Windower: Feature Extraction for Real-Time DDoS Detection Using Machine Learning

Patrik Goldschmidt
Brno University of Technology, Brno, Czech Republic
Kempelen Institute of Intelligent Technologies, Bratislava, Slovakia
igoldschmidt@fit.vut.cz

Jan Kučera
Brno University of Technology, Brno, Czech Republic
CESNET a.l.e., Prague, Czech Republic
jan.kucera@cesnet.cz

***Abstract*—Distributed Denial of Service (DDoS) attacks are an ever-increasing type of security incident on modern computer networks. To tackle the issue, we propose Windower, a feature-extraction method for real-time network-based intrusion (particularly DDoS) detection. Our stream data mining module employs a sliding window principle to compute statistical information directly from network packets. Furthermore, we summarize several such windows and compute inter-window statistics to increase detection reliability. Summarized statistics are then fed into an ML-based attack discriminator. If an attack is recognized, we drop the consequent attacking source's traffic using simple ACL rules. The experimental results evaluated on several datasets indicate the ability to reliably detect an ongoing attack within the first six seconds of its start and mitigate 99% of flood and 92% of slow attacks while maintaining false positives below 1%. In contrast to state-of-the-art, our approach provides greater flexibility by achieving high detection performance and low resources as flow-based systems while offering prompt attack detection known from packet-based solutions. Windower thus brings an appealing trade-off between attack detection performance, detection delay, and computing resources suitable for real-world deployments.**

***Index Terms*—network intrusion detection, NIDS, DDoS mitigation, real-time, stream data mining, machine learning**

I. Introduction

Distributed Denial of Service (DDoS) is one of the most prevalent cyber-attacks on today's Internet. It is described as a deliberate attempt to render a target system unavailable, threatening the key cybersecurity goal – availability. The attack is typically performed via a large number of geographically distributed computers known as a botnet. Such computers simultaneously generate a huge amount of requests to overwhelm a target system or its underlying network architecture.

DDoS usage has grown massively since the first attack [1] reported in 1999. Cisco predicts 15.4 million DDoS attacks in 2023 [2]. In Q1/2023, Cloudflare observed an increase in hyper-volumetric attacks, peaking above 71 Mrps [3]. Volumetric attacks like DNS amplification, SYN flood, GRE flood, and other TCP/UDP floods are the most popular [3], [4].

In general, defense against cyberattacks spans three high-level objectives – prevention, detection, and reaction [5]. It is well-known that perfect prevention cannot be achieved, whereas the reaction implicitly assumes that the attack has already happened [6]. In this light, attack detection is crucial in successfully reacting toward harm minimization or attack deflection (mitigation).

Attack detection is performed by Intrusion Detection Systems (IDSs), which monitor the activity of a protected system, intending to identify its potential unauthorized use, misuse, or abuse [7]. Identified malicious activities are collected centrally and presented to a security officer. Afterward, either automated or manual reaction can take place.

Most notable aspects of IDSs [8] include: I) Information source, specifying whether network traffic (Network IDS – NIDS) or end-host data are analyzed; II) Detection approach, defining a detection principle – known signatures or deviations from a norm; III) Detection time, putting constraints on the detection delay; and IV) Response type, as whether the system actively participates in attack mitigation after its detection.

We present Windower, a feature-extraction method for real-time network-based intrusion (particularly DDoS) detection at the network perimeter. It processes packet headers and utilizes stream data mining and windowing techniques to extract relevant traffic statistics. We designed Windower to detect flooding attack and periodicity patterns, but it also proved efficient against low-rate DoS. To demonstrate its capabilities, we employed KitNet, an autoencoder ensemble from the Kitsune [9] NIDS, as an anomaly-based attack discriminator and simulated both detection and mitigation scenarios.

As most current machine learning (ML)-based NIDS research implicitly assumes off-line scenarios [10], [11], we specifically aimed to propose a practical and reliable method for real-time attack detection and consequent mitigation. It also allows stream learning, recommended for real-time cybersecurity solutions [12]. The paper's main contributions are:

- We propose a packet-based sliding window method[1] to compute traffic statistics from packet headers using stream data mining, suitable for real-time intrusion detection and mitigation.
- We introduce a specific feature set particularly for volumetric DDoS but also prove its ability to detect low-rate DoS attacks. Moreover, features are detector-independent, allowing both misuse- and anomaly-based approaches.
- We demonstrate Windower's performance in terms of detection rate and latency on several public datasets (CAIDA, CTU-13, SUEE-2017, UNSW-NB15) and compare it to the state-of-the-art packet-based NIDS.

[1]Code publicly available at: https://github.com/xGoldy/Windower

This paper firstly looks at problems of flow-based NIDS for real-time detection in Sec II. We further discuss related work in Sec III and method design in Sec. IV. Experimental results are presented in Sec. V, followed by rigorous discussion in Sec. VI. Finally, Sec. VII concludes the paper.

II. Flow-Based DDoS Detection Pitfalls

The current mainstream way for ML-based DDoS detection predominantly utilizes network flows. This fact is mainly attributed to the wide availability of flow-based datasets, e.g., CIC-IDS2017 [13], CIC-DDoS2019 [14], and ISCX2012 [15], currently considered the most popular for research [10]. Such approaches typically achieve detection rates above 99% [10]. However, in addition to losing information about packet payloads, flow-based detection suffers from two more defects:

1) Network flows provide no communication context.
2) Detection delays occur due to flow creation mechanism.

A network flow captures only a single data exchange between a client and a server, providing no information context about the client's previous communication. Therefore, some attacks using port randomization (e.g., tools HOIC or XOIC incrementing source ports) or port scans create a separate flow for each attack packet. The capabilities of flow-based methods without flow correlation are thus significantly hampered.

Flow-based methods suffer from delays due to creation and export intervals. Typically, a flow terminates after observing a TCP FIN flag or when a timeout occurs. However, attacks like SYN Flood do not carry FIN flags, so flows must be terminated upon timeout. Moreover, observed flows are exported in bulks, so dozens of seconds might pass until the flow data arrives at a flow collector. Afterward, a NIDS still needs to access the collector to retrieve the entries, adding additional delay.

Although the network flow collection is widely deployed and well-matured, it was created with the aim of monitoring and not directly for cyberthreat detection. Therefore, flow-based attack detection might be unsuitable if a near-instant reaction to the attack is required, as it can be significantly delayed, and additional post-processing might be required.

As a possible solution, we propose a packet stream method to compute statistics via a sliding window aggregated based on the source IP. In such a way, it provides the client's communication context by design, thus avoiding the necessity of flow correlation. Since no export must occur, the detection delay is minimized to a few seconds after the attack begins.

