NetRadar: Enabling Robust Carpet Bombing DDoS Detection

Junchen Pan
Tsinghua University
pjc21@mails.tsinghua.edu.cn

Lei Zhang*
Zhongguancun Laboratory
zhanglei@zgclab.edu.cn

Xiaoyong Si
Tencent Technology (Shenzhen)
seansi@tencent.com

Jie Zhang
Tsinghua University
jie-zhan22@mails.tsinghua.edu.cn

Xinggong Zhang
Peking University
zhangxg@pku.edu.cn

Yong Cui*
Tsinghua University
cuiyong@tsinghua.edu.cn

***Abstract*—Carpet bombing attack, a growingly prevalent variant of Distributed Denial of Service (DDoS), floods multiple servers in the victim network simultaneously, minimizing per-flow malicious traffic throughput to evade detection. The aggregated malicious traffic overwhelms network access points (e.g., gateways), causing a denial of service. Moreover, advanced attackers employ application-layer attack methods to generate malicious traffic inconspicuous in both semantic and traffic volume, failing existing DDoS detection mechanisms. We propose NetRadar, a DDoS detector that achieves accurate and robust carpet bombing detection. Leveraging a server-gateway cooperation architecture, NetRadar aggregates both traffic and server-side features collected across the victim network and performs cross-server analysis to locate victim servers. To enable server-assisted carpet-bombing detection, a general server-side feature set compatible with diverse services is introduced, alongside a robust model training method designed to handle runtime feature mismatch issues. Furthermore, an efficient cross-server inbound traffic analysis method is proposed to effectively exploit the similarity of carpet bombing traffic while reducing computational overhead. Evaluations on real-world and simulated datasets demonstrate that NetRadar achieves better detection performance than state-of-the-art solutions, achieving over 94% accuracy in all carpet bombing detection scenarios.**

I. Introduction

Carpet bombing attack, a variant of Distributed Denial of Service (DDoS), recently shakes the balance between DDoS defenders and attackers [1]–[8]. By flooding numerous servers in the target victim network simultaneously, the carpet bombing attack evades DDoS detection with a lower per-flow malicious traffic throughput than that in traditional DDoS. Malicious traffic to numerous victim servers aggregates and overwhelms the network access points (e.g., gateways), making all online services in the victim network unavailable. Existing DDoS defense systems fail to locate carpet bombing victim servers precisely in real time, resulting in increased service delays and inefficient use of DDoS mitigation resources.

*Corresponding authors

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.242118
www.ndss-symposium.org

Since the carpet bombing attack strategy can be applied to existing botnets and attack scripts with little extra cost, this DDoS variant has quickly become a popular weapon for DDoS attackers. Reports from academia and industry have pointed out that the frequency of carpet bombing attacks is increasing [1]–[5], [8], with increases over 30% yearly [3].

Carpet bombing can be divided into two categories according to different per-flow malicious traffic throughput [1]. High-rate carpet bombing attacks are comparatively easier to detect as the number of victim servers is insufficient to disperse the per-flow traffic volume below benign traffic. Low-rate carpet bombing attacks, on the other hand, generate malicious traffic to numerous victim servers with a much lower per-flow traffic volume to evade detection. Although the aggregated traffic throughput anomaly is evident (as the attacker tries to disrupt benign services), individual attack flows are virtually indistinguishable from benign ones in throughput, which challenges the DDoS detection.

Among low-rate carpet bombing attacks, those based on application-layer attacks are most challenging. Unlike volumetric DDoS, application-layer attack traffic closely mimics legitimate service traffic in semantic characteristics [9]–[11]. As a result, during a low-rate carpet bombing attack launched with application-layer attack methods, the malicious traffic is indistinguishable from benign traffic in terms of both traffic volume and semantics. This renders semantics analysis [12]–[14] and heavy-hitter-based detection [12], [15] ineffective, and causes performance degradation in existing traffic analysis detectors [16]–[18]. Similar evasion strategies are used in link flooding attacks, demonstrating their effectiveness in bypassing defense [19].

Fortunately, the unique property of carpet bombing, which spreads malicious traffic, presents us with new opportunities. Firstly, victim servers can provide effective detection assistance during carpet bombing attacks. Victim servers can perform more detailed traffic monitoring and behavioral analysis based on application-layer information, such as decrypted packet payload, request URL, etc., which is particularly effective for low-rate application-layer carpet bombing detection. Besides, carpet bombing's distributed nature prevents individual servers from being overloaded, allowing for the deployment of sophisticated feature extraction and real-time monitoring. Secondly, the similarity between malicious flows is an effective carpet bombing indicator. Carpet bombing traffic to different servers shares similar features as they are typically generated by the same botnet and automatic scripts [7], [20]. This means that multiple victim servers simultaneously receive seemingly normal but highly correlated traffic. By analyzing traffic to multiple victim servers simultaneously (or cross-server analysis for short), we can exploit malicious traffic similarity and accurately identify victim servers.

