DDoS-as-a-Smokescreen: Leveraging Netflow Concurrency and Segmentation for Faster Detection

Brian Ricks
Computer Science Department
University of Texas at Dallas
Richardson, USA
Email: bwr031000@utdallas.edu

Patrick Tague
ECE Department
Carnegie Mellon University
Moffett Field, USA
Email: tague@cmu.edu

Bhavani Thuraisingham
Computer Science Department
University of Texas at Dallas
Richardson, USA
Email: bxt043000@utdallas.edu

***Abstract*—In the ever evolving Internet threat landscape, Distributed Denial-of-Service (DDoS) attacks remain a popular means to invoke service disruption. DDoS attacks, however, have evolved to become a tool of deceit, providing a smokescreen or distraction while some other underlying attack takes place, such as data exfiltration. Knowing the intent of a DDoS, and detecting underlying attacks which may be present concurrently with it, is a challenging problem. An entity whose network is under a DDoS attack may not have the support personnel to both actively fight a DDoS and try to mitigate underlying attacks. Therefore, any system that can detect such underlying attacks should do so only with a high degree of confidence. Previous work utilizing flow aggregation techniques with multi-class anomaly detection showed promise in both DDoS detection and detecting underlying attacks ongoing during an active DDoS attack. In this work, we head in the opposite direction, utilizing flow segmentation and concurrent flow feature aggregation, with the primary goal of greatly reduced detection times of both DDoS and underlying attacks. Using the same multi-class anomaly detection approach, we show greatly improved detection times with promising detection performance.**

***Index Terms*—DDoS, distributed denial-of-service, smokescreen, DDoS-as-a-smokescreen, DaaSS, anomaly detection, intrusion detection, concurrent flow, segmented flow, netflow, underlying attack**

I. Introduction

Throughout the Internet age, network intrusions have been a concern, with detection strategies dating all the way back to D.E. Denning's general intrusion detection model in 1987 [1]. Indeed, network intrusion detection was largely an afterthought in the late 1980s, up until the Morris Internet Worm nearly brought down the network in November of 1988 [2]. Since then, network-based attacks have grown considerably, fueled in part by a widespread adoption of the Internet [3].

One common network attack has been prevalent: the Distributed Denial-of-Service (DDoS). This attack involves many often hijacked computers to concurrently flood a target network, with goals ranging from denying customer access to providing cover for some other nefarious deed. This latter goal, using a DDoS attack as a distraction or smokescreen to obfuscate another attack, is called *DDoS-as-a-Smokescreen (DaaSS)* [4], and often are leveraged for purposes of data exfiltration or financial theft. DaaSS attacks can be especially challenging for information technology support (IT) to mitigate as the obfuscated attack may not be noticeable at first (or ever), but the DDoS itself presents problems that need to be mitigated immediately. DaaSS attackers exploit this need to mitigate DDoS attacks in the present, along with the expectation of finite IT resources.

While DaaSS attacks have been discussed widely in industry for at least a decade, they are only recently gaining traction as an area of active research [4], [5]. DaaSS attacks require intrusion detection methods to be able to reason about multiple concurrent threats, and deduce the correct attack. Furthermore, the concurrent attack being obfuscated is assumed to be anomalous, requiring anomaly detection techniques which may be prone to false positive detection or classification [6]. False positive DaaSS classification is especially destructive in this setting, as it could result in members of IT diverting from DDoS mitigation to mitigation of a non-existent underlying attack.

Our previous work in DaaSS detection utilized network flow aggregation [4], which merged netflows to boost aggregate attributes, enabling strong DDoS detection performance, but with a tradeoff of long detection times and potentially less sensitivity to benign behavior and/or the underlying anomalies which comprise a DaaSS attack. In this work, we approach it from the opposite direction, using flow segmentation with attribute augmentation to enable shorter duration network flows suitable for both DDoS detection and underlying anomaly detection.

A key observation that provides motivation for this approach is similarity that often exists between DDoS and some benign traffic. For example, HTTP, request-based DDoS traffic may be very similar to legitimate, benign HTTP traffic, when looking at each request in isolation. However, the volume of these requests will be much higher when the DDoS is present. By leveraging this additional information in the form of concurrent flow attributes, we hope to overcome the natural information loss from segmenting netflows.

The primary goal of this work is to shorten DaaSS detection times, while hopefully maintaining a high level of DaaSS detection performance. More specifically, we aim to achieve the following:

- Smaller duration flows with a robust attribute space utilizing network flow concurrency, in a format suitable for

model learning and inference tasks specifically targeting DaaSS classification.
- Lower DDoS and DaaSS detection times from an order of minutes to an order of seconds.
- Maintain compatibility with existing learning and inference methods for DaaSS detection.
- Maintain classification performance with the aggregated netflow baseline.