III. Related Work: Real-Time ML DDoS Detection

Real-time attack mitigation is the main design goal of practical anti-DDoS solutions [16]. Delays of dozens of seconds or even minutes might thus be unacceptable. In general, there are three main approaches to speed up the detection process: 1) *Window aggregates*, 2) *Per-packet processing*, 3) *Subflows*.

A. Window Aggregates

As the whole data stream cannot be processed at once, windowing splits a (potentially infinite) data stream into logical subsets (windows). Applying operations separately to each window allows for partial stream analysis. Thus, it enables to detect and react upon events (e.g., attacks) in a timely manner.

Time-based windows are based on timestamps, whereas *count-based* windows rely on the order of processed elements (packets/flows) as they arrive from the network.

Vijazasarathy [17] detects DDoS using TCP flags and durations of count-based windows with the Naive Bayes classifier. Similarly, Mousavi and St-Hilaire [18] proposed a DDoS detection for SDN based on the entropy of destination IP addresses within five 50-packet windows and thresholding. Based on Jensen-Shannon Divergence, Bhandari [19] differentiates DDoS attacks and flash events within short 1-second windows.

Aggregates can also be created upon flows. For instance, GEE [20] aggregates flows per source IP inside 3-minute windows. Although such methods achieve relatively high accuracy, timely attack detection is degraded due to NetFlow properties (Sec. II) along with too long window lengths.

B. Per-Packet Processing

Per-packet NIDSs evaluate each packet within the attack discriminator separately. Such systems utilize statistical techniques like n-grams [21] or signatures (e.g., Snort [22]) to search specific patterns within packets. Despite the signatures' efficiency, the research has moved towards ML, especially Recurrent Neural Networks (RNNs), due to their ability to maintain context. DeepDefense [23] uses a bi-directional LSTM-RNN with the last 100 packets, achieving $\sim 98\%$ accuracy and a 1-2% error rate. LSTM-BA [24] improved its performance by combining an RNN with a Naive Bayes classifier.

Per-packet methods are sometimes combined with windowing for additional context. Kitsune [9] uses per-packet feature extraction and KitNET, an ensemble of autoencoders – unsupervised artificial neural networks, for anomaly detection. It keeps the communication context for channels (conversations between two hosts) using five damped time windows, computing 115 features per packet. Chronos [25] employs the same feature extraction as Kitsune, but their handling and model architecture differ. Finally, Doshi et al. [26] combine stateful window along with stateless packet-header features.

Although the discussed methods achieve promising results and Kitsune sparked great interest within the community, they still require evaluating every packet in the detection model. Despite sufficient for smaller-scale local networks, the per-packet approach might be infeasible for larger networks with only a few dozen nanoseconds to process a packet available.

To enable DDoS detection and mitigation in high-speed networks, we argue that one must classify at higher-level abstraction, i.e., windows or network flow aggregates. As outlined in Sec. II, flow-based methods might add additional delay. Therefore, window-based methods seem to be a promising research direction for real-time detection purposes.

C. Subflows

A subflow is formed by the same 5-tuple (IPs, ports, protocol) as a regular flow but is restricted by the number of packets and/or its duration. Therefore, only the first n

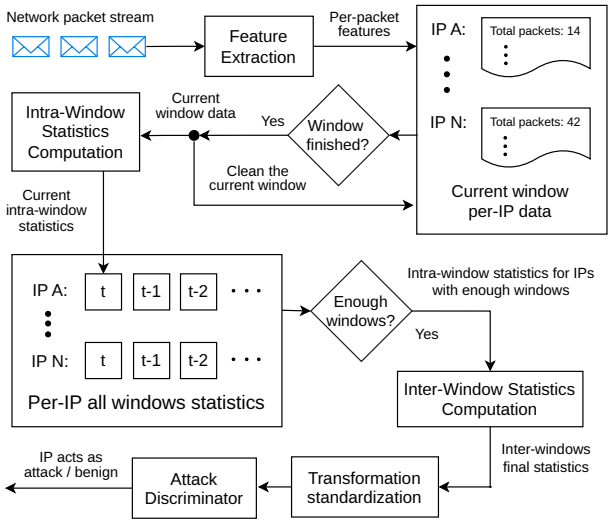

Fig. 1: Architecture of the proposed DDoS detection system.

packets of the flow are examined, or it is terminated after a certain time. These approaches are plausible since they can utilize existing NetFlow architecture while still speeding up the detection. Nevertheless, the client's communication context is still missing, requiring additional post-processing (e.g., as in the NeMo DDoS software [27]) for better performance.

AE-D3F NIDS [28] uses an autoencoder to detect DDoS via features like packet/byte counts, packet sizes, and TTLs based on 10-second bi-directional subflows. Similarly, Lucid [29] aggregates packets within a subflow and performs detection using a Convolutional Neural Network (CNN). Liang and Znati [30] classify the whole flow based on its first N packets via an LSTM model. These designs are less computationally demanding than per-packet approaches but still might be insufficient for real-time operation. Another drawback is that these methods ([29], [30]) expect an input of a certain length, so dummy packets are needed for less than N-packet flows.

IV. System Design

Our main design objective was to provide a lightweight method aiming for the best trade-off between attack mitigation metrics and detection timeliness. We thus had to consider: I) providing attack-distinguishing features in a timely manner and II) evaluating such features quickly enough to keep up with the network. Despite achieving similarly high detection rates [31], flow-based data are not always sufficient for timely detection, whereas processing every packet might be infeasible in large-scale networks. Therefore, we opted to employ sliding-window statistics aggregated per source IP.

Since we aggregate the data based on the source IP address, we expect the same-source traffic to be routed via the same path so it can be captured. Actually, the detection method only examines incoming (potentially malicious) traffic and does not care about the responses, thus monitoring the packet-wise traffic in an uni-flow manner. Relying only on ingress traffic effectively decreases the detection latency, resource requirements, and relaxes the constraints on traffic routing.

Windower is depicted in Fig. 1. Firstly, we extract header values from each packet (Tab. I). They are used to compute the current window statistics, which we aggregate by their source IP. After a specified amount of windows is reached, window-wise statistics are summarized, and standard deviations are computed between windows (Sec. IV-A). Such summaries (Tab. II) are fed into a model to determine whether the host resembles anomalous behavior (Sec. IV-C). After detection, attack mitigation is launched.

A. Feature Extraction and Statistics Computation

As illustrated by Fig. 1, our method is designed to process a raw network data stream. However, instead of processing every packet within the attack discriminator, we extract relevant packet features and compute communication statistics for each source IP address using summarized sliding windows for attack detection. This process is divided into several steps:

1) Packet Feature Extraction: We utilize only packet headers (Tab. I) instead of payloads, making the method application-level independent and unaffected by encryption. Features can characterize various DDoS attacks with temporal and packet sizing patterns different from benign traffic.