Implementing server-assisted detection and cross-server traffic analysis presents several technical challenges. Firstly, leveraging server-side features and performing cross-server analysis requires a network-wide feature collection mechanism. A server-gateway cooperation architecture must be established to deploy these mechanisms with minimized cooperation cost. Secondly, victim servers run a variety of different online services with heterogeneous service information, which necessitates selecting general enough features to be compatible with most services. Additionally, the server-side features collection may be affected by the DDoS mitigation process during the attack, leading to detection performance degradation. Thirdly, leveraging cross-server analysis in the detection model introduces high computational overhead. The increased input size (features of multiple servers at each inference) incurs higher feature space complexity and requires a larger model to achieve sufficient detection accuracy, resulting in increased training cost and reduced inference throughput.

We address these challenges and propose NetRadar, a DDoS detector that achieves accurate and robust carpet bombing detection. In NetRadar, the central feature collection point and cross-server analysis model are deployed at the gateway to minimize feature transmission overhead while ensuring rapid detection and mitigation of evolving attacks. To enable general server-side feature extraction, we propose a set of features available in most services through a lightweight behavioral model of online services. We develop a robust server-assisted model training method to handle the runtime feature mismatch issue under the server-gateway cooperation architecture. As for efficient cross-server analysis, based on our analysis of set-structured properties of features from multiple servers in carpet bombing detection tasks, we leverage the permutation-equivariant model architecture to improve model scalability and design a sort-based group function specifically optimized for carpet bombing traffic similarity analysis.

To sum up, we make following contributions:

- A server-gateway cooperation architecture that aggregates the available traffic and server-side features across the network with minimized feature transmission overhead, enables cross-server analysis for carpet bombing detection, and coordinates with DDoS mitigation systems.
- A set of general server-side features that captures service access patterns for effective carpet bombing detection and a robust model training method that handles runtime feature mismatch issues due to traffic scrubbing.

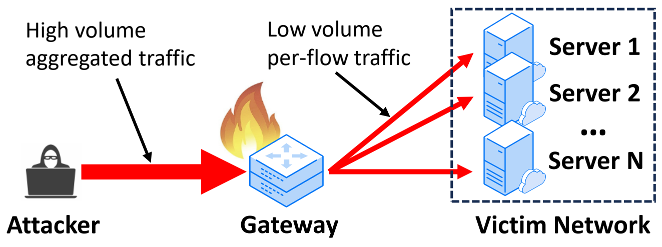

Fig. 1: Carpet Bombing DDoS

- An efficient and scalable cross-server inbound traffic analysis method that analyzes features from multiple servers simultaneously and exploits carpet bombing traffic similarity efficiently.

We evaluate NetRadar with both real-world and simulated carpet bombing traces in the simulation environment. The result shows that NetRadar can detect both high-rate and low-rate carpet bombing DDoS with high detection accuracy, with up to 40.55% improvement compared with the state-of-the-art. In the extreme covert low-rate application-layer carpet bombing attack scenarios, NetRadar is still able to identify 85% victim servers even if the malicious traffic throughput is less than 10% of the total traffic volume.

II. Background and Motivation

A. Carpet Bombing DDoS

Carpet bombing attacks have become one of the most severe threats to network services in recent years [1]–[8]. As Figure. 1 shows, unlike traditional DDoS targeting a single IP address, carpet bombing attacker sends malicious traffic to numerous servers owned by the victim network simultaneously. Malicious traffic to victim servers aggregates at the gateway and exhausts bandwidth, making services in the victim network unavailable. Carpet bombing attacks can be launched with existing DDoS techniques. The attackers obtain the victim network's address information in advance and then generate malicious traffic across these IP addresses to launch a carpet bombing attack. As victim servers typically share a common gateway, the malicious traffic volume to individual servers can be significantly reduced without weakening the damage to the victim network, which confuses DDoS detection and results in heavy DDoS mitigation resource consumption.

The key difficulty in defending against carpet bombing attacks is to identify the victim servers precisely in real time. Nowadays, DDoS defense systems consist of continuously running detection systems and on-demand mitigation systems [21], [22]. During attacks, the detection system provides a list of victim IP addresses, and the mitigation system activates defense policies for these IPs to handle DDoS traffic. During a carpet bombing attack, although the aggregated traffic throughput anomaly can be easily detected, identifying which servers are under attack is significantly more difficult. If the detection system fails to pinpoint victim servers in real time, it will force the defender to allocate more mitigation resources to protect benign services, even if the attack traffic volume is far less than these resources can handle. Besides, an imprecise victim server list may also result in increased delay and packet loss of benign services, as benign traffic is rerouted to the mitigation system before it reaches the victim servers.

Carpet bombing attacks can be divided into two categories according to their per-flow malicious traffic volume [1]. (1) **High-rate Carpet Bombing.** Attackers generate malicious traffic to multiple servers and rapidly shift targets during the attack. While the abnormally high per-flow traffic volume makes detection easier, the attack's dynamic nature challenges the defense system's response time. The report shows that, in most cases, the attack lasts for less than 10 seconds on each victim server [1], leaving the defense system extremely limited time to react. (2) **Low-rate Carpet Bombing.** Attackers target more servers in the victim network than high-rate carpet bombing, and thus the per-flow malicious traffic volume is significantly reduced. Low-rate carpet bombing is more challenging to detect as it generates concealed low-volume malicious traffic to each victim server. The report shows that more than three-quarters of carpet bombing attacks exhibit low-volume characteristics [1].