The rest of this paper is structured as follows. Section II discusses relevant fundamentals and related work, including the DDoS-as-a-smokescreen threat model. Section III introduces our segmented netflow approach, discussing at depth flow segmentation and concurrency feature augmentation. Section IV discusses the learning and detection (classification) approach we apply our segmented netflow data to. Section V discusses our experiments, including the PCAPs from which we generate our flow data, experimental methodology, results and discussion.

II. Preliminaries

In the following subsections we discuss two important preliminaries for the rest of the paper: network flows, and the DDoS-as-a-smokescreen (DaaSS) threat model.

A. Network Flows

Network data captures are typically represented as packets, using formats such as PCAP. Raw packet data is rich in information not only about individual packets, but timing in relation to other packets. Unfortunately, raw packet captures alone lack explicit structure that can be leveraged to generalize the data into a model suitable for inference tasks. To provide such structure, PCAP data can be parsed into network flows (netflows, or simply *flows*), with each flow representing a sequence of packets that are grouped by some criteria, such as common source address/port, common destination address/port, and protocol. Each flow comprises aggregate attributes to characterize the flow, such as number of packets in the flow, and total byte count of the flow. Netflow data can be mapped directly to propositional learning tasks, with flows corresponding to examples, and attributes as features, where it is assumed that the feature cardinality is fixed and identical for each flow.

Network flows can be parsed from packet data in a uni-directional or bi-directional manner, with bi-directional flows encompassing packet traffic from both source-to-destination and destination-to-source [7]. On the one hand, uni-directional flows induce an independence between the two directions of a flow; on the other hand, bi-directional flows lose the direction-specific aggregate attributes. In this work, we use bi-directional flows with uni-directional attribute augmentation to preserve some uni-directional flow information.

Many netflow standards exist that are widely used in industry, such as Argus [8]. The Argus netflow standard comprises a base set of attributes which serve as a foundation for many extended-attribute netflow formats [4], [9], as well as for our proposed segmented netflow format. Some popular network intrusion datasets are generated from Argus, such as CTU-13, a real-world botnet dataset utilizing bi-directional network flows [7], [10].

Models trained from netflow data belong to a methodology called anomaly detection, and can sometimes have issues discriminating among many different traffic behavioral profiles. For example, the aggregated netflow format exists primarily for training models that can detect high volume DDoS traffic. Unfortunately a trade-off with flow aggregation is additional loss of information, possibly leading to a loss of sensitivity of a model to detect (classify) other traffic behavior. This also leads to increased detection times as all flows in the aggregate must be completed (connection closed or time out) before they can be used for classification [4]. Indeed, anomaly detection methodologies in general have a hard time finding a footing in industry due to potentially unacceptable false positive rates [6].

B. DDoS as a Smokescreen

While DDoS attacks have been a relatively common Internet occurrence since the 1990s, *DDoS as a Smokescreen* emerged about a decade ago, with the now infamous 2011 DaaSS attack on the Sony PlayStation network, in which personal data from 77 million customers was exfiltrated [11].

The DaaSS threat model can be considered a constrained version of the *Coordinated Attack with Role Distribution (CARD)* threat model discussed by Samarji et al. [12]. The key difference here is that the underlying anomaly must occur concurrently with an active DDoS specifically, a characterization which presents unique new ways to approach detection tasks [4]. For example, because the presence of a DDoS is a prerequisite for a DaaSS attack, we can constrain our detection search space to encompass only those time periods in which a DDoS is present. Furthermore, DDoS attacks themselves constrain nominal traffic behavioral patterns through the denial-of-service they induce.

The DDoS should be noticeable to the point of invoking a response from IT, afterall, it would not serve as a good smokescreen if it did not do so. These DDoS attacks often manifest in high-throughput flooding from many source hosts, perhaps from botnets hired on contract [13].

III. Segmented Netflow Generation

Netflows can span long periods of time, which naturally lends to potentially long wait times for real-time inference tasks. Conversely, segmented or short duration flows may not encompass enough information to adequately classify different traffic behavioral patterns. In the following subsections, we propose a method to segment flows and augment the segments with flow concurrency attributes, enabling classification of segments directly before their respective flows are complete.

Our generation approach takes as input raw packet data (PCAP), and outputs flow segments as a propositional flat-file. Each row represents a segment as a comma delimited list of attribute values. We next discuss in more depth how these segments, and their attributes, are defined and generated from PCAP data.