2) Sliding Window Per-Source Aggregation: We aggregate packet features by their source IP address inside a sliding window of t seconds. Aggregation per source IP creates a natural communication context, providing rigorous patterns of a single host across multiple network flows. No additional correlation between network flows is hence required.

Aggregation within the current window is done by counts, sums, averages, standard deviations, minimums, maximums, unique counts, and an entropy estimation for every unique IP in the window (Tab. II). As the mechanism might need to process dozens of gigabits per second, it would be infeasible to store extracted features from all packets within each window.

Counts, sums, minimums, and maximums can be computed on the go by a simple update. However, computations of regular average and variance assume that all the samples are available at the time of the computation. For this reason, we utilize data stream (running) algorithms to compute such statistics without storing samples. The streaming mean is computed using Eq. 1, and we use Welford's algorithm [32] for variance. It maintains an auxiliary value s_i updated for every element (Eq. 2) along with running mean $\overline{x}_i$ (Eq. 1). As the window finishes, the variance estimate is computed using Eq. 3, and the standard deviation then simply as $\sigma{=}\sqrt{s^2}$.

$$\overline{x}_n = \overline{x}_{n-1} + \frac{x_n - \overline{x}_{n-1}}{n} \quad (1)$$

$$s_n = s_{n-1} + (x_n - \overline{x}_n)\cdot(x_n - \overline{x}_{n-1}) \quad (2) \qquad s^2 = \frac{s_n}{n-1} \quad (3)$$

Computing the unique source port count would normally be trivial via the cardinality of a set. Nevertheless, sets require

TABLE I: List of all extracted packet features.

# **Packet Features** (*I=Integer, S=String, B=Boolean*)		
1. Timestamp (I)	5. Destination IP (S)	7. Headers length (I)
2. Source IP (S)	6. Destination port (I)	8. Payload length (I)
3. Source port (I)	4. L4 protocol (I)	9. IP fragmented (B)

TABLE II: List of all intra- and inter-window statistics.

#	Intra-Window Statistics	
1.	*src_ip*	IP address for the corresponding statistics
2.	*window_count*	Number of summarized windows
3.	*window_span*	The first and the last window ID difference
4.	*pkts_total*	Total number of packets
5.	*bytes_total*	Sum of bytes of all packets
6.	*pkt_rate*	Estimate of a packets per second (pps) value
7.	*byte_rate*	Estimate of a bytes per second (bps) value
8.	*pkt_arrivals_avg*	Average time between packet arrivals
9.	*pkt_arrivals_std*	Standard deviation between packet arrivals
10.	*pkt_size_min*	Overall minimum packet size
11.	*pkt_size_max*	Overall maximum packet size
12.	*pkt_size_avg*	Average of packet sizes
13.	*pkt_size_std*	Standard deviation of packet sizes
14.	*proto_tcp_share*	TCP traffic share
15.	*proto_udp_share*	UDP traffic share
16.	*proto_icmp_share*	ICMP traffic share
17.	*port_src_unique*	Number of unique source ports
18.	*port_src_entropy*	Source port entropy
19.	*conn_pkts_avg*	Average number of socket-to-socket transfers
20.	*pkts_frag_share*	Fragmented packets share
21.	*hdrpkt_ratio_avg*	Average of header to packet size ratio

#	Inter-Window Statistics (*Std=Standard Deviation*)	
22.	*pkts_total_std*	Std of a total number of packets
23.	*bytes_total_std*	Std of a total number of bytes
24.	*pkt_size_avg_std*	Std of packet size averages
25.	*pkt_size_std_std*	Std of packet size stds
26.	*pkt_arrivals_avg_std*	Std of average time between packet arrivals
27.	*port_src_unique_std*	Std of number of unique source ports
28.	*port_src_entropy_std*	Std of source port entropy values
29.	*conn_pkts_avg_std*	Std of packet count per connection averages
30.	*pkts_frag_share_std*	Std of fragmented packets share
31.	*hdrpkt_ratio_avg_std*	Std of header to whole packet ratios
32.	*main_proto_ratio_std*	Std of ratio of the dominant L4 protocol
33.	*intrawin_activity_ratio*	IP estimate within the windows
34.	*interwin_activity_ratio*	IP estimate during the period

saving every unique element in the memory. Instead, we rely on HyperLogLog (HLL) [33], a probabilistic structure for reducing space demands. It cannot give a definite answer but provides an approximation within some maximum error range [34]. Using HLL, we achieve a standard error of 4.6% with less than 1 kB of memory per single window entry.

Lastly, we compute the client's source port entropy (the amount of randomness). Although several techniques for calculating streaming entropy exist [35], [36], we opted for regular Shannon Entropy [37] computed from a sample of $n{=}40$ observed source ports obtained by Reservoir sampling [38].

3) Intra-Window Statistics: After the current window finishes, a new one is started. However, some information is not used as collected, but additional (intra-window) statistics are computed (Tab. II). Windows are considered valid when they contain at least p packets. Such a restriction aims to reduce statistical noise introduced by small sample sizes. This phase also computes running variances and other statistics, like the average number of packets per connection.

4) Window Summarization and Inter-Window Statistics: Using only one window might not properly capture the variability of communication over time. For instance, flash events resemble identical characteristics to a DDoS [39]. When a flash event is captured by a single window, its features might look similar to an attack. To make Windower more robust, we utilize multiple windows to describe communication patterns more reliably. The functionality is achieved by 1) summarizing multiple windows and 2) computing additional inter-window statistics (Tab. II). Similarly to the minimum packet limit, we set the minimum window count w for noise reduction.

After collecting w windows, the features are summarized via arithmetic means over available per-IP window statistics. We compute TCP, UDP, and ICMP traffic shares, as well as the header-to-payload length ratio. For specific attacks, e.g., SYN flood, the attacker sends packets without payload. These features can thus help to reveal noticeable attack patterns.

Inter-window statistics are computed as standard deviations of corresponding statistics across windows. Their rationale is that regular benign traffic would likely have a substantial variance over machine-generated attacks. Attack tools usually limit an attack to a specific bitrate, achieving high uniformity, thus leading to low variance across multiple time windows. The premise might be incorrect for pulsing DDoS [40] or benign streaming services. Nevertheless, we compensate for it with other features (packet size analysis and packet/byte rates).