Among carpet bombing attacks, low-rate application-layer carpet bombing is the most challenging to detect. Firstly, application-layer DDoS methods [9]–[11] easily bypass traditional DDoS detection based on semantics analysis [12]–[14]. Besides, low-rate carpet bombing incurs little traffic throughput change on each victim server, rendering most heavy-hitter-based attack detection methods ineffective [12], [15]. Malicious traffic is not obviously different from normal traffic in terms of both semantics and traffic volume, posing a severe challenge to traffic-feature-only detectors [16]–[18]. Similar evasion tactics are also used in link flooding attacks (LFA) to evade detection [19]. However, existing LFA detection methods [23]–[25] typically require monitoring data from various locations across the network topology, which is hard to implement in the victim network.

B. Threat Model

We focus on detecting carpet bombing DDoS, especially the most challenging low-rate carpet bombing that generates malicious traffic with application-layer attack methods in this work. Attackers have full knowledge of the victim address pool and generate malicious traffic to multiple hosts in the victim network simultaneously. Malicious traffic aggregates at the gateway and causes a denial of service.

Specifically, we model the behavior of the carpet bombing attacker with a simple model, as shown in Figure. 2. Suppose an attacker with M DDoS malicious traffic generation budget targets a victim network with N different servers. During an attack, the attacker randomly selects k victim servers and distributes portions of budget M among them at each time slot t_n. Malicious traffic budget M is larger than the victim network gateway capacity, and thus benign network services are cut off. This model, although extremely simplified, reveals the complexity of the carpet bombing attack. For

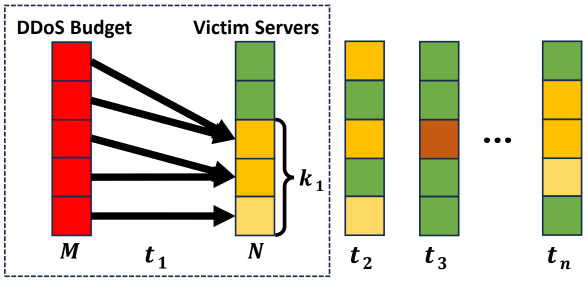

Fig. 2: Carpet Bombing Attacker Behavior

each time interval, the attacker may select different server combinations and redistribute the traffic budget, creating an exponentially expanding attack feature space. The attacker can constantly change these configurations during the attack to evade detection. This characteristic of rapid changes in time (temporal variation in target selection) and space (spatial distribution across multiple targets) can easily bypass the ISP network's sampling-based detection methods [26], rendering the defence ineffective.

We deploy NetRadar as an off-path carpet bombing detector at the victim network. NetRadar performs analysis on replicated traffic and does not interfere with traffic forwarding. NetRadar locates victim IPs (used interchangeably with servers/flows in this context) being attacked in real time and requests DDoS mitigation assistance from local Anti-DDoS devices or from upstream mitigation service providers (e.g., a scrubbing center), based on the attack volume. As for attackers, we assume that they have a limited number of malicious traffic generation methods, so multiple victim servers receive similar attack traffic. While advanced attackers may employ resource-intensive techniques such as deep learning to craft server-specific malicious traffic and eliminate similarities, the carpet bombing attacks currently confronting the industry are in fact derived from existing attack methods [1]–[3], [6]–[8], which is precisely the threat NetRadar is designed to address.

C. Opportunities

Although carpet bombing poses a significant challenge in DDoS detection, its distributed nature across multiple victim servers simultaneously creates new detection opportunities. During a traditional DDoS, available features for detection can be collected at the access point (gateway, for example). However, carpet bombing attacks spread malicious traffic to multiple servers to disperse features and avoid detection. Therefore, we need to collect features across the network and leverage the strengths of devices distributed in the network to build an effective carpet bombing detector. Specifically, we can identify carpet bombing based on extra features from victim servers (Server-Assisted Detection) and analyze traffic to multiple hosts together to exploit the similarity of malicious flows (Cross-Server Inbound Traffic Analysis).

Server-Assisted Detection. Compared with the DDoS detection system running on the gateway, the victim servers can achieve more detailed monitoring of traffic semantics and context. The servers have access to application-layer information, such as decrypted packet payload, request URL, etc., which are particularly useful in improving low-rate application-layer carpet bombing detection performance. For example, if the attacker requests large file downloads on multiple servers to launch a carpet bombing attack, it is difficult to accurately determine whether an anomaly stems from an attack or normal business fluctuations based on traffic features alone. This is because the malicious traffic remains semantically legitimate and its behavior is indistinguishable from that of normal users. On the contrary, with the help of the extra server-side features, the detection system can easily identify malicious behavior patterns: similar resource requests occurring simultaneously across multiple servers, with abnormally high bandwidth consumption. Besides, the feasibility of server-assisted detection is ensured by carpet bombing's distributed nature: the attackers must keep the malicious traffic throughput to each server at a low level to avoid being detected, which prevents server overload and preserves computational resources to assist detection. Moreover, in common targets of carpet bombing attacks, such as data centers and enterprise networks [7], operators are usually capable of monitoring the servers, thereby enabling the deployment of additional feature extraction mechanisms.