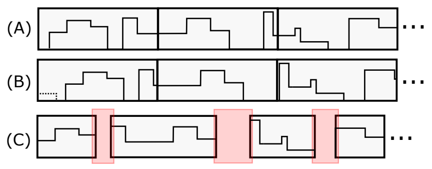

Fig. 1. Effects of byte padding on a fixed length segmentation scheme. The boxes represent flow segments, and the patterns inside represent packet byte aggregate values. The sequence of segments denoted (A) illustrates a fixed length sequence in which the patterns contained in each segment are similar. The sequence denoted (B) shows what happens when a small amount of byte padding is introduced, denoted by the dotted pattern. The packet byte aggregate values change, and rather significantly for the second segment. However, the sequence denoted (C) uses the gap interval, denoted by the red box highlights. Notice that the segments are not fixed length, and have segmented along regions where the flow had no traffic activity. Even with a shift, the second segment onward would be unaffected.

A. Flow Segmentation

Segmentation follows along two criteria: *gap interval* and *maximum duration*. Gap interval, which we denote as λ_g, is defined as the minimum amount of time in which a netflow exhibits no traffic in both directions, and provides a natural method of segmenting many real-world traffic patterns. Maximum duration, which we denote as λ_d, provides an upper bound to how long in time a flow is allowed to be. This upper bound is used to force segmentation for continuous traffic flows that otherwise would never be segmented according to the gap interval, such as DDoS traffic.

Intuition behind using gap interval as a segmentation criteria follows from the observation that many traffic patterns on the internet are in the form of request/response, with gaps (even small ones) appearing when either the client or server is processing a request or response. A gap interval helps to preserve general traffic patterns within a flow by assuming that inter-gap traffic for a specific flow type will follow similar patterns. Gap interval provides a principled way to preserve such 'sub-patterns' within a flow, and should provide resilience to shifts in these patterns in segments due to timing variation. And example of how shifts can affect segment patterns is illustrated in Figure 1.

The maximum duration criteria enforces a maximum interval (in time) before segmentation must occur. The primary use case is continuous traffic flows, such as DDoS or file downloads. This criteria provides an upper bound on detection times by enforcing a maximum segment length. Note that the traffic sub-pattern shift problem discussed above largely does not apply here, as continuous traffic itself is often bounded by available bandwidth, and generally resembles a uniform pattern.

Clearly, the value of the gap interval is important: too small a gap may result in over-segmentation, too large and segmentation may fall back to the maximum duration criteria. Over-segmentation may lead to over-fitting when learning a model, possibly resulting in poor classification performance.

B. Flow Concurrency

While flow segmentation should lead to faster inference times, to enable potentially *good* inference, we need to make up for the inherent information loss that segmentation brings. To this end, we define additional attributes which relate flow segments to other netflows. We say that a flow segment i belonging to flow j, denoted s_{ij} is *concurrent* to a flow k, denoted w_k, where $j \neq k$, if s_{ij} and w_k share a common destination address, port and protocol, and if they overlap in time. This time overlap is true if:

$$\min[t_e^*(w_k), t_e(s_{ij})] - \max[t_s(w_k), t_s(s_{ij})] \geq 0, \qquad (1)$$

where t_s is a flow or segment start timestamp, t_e is a segment end timestamp, and t_e^* is the last known packet arrival for a flow.

Equation 1 would be applied as the last packet arrives to a segment. Unfortunately we may not know exactly which packet is the last one, for example if packets simply stop arriving before any segmentation criteria can be met, we will not know if any additional packets may arrive. Therefore, $t_e(s_{ij})$ for segment s_{ij} can only be known with certainty by applying a packet timeout and retroactively calculating concurrency when the timeout is reached, potentially causing other problems with concurrent flows that have had packet arrivals during the timeout period.

Therefore, our implementation to calculate concurrency assumes that *any* arriving packet could be the last one of a segment, and concurrency is calculated on a per-packet basis as packets arrive. This may seem intractable from a computational standpoint, but we can increase efficiency by exploiting a key observation from this approach: each packet that arrives is the current one, and packets cannot arrive at the same time.[1] In other words, when a packet arrives, we know that there are no packets that have arrived after it. Thus, for a given segment s_{ij} and any flow w_k, $j \neq k$, we have that $t_e^*(w_k) < t_e(s_{ij})$, and can eliminate $t_e(s_{ij})$ from the computation. Similarly, we can eliminate $t_s(w_k)$, as flow w_k's start time no longer has influence on concurrency (we do not record degree of concurrency, only if a concurrency exists). Therefore, Equation 1 can be reduced to:

$$t_e^*(w_k) > t_s(s_{ij}), \qquad (2)$$

in that if a flow w_k's end time is greater than segment s_{ij}'s start time, w_k is concurrent to s_{ij}.[2]

Keeping our feature space fixed, we generate aggregate features that capture similar flow statistics as the standard netflow aggregates. Representing the number of concurrent flows is simply a summation, however, representations of total packet count and total packet bytes among all concurrent flows cannot be simple summations as, like above, we do not know if any concurrent flow has completed or if other packets

[1]We assume packet capture is on a single interface, therefore enforcing sequential packet arrival.