We also estimate the client's intra- and inter-window activity during the analyzed period. Intra-window activity is based on a communication gap at the start and end of the window (Eq. 4). Inter-window activity is given as a ratio of the summarized window count to all possible windows within the range (Eq. 5). Due to the p packets window validity requirement, a client might not produce enough data to fill in the window, whereas a flooding attack should have both values close to 1.

$$a_{\text{intra}} = \frac{t_{\text{last-pkt}} - t_{\text{first-pkt}}}{t_{\text{win-len}}} \quad (4) \qquad a_{\text{inter}} = \frac{\#_{\text{windows-summarized}}}{\text{ID}_{\text{last-win}} - \text{ID}_{\text{first-win}}} \quad (5)$$

5) Statistics Preprocessing: Statistics computed in the previous phase require preprocessing before being fed into the attack discriminator. Although the source IP is important for alerting and mitigation, we drop it before classification to prevent evaluation bias [41]. We further drop features *window_count* and *window_span*, which can be useful during postprocessing to adjust the decision confidence but are not helpful during the classification itself. Therefore, 31 features to process within the detection model are left. Since all are numerical, no encoding is required. Nevertheless, we rescale them with z-score normalization to maximize the performance.

B. Hyperparameters Summary

In summary, we can tune the following parameters influencing the method's performance:

- *Window length* (t): Length of a sliding window in seconds.
- *Packet count* (p): We consider a window valid only if a certain source host sent at least p packets within it.
- *Window count* (w): The feature vector is finalized once at least w windows are collected in the last v seconds.
- *Window validity* (v): Windows older than v seconds are outdated and not considered for statistics computation.

As a consequence of our design, an attack can only be detected in $t \cdot w$ seconds, supposing that each window has at least p packets and no window got invalidated due to exceeding

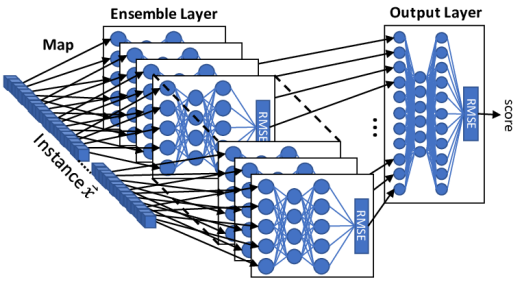

Fig. 2: KitNet [9] architecture, the ensemble of autoencoders.

maximum validity time v. Longer windows are considered more reliable (decreasing false positives), while shorter allow faster detection. Packet count p assures reliable intra-window, while window count w ensures reliable inter-window statistics.

C. Attack Detection

For a demonstration of the attack-detection capabilities, we employed KitNet [9]. However, the extracted features can be utilized by both misuse- and anomaly-based detectors.

KitNet (Fig. 2) is an ensemble of k three-layer autoencoders (AEs). It is a part of Kitsune [9], a state-of-the-art anomaly-based NIDS for online attack detection. Each ensemble AE measures the abnormality of its respective subspace $\vec{v}_i$, given by input $\vec{x}$ mapping f into $k-1$ subspaces, producing a Root Mean Squared Error (RMSE) value. RMSEs are then led to the autoencoder at the output layer, giving a final RMSE value.

During training (with benign only), the model first learns the feature mapping function f using agglomerative hierarchical clustering. In our case, $\vec{x}$ is a vector of 31 elements. Individual autoencoders then learn distributions of their respective subspaces $\vec{v}_i$, and the output layer AE learns the relationship between subspace abnormalities and naturally occurring noise.

The evaluation phase uses learned mapping f and individual autoencoder distributions to produce the final RMSE score. We compare it with the predefined threshold τ, $\mathrm{RMSE} > \tau$ representing an anomaly. During mitigation, we correlate anomalies with the acting source IP to drop its packets. Further KitNet details can be found in the original Kitsune paper [9].

V. Evaluation

We evaluate Windower using four packet-based datasets. This section describes our setup and analyzes the performance compared with Kitsune [9]. We also analyze the performance by varying windowing configurations and discuss the detection speed, KitNET's complexity, and the runtime performance.

A. Utilized Datasets

The selection of a suitable dataset was non-trivial as the NIDS domain suffers from a long-term shortage of standardized quality datasets [6], [42], [43]. Although various datasets were released recently [44], none cover a wide variety of DDoS attacks and provide packet-based data suitable for our purposes. Datasets are very limited in the attack scope or the number of attacking hosts. For instance, CIC-DDoS2019 [14] provides various DDoS attacks, but all its attack traffic comes from a single IP address. Since we perform aggregation based on source IPs, we could not use it because it produced too few aggregated window entries, insufficient for reliable evaluation.

As no dataset is perfect [45], we used multiple ones to evaluate our work in different scenarios and thus strengthen the achieved results and stated claims. When discussing the CAIDA dataset, we refer to a custom mix of the CAIDA DDoS Attack 2007 dataset [46] and the CAIDA Anonymized Internet Traces [47]. Since the DDoS dataset contains only attack traffic, we mixed it with benign traffic from CAIDA Traces. CTU-13 [48] is a dataset of real botnet traffic. We extracted only DDoS from the scenario 45, excluding any C&C traffic. Similarly, we extracted only packets corresponding to the DoS traffic class along with a subset of normal traffic from UNSW-NB15 [49]. Former datasets consist primarily of flood attacks. In order to test the performance against slow DoS attacks as well, we used the SUEE8 variant of the SUEE-2017 [50].

For our purposes, we extracted only the forward direction (targeting the protected network) without responses and created a train set (only benign) and a test set (both benign and malicious traffic) for each dataset, assuming distinct attacking and legitimate IP addresses. We thus label the datasets via attackers' IPs. The number of packets, along with their class affiliation, is given in Tab. IV. Full details about the dataset compilation process are provided on our GitHub (Sec. I).

B. Experiments Setup

We implemented Windower in Python to process the discussed datasets. The evaluation utilized a full feature set – 31 features after preprocessing (Sec. IV-A). A fair comparison with Kitsune was achieved using the same attack detector – KitNET, with equal parameters, i.e., m=10 (the maximum number of inputs for each AE) and the same feature mapping procedure. Performance comparisons of our system with Kitsune are based on its original Python implementation [9]. During training, we always use 10% of the training set for feature mapping. Unless stated otherwise, we set the Windower's parameters as follows: t=1, p=10, w=6, and v=120, i.e., aggregate features across six 1-second long windows.

We executed the experiments on a single CPU core (Intel Xeon E5-2630v3) and 64 GB of RAM. Given modern NICs support of distributing packets into multiple receive queues processed by separate cores, the implementation can be easily parallelized and accelerated in the future.

C. Per-Packet Mitigation Analysis

We evaluate detection capabilities using a simulated mitigation process based on RMSE scores. Fig. 3 shows the RMSE scores assigned to each packet in the CAIDA dataset for both Kitsune and Windower. We use different colors to distinguish scores of benign (green) and attack (red) packets. Each data point in the figure represents a single packet classification.

For Kitsune (Fig. 3a), the output layer produces the RMSE score for each packet as an implicit result of the per-packet processing approach. We can compare this score with a predefined threshold τ to decide whether to drop or pass the packet.