Cross-Server Inbound Traffic Analysis. By collecting features from the entire victim network and conducting centralized analysis on them, we can locate carpet bombing attacks based on a comprehensive view of the victim network. As we analyzed in §II-B, an attacker can easily alter traffic throughout to each host to maximize the detection difficulty of the traditional per-flow detection schemes. But as we know, in practice, DDoS attacks these days are launched with botnets and automatic scripts [20], [27], leading to inherent similarities across malicious flows on both traffic and server-side features. In fact, analysis of existing botnet codes [7] shows that attackers simply randomize the target IPs when launching carpet bombing attacks to minimize the traffic generation cost. Compared with randomly distributed benign traffic, the similarity between malicious traffic features is much higher. The cross-server inbound traffic analysis models analyze traffic to multiple servers simultaneously and utilize the similarities among malicious traffic to achieve better detection accuracy.

Based on these observations, we propose NetRadar, a system that makes full use of the available features for carpet bombing detection across the victim network with server-assisted detection and cross-server inbound traffic analysis.

III. NetRadar Architecture

To enable server-assisted detection and cross-server inbound traffic analysis, traffic and server-side features need to be collected and analyzed in a centralized manner. Therefore, a server-gateway cooperation architecture is required to realize information collection while collaborating with DDoS mitigation devices (e.g., firewalls) during the attack. NetRadar uses a simple server-gateway cooperation mechanism where servers in the victim network transmit server-side features to the upstream cross-server analysis detector located at the gateway. This architecture is motivated by the following considerations:

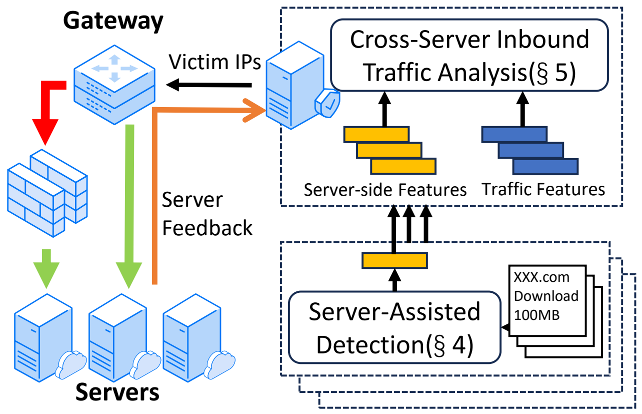

Fig. 3: NetRadar Architecture

(1) Minimized Feature Transmission Overhead. Since server-side features are distributed across the victim network, feature transmission is unavoidable when gathering information from servers. From a topological perspective, the most reasonable choice is to place this central feature collection point at the gateway, through which all traffic of the victim network passes. Features from server systems can be piggybacked on outbound traffic to the gateway, minimizing transmission overhead. Besides, extracting traffic features directly at the gateway provides additional transmission overhead savings. Although servers can collect traffic features, gathering these features from servers requires extra bandwidth costs in an already overwhelmed victim network. Therefore, in NetRadar, servers only extract and transmit features unavailable at the gateway. The gateway collects traffic features locally and merges them with server-collected features for further analysis. This architecture minimizes the feature transmission cost.

(2) Rapid Detection of Evolving Attacks. Different from traditional single-target DDoS, carpet bombing attacks dynamically change target servers during the attack to challenge the DDoS defense system's response time. To block carpet bombing in time, defense strategies such as blocking and rate-limiting should be applied and revoked quickly accordingly. Industry reports indicate that the attack duration experienced by each victim server may be less than 10 seconds [1], necessitating equally responsive defense mechanisms.

Timely detection of such rapidly evolving attacks requires direct analysis of original traffic to immediately identify malicious behavior patterns. As we all know, to filter malicious traffic before it affects benign services, DDoS mitigation devices need to be located upstream of the victim servers. In other words, traffic to the servers is already processed by these DDoS mitigation devices if defense strategies are applied. Therefore, deploying the detection model on the servers creates a fundamental limitation that detection results will affect the feature collection process, causing a deadlock. Such a detection model based on server-collected features cannot determine whether the attack has stopped or not, as in both cases, traffic received on the server is free of malicious traffic. In summary, collecting all features on server systems does not work well with DDoS mitigation devices, which makes this solution unfeasible. Therefore, NetRadar collects traffic features and deploys the detection model on the gateway, enabling effective detection of evolving attacks while maintaining compatibility with existing mitigation infrastructure.