[2]Equation 2 is strictly greater than, due to a 1-packet segment having a duration of 0 by definition, and thus as the only packet of the segment and the last packet to have arrived, cannot be concurrent to any flow.

will from those flows will arrive in the future. Concurrent summation aggregates give monotonically increasing values as concurrent flows progress in time due to new packet arrivals adding to their values. Thus, these aggregates correlate timing information of concurrent flows which is not desirable. For example, suppose two concurrent flows exhibit the same traffic patterns, but one started before the other. The one that started first will have aggregate values at least as high as the second, due to possibly more packet arrivals.

To compensate for this, we define *average* aggregates for concurrent total packet count and total packet bytes as:

$$f_{avg} = \sum_{w \in W_c} \frac{\sum_{s \in w} v_s}{|w|}, \tag{3}$$

where W_c is the set of concurrent flows, v_s is an attribute value from a segment s used in the aggregate sum, and $|w|$ is the cardinality of segments in a given flow w. In this way, we sum up average aggregate values from each concurrent flow, which can be looked at as normalizing each flow by the current number of segments that belong to each. Segments are bounded by the max duration parameter λ_d, so any flow w that is longer in duration than λ_d will have $|w| > 1$.

To summarize, we added a total of 11 new attributes to the existing bi-directional Argus base to form our segmented netflow format:

- Flow statistical attributes: uni-directional aggregates of packet count and packet bytes, in both directions
- Concurrent flow attributes: bi-directional aggregates for concurrent flow count, average packet count, and average packet bytes. Additional uni-directional aggregates for concurrent average packet count and average packet bytes, in both directions.

IV. Learning and Inference

The tasks of training a model and performing inference (classification or detection) on that model is provided by a method given in our previous work, called N1 clustering [4]. We treat this as an off-the-shelf solution in which segmented flow data (Section III-A) is given as the dataset.

N1 clustering is an extension of k-prototypes unsupervised hard clustering [14], enabling an unobserved (unknown) class along with multiple observed (known) classes. k-prototypes is itself an extension of k-means hard clustering, providing numerical and categorical features among a fully observed space of classes. N1 clustering can be considered a generalization of one-class unsupervised hard clustering.

In our setting, we have three classes: *nominal*, *DDoS*, and *anomaly*, in which the first two classes are observed. It is expected that training datasets comprise unlabeled segments corresponding to the observed classes only, as observed anomalies are not really anomalies. Observed classes are represented in the model as clusters, while the unobserved class can be visualized in a open world as encompassing all space that is not occupied by any cluster. This implies that the clusters are bounded, which is accomplished by setting an

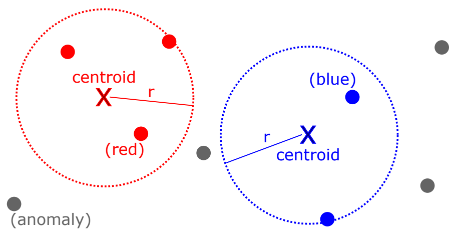

Fig. 2. Visualization of Inference in N1 clustering. In this example we have three classes: *red*, *blue*, and *anomaly*. Various points represent flow segments, and their color their classification. Segments classified as *anomaly* reside outside the two clusters, with cluster boundaries defined by their radius r. The space external to all clusters is an open world.

explicit radius per cluster [4], [15]. More formally, this radius, denoted as r_j, is defined as:

$$r_j = \alpha_r \max_{\mathbf{x}_i \in \mathbf{c}_j} d(\mathbf{x}_i, \mathbf{c}_j), \tag{4}$$

in which $\mathbf{c}_j$ is the cluster upon which radius r_j is assigned, $\mathbf{x}_i$ is a datapoint (flow segment) assigned to cluster $\mathbf{c}_j$, and α_r is a tuneable hyperparameter to control classification sensitivity. Distance metric $d(\mathbf{x}_i, \mathbf{c}_j)$ represents the Euclidean distance between datapoint $\mathbf{x}_i \in \mathbf{c}_j$ and cluster $\mathbf{c}_j$'s centroid.

An example (segment) is classified according to the cluster it is closest to, provided the example is within the cluster. An example that is not within any cluster is classified as an anomaly. More specifically, an example $\mathbf{x}_i$ is classified as an anomaly if $\forall j \; d(\mathbf{x}_i, \mathbf{c}_j) > r_j$ [4]. Figure 2 illustrates inference in N1 clustering, using a three-class example in which classes *red* and *blue* could represent *DDoS* and *nominal*, and with class *anomaly* representing the underlying concurrent attack in a DaaSS.