TABLE III: Comparison of mitigation performance using TPR for FPR={0.002, 0.005, 0.01, 0.02, 0.05}.

Dataset	FPR=0.002		FPR=0.005		FPR=0.01		FPR=0.02		FPR=0.05	
	Kitsune	Windower	Kitsune	Windower	Kitsune	Windower	Kitsune	Windower	Kitsune	Windower
CAIDA	0.00000	0.98964	0.00000	0.98964	0.00000	0.98964	0.00000	0.98964	0.00000	0.98964
CTU-13	0.00000	0.99859	0.00000	0.99859	0.00000	0.99859	0.00000	0.99859	0.00000	0.99859
UNSW-NB15	0.54952	0.57073	0.60907	0.66585	0.73372	0.73230	0.81191	0.86095	0.85211	0.97638
SUEE-2017	0.07270	0.22996	0.23747	0.36122	0.33350	0.91729	0.48011	0.91729	0.92678	0.91729

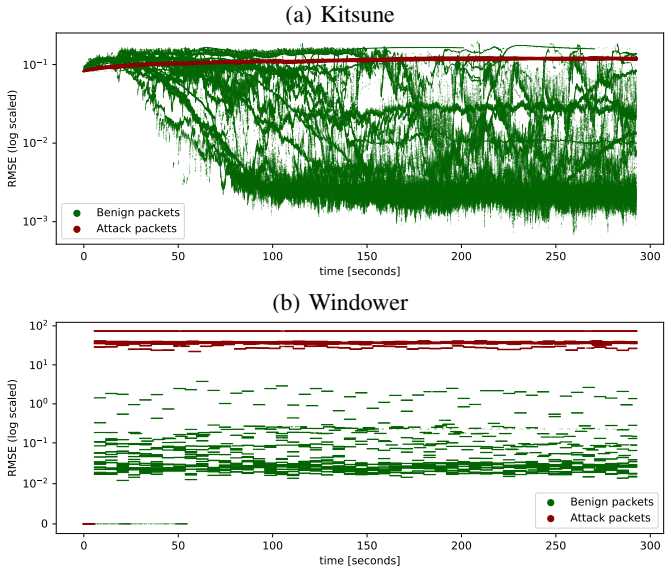

Fig. 3: Visualization of the per-packet Root Mean Square Error (RMSE) values evaluated using the CAIDA dataset.

Unlike Kitsune, Windower (Fig. 3b) always produces an RMSE score for a specific source IP address and its corresponding feature vector based on an aggregated set of time window statistics. To plot Fig. 3b and to fairly compare both solutions, we thus assign RMSE scores to packets during postprocessing according to the latest source IP address classification. In practice, we would directly compare the score of each source IP address classification with τ to immediately decide whether to drop or allow all its packets. For this reason, in Fig. 3b, the RMSE values for individual packets create 6-second segments (lines) as we are dealing with the time-based source IP aggregated windows.

The approach using source IP window-based classification will always introduce a small number of false negatives by design. In the first 50 seconds of CAIDA, in Fig. 3b, one can see several packets assigned zero RMSE score. These packets will be allowed due to not having enough collected data (at least six 1-second time windows) to classify their source IP address, so their RMSE value is undefined. Unless stated otherwise, we always include such false negatives caused by detection delays in the performance evaluation metrics.

As evident from Fig. 3, Windower outperforms Kitsune on the CAIDA dataset. For Windower, we can easily find a threshold, e.g., τ=10, that ideally separates legitimate (green) and attack (red) traffic using the RMSE scores. In contrast to Kitsune, we can only find such τ by introducing a non-negligible number of false positives or negatives.

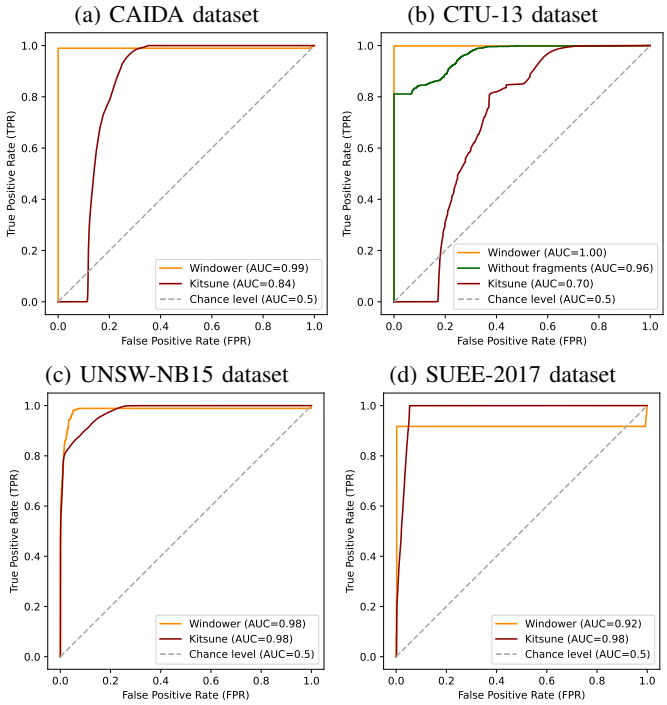

Fig. 4: Mitigation performance depicted using ROC curves.

D. Mitigation Performance

We investigate the method's performance regardless of a specific τ using an ROC (Receiver Operating Characteristics) curve. It plots true (TPR) versus false positive rates (FPR) with regard to increasing threshold. The numerical representation of the classifier's performance is given by the Area Under the ROC Curve (AUC), i.e., AUC=1.0 being a perfect score.

Fig. 4 compares Windower (orange) against Kitsune (red) in terms of mitigation performance using ROC curves. In cases of CAIDA and CTU-13, Windower significantly outperforms Kitsune. For both datasets, Kitsune introduces a substantial amount of false positives (above 30%) to detect a similar number of malicious packets as Windower. From the AUC score perspective, Windower achieves AUC=0.99 for CAIDA and AUC=1.00 for CTU-13, respectively, while Kitsune reaches only AUC=0.84 for CAIDA and AUC=0.70 for CTU-13.