NetRadar Workflow. Based on these considerations, NetRadar employs the server-gateway cooperation architecture illustrated in Figure. 3. The detection model is deployed at the gateway of the victim network, where it aggregates both server-collected and gateway-collected features. Servers extract crucial service features that are unavailable on network devices and send them to the upstream detection model to enable server-assisted detection. Traffic features are extracted from replicated traffic at the gateway, and then merged with the server-side features transmitted from victim servers. Finally, based on both traffic and server-side features, NetRadar performs cross-server inbound traffic analysis and identifies victim IPs receiving malicious traffic. Then, according to the victim IPs list, malicious traffic is rerouted to the DDoS mitigation device while benign traffic passes through the gateway. The victim IPs list is updated continuously during the attack to block dynamic carpet bombing attacks and withdraw outdated defense policies in time. We propose the detailed design of Server-Assisted Detection and Cross-Server Inbound Traffic Analysis in the following sections.

IV. Server-Assisted Detection

During a carpet bombing attack, victim servers can provide critical server-side features to enhance attack detection. NetRadar uses a server-gateway cooperation architecture where servers are only responsible for feature extraction. To enable a general server-assisted detection, the server-side features must be widely available across most services and effective for carpet bombing detection. Besides, due to traffic scrubbing by DDoS mitigation devices, certain traffic may be invisible to servers, causing the runtime feature mismatch issue between server-collected features and gateway-collected features.

A. General Server-side Features

To ensure server-side feature compatibility across diverse services, we developed a lightweight behavioral model for online services, enabling effective extraction of service access patterns. We abstract the behavior of online services in the simplest way, where each request triggers a specific corresponding reply. In this model, when receiving a request, the server locates the requested resource and generates a reply. The model assumes that servers can identify requests for the same resources to pinpoint frequently accessed ones. Additionally, we assume servers can acquire the size of the requested resources, either in advance or after processing the requests (if the requested resource is not static). This model, although extremely simplified, matches the behavior of most online services, such as website and file download services. NetRadar employs this model to unify features across different services. An identical feature extraction method is used across all victim servers. We avoid extracting features for each specific service because this will lead to feature heterogeneity, making it difficult for us to analyze the similarity of attack behaviors.

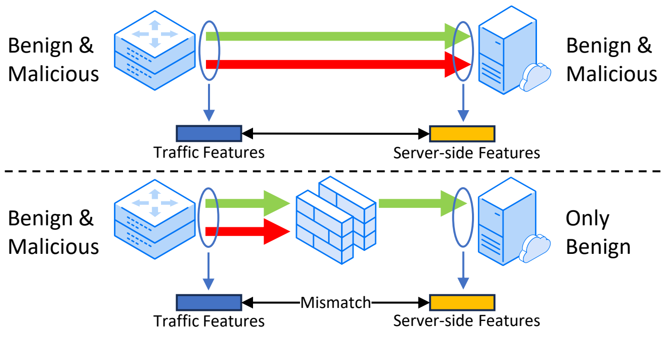

Fig. 4: Feature Mismatch when DDoS Mitigation is On

From this behavior model, we select **resource hit frequency** and **active resource size** as server-side features. DDoS attacks are typically launched by repeatedly requesting high-bandwidth resources. Consequently, requests that repeatedly access the same resource or incur high throughput traffic transmission are most likely to be malicious ones. Therefore, NetRadar collects access times of the most frequently requested resources and the total size of the resources currently being requested within a given time window as server-side features. These features are effective in identifying suspicious traffic and can be easily extracted from most online services. As for malformed packets that are generated by volumetric DDoS scripts, we treat them as special requests for the zero-size resource, allowing the detection model to accurately determine the presence of volumetric attacks through special resource size and extremely high request frequency.

B. Robust Server-Assisted Model

In NetRadar, traffic features are collected on the gateway while service access pattern features are collected on the victim servers. This distributed collection mechanism introduces a critical challenge during carpet bombing attacks: when detected, victim server traffic is redirected through DDoS mitigation devices, creating a runtime feature mismatch between server-collected and gateway-collected features. As shown in Figure. 4, when the mitigation is off, both victim servers and the gateway receive complete traffic, allowing direct feature merging. However, when the mitigation is on, the traffic received on victim servers has already been checked and filtered by the DDoS mitigation device, and thus the features collected on servers only represent benign traffic status. In other words, the service access pattern features cannot provide malicious traffic information to enhance DDoS detection in this situation. Moreover, the activation status of the mitigation is not fixed, but is adjusted by the real-time defense strategy. This means that our solution must adapt to both states and detect DDoS accurately when some server-side features are misleading. Otherwise, the model performance is likely to be influenced by the previous decision, resulting in repeated fluctuations in the detection results and DDoS defense failure.

We introduce random erasing to the model training process to solve the runtime feature mismatch issue. Random erasing is a data augmentation technique that randomly masks some features during the training process to make the model robust [28]. In our server-gateway cooperation architecture, traffic features are always extracted from complete traffic, as traffic is not processed by the firewall when collected on the gateway. Therefore, to train a model that works well in both states mentioned above, we apply random erasing to the server-side features specifically to simulate the situation where some servers extract features from firewall-processed traffic.