A. Post-Inference Classification

Our segmented netflows lead to datasets in which the examples comprise flow segments. Because flow segments are propositional examples, they follow the IID assumption, potentially leading to flows being classified to multiple labels.

On the surface it may not seem to be a big deal, after all, in an online setting we are using segment classification directly. However, inconsistent segment classifications can give us insights into the flows of which they belong. Using the data exfiltration task presented in Section III-A, suppose that a majority of segments from a common flow are misclassified as *DDoS*, but a few are correctly classified as *anomaly*. The 'anomaly' segment classifications provide strong evidence that the flow is indeed an underlying anomaly, while the DDoS segment classifications give insights into the behavior of the underlying anomaly; more specifically, that the obfuscated attack is designed to resemble a DDoS. In this case, the netflow classification inconsistency can be resolved to *anomaly*, with additional information letting IT know the behavior of the anomaly.

In Section V-B we discuss the rules by which we resolve classification inconsistencies among segments that belong to a common netflow.

Scenario	λ_g (seconds)	λ_d (seconds)	# segments	# flows
Training	2	30	441847	-
	2	50	441847	-
DDoS-1	2	30	109305	104858
	2	50	109305	104858
DaaSS-1	2	30	118942	113616
	2	50	118937	113616
DaaSS-2	2	30	122279	116494
	2	50	122277	116494

TABLE I
THE DATASETS (SCENARIOS) USED IN OUR EXPERIMENTS. INFORMATION MAPPING SEGMENTS TO THEIR FLOWS IS ONLY USED DURING INFERENCE TIME, HENCE THIS INFORMATION WAS NOT SAVED FOR THE TRAINING SCENARIOS.

V. EXPERIMENTS

In the following subsections we discuss our dataset used for experimentation, the experimental methodology, and results obtained.

A. Dataset

We use DDoS and DaaSS PCAPs from our previous work to generate our segmented flow data [4]. Each training and test PCAP was generated from the EMEWS framework [16], which provides client-side autonomous network traffic generation. Capturing human-client traffic on real-world networks may lead to privacy concerns, whereas capturing traffic from human volunteers on a network testbed may lead to scaling issues, and also potential behavioral deviations simply due to the volunteers 'being watched' [17]. To promote large scale networks in a lab setting while maintaining a real-world network stack per network node, EMEWS was deployed on the CORE network emulator [18], with a 312 node topology, containing 130 benign HTTPS clients, 31 benign SSH clients, and an 89 node botnet to carry out the DDoS attack.

The training set incorporates both benign and DDoS traffic, but no underlying anomalies. DDoS test sets do not capture any underlying anomalies, whereas DaaSS test sets do. The DDoS behavior resembles an amplification attack [19], but rather than using more traditional means such as DNS exploitation, here HTTPS is used, inducing many very small but dense TCP flows. This type of DDoS behavior provides a challenge, as flow segmentation is impractical. However, the underlying anomalies captured resemble an insider data exfiltration task, resulting in long, bursty flows which may be suitable for segmentation.

The PCAP data was parsed to our segmented netflow format using λ_g and λ_d values as given in Table I. As most flows are DDoS, which are small in duration and usually not segmented (according to the λ_g and λ_d values we used), the number of segments that comprise a dataset is not that much greater than the number of flows. When generating segmented flows, we discarded the first 30 seconds of packets and any flows which started within the last 30 seconds of the capture, due to outliers captured from CORE network setup and tear-down.

B. Experimental Methodology

Experiments compare our approach to the aggregated netflow baseline in both detection times and performance metrics. When discussing our results, we will refer to each dataset from Table I as a scenario.

As the original N1 clustering algorithm was written in Python2, we ported it to Python3, and additionally augmented it with post inference classification resolution (Section IV-A). When training, k-prototypes was configured for 10 runs, and the best run (model) picked based on cost. For each run, N1 clustering initializes centroids using the method given in Cao et al. [20] for categorical features, and sampling over normal distributions derived from the training examples for numerical features [4]. This learned model was then applied to all the test sets in a batch, varying α_r and recalculating cluster radii between batch predictions.

One challenge in using clustering for a dataset with heavy class imbalance, such as ours, is the potential for one class to represent some or all learned clusters. In our setting, sometimes the best learned model would comprise two clusters with very little separation, both representing DDoS traffic. For evaluation, N1 clustering uses a baseline dataset of representative examples to learn the appropriate cluster labels, and when both learned clusters represent the same class, this step fails. In such cases, trying to train a model again would usually work.