In contrast, the mitigation performance is more or less comparable for UNSW-NB15 and SUEE-2017. Regarding the AUC score, both methods achieved AUC=0.98 for UNSW-NB15, and Kitsune (AUC=0.98) even outperformed Windower (AUC=0.92) for SUEE-2017. However, Windower still introduces fewer false positives with significantly fewer computational requirements (Sec. V-F). We give detailed TPR/FPR results in Tab. III. In the DDoS mitigation case, we are more

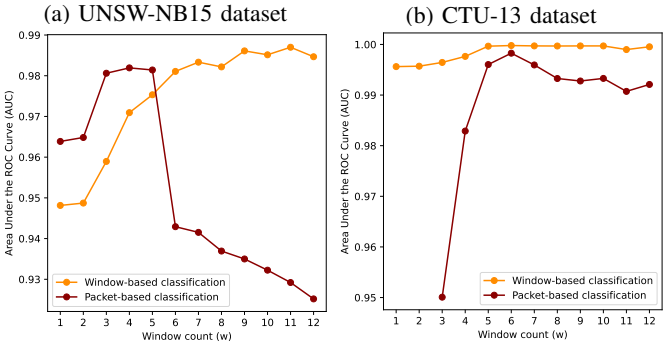

Fig. 5: Windows count (w) analysis using AUC.

concerned with false positives than false negatives. We thus claim that Windower still achieves decent performance for slow attacks with meaningful benefits over Kitsune.

Analysis of CTU-13 revealed that a significant portion of attack traffic is fragmented. For this reason, we analyzed Windower's performance with and without two fragmentation-related features. As seen in Fig. 4b, the performance without fragmentation (green) achieves an AUC of 0.96. With fragmentation, even AUC=1.00 is achieved. Nevertheless, both variants significantly outperform Kitsune (AUC=0.70).

E. Window Count Impact: Detection Delay vs. Accuracy

Fig. 5 investigates the impact of the number of collected windows on the Windower's performance. It shows the AUC score (y-axis) for the UNSW-NB15 and CTU-13 datasets with the varying parameter w – the minimum number of collected windows. We present the score from two different viewpoints:

1) *Window-based classification* (orange) shows the performance without the impact of false negative packet classification (discussed in Sec. V-C) caused by the delay in collecting the minimum number of windows. It reveals the pure success rate of source IP address classifications (benign vs. attack source). The more windows we use (the more statistics we have collected), the higher the classification accuracy.

2) *Packet-based classification* perspective (red) considers the impact of the delay in collecting the specified number of windows. For instance, Windower performs best if w=5 for UNSW-NB15 and w=6 for CTU-13, respectively. Indeed, the more windows we use (the more time we need for collection), the higher the false negative rate effect. The parameter w generally enables fine-tuning the trade-off between detection accuracy and its delay in real-time mitigation.

F. Runtime Performance

Tab. IV shows the runtime performance and compares Windower with Kitsune for all datasets. As apparent, Windower outperforms Kitsune in both training and evaluation runtimes. For illustration, Kitsune needs 83.1 min. for training, while the Windower needs only 14.1 min. (∼6x faster) using CAIDA. In some cases, the speed-up is even more significant (an order of magnitude) for the CTU-13 and SUEE-2017 datasets.

Although Kitsune relies on the ensemble of small autoencoders to increase its speed, it must still evaluate each incoming packet. In addition, the computational complexity of the ensemble itself grows with the number of features. Kitsune's feature extractor derives more than a hundred features for each packet to maintain its context aggregated based on source IP address, as well as channel and socket perspectives.

In contrast, Windower reduces the feature count (only 31) by using statistical features computed across multiple windows. A smaller feature set results in fewer ensemble autoencoders to train and evaluate. Tab. IV also presents the number of autoencoders for individual datasets (*#AEs*). As demonstrated, Windower requires up to 2-3x less AEs.

More importantly, we substantially lower the number of ensemble evaluations (*AE evals*). For Kitsune, it coincides with the packet count as it works on a per-packet basis, e.g., ∼3M AE evaluations using CAIDA. On the other hand, Windower classifies window-based feature aggregates and blocks packets based on source IP addresses. Using the same dataset, it needs no more than 3k evaluations (three orders of magnitude less). For this reason, Windower is fundamentally faster and still offers similar, or even better, mitigation performance.

VI. Discussion

In this section, we discuss possibilities of advanced mitigation, elaborate on real-time operation, and outline probable adversarial behavior and attacks against the system itself.

A. Intelligent Mitigation

In the previous section, we demonstrated the mitigation capabilities using an Access Control List (ACL) mechanism. If an anomalous IP is determined, it is denylisted, and all its consequent traffic is explicitly dropped. We further suggest performing ACL filtering before the Windower processing itself. This can save resources by discarding data from potentially malicious clients using hardware, hence not analyzing the traffic of denylisted hosts. After a certain period, the denylist entry expires, and the host's traffic is allowed again.

More advanced mitigation can be achieved by signature or rule derivation. For instance, if Windower detects a host acting anomalously, it could sample its future traffic. Afterward, algorithms based on string matching [51], state machines [52], string patterns or semantic conditions [53], or AI [54] can be applied to fill up the database of mitigation rules.

Assuming that the same malicious tool generated the attack, there should be similarities in its traffic. If an attacker had not utilized any obfuscation (discussed in Sec. VI-C), the created rules should generalize and block the traffic even from clients not analyzed by the attack detection mechanism.

B. Real-Time Performance

Windower aims to provide reliable real-time DDoS attack detection – a trade-off between detection performance and delay. Per-packet methods (hypothetically) offer the best timeliness, as we know whether to drop or pass a packet instantly. However, deep ML models are too complex to process every packet and meet throughput demands. We thus investigated how to remove the model evaluation from the data path and

TABLE IV: Comparison of Windower's and Kitsune's runtime performance.

Dataset	# packets			Kitsune runtime performance				Windower runtime performance			
	trainset	*evalset (attack/all)*		*train* $[min]$	*eval* $[min]$	*# AEs*	*AE evals*	*train* $[min]$	*eval* $[min]$	*# AEs*	*AE evals*
CAIDA	2 003 716	944 764	3 067 177	83.1	76.4	31	3 067 177	14.1	23.6	9	2 789
CTU-13	21 498 729	190 859	18 807 679	2 464.9	6 538.8	16	18 807 679	197.4	178.6	8	2 675
UNSW-NB15	3 179 605	140 360	10 126 614	80.1	150.7	15	10 126 614	26.8	90.5	13	15 182
SUEE-2017	104 313	15 694	291 384	22.6	148.9	22	291 384	1.2	3.2	14	6 812

limit the number of its evaluations to allow for high packet processing speeds and keep up with the incoming traffic.

Our current Windower's implementation as a single Python process achieves a throughput of $\sim$2 kpps. However, our design enables almost linear parallelization via source IP hashing along with separating windowing and attack detection. In such a scheme, the data plane handles feature extraction and intra-window statistics computation. In contrast, the control plane finalizes the streaming and inter-window statistics and runs the attack discriminator upon finishing the window. The model evaluation is thus removed from the data path, speeding up the overall throughput, while the detection delay is controlled via windowing parameters – the window length and the minimum number of summarized windows.

We highlight that Windower's deployment cost is also significantly lower by not evaluating every packet in the attack discriminator. The proposal thus brings a suitable practical trade-off for precise attack detection with regard to computing resources by redesigning only the data preprocessing.

C. Adversarial Considerations

While Windower achieves promising results and has several strengths, it is still not entirely foolproof to various adversarial techniques – most notably, source IP randomization. Below, we examine variants depending on the adversary's knowledge.

Black-box adversaries lack the system knowledge and thus cannot tailor attacks to overcome the detection mechanism. This is a common assumption for DDoS, which relies on an overwhelming traffic volume rather than attack sophistication. Adversaries often use well-known flooding or slow DoS [55]. Despite being designed against flooding attacks, our method shows potential in detecting slow DoS attacks as well.

Our experiments demonstrate the effectiveness against flooding, sharing common characteristics like increased request rates, specific packet timing, or sizing. Since we do not analyze upper protocol headers, this also applies to multi-vector attacks. Adaptive adversaries may use obfuscation like varying packet sizes, inter-packet timings, or payload randomization. Although it could degrade detection capabilities, we argue that normal behavior could hardly be simulated so that some attack patterns would remain visible. Additionally, the need for obfuscation techniques increases the attacker's cost.

A significant threat to our method is source IP randomization. While the method can efficiently handle common source address spoofing (unless the IP stays the same), it struggles when each packet's source IP is randomized, leading to increased memory consumption and statistics computation issues. We address the memory concerns by limiting stored historical windows and by dropping inactive ones with little traffic. However, the computation issue cannot be handled at the method's level. No statistics can be computed if a per-source IP window contains only one packet. It needs to be tackled outside, e.g., by reverse path forwarding [56], [57] or a specific NAT setup to mitigate IP spoofing. Another way of solving the issue is by detecting an attack via correlation [58] or by dropping spoofed packets via derived rules (Sec. VI-A).

Grey-box adversaries possess some system knowledge. Although not typically considered for NIDS [59], we theorize that such adversaries might predict the usage of anomaly-based NIDS and windowing (common for real-time systems). They may then employ adversarial learning, i.e., gradually retrain the system by low amounts of attack traffic to decrease attack sensitivity [60], supposing used incremental learning to tackle concept drift [61]. The windowing can be challenged by pulsing DDoS, reducing efficiency for most systems with only a single window. Nevertheless, our mechanism utilizes multiple ones. Therefore, despite the pulsing effectively lowering the averages, the inter-window variance would stay higher and rather consistent, providing clues about anomalous activity.

White-box (full knowledge) scenarios are less likely as details of NIDSs are typically well protected [62], and much more devastating attacks such as data theft, malware, or ransomware would become more attractive options instead [6].

VII. Conclusions

This work has presented Windower, a feature extraction mechanism based on sliding window and stream data mining. The windowing mechanism first collects per-source IP statistics within a short time frame. They are further summarized with several previous windows to determine more reliable hosts' communication patterns and temporal variability.

Windower achieved promising results, mitigating 99% of malicious DDoS flooding traffic on two public datasets with only 0.2% of false positives. With 1% of false positives, it also successfully filtered 92% of low-rate DoS traffic. Our solution outperforms Kitsune [9] while improving the runtime performance by more than an order of magnitude, making our method more suitable for real-world scenarios.

Windower currently treats statistics from different IPs independently. Our future plans involve exploring correlations between statistics from various hosts, aiming to reduce false positives and enhance flash crowd events resistance. Additionally, enriching the feature set with application-level features might help to characterize modern L7 (D)DoS attacks better.

Acknowledgements

We thank the project e-INFRA CZ (LM2023054) granted by the Ministry of Education, Youth and Sports of the Czech Republic and Brno University of Technology (FIT-S-23-8141).

REFERENCES

[1] P. J. Criscuolo, "Distributed denial of service: Trin00, tribe flood network, tribe flood network 2000, and stacheldraht ciac-2319," *Department of Energy Computer Incident Advisory Capability (CIAC), UCRL-ID-136939, Rev*, vol. 1, 2000.

[2] Cisco Systems Inc., "Cisco Annual Internet Report (2018–2023) White Paper," Cisco Systems Inc., San Jose, CA 95134 USA, Tech. Rep., 2018, updated March 2020. Accessed 2023-09-03. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[3] O. Yoachimik and J. Pacheco, "DDoS threat report for 2023 Q1," Cloudflare, Inc., Tech. Rep., Apr. 2023, acessed 2023-09-03. [Online]. Available: https://blog.cloudflare.com/ddos-threat-report-2023-q1/

[4] O. Kupreev, A. Gutnikov, and Y. Shmelev, "DDoS attacks in Q3 2022," Kaspersky, Tech. Rep., Nov. 2022, accessed 2023-09-03. [Online]. Available: https://securelist.com/ddos-report-q3-2022/107860/

[5] H. Yang, H. Luo, F. Ye, S. Lu, and L. Zhang, "Security in mobile ad hoc networks: challenges and solutions," *IEEE Wireless Communications*, vol. 11, no. 1, pp. 38–47, 2004.

[6] G. Apruzzese, P. Laskov, and J. Schneider, "SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2023, pp. 592–614.

[7] B. Mukherjee, L. Heberlein, and K. Levitt, "Network intrusion detection," *IEEE Network*, vol. 8, no. 3, pp. 26–41, Jun. 1994.

[8] A. Thakkar and R. Lohiya, "A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions," *Artificial Intelligence Review*, vol. 55, no. 1, pp. 453–563, Jan 2022.

[9] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[10] M. Mittal, K. Kumar, and S. Behal, "Deep learning approaches for detecting DDoS attacks: a systematic review," *Soft Computing*, vol. 27, no. 18, pp. 13 039–13 075, Jan. 2022.

[11] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, Jun 2018.

[12] F. Ceschin, M. Botacin, A. Bifet, B. Pfahringer, L. S. Oliveira, H. M. Gomes, and A. Grégio, "Machine learning (in) security: A stream of problems," *Digital Threats*, sep 2023.

[13] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *4th International Conference on Information Systems Security and Privacy*, Jan. 2018, pp. 108–116.

[14] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8.

[15] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[16] A. Bhardwaj, V. Mangat, R. Vig, S. Halder, and M. Conti, "Distributed denial of service attacks in cloud: State-of-the-art of scientific and commercial solutions," *Computer Science Review*, vol. 39, p. 100332, 2021.

[17] R. Vijayasarathy, S. V. Raghavan, and B. Ravindran, "A system approach to network modeling for DDoS detection using a Naïve Bayesian classifier," in *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, 2011, pp. 1–10.

[18] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, 2015, pp. 77–81.

[19] A. Bhandari, K. Kumar, A. L. Sangal, and S. Behal, "An anomaly based distributed detection system for DDoS attacks in Tier-2 ISP networks," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 1, pp. 1387–1406, Jan 2021.