Specifically, for each test scenario, we preprocessed two versions of features, namely $Feature_{complete}$ (collected on original traffic) and $Feature_{benign}$ (collected on firewall-processed traffic). For each training sample (which contains features of multiple servers for cross-server analysis), we randomly select a subset of servers and replace their server-side features with the $Feature_{benign}$ versions. And the number of the erased server-side features is also randomly selected from zero to the maximum host count in each inference during the training process. In other words, this approach trains the model to handle all possible operational states - from relying solely on gateway-collected traffic features when all server-side features are filtered, to utilizing complete features when mitigation is off, and every intermediate scenario. Consequently, NetRadar develops robust detection capabilities that are effective regardless of the mitigation state, successfully addressing the runtime feature mismatch issue in our server-gateway cooperation architecture.

Workflow Summary. NetRadar detects carpet bombing attacks by leveraging both traffic features and service access pattern features. The gateway collects traffic features—specifically, throughput, packet length, and inter-packet delay. Meanwhile, service access pattern features, namely resource hit frequency and active resource size (§IV-A), are collected by the server and subsequently transmitted to the gateway for centralized analysis. This process is applied consistently during both the training data collection phase and the real-time feature extraction phase after deployment. To address the runtime feature mismatch issue during inference, we introduce a training phrase enhancement mechanism aimed at building a more robust detection model (§IV-B).

V. Cross-Server Inbound Traffic Analysis

As we analyzed in §II-C, carpet bombing DDoS attackers generate similar malicious traffic to multiple servers in the victim network. Thus, analyzing feature similarities across servers proves to be an effective carpet bombing detection mechanism. Especially when faced with low-rate carpet bombing, there is virtually no difference between malicious traffic and benign traffic unless we analyze traffic to multiple servers simultaneously. However, taking advantage of the similarity requires costly cross-server analysis, necessitating a detection model capable of processing features of multiple flows simultaneously. To achieve efficient detection, we propose a scalable cross-server analysis model structure featuring a permutation-equivariant feature transformation function specially designed for carpet bombing detection.

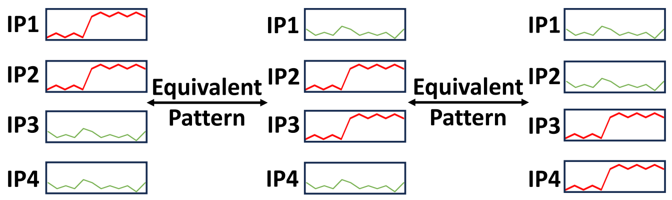

Fig. 5: Example of Equivalent Carpet Bombing Patterns

A. Scalable Analysis on Traffic Feature Set

Cross-Server Analysis Scalability Issue. In the traditional per-flow traffic analysis scheme [16]–[18], the detection model processes the features of one flow at a time. The model determines whether there is an anomaly only based on the isolated features of each flow. In contrast, a detection model based on cross-server analysis takes features of more than one flow as input in each inference to model inter-flow similarities. This approach is promising in improving carpet bombing detection accuracy (as we analyzed in §II-C), but the increased input dimensionality poses challenges to model design.

The naive solution of constructing a large neural network to process multiple server features simultaneously incurs high performance overhead. In order to model larger inputs, the model size inevitably needs to be increased accordingly. However, this approach does not scale well as the model size is correlated with the victim subnet size. A possible workaround is to process a portion of the flows at a time to keep a reasonable model size. However, if the number of simultaneous inputs is too small, it will limit the model's performance as its analysis is based on incomplete local features. Besides, larger models demand longer inference times, rendering them impractical for real-time DDoS detection.

An alternative approach is to build a model based on summarized subnet features, which reduces model complexity by compressing input dimensionality. For example, statistical subnet features like average traffic throughput of the victim network can represent the overall properties of traffic to multiple servers. A model based on these summarized features enables network-wide analysis with reasonable model complexity, as input feature size is significantly reduced. However, we sacrifice model performance as these specific features are hand-crafted. What's more, we cannot pinpoint victim servers in one inference as we give up per-flow features, and we need to traverse possible server combinations to finally locate the victims, which introduces exponential overhead.

Features of Multiple Servers as A Set. NetRadar solves the cross-server analysis scalability issue with a key observation that features of multiple victim servers can be treated as a set. Specifically, when we detect carpet bombing DDoS in a victim

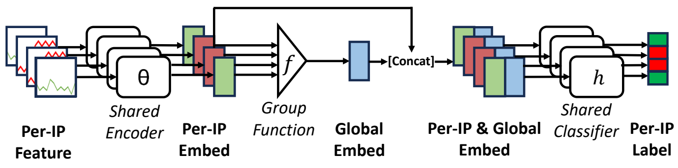

Fig. 6: Permutation-equivariant Model Structure

network, we mainly rely on overall summarized features and similarities of individual flows. The order of flow features is not crucial to the detection process. For example, as Figure. 5 shows, if a victim network contains 4 servers and a carpet bombing attacker sends malicious traffic to 2 of them, we detect the existence of the attack by the increase in the overall traffic volume and identify the victim servers by identifying the similarity among features, none of the detection process depends on the order of these features. In other words, in Figure. 5, whether the malicious traffic hits IP1&2, IP2&3, or IP3&4, the analysis process remains the same. Considering this unordered property of features, we can significantly reduce the feature space as the equivalent variants of different input orders are eliminated, which simplifies the detection model and thus improves scalability.