We present our experimental results as a single classification per netflow. This has the added benefit of providing a direct comparison to the baseline format, which comprise aggregated netflows. Our rules for resolving segment classification conflicts is straightforward: if any segment of a common flow is classified as *anomaly*, classify the flow itself as *anomaly*. Underlying anomalies during a DaaSS attack can be very subtle, and any segment classified as such should be taken seriously. At the same time, we hope to keep false positives low, as they could divert IT resources away from the active DDoS.

If no segments in a common netflow classify as *anomaly*, then take a majority consensus among the segment classifications. For *nominal* or *DDoS* (resolved) classified flows, a misclassification of the other is still considered a true negative, as the flow was not misclassified as *anomaly*. In this case, IT does not have to change their current mitigation strategy. For a netflow that actually is an underlying anomaly, this would be considered a false negative, which still would result in the mitigation strategy of IT being unaffected, though not as ideal.

Detection times are defined as the duration from the start of the first segment of a flow (either DDoS or underlying anomaly) to a segment within that flow which correctly provides the resolved classification. For DDoS detection times, we use the first DDoS flow which provided the correct resolved classification. For underlying anomalies, we present the detection times for all flows in which the resolved classification was correct. Our segmented netflow format does not capture duration between segments, thus, we take the value of the gap interval criteria, λ_g. This gives us an upper bound (or worst case) on what the detection times could be. Because we are comparing flow-based detection times, we do not consider the time from the beginning of the scenario. Per-flow generation and inference times are negligible and not included in the duration calculation.

C. Results

Table II breaks down our results in terms of classification performance, in which we achieve the same performance using $\lambda_d = 30$ seconds and $\lambda_d = 50$ seconds, given $\lambda_g = 2$ seconds. The number of segments which comprise our underlying anomaly flows did decrease as λ_d increased, implying that the underlying anomalies exhibited rather dense traffic patterns. Indeed, some of the segments have a duration at or close to the maximum duration, λ_d, implying that these segments were segmented using the maximum duration criteria instead of gap interval.

The choice of gap interval, $\lambda_g = 2$, was based on the observation that for the maximum duration criteria to be of value, $\lambda_g < \lambda_d$, otherwise segmentation may always use maximum duration for the criteria. Therefore, larger λ_g values would lead to larger λ_d values, and based on our underlying anomaly detection times (Table III) and traffic patterns, larger values for λ_d would most likely increase these times. Furthermore, larger values such as $\lambda_g = 15$ seconds resulted in learning difficulties where N1 clustering could not learn a model with distinct clusters, even after numerous attempts. A potential solution is to change the cluster centroid initialization method to one that provides better guidance, such as using labeled data as a seed.

The best value for α_r ended up matching the best value from the aggregated netflow baseline. Even though we are using a new flow format for input to N1 clustering, the best results still are from an α_r value which expands the learned radius of each cluster by roughly 50%. This may suggest that good choices for α_r are dependent more on the captured traffic patterns rather than the specific network flow format and attributes provided.

For scenario *DaaSS-1*, flow segmentation helped to reduce DaaSS detection times, by segmenting the often long flows that comprised underlying anomalies. At $\alpha_r = 1.5$ seconds, we were able to detect all underlying anomalies without false positives, for all λ_g and λ_d values presented. Scenario *DaaSS-2* was more challenging, and difficult to correctly classify underlying anomalies from benign traffic. While we could not detect any underlying anomalies for this scenario, there were no false positives detected that may lead IT astray.

As given in Table III, DDoS flow detecton times were orders of magnitude smaller with segmented netflows. This is expected, given that the primary purpose of flow aggregation is for DDoS traffic classification, with our analogue being concurrent flow attributes. Underlying anomaly detection times were closer, though the segmented netflow approach still held a sizable advantage for scenario *DaaSS-1*. Unfortunately we were unable to classify any underlying anomalies for scenario *DaaSS-2* without lowering α_r to such a value as to induce false positives.

D. Discussion

Aggregated netflows are naturally strong with DDoS detection through building up by summation the limited set of attributes available, differentiating them to other flow types, which enables a learning algorithm to learn a separation rather easily. Segmented netflows are also strong in DDoS detection, and can provide this detection much faster due to smaller duration flows/segments to classify.

Concurrent flow attributes help greatly in maintaining DDoS classification performance, and thanks to the specific behavioral characteristics of the DDoS, we achieved these detection times without needing to focus on λ_g or λ_d values. In fact, in order for λ_d to have any effect on DDoS flows, it would need to be set to < 2.0 seconds (depending on the scenario - see Table III) for any segmentation to occur. λ_g doesn't play a role here, as it would need to be set so low as to induce near 1-packet flows, rendering the dataset useless. What about UDP-based dense DDoS traffic? In this case, there is no natural flow structure, and segment length would be determined by λ_d. In our setting, this would raise detection times for DDoS attacks to at least λ_d. One way to avoid this issue would be to have multiple λ_d parameters, one for TCP flows and one for UDP flows.

1) Separation of Benign and DDoS Traffic: For every scenario, a majority of the benign segments were classified as *DDoS*. This is due to a majority of benign traffic captured being HTTPS flows, resembling the DDoS traffic. This similarity presents a problem with classification *during an active DDoS*. Note that before the DDoS begins, the number of flows concurrent to benign HTTPS traffic will be much much less, as the DDoS itself comprises many concurrent flows. Therefore, it is reasonable to expect that concurrent flow attribute values of benign segments before a DDoS would closer match classification to the benign cluster, and limit the number of misclassifications of benign traffic (we do not want the system to give a false alert of a DDoS that is not actually present). As explained in Section IV-A, misclassifications of benign segments as *DDoS* during an active DDoS is okay as it will not affect IT actions.

VI. Limitations and Future Work

Our PCAP data only captured traffic during an active DDoS. Using the same PCAPs as our prior work gave us a baseline for comparison, but at the expense of being unable to observe how our segmented netflow approach handles DDoS detection before and after an active DDoS. We need to put our theory to practice and capture PCAP network data from before, during, and after an active DDoS.

We also need to augment our segmented flow generator with additional output related to duration between segments in a flow. This not only would help give us more precise underlying anomaly detection times, but also may be useful as an additional attribute to the segmented netflow format.

Scenario	λ_g (seconds)	λ_d (seconds)	# DaaSS flows	$\alpha_r = 1.0$			$\alpha_r = 1.25$			$\alpha_r = 1.5$		
				#FP	Precision	Recall	#FP	Precision	Recall	#FP	Precision	Recall
DaaSS-1	2	30	2 (27)	1	67%	100%	1	67%	100%	0	100%	100%
	2	50	2 (22)	1	67%	100%	1	67%	100%	0	100%	100%
Aggregated (baseline)			3	1	75%	100%	0	100%	100%	0	100%	100%
DaaSS-2	2	30	7 (63)	0	-	-	0	-	-	0	-	-
	2	50	7 (61)	0	-	-	0	-	-	0	-	-
Aggregated (baseline)			9	1	85.71%	66.67%	0	100%	66.67%	0	100%	66.67%

TABLE II

DaaSS scenarios (those which included underlying anomalies). Our segmented netflow approach is presented with the baseline aggregated netflow approach. For # DaaSS flows, the values in parenthesis are the total number of segments that belong to underlying anomaly flows. Note that we were unable to detect any underlying anomalies in scenario *DaaSS-2*, and thus have no precision and recall metrics.

Scenario	λ_g (seconds)	λ_d (seconds)	Detection (seconds)	
			DDoS	Anomalies
DDoS-1	2	30	2.0	-
	2	50	2.0	-
Aggregated (baseline)			179.6	-
DaaSS-1	2	30	1.7	73.3
				43.3
	2	50	1.7	78.5
				58.3
Aggregated (baseline)			179.8	176.7
				119.0
				112.1
DaaSS-2	2	30	1.7	-
	2	50	2.3	-
Aggregated (baseline)			178.5	105.1
				83.4
				72.3
				178.1
				65.9
				33.9

TABLE III

DDoS and underlying anomaly detection times, comparing our segmented netflow approach to the aggregated netflow baseline, with $\alpha_r = 1.5$. DDoS detection times are calculated using the first DDoS flow that is correctly classified as a DDoS. Scenario *DDoS-1* did not include any underlying anomalies, whereas for scenario *DaaSS-2*, our segmented netflow approach could not detect any underlying anomalies.

The fixed structure of propositional datasets requires us to resort to aggregates such as counts to represent variable length features such as concurrent netflows. Network data itself is naturally rich with these types of one-to-many or many-to-many relationships, which unfortunately are difficult to represent in a fixed-length vector format without resorting to using aggregates, and the associated information loss that comes thereof. Indeed, the Argus netflow standard makes the IID assumption between flows, which is not only inaccurate in real-world networks, but is also inadequate in a segmented netflow setting for classification.

In addition to the information loss present with such aggregates, we lose many relations within our naturally rich relational data simply because we cannot represent them within a fixed feature space. For example, while concurrent flow counts provide some additional information to a segment, we lose relationships among the specific concurrent flows, which we cannot represent due to a variable feature space in which these relational features induce. Such relationships include specific traffic patterns present in concurrent netflows which could be used to aid classification.

One elegant solution to this limitation, and a direction of future work, is to use statistical relational methods for both model learning and inference tasks. Relational methods utilize datasets which contain grounded logic predicates, along with modes that define structure among the predicates and constrain the search space during model learning [21]. Of particular interest are one-class classification (OCC) methods that could be used for pairwise classification, generalizing OCC to the use-case we require for DaaSS detection [22], [23].

VII. Conclusion

In this work we introduced a segmented netflow format utilizing concurrent flow attributes, with the primary goal of decreasing detection times of the two key components of a DaaSS attack: DDoS and underlying anomalies. Our results show promise, and by addressing some limitations of the work (Section VI), we believe we can achieve at least as good classification performance as the aggregated netflow baseline, including the ability to detect underlying anomalies in harder scenarios, such as *DaaSS-2*.

References

[1] D. E. Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, Feb 1987.

[2] E. H. Spafford, "The Internet Worm Incident," in *ESEC '89*, C. Ghezzi and J. A. McDermid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 446–468.

[3] V. Yegneswaran, P. Barford, and J. Ullrich, "Internet intrusions: Global characteristics and prevalence," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 138–147.

[4] B. Ricks, B. Thuraisingham, and P. Tague, "Lifting the Smokescreen: Detecting Underlying Anomalies During a DDoS Attack," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2018, pp. 130–135.

[5] M. Junger, A. Abhishta, and B. Nieuwenhuis, "Crime Chain: the Link between DDoS Attacks and Phishing," University of Twente, Tech. Rep., 06 2021.

[6] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 305–316.

[7] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[8] Qosient, "Audit Record Generation and Utilization System (Argus)," https://qosient.com/, 2021.

[9] Y. Feng, J. Li, L. Jiao, and X. Wu, "Botflowmon: Learning-based, content-agnostic identification of social bot traffic flows," in *2019 IEEE Conference on Communications and Network Security (CNS)*, 2019, pp. 169–177.

[10] Stratosphere, "Stratosphere laboratory datasets," 2015, retrieved March 13, 2020, from https://www.stratosphereips.org/datasets-overview.

[11] P. Wagenseil, "Sony Blames Anonymous for PlayStation Network Attack," http://www.nbcnews.com/id/42909386/ns/technology_and_science-security/t/sony-blames-anonymous-playstation-network-attack/, 2011.

[12] L. Samarji, F. Cuppens, N. Cuppens-Boulahia, W. Kanoun, and S. Dubus, "Situation Calculus and Graph Based Defensive Modeling of Simultaneous Attacks," in *Cyberspace Safety and Security*. Springer International Publishing, 2013, pp. 132–150.

[13] M. Karami and D. McCoy, "Understanding the Emerging Threat of DDoS-as-a-Service," in *Presented as part of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats*. Washington, D.C.: USENIX, 2013.

[14] Z. Huang, "Clustering Large Data Sets with Mixed Numeric and Categorical Values," in *In The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 1997, pp. 21–34.

[15] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and Novel Class Detection in Concept-Drifting Data Streams Under Time Constraints," *IEEE Trans. on Knowl. and Data Eng.*, vol. 23, no. 6, pp. 859–874, Jun. 2011.

[16] B. Ricks, P. Tague, and B. Thuraisingham, "Large-Scale Realistic Network Data Generation on a Budget," in *19th International Conference on Information Reuse and Integration (IRI)*. IEEE, 2018.

[17] M. Bateson, D. Nettle, and G. Roberts, "Cues of being watched enhance cooperation in a real-world setting," *Biology Letters*, vol. 2, no. 3, pp. 412–414, 2006.

[18] J. Ahrenholz, "Comparison of CORE Network Emulation Platforms," in *MILCOM 2010 Military Communications Conference*, Oct 2010, pp. 166–171.

[19] H. van Wieren, "Signature-based ddos attack mitigation: Automated generating rules for extended berkeley packet filter and express data path," November 2019.

[20] F. Cao, J. Liang, and L. Bai, "A New Initialization Method for Categorical Data Clustering," *Expert Syst. Appl.*, vol. 36, no. 7, pp. 10 223–10 228, Sep. 2009.

[21] L. D. Raedt, K. Kersting, and S. Natarajan, *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Morgan & Claypool Publishers, 2016.

[22] T. Khot, S. Natarajan, and J. W. Shavlik, "Relational One-Class Classification: A Non-Parametric Approach," in *AAAI*, 2014.

[23] S. Natarajan, B. N. Saha, S. Joshi, A. Edwards, T. Khot, E. M. Davenport, K. Kersting, C. T. Whitlow, and J. A. Maldjian, "Relational learning helps in three-way classification of alzheimer patients from structural magnetic resonance images of the brain," *International Journal of Machine Learning and Cybernetics*, vol. 5, pp. 659–669, 2014.