[20] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan, "GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection," in *2019 IEEE Conference on Communications and Network Security (CNS)*, 2019, pp. 91–99.

[21] K. Wang and S. J. Stolfo, "Anomalous Payload-Based Network Intrusion Detection," in *Recent Advances in Intrusion Detection*, E. Jonsson, A. Valdes, and M. Almgren, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 203–222.

[22] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99. USA: USENIX Association, 1999, p. 229–238.

[23] X. Yuan, C. Li, and X. Li, "DeepDefense: Identifying DDoS Attack via Deep Learning," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2017, pp. 1–8.

[24] Y. Li and Y. Lu, "LSTM-BA: DDoS Detection Approach Combining LSTM and Bayes," in *2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)*, 2019, pp. 180–185.

[25] M. A. Salahuddin, V. Pourahmadi, H. A. Alameddine, M. F. Bari, and R. Boutaba, "Chronos: DDoS Attack Detection Using Time-Based Autoencoder," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 627–641, 2022.

[26] R. Doshi, N. Apthorpe, and N. Feamster, "Machine Learning DDoS Detection for Consumer Internet of Things Devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 29–35.

[27] GÉANT Security, "Nemo ddos software," accessed 2024-01-11. [Online]. Available: https://security.geant.org/nemo-ddos-software/

[28] K. Yang, J. Zhang, Y. Xu, and J. Chao, "DDoS Attacks Detection with AutoEncoder," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[29] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del Rincón, and D. Siracusa, "Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 876–889, 2020.

[30] X. Liang and T. Znati, "A Long Short-Term Memory Enabled Framework for DDoS Detection," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[31] G. Olimpio, P. F. C. Silva, L. Camargos, R. S. Miani, and E. R. de Faria, "Intrusion detection over network packets using data stream classification algorithms," in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2021, pp. 985–990.

[32] B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[33] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm," in *AofA: Analysis of Algorithms*, ser. DMTCS Proceedings, P. Jacquet, Ed., vol. DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07). Discrete Mathematics and Theoretical Computer Science, 06 2007, pp. 137–156.

[34] A. Singh, S. Garg, R. Kaur, S. Batra, N. Kumar, and A. Y. Zomaya, "Probabilistic data structures for big data analytics: A comprehensive review," *Knowledge-Based Systems*, vol. 188, p. 104987, 2020.

[35] N. J. Harvey, J. Nelson, and K. Onak, "Sketching and Streaming Entropy via Approximation Theory," in *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, 2008, pp. 489–498.

[36] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data Streaming Algorithms for Estimating Entropy of Network Traffic," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '06/Performance '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 145–156.

[37] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[38] K.-H. Li, "Reservoir-Sampling Algorithms of Time Complexity O(n(1 + Log(N/n)))," *ACM Transactions on Mathematical Software*, vol. 20, no. 4, p. 481–493, Dec. 1994.

[39] S. Behal, K. Kumar, and M. Sachdeva, "Characterizing DDoS attacks and flash events: Review, research gaps and future directions," *Computer Science Review*, vol. 25, pp. 101–114, 2017.

[40] X. Luo and R. K. C. Chang, "On a New Class of Pulsing Denial-of-Service Attacks and the Defense," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005, San Diego, California, USA*. The Internet Society, 2005. [Online]. Available: https://www.ndss-symposium.org/ndss2005/new-class-pulsing-denial-service-attacks-and-defense/

[41] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 3971–3988.

[42] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, Oct. 2021.

[43] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 305–316.

[44] Z. Yang, X. Liu, T. Li, D. Wu, J. Wang, Y. Zhao, and H. Han, "A systematic literature review of methods and datasets for anomaly-based network intrusion detection," *Computers & Security*, vol. 116, p. 102675, May 2022.

[45] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, pp. 147–167, 2019.

[46] "The CAIDA UCSD "DDoS Attack 2007" Dataset," accessed: 2023-10-03. [Online]. Available: https://www.caida.org/catalog/datasets/ddos-20070804_dataset/

[47] "The CAIDA UCSD Anonymized Internet Traces," accessed: 2023-10-03. [Online]. Available: https://www.caida.org/catalog/datasets/passive_dataset/

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

[49] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, Nov. 2015, pp. 1–6.

[50] T. Lukaseder, "Github: 2017-SUEE-data-set," 2017, accessed: 2023-10-03. [Online]. Available: https://github.com/vs-uulm/2017-SUEE-data-set

[51] R.-T. Liu, N.-F. Huang, C.-H. Chen, and C.-N. Kao, "A Fast String-Matching Algorithm for Network Processor-Based Intrusion Detection System," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 3, p. 614–633, aug 2004.

[52] C. R. Meiners, J. Patel, E. Norige, E. Torng, and A. X. Liu, "Fast Regular Expression Matching Using Small TCAMs for Network Intrusion Detection and Prevention Systems," in *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. USENIX Association, 2010, pp. 111–126.

[53] P.-C. Lin, Y.-D. Lin, and Y.-C. Lai, "A Hybrid Algorithm of Backward Hashing and Automaton Tracking for Virus Scanning," *IEEE Transactions on Computers*, vol. 60, no. 4, pp. 594–601, 2011.

[54] M. Zadnik and E. Carasec, "AI infers DoS mitigation rules," *Journal of Intelligent Information Systems*, vol. 60, no. 2, pp. 305–324, Aug 2022.

[55] W. Zhijun, L. Wenjing, L. Liang, and Y. Meng, "Low-Rate DoS Attacks, Detection, Defense, and Challenges: A Survey," *IEEE Access*, vol. 8, pp. 43 920–43 943, 2020.

[56] D. Senie and P. Ferguson, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827, May 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2827

[57] K. Sriram, D. Montgomery, and J. Haas, "Enhanced Feasible-Path Unicast Reverse Path Forwarding," RFC 8704, Feb. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8704

[58] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, and F. Tang, "Discriminating DDoS Attacks from Flash Crowds Using Flow Correlation Coefficient," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 1073–1080, 2012.

[59] G. Apruzzese, P. Laskov, E. Montes de Oca, W. Mallouli, L. Brdalo Rapa, A. V. Grammatopoulos, and F. Di Franco, "The Role of Machine Learning in Cybersecurity," *Digital Threats*, vol. 4, no. 1, pp. 1–38, Mar. 2023.

[60] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, p. 3448–3470, Aug. 2007.

[61] C. Nixon, M. Sedky, and M. Hassan, "Reviews in Online Data Stream and Active Learning for Cyber Intrusion Detection - A Systematic Literature Review," in *2021 Sixth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2021, pp. 1–6.

[62] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, "Modeling Realistic Adversarial Attacks against Network Intrusion Detection Systems," *Digital Threats*, vol. 3, no. 3, feb 2022.