In fact, we argue that a carpet bombing detection model should avoid being affected by the order of input features to achieve robust detection. As we analyzed in II-B, carpet bombing attackers can easily change the targets and send malicious traffic to another set of victim servers, IP1&4 in Figure. 5 for example. If a detection model's performance is strongly related to the order of input features, it will be easily breached by a dynamic carpet bombing attack. On the other hand, although it is theoretically possible to train a model that can accept inputs in different orders by constructing a large training dataset containing all possible input orders, training a model in such a way requires a huge amount of training time and an unacceptable large model size. If we can eliminate equivalent patterns in model design, it will be much easier to implement such a robust model.

Permutation-equivariant Model Structure. As we analyzed above, the input of the cross-server analysis model is set-structured. Deep learning on set-structured data has been explored in point cloud data analysis [29], where the coordinate characteristics of multiple points in space also conform to the characteristics of unordered sets. Inspired by prior works of deep learning on point cloud data, NetRadar uses the permutation-equivariant model structure to enable efficient cross-server inbound traffic analysis.

As Figure. 6 shows, in a permutation-equivariant model, per-IP (or per-server) features are embedded with a shared encode MLP θ. Then, these per-IP embeddings are summarized by group function *f*, and a global feature representing all inputs is generated. After that, this global embedding is copied several times, concatenated with the previous per-IP embedding, and processed by the subsequent classifier model h. Since the classification model h processes both per-IP feature and global feature, it enables cross-server analysis. Note that both θ and h are shared MLP across per-IP features, which means the processing on each flow is exactly the same and thus the order can be swapped.

The core structure of the permutation-equivariant model is the group function f that aggregates the features from multiple inputs. The function f is a symmetry function that takes multiple vectors and outputs a vector invariant to the order of these vectors. Specifically, for any permutation π, group function f on a set of elements $\{x_1, x_2, \ldots, x_n\}$ satisfies the following permutation-equivariant properties:

$$f(\{x_1, x_2, \ldots, x_n\}) = f(\{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\}) \quad (1)$$

In other words, for the model in Figure. 6, no matter how the input order of per-IP embedding changes, the global embedding is always the same. Every part of the model can achieve permutation-equivariant, making it particularly suitable for set data processing, as the order of input is eliminated. This allows us to implement a highly scalable cross-server analysis model and reduce the difficulty of training without sacrificing detection accuracy.

B. Group Function for Carpet Bombing Detection

By treating features from multiple servers as a set, the permutation-equivariant model structure can effectively reduce model complexity and improve scalability. We design a special group function f for the carpet bombing detection task.

Individual Similarities rather than Overall Properties. Prior works using permutation-equivariant model structure have focused on the overall properties of the feature set. For example, in a 3D object classification task [29], the overall shape composed of multiple point coordinates is the most important group feature for object classification. In a switch buffer management task [30], the overall load of the switch composed of the status of multiple ports, is the most important group feature to make better resource scheduling actions. Thus, these works use the maximum, minimum, and average of the input features as the group function *f*, which can summarize the overall characteristics of all inputs.

However, in the carpet bombing detection task, we focus on the similarities between individual elements within the set rather than the overall characteristics. If we directly use the function f that summarizes group features, the feature of each flow that is useful in similarity analysis will be discarded. The overall properties may be useful in detecting whether a carpet bombing attack has occurred, but they are less effective in identifying specific servers under attack. Take traffic throughput as an example, traditional group function f summarizes overall characteristics of traffic throughput to multiple servers and outputs overall victim network traffic load, which is a feature suitable for attack discovery rather than victim servers identification. As we analyze in §II-C, a detection model that identifies the similarity of malicious flows is suitable for carpet bombing detection, and thus we need to preserve the characteristics of each flow in the global embedding. Besides, we still need to ensure that the properties

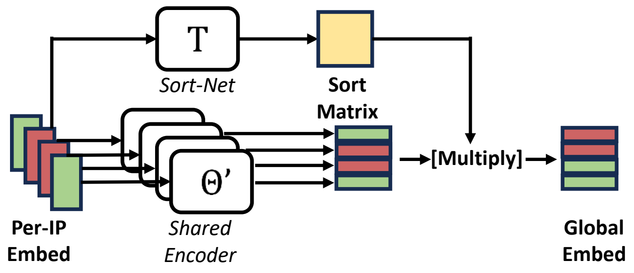

Fig. 7: Sort-based Group Function

of the symmetric function are satisfied to gain the advantage of the permutation-equivariant model structure.

Group Function for Similarities Analysis. To simultaneously achieve permutation-equivariance and preserve individual flow information in global embedding, we need to design an order arrangement method that any set of inputs can obtain a stable order. A simple solution is to sort the original features from multiple servers and ensure a stable order before the model processes these inputs. However, this order changes significantly even if there are small fluctuations in features, which poses a negative impact on model training. We desire a relatively fixed order of malicious and benign features to reduce subsequent classifier model training difficulty. In NetRadar, the neural network compresses the original features, and we sort them based on their embeddings to achieve a more stable order than sorting them based on the original features.

We propose a sort-based group function for the permutation-equivariant model in carpet bombing detection tasks. As Figure. 7 shows, the group function takes multiple per-IP embeddings and outputs a global embedding that contains the compressed information of all per-IP features. We first use a shared neural network to further compress per-IP embeddings. Then we construct the sorting matrix with a separate Sort-Net and sort the compressed embeddings with this matrix. The sorted compressed embeddings are then used as the global embeddings of the permutation-equivalent model.

In order to construct the sorting matrix, we introduce a new loss function to train the Sort-Net. Sort-Net is a mini-network that takes per-IP embeddings as input and generates the sort matrix. We add a regularization term to the training loss to ensure that the sorting matrix is a permutation matrix:

$$L_{sort} = \|I - AA^T\|^2 + \lambda(\Sigma \frac{(sum(a_i) - \max(a_i))}{\max(a_i)}) \tag{2}$$

where A is the sorting matrix predicted by Sort-Net, a_i is row i of the sorting matrix. The first term ensures that the sorting matrix is close to an orthogonal matrix, as in PointNet [29]. The second term ensures that each row contains only one valid parameter. Since we restrict the matrix to be orthogonal with the first term, the matrix that satisfies both restrictions is the permutation matrix, which guarantees that each feature is assigned to exactly one position. In other words, a matrix that satisfies the above loss function will rearrange the input, which can be used to sort the per-IP embeddings.

Workflow Summary. Traffic and server-side features are aggregated at the gateway through the method described in §IV. Since features from multiple servers can be treated as set-structured data in the context of carpet bombing attack detection, NetRadar employs an efficient permutation-equivariant model structure to build the detection model (§V-A). Features from individual servers are organized into several groups, with each group processed by the model to generate detection results for the corresponding servers. Additionally, to better adapt the model to carpet bombing attack detection, we have improved the Group Function (§V-B).

VI. Evaluation

In this section, we evaluate NetRadar from three aspects: (1) End-to-end performance on carpet bombing DDoS detection. NetRadar achieves better detection accuracy compared with state-of-the-art DDoS detection methods on both high-rate and low-rate carpet bombing DDoS detection; (2) Detection robustness. NetRadar achieves robust detection even in extreme covert low-rate carpet bombing attack detection tasks; (3) Design validation. We further test the performance gain of the detailed design of NetRadar and the hyperparameters that affect the performance of NetRadar.

A. Experiment Setup

Server-side Features Generation. NetRadar analyzes both traffic and service access pattern features to detect carpet bombing DDoS, thus we need to generate not only the traffic dataset but also the corresponding server-side feature dataset. We build the test dataset based on the CIC-IDS{2017,2018} [31]. The CIC-IDS dataset's packet trace captures the complete packet payload, from which we can extract server-side features. We select HTTP traffic from the dataset and extract the URL of each five-tuple flow from the packet payload. These URLs identify flows accessing similar resources, while the total bytes of the flow represent the resource size. Based on this, we subsequently compute the resource hit frequency and active resource size features.

Carpet Bombing Traffic Generation. To evaluate NetRadar, we use both real-world and simulated carpet bombing packet traces. We set up one high-rate carpet bombing detection task and two low-rate carpet bombing detection tasks in which attackers use different malicious traffic generation methods. For benign background traffic, flows from CIC-IDS-2017 Monday, which contain only benign activities, are duplicated and assigned with random flow start timestamps to construct a dataset of 120 seconds. Then we generate three carpet bombing traffic datasets and merge them with the benign traffic dataset above to create three DDoS detection test settings. The details of the carpet bombing datasets are as follows:

- Volumetric DDoS (High-rate): A reconstructed high-rate carpet bombing attack based on a real-world dataset from a top-tier cloud service provider. The original dataset contains a 2-second packet trace captured on 256 victim servers during a carpet bombing attack. Due to privacy concerns and hardware limitations, we cannot obtain a

TABLE I: Detection Accuracy of NetRadar and Baselines in Different Carpet Bombing Detection Tasks.

Attack Traffic	Volumetric DDoS			Low-rate HTTP			Synchronous Download		
Metric	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Lemon	99.28%	99.64%	99.61%	53.04%	53.41%	81.32%	54.23%	52.55%	86.47%
Kitsune	99.95%	99.99%	99.95%	63.22%	34.70%	52.66%	89.86%	96.07%	72.36%
Flowlens	99.99%	99.99%	99.99%	92.90%	94.92%	93.69%	90.31%	93.37%	83.57%
Whisper	99.69%	99.81%	99.86%	92.77%	95.34%	90.18%	86.63%	84.14%	85.18%
NetRadