systems applications*; • **Computer systems organization** → *Distributed architectures*;

KEYWORDS

DDoS Detection, DDoS Mitigation, Machine Learning, Distributed, Big Data, Analytics

BDCAT'17, December 5–8, 2017, Austin, TX, USA

ACM ISBN 978-1-4503-5549-0/17/12…$15.00
https://doi.org/10.1145/3148055.3148077

ACM Reference Format:
J. J. Villalobos, Ivan Rodero, Manish Parashar. 2017. An Unsupervised Approach for Online Detection and Mitigation of High-Rate DDoS Attacks Based on an In-Memory Distributed Graph Using Streaming Data and Analytics. In *Proceedings of BDCAT'17*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3148055.3148077

1 INTRODUCTION

The distributed nature of a DDoS attack makes it significantly more powerful, as well as more difficult to detect and block its source. DDoS attacks are coordinated, launched using a large number of hosts that have been compromised at an earlier stage, most commonly by means of spreading malware. Depending on the intensity of attack packets, the traffic volume and the number of hosts used to attack, the consequences can be catastrohpic. If the perpetrators are able to arrange a large number of compromised hosts, an entire network may be disrupted within a very short period of time, and that is what we classify as a high-rate, or volumetric attack.

If 2016 was the year of DDoS with major disruptions in terms of technology, attack scale and impact on our daily life, now that Internet of Things (IoT) has reached critical mass, millions of IoT devices can be leveraged to coordinate colossal attacks [17] [16] [21]. According to the Worldwide Infrastructure Security Report [23], the largest attack reported by a respondent in 2016 was 500 Gbps, with others reporting attacks of 450 Gbps, 425 Gbps and 337 Gbps. The trend of significant growth in the top-end size of DDoS attacks continues year-over-year.

Increased interest in DDoS detection and mitigation services continues [14], online detection mechanisms have the potential to solve the difficult problem of preventing, detecting and mitigating DDoS attacks. Many studies have been published on data mining and machine learning techniques such as classification or clustering, however, for these mechanisms to be really effective in detecting high-rate DDoS attacks, they have to be distributed, unsupervised, capable of scaling out linearly and as close as possible to the edge of the network in order to let detection happen early and in a timely manner.

NetFlow analyzers remain the most effective and the most commonly deployed way of detecting threats. NetFlow technology is a prevalent IP traffic analysis and measurement standard in the Internet that enables supported devices or applications to collect IP traffic statistics on their interfaces and to expose them as NetFlow records towards one or more NetFlow collectors for analysis. The flow information contains information such as source IP address,

destination IP address, source port, destination port, number of packets in the flow, size of the flow in octets, protocol number, protocol flags, duration of the flow, type of service, etc. Our proposed mechanism is based on the unsupervised online analysis of NetFlow data coming from multiple sources and analyzed via a distributed in-memory graph that stores a holistic representation of the state of the participating networks in near real-time. Provided that in this context the size and velocity of the data are massive, the design, development and implementation pose several challenges such as a timely and continuous analysis, an efficient use of memory and processors, a high degree of portability in terms of technology and the robustness of the entire solution.

Our contribution in this paper is twofold: (i) a technology independent big data and analytics architecture for online volumetric DDoS attack detection that can be deployed not only in a distributed fashion due to its scalable and distributed design, but also in a local environment for research-based testbeds and (ii) a distributed shared-nothing in-memory graph data structure that holds a sliding window view of the entire network in such a way that is optimal for the application of streaming machine learning techniques at scale.

The organization of the remaining part of the paper is as follows. Section 2 discusses the background and related work. Section 3 describes the architecture and data structures. Section 4 presents our experimental evaluation. Our conclusion and directions of future work are provided in section 5.

2 BACKGROUND AND RELATED WORK

Chen et al [31] presented a distributed approach to detect DDoS flooding attacks based on traffic fluctuations at Internet routers or at gateways of edge networks. They approached the challenge by monitoring the traffic at the superflow level to detect abrupt traffic changes across multiple network domains at the earliest time, a superflow contains all the packets destined for the same network domains from all possible source IP addresses. Berral et al [19] proposed a mechanism based on an overlay network whose nodes are equipped with detection and classification capabilities, nodes exchange gossiping about possible threats and warnings about declared threats. The chosen nodes must be key nodes like backbone routers, firewalls, etc. and in the extreme case all the routing nodes from the network should be chosen, unfeasible in practice though. Moreover, all traffic towards a node, including legitimate traffic, would be blocked when the mechanism detects an attack. Zeyu et al [30] and Han et al [12] proposed a collaborative DDoS detection mechanism based on traffic classification that required training in order to be effective. Our proposed model overcomes many of these limitations; scales out linearly, its deployment model can be adapted to any possible scenario, and it has been tested with real-world datasets.

Nguyen et al [24] conducted a comprehensive survey which has served as the base for many other researches, they presented machine learning (ML) applications to IP traffic classification and discussed a number of key requirements for the employment of ML-based traffic classifiers in operational IP networks. Kato et al [18] utilized ML techniques to study the patterns of DDoS attacks and detect them. Among others, the features extracted included source IP address, time interval in seconds between packets and packet size in bytes from the dataset. The experimental evaluation consisted in training a Support Vector Machine (SVM) and testing the detection system using a testing dataset. Robinson et al [28] conducted an experimental evaluation to rank ten different supervised ML algorithms. The experiment was divided into several phases: packet header parsing, feature extraction, normalization, classification and evaluation of metrics, and ranking of algorithms. Purnawansyah et al [25] implemented and explored K-Means [27] as a clustering algorithm for bandwidth usage. The results showed that the K-Means method can perform clustering with 3 and 4 clusters and that could be a recommendation on bandwidth management for network administrators in order to plan, share, and control bandwidth. Wayan et al [26] presented a modified K-Means algorithm using timestamp initialization and showed that it can eliminate the determination of K-cluster that affects detection rate and false positive rate when using different K-cluster. Their research also used a windowing technique to obtain a more efficient process to detect anomalous traffic. Raimir et al [6] work focused on the stage of short-term traffic prediction using Principal Components Analysis (PCA) as a technique for dimensionality reduction and a Local Linear Model based on K-Means as a technique for prediction and trend analysis. The results validated with data on a real network presented a satisfactory margin of error for use in practical situations. More recently, Taimur et al [3] proposed a two-phased ML classification mechanism using NetFlow as input data. The individual flow classes are derived per application through K-Means and are further used to train a C5.0 decision tree classifier. As part of the validation, the initial unsupervised phase used flow records of fifteen popular Internet applications that were collected and independently subjected to K-Means clustering to determine unique flow classes generated per application. Our work leverages the well-known K-Means ML algorithm, however, it does so in an online fashion, on top of a distribution processing engine, and with real Internet traffic datasets.

Do Quoc Le et al [20] proposed a novel approach to detect anomalous network traffic based on graph theory concepts such as degree distribution or maximum degree. McGregor et al [1] combined existing data stream techniques with ideas from approximation algorithms and graph theory. Zhang et al [32] proposed a sliding window graph model (SWG) from the perspective of complex network, to study changes of interactions of end-hosts in a day for four applications. Crouch et al [5] presented an extensive set of positive results including algorithms for constructing basic graph synopses like combinatorial sparsifiers and spanners as well as approximating classic graph properties such as the size of a graph matching or minimum spanning tree. Our research differentiates from previous work in terms of how the graph models the network through a split source-destination view, how the graph is partitioned between multiple threads, and how graph traversal is minimized with the help of an in-thread simple cache.

In the recent years there has been growing interest in the application of big data and analytics [4, 13, 22] to the field of DDoS, and actually, existing open-source architectures [9, 15] for online monitoring and analysis have proven to work extremely well; however, they seem to be focused on monitoring multi-gigabit links, unlike our proposed architecture whose principal advantage is its

distributed foundation that theoretically let us decompose the network beyond the edge, and hence, lets us in into the field of IoT defense for DDoS attacks.

3 ARCHITECTURE

The proposed architecture is open, distributed and scalable. As illustrated in Figure 1, the architecture is composed of one or more core nodes, one or more edge nodes and one or more external agents. On the one hand, an edge node is coupled with one or more core nodes and one or more external agents and it has the following responsibilities:

- Continuous reception of NetFlows from its associated external agents and continuous streaming of NetFlows to its main core node.
- Listening and reacting to the command and control packets sent from its active core node.

On the other hand, a core node is connected to at least an edge node and shall be coupled with other core nodes as well. A core node is responsible for:

- Online parsing, processing and aggregation of NetFlow data sent by each one of the associated edge nodes.
- Processing, aggregation and consolidation of information shared with other core nodes.
- Global aggregation of multiple produced datasets.
- Application of ML algorithms.
- Decision making and command and control of its edge node(s).

The communication channels between the edge nodes and the core nodes for the NetFlow data transfer and for the command and control interface, and between the core nodes for the inter-core data sharing must happen via an overlay network that guarantees a minimum throughput and low latency in order to avoid being impacted by the conditions of the network at any given time. This could be achieved by means of Virtual Private Network (VPN) and Quality of Service configurations (QoS).

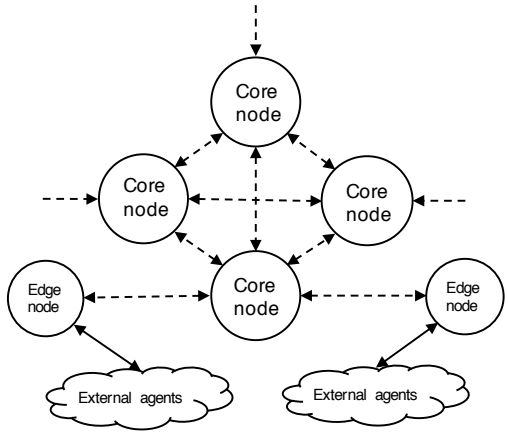


Figure 1: Architecture overview

3.1 Edge nodes

Edge nodes are thin components whose provisioning does not need much memory or processing power. The ideal location for edge nodes is close to the networked components they are receiving data from, such as at the edge of the networks or at intermediate network nodes (e.g. a point-of-presence), where the integration with the existing network equipment is more secure and where the latency is minimal; however, an edge node can be located anywhere as long as there is a reliable communication path between the external agents, the edge node and the core node involved in the pipeline. Figure 2 details the two subcomponents that run within an edge node:

- Edge Ingestion Engine (EIE) : Implements the NetFlow data collection process and the streaming mechanism that is in charge of shipping the collected data towards a core node.
- Edge Reaction Engine (ERE) : Implements actions based on the command and control packets received from its active core node.

The implementation of the ERE actions is open and flexible, e.g. block or rate-limit traffic to or from a specific service via the Application Programming Interface (API) of the involved devices. The definition of these actions is crucial for the attack mitigation endeavor.

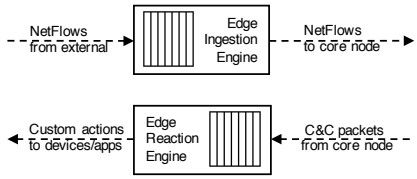


Figure 2: Edge node

3.2 Core nodes

Unlike edge nodes, core nodes are thick components, the heavy lifting is conducted on the core nodes. The core nodes implementation is not bound to the provisioning model, it can be a cluster of bare metal machines, virtual machines or even a pool of containers, actually a core node could be provisioned across geographically distributed locations for scalability and redundancy reasons.

As shown in Figure 3, a core node has the following subcomponents:

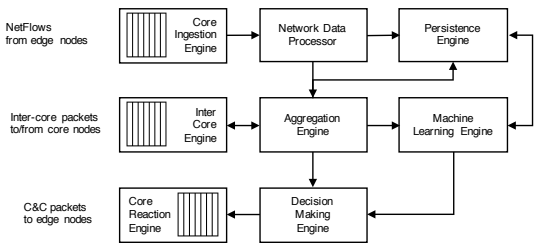


Figure 3: Core node

- Core Ingestion Engine (CIE) : Implements the core ingestion queues where the EIE on the edges nodes publish the NetFlows to. It also implements the NetFlow partitioning algorithm.
- Network Data Processor (NDP) : Implements an in-memory sharded graph data structure and is responsible for feature extraction.
- Aggregation Engine (AGE) : Aggregates the data received from the multiple NDR processes.
- Machine Learning Engine (MLE) : Implements machine learning algorithms and runs them on the aggregated data.
- Decision Making Engine (DME): Decisions are made based on the input from the AGE and the MLE.
- Inter Core Engine (ICE) : Implements the communication protocol for inter-core information sharing.
- Core Reaction Engine (CRE) : Implements the edge nodes command and control based on the input from the DME.
- Persistence Engine (PTE) : Data consolidation for offline and forensics analysis.

3.3 External agents

An external agent is just any entity capable of sending NetFlows to a log collector, for example, edge routers, NetFlow exporters, specific applications or even home gateways, which are emerging as a key element of bringing legacy and next-gen devices to the Internet of Things (IoT).

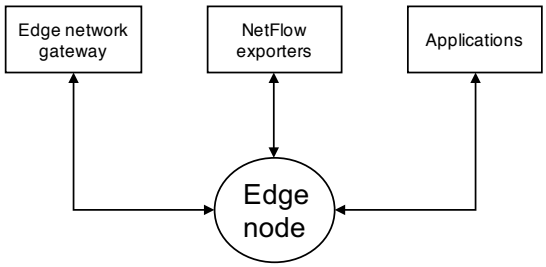


Figure 4: External agents

3.4 Data structure

One of the main aspects of our work is the data structure that holds the state of the network and how this data structure is continuously updated and analyzed. The state of the entire network is stored on an in-memory shared-nothing sharded directed multidigraph whose analysis is based on a sliding window mechanism. Despite the sliding window model has become a popular model for processing infinite data streams [1] [32] [5] [2] and plenty of research has been conducted around the area of graph algorithms, our work focuses on avoiding graph traversal as much as possible by increasing data locality and caching, otherwise the processing capacity would be prohibitive at the speed that the data flows through the pipeline. This data structure is simple, yet powerful, it is technology agnostic, allowing us to adapt it to virtually any distribution processing engine.

The NDP is a multi-threaded process where every thread has a partial fragment of the whole graph that a core node manages. Even though a core node does not store other core nodes graphs, it can get partial graphs from other core nodes via the ICE.

Let there be n the number of NDP threads, i be the NDP thread identifier, the graph for a given NDP thread is given by the expression:

$$G_i = (V_i, E_i) \tag{1}$$

, where V_i is the set of vertices and E_i is the set of directed edges or ordered pairs of vertices; therefore, the global graph managed by the core node is described by the expression:

$$G = G_1 \cup ... \cup G_n \tag{2}$$

Each NetFlow ingested by the CIE goes through a partitioning algorithm that decides which NDP thread has to receive it, then once inside a NDP thread, the NetFlow is modeled as two vertex objects that represent each host in the NetFlow and one directed edge object that represents the properties of the NetFlow. The vertex objects and the edge objects are kept on a hash map and a linked hash map respectively.

Due to the cost of traversing the graph, especially when there is inter-process communication involved, we avoid it when not strictly necessary by increasing data locality using a combination of a partitioning algorithm, a double stream that lets us have different views for the source and destination traffic, and a features cache for every vertex that any given NDP thread is responsible for.

The sequence of NetFlows listed in Table 1 forms the graph represented in figure 5, however, after going through the CIE and the NDP, as shown in Figure 6, each one of the threads has a different fragment, the gray vertices are the vertices that are stored, along their caches, on that thread, the white vertices are just implicit references stored on the graph edges connecting them.

Table 1: NetFlow sequence example

Time	Source	Destination
t0	v1	v2
t1	v2	v1
t2	v1	v3
t3	v2	v4
t4	v2	v5
t5	v3	v2
t6	v3	v4

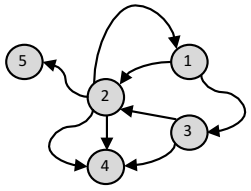


Figure 5: Complete graph

Table 2 describes which features are extracted and cached in-thread.

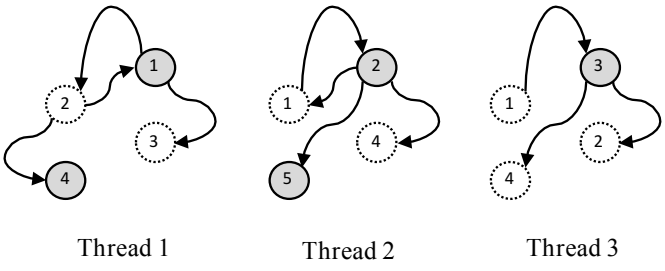


Figure 6: Distributed graph

On the event of the failure of a NDP process there is no graph rebalancing, another NDP process will take the place of the failed one populating its data structure from the last checkpoint available on the PTE. Edge eviction based on the window width happens at configurable regular intervals.

Table 2: Vertex cache

Feature	Description
inDegree	Count of incoming edges for the vertex
outDegree	Count of outgoing edges for the vertex
inPackets	Sum of packets for the vertex incoming edges
outPackets	Sum of packets for the vertex outgoing edges
inOctets	Sum of octets for the vertex incoming edges
outOctets	Sum of octets for the vertex outgoing edges

3.5 Machine Learning Engine

The Machine Learning Engine (MLE) is responsible for the execution of different machine learning algorithms on the extracted features received from the Aggregation Engine (AGE) where the aggregation stage has taken place, and for passing it over to the Decision Making Engine (DME).
As described in Table 2, six features have been selected as the foundation for the machine learning analysis, and the following data structures are built and normalized, in-flight, and for each network protocol, to serve as input data for the machine learning algorithms:

- One two-dimensional (2D) array per feature where the first dimension represents the inbound traffic, the second dimension represents the outbound traffic, and each point represents a vertex.
- One three-dimensional (3D) array per traffic direction where each dimension represents one feature, and each point represents a vertex.

The machine learning two-step process is composed of a quick and lightweight hint step and a machine learning algorithm execution step which is significantly more costly and therefore it is launched only when tipped off by the hint step:

3.5.1 Hint step. The two-dimensional arrays are continuously analyzed on a per-column basis in order to determine which protocols and ports have an anomalous distribution. Statistical metrics are calculated and a list of candidate tuples (protocol, port) is returned for the next step to process.

```
List<Prot> hintedProtocols = new ArrayList<Prot>();
for (Prot protocol : protocols) {
  for (Feature feature : protocol.features) {
    for (Direction direction : feature.directions) {
      if (hint(direction)) hintedProtocols.add(protocol);
    }
  }
}
```

Listing 1: Machine Learning Engine Hint step

3.5.2 ML step. This step takes in the list of candidate services generated during the hint step and applies machine learning algorithms.

```
for (Prot protocol : hintedProtocols) {
  for (Direction direction : protocol.directions) {
    machine_learning(algorithm, protocol, direction);
  }
}
```

Listing 2: Machine Learning Engine ML step

3.6 Online analysis pipeline

Network data, in the form of NetFlows, is fed into a core node via its Core Ingestion Engine (CIE) where it is decoded and via the partitioning algorithm sent to both the source and destination streams by hashing out the tuple *(IP address, protocol, port)* of both the source vertex v_1 and the destination vertex v_2, respectively. The reason for sending it to two different streams, and possibly ending up in two different NDP threads, lays in the fact that the graph sharding strategy is based upon the vertices, and a vertex can, and most likely does, have both incoming and outgoing edges.
Figure 7 illustrates how a NDP thread processes a parsed NetFlow. Every time a NDP thread receives a NetFlow, it adds a new edge $e_{1\text{-}2}$ to its local graph, updates the local cache for v_1 and v_2, and immediately passes the updated features *inDegree, outDegree, inPackets, outPackets, inOctets, outOctets* to the AGE. Updates are consolidated via the PTE in batches to minimize the latency of in-flight data. It is also possible that the NetFlow received is also persisted via the PTE for offline analysis, however the volume of data can be massive and therefore the retention period must be chosen with care. In any case, although the PTE is part of the architecture and it actually is a very important component, its internal details and its participation in the pipeline is out of the scope of this work and will be obviated from this point onwards.

The AGE role is crucial, as shown in Figure 8, the AGE performs the aggregation and forwards the aggregated data to the MLE and to the DME, which is the component in charge of signaling the CRE to assemble a reaction packet and to send it to the ERE. In turn, the ERE at the edge node will process the reaction packet and will perform the associated actions which can be virtually anything programmatically possible such as disabling a port, throttling an interface, changing traffic class priorities, updating routing tables, sending an API request to an application, etc. Once the action has

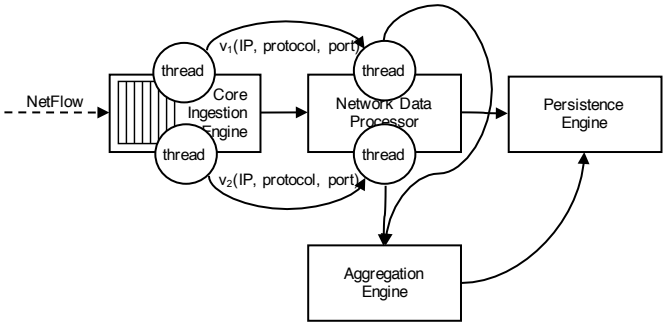


Figure 7: CIE, NDP, AGE, PTE

been executed, the ERE will assemble and send back to the CRE an acknowledgement reaction packet whose payload will contain information related to the processing of the reaction packet like for instance the exit code or the output, if any, of the action.

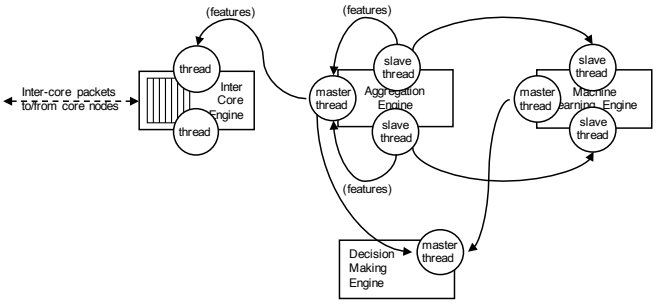


Figure 8: AGE, MLE, DME

4 EXPERIMENTAL EVALUATION

For our experimental evaluation we have implemented one edge node, one core node and several external agents that mimic network gateways by replaying real NetFlow traffic. The core node has been implemented on top of Apache Storm [11], a distributed processing engine, and Apache Kafka [7], a distributed message broker. In Storm, the structure of a distributed computation is referred to as a topology and is made up of streams of data, spouts (stream producers), and bolts (operations). Storm topologies are roughly analogous to jobs in batch processing systems such as Hadoop. However, while batch jobs have clearly defined beginning and end points, Storm topologies run forever, until explicitly killed or undeployed. Kafka lets us publish and subscribe to streams of records, lets us store streams of records in a fault-tolerant way, and lets us process streams of records as they occur. Our research scope does not consider the implementation of noither the ICE or the PTE, and the CRE is partially implemented because its role is not key for the contribution of our research.

We have leveraged spare resources on a Mesos [8] cluster where we have provisioned four Apache Kafka brokers and four Apache Storm supervisors with two workers per supervisor. All the processing is part of a Storm topology; the CIE is modeled as a series of spouts that harvest the ingestion queues and one bolt that takes care of the NetFlow decoding, the rest of the subcomponents have been implemented as bolts. The MLE has been implemented as a topology based on Apache SAMOA [10], where distributed streaming ML algorithms can be developed and executed.

We have run two experiments using a real-world DDoS attack dataset, an attack mostly based on the DNS protocol, a reflection and amplification DDoS attack captured at Merit's border router in SFPOP [29]. The edge node EIE subcomponent receives NetFlow streams from five different devices for one hour.

4.1 Experiments

Two experiments have been run with the same dataset but with different sliding window sizes. Window size was chosen arbitrarily in order to study the effects of different window sizes. One experiment was conducted using a one-minute window and another experiment was conducted using a ten-minute window, in both cases no sampling has been enabled, all the NetFlow data has been analyzed. It is important to mention that no prior training has been conducted, actually no training is needed at all given that our implementation is based on a two-step unsupervised approach, the first step (hint) being based on statistical metrics on each single dimension of the two-dimensional features arrays, and the second step (ML) being based on K-Means clustering.

The time slice of the dataset ingested starts with an ongoing attack and the DME instantly flags IP_v as an attacked node. All the IP addresses contained in the dataset are anonymized, the last 11 bits of source and destination IPs have been obfuscated with zeros, however for privacy reasons we are referring to each IP using a variable.

4.1.1 Hint step. The first step of the MLE adds the alleged victim to the candidate list for further analysis on all three features of the two-dimensional array. It can be observed that on the UDP/53 2D array hint step for the vertex degree feature, there is a suspicious imbalance for UDP:

```
2D array edges (sorted desc):
    IN  OUT   IP
996598 7088 <IP_v>
201652    0 <IP_a>
183815    0 <IP_b>
...
```

The hint step analysis on the UDP/53 2D array for the octets feature shows exactly the same pattern, the candidate is suspicious on this feature as well.

```
2D array packets (sorted desc):
        IN   OUT   IP
4266185242 41613 <IP_v>
   7289667   117 <IP_e>
   1906266     0 <IP_b>
...
```

The hint step analysis on the UDP/53 2D array for the packets feature shows yet again the same pattern, the candidate is suspicious on this feature as well.

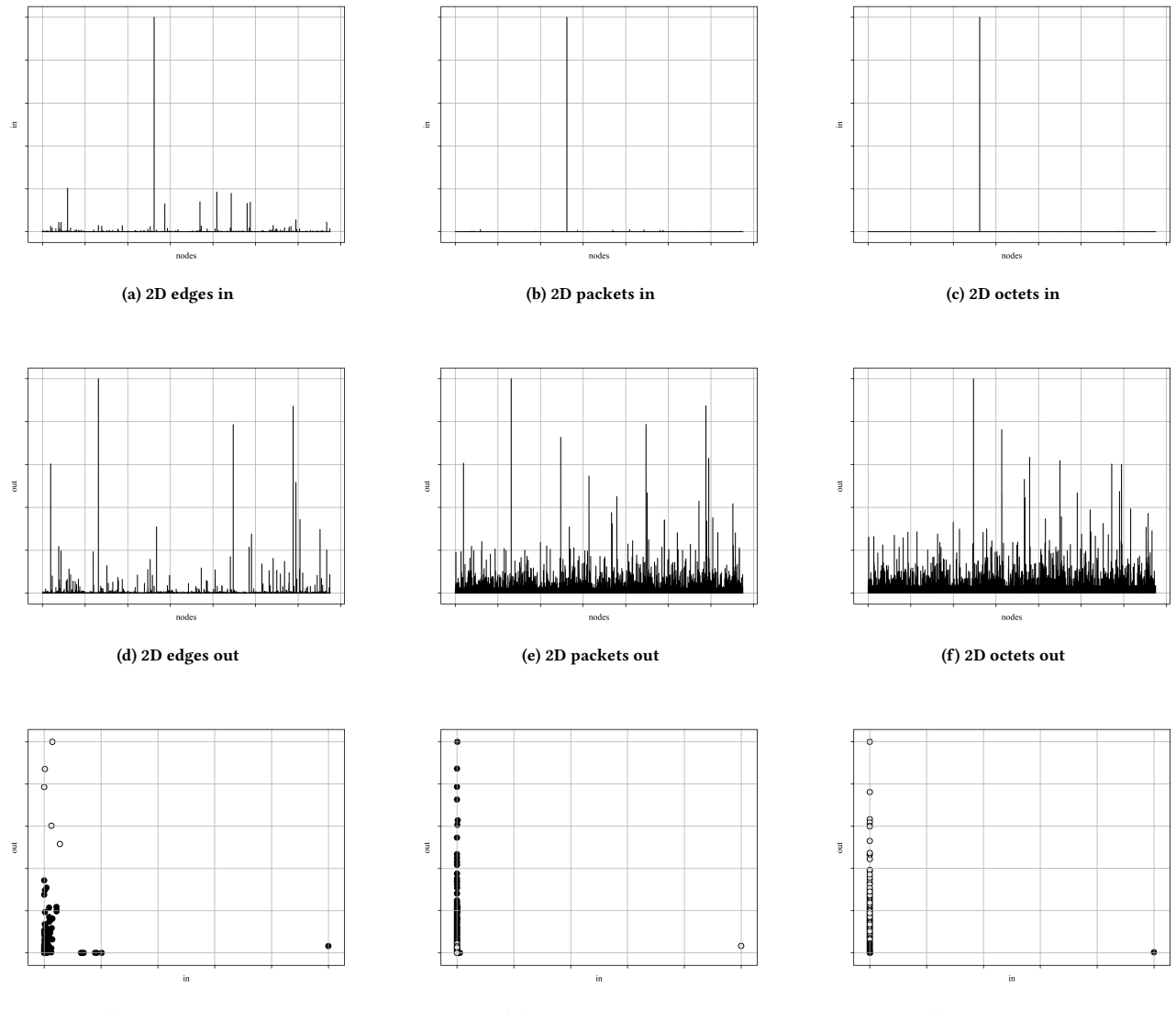
```
2D array packets (sorted desc):
     IN OUT   IP
3473270 530 <IP_v>
  23768   0 <IP_b>
  23083   0 <IP_a>
...
```

Our candidate has been strongly confirmed, three out of three features tagged the UDP/53 protocol, and thus the merge process passes it to the ML step, actually it would have been enough to score two out of three. Figures 9a, 9b, 9c, 9d, 9e, 9f, 9g, 9h, 9i report the visual representation of the Hint step when an attack is active. Figures 12a, 12b, 12c, 12d, 12e, 12f, 12g, 12h, 12i report the visual representation of the Hint step when there is no active attack.

4.1.2 ML step. Once the three-dimensional arrays are processed by the K-Means algorithm directed to the specific protocol and port provided by the hint step, then the candidate is confirmed as victim and the output of the MLE, a vertex key, is passed on to the DME which builds a query against the core node graph and the returned result is used to create the command and control packet to signals the edge core nodes. Figures 10a, 10b show the visual representation of the MLE and the DME when the hinted candidate is considered as confirmed, and therefore a DDoS attack is confirmed and detected. Figure 10a is conclusive, the inbound dimension of the three-dimensional array for the given protocol isolates one IP address in a cluster meaning that that is the victim.

```
3D array inbound:
  Cluster 0 ['...', '...', ...]
  Cluster 1 ['<IP_v>']
```

Our experimental setup implements an API that lets us send signals to interact with the in-flight data, internal buffers, etc. Figure

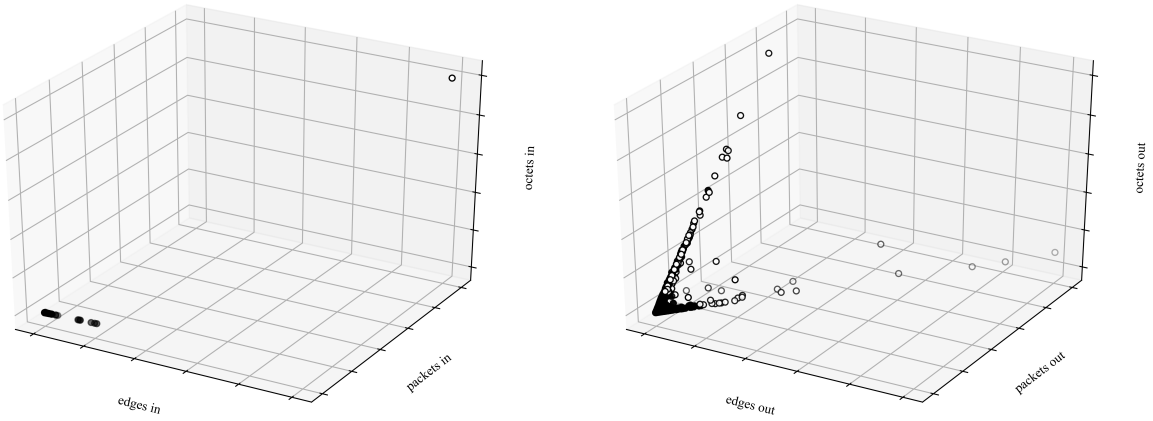


(a) 2D edges in **(b) 2D packets in** **(c) 2D octets in**

(d) 2D edges out **(e) 2D packets out** **(f) 2D octets out**

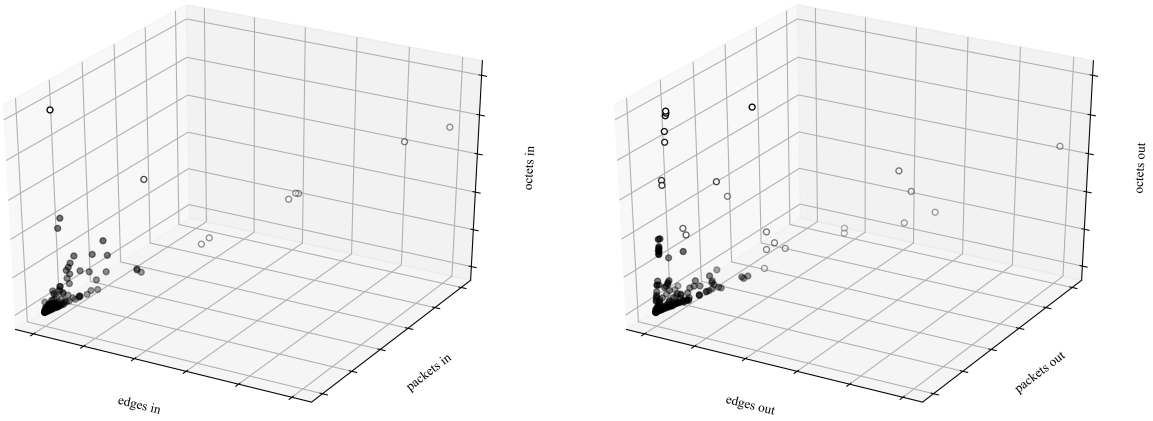
(g) 2D edges clusters **(h) 2D packets clusters** **(i) 2D octets clusters**

Figure 9: Hint step - Attack detected



(a) 3D inbound **(b) 3D outbound**

Figure 10: ML step - Attack detected



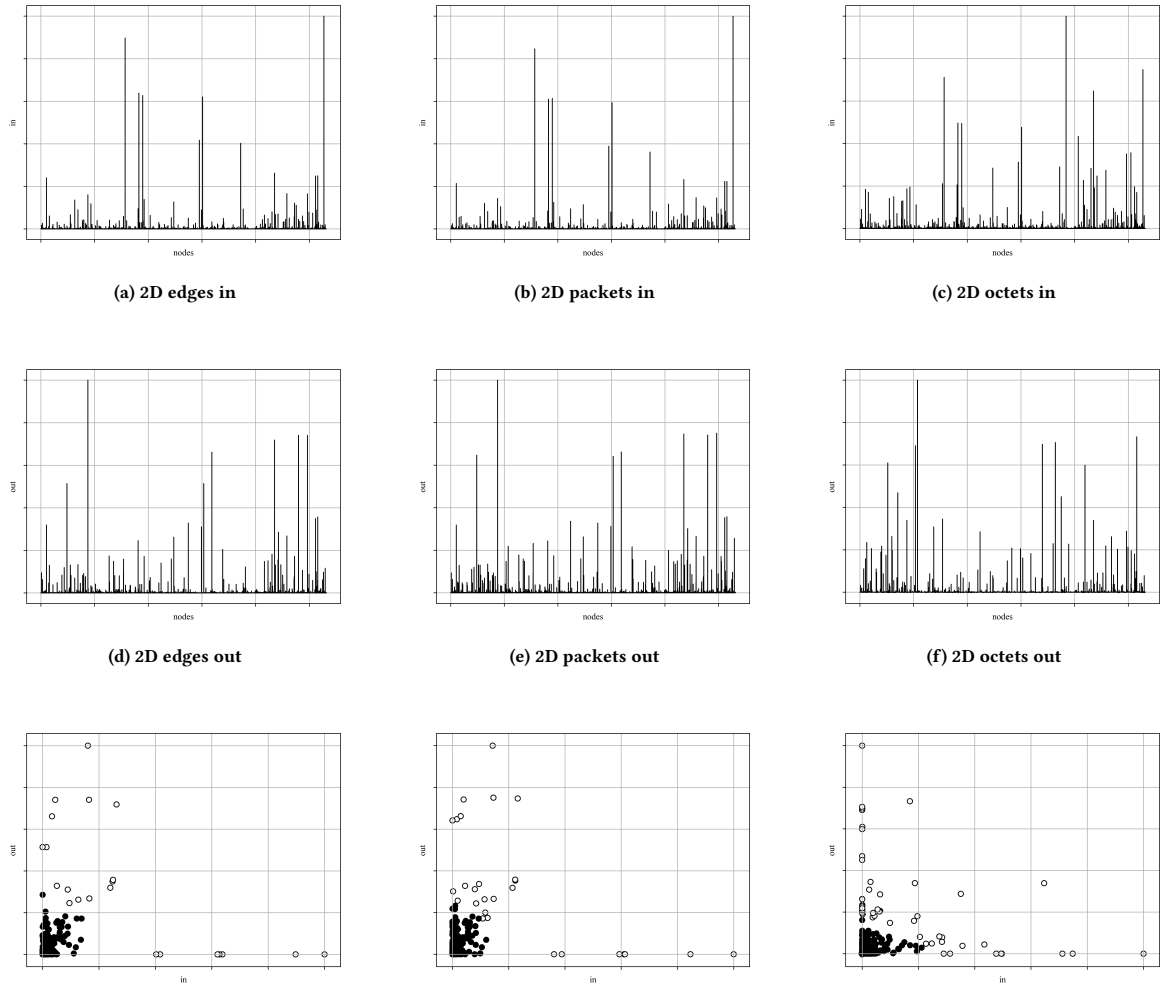
(a) 3D inbound **(b) 3D outbound**

Figure 11: ML step - No attack detected

11 visually describes the internal state of the MLE and the DME when no attack was detected. The difference with the state of the MLE when an attack is ongoing is remarkable.

5 CONCLUSIONS AND FUTURE WORK

The number and complexity of DDoS attacks will keep growing. Attackers are likely to avoid generating any traffic with unique characteristics that stand out, and therefore, invalidate the defense systems that are signature-based or trained for specific traffic patterns. Most existing DDoS defense methods are very specific and are developed to counter very concrete types of DDoS attacks, focused on a pre-defined group of protocols, and most of times have been validated using controlled network environments and/or using synthetic datasets. A generic DDoS defense system that can identify any type of DDoS attack that might occur in a real network environment, regardless of protocol and network layer, does not exist yet. Designing such a defense system with generic features is a challenge. The research we have presented on this paper works towards that direction, it aims for an open and generic architecture that can not only be easily extended and integrated with the current network infrastructure, but also serves as a technology-independent testbed that supports multiple, if not any, deployment models. The experiments we have run have proven the proposed architecture as a starting point for such a goal. Moreover, the unsupervised ML approach tested in our experiments has proven to be successful, the attack was instantly detected. One of our observations that require further experimentation and analysis is the fact that with a wider sliding window, the detection accuracy seems to have increased singnificantly. It has to be taken into account that increasing the sliding window requires more memory space, and



(a) 2D edges in (b) 2D packets in (c) 2D octets in

(d) 2D edges out (e) 2D packets out (f) 2D octets out

(g) 2D edges clusters (h) 2D packets clusters (i) 2D octets clusters

Figure 12: Hint step - No attack detected

also that the graph edges eviction process takes longer to complete on every iteration. Our experimental evaluation has been based on a somewhat limited implementation of our proposed architecture, future studies on the current topic are therefore recommended, further work needs to be done to test the presented architecture with different network paradigms, e.g. going beyond the edge of the network and concentrating on IoT and fully decentralized conversations between the agents, edge nodes and core nodes.
Therefore, our future work includes, on the one hand, conducting experiments using other datasets containing different types of attacks, both synthetic and real captured at different locations, and on the other hand, exploring the IoT paradigm by adapting the presented mechanism and data structure to work well at the very edges of the network ending up with a totally distributed mechanism by means of a peer-to-peer network and gossip protocol.

ACKNOWLEDGMENTS

This research is supported in part by NSF via grant ACI 1464317. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI^2).

REFERENCES

[1] McGregor A. 2014. Graph stream algorithms: a survey. *ACM SIGMOD* 43, 1 (2014), 9–20.
[2] Eran Assaf, Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2017. Counting distinct elements over sliding windows. *Proceedings of the 10th ACM International Systems and Storage Conference SYSTOR'17*.
[3] Taimur Bakhshi and Bogdan Ghita. 2016. On Internet Traffic Classification: A Two-Phased Machine Learning Approach. *Computer Networks and Communications* (2016).
[4] Mar Callau-Zori, Ricardo Jiménez-Peris, Vincenzo Gulisano, Marina Papatriantafilou, Zhang Fu, and Marta Patiño-Martínez. 2013. STONE: a stream-based DDoS defense framework. *Proceedings of the 28th Annual ACM Symposium on Applied Computing SAC'13*.

[5] Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. 2013. Dynamic Graphs in the Sliding-Window Model. (2013), 337–348.

[6] Raimir Holanda Filho and José Everardo Bessa Maia. 2010. Network traffic prediction using PCA and K-means. *IEEE Network Operations and Management Symposium NOMS'10* (2010).

[7] Apache Foundation. [n. d.]. Apache Kafka. ([n. d.]). http://kafka.apache.org/

[8] Apache Foundation. [n. d.]. Apache Mesos. ([n. d.]). http://mesos.apache.org/

[9] Apache Foundation. [n. d.]. Apache Metron. ([n. d.]). http://metron.apache.org/

[10] Apache Foundation. [n. d.]. Apache SAMOA. ([n. d.]). https://samoa.incubator.apache.org/

[11] Apache Foundation. [n. d.]. Apache Storm. ([n. d.]). http://storm.apache.org/

[12] Zilong Han, Xiaofeng Wang, Fei Wang, and Yongjun Wang. 2012. Collaborative Detection of DDoS Attacks Based on Chord Protocol. In *IEEE 9th International Conference on Mobile Adhoc and Sensor Systems MASS'2012*. Las Vegas, Nevada, USA.

[13] Chang-Jung Hsieh and Ting-Yuan Chan. 2016. Detection DDoS attacks based on neural-network using Apache Spark. *IEEE International Conference on Applied System Innovation ICASI'16'*.

[14] Mattijs Jonker, Anna Sperotto, Roland van Rijswijk-Deij, Ramin Sadre, and Aiko Pras. 2016. Measuring the Adoption of DDoS Protection Services. *Proceedings of the 2016 Internet Measurement Conference IMC'16*.

[15] Michael Kallitsis, Stilian A. Stoev, Shrijita Bhattacharya, and George Michailidis. 2016. AMON: An Open Source Architecture for Online Monitoring, Statistical Analysis, and Forensics of Multi-Gigabit Streams. *IEEE Journal on Selected Areas in Communications*.

[16] Kaspersky 2017. Kaspersky Lab Report on DDoS Attacks in Q1 2017. (2017). Retrieved May 31, 2017 from https://usa.kaspersky.com/about/press-releases/2017$_k$aspersky-lab-report-on-ddos-attacks-in-q1-2017-the-lull-before-the-storm

[17] Kaspersky 2017. Kaspersky Lab Report on DDoS Attacks in Q4 2016. (2017). Retrieved May 31, 2017 from https://usa.kaspersky.com/about/press-releases/2017$_k$aspersky-lab-q4-2016-ddos-attack-report-shows-record-breaking-data-for-the-year

[18] Keisuke Kato and Vitaly Klyuev. 2014. An Intelligent DDoS Attack Detection System Using Packet Analysis and Support Vector Machine. *International Journal of Intelligent Computing Research IJICR'14'* 5, 3 (2014).

[19] Berral J. L., Poggi N., Alonso J., Gavaldà R., Torres J., and Parashar M. 2008. Adaptive distributed mechanism against flooding network attacks based on machine learning. In *Proceedings of the 1st ACM workshop on Workshop on AISec AISec'08)*. Alexandria, Virginia, USA, 43–50.

[20] Do Quoc Le, H. Taeyoel Jeong, Eduardo Roman, and James Won-Ki Hong. 2011. Traffic Dispersion Graph Based Anomaly Detection. *SoICT 2011* (Oct. 2011).

[21] Minzhao Lyu, Dainel Sherratt, Arunan Sivanathan, Hassan Habibi Gharakheili, Adam Radford, and Vijay Sivaraman. 2017. Quantifying the reflective DDoS attack capability of household IoT devices. *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks WiSec'17*.

[22] Masataka Mizukoshi and Masaharu Munetomo. 2015. Distributed denial of services attack protection system with genetic algorithms on Hadoop cluster computing framework. *IEEE Congress on Evolutionary Computation CEC'15*.

[23] Arbor Networks. [n. d.]. Worldwide Infrastructure Security Report. ([n. d.]). https://www.arbornetworks.com/images/documents/WISR2016$_E$N$_W$eb.pdf

[24] Thuy T.T. Nguyen and Grenville Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials* 10, 4 (2008), 56–76.

[25] Purnawansyah and Haviluddin. 2016. K-Means clustering implementation in network traffic activities. *International Conference on Computational Intelligence and Cybernetics CYBERNETICSCOM'16'* (2016).

[26] I Wayan Oka Krismawan Putra, Yudha Purwanto, and Fiky Yosef Suratman. 2015. Modified K-means algorithm using timestamp initialization in sliding window to detect anomaly traffic. *International Conference on Control, Electronics, Renewable Energy and Communications ICCEREC'15* (2015).

[27] Jianpeng Qi, Yanwei Yu, Lihong Wang, and Jinglei Liu. 2016. A survey of techniques for internet traffic classification using machine learning. *IEEE International Conferences on Big Data and Cloud Computing BDCloud'16'* (2016).

[28] Rejimol Robinson R R and Ciza Thomas. 2015. Ranking of Machine learning Algorithms Based on the Performance in Classifying DDoS Attacks. *IEEE Recent Advances in Intelligent Computational Systems RAICS'2015'*.

[29] IMPACT Cyber Trust. [n. d.]. A reflection and amplification DDoS attack. http://dx.doi.org/10.23721/105/1354086. ([n. d.]). https://doi.org/10.23721/105/1354086

[30] Zeyu X., Yongjun W., and Wang X. 2013. Distributed Collaborative DDoS detection method based on traffic classification features. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering ICCSEE'2013*. P.R. China.

[31] Chen Y. 2007. *Collaborative Detection of DDoS Attacks over Multiple Network Domains*. IEEE, 1649–1662.

[32] Xinyu Zhang, Ke Yu, Jin Yang, and Chunying Xu. 2014. A sliding-window-based graph model for dynamic characteristics analysis of Internet traffic. (2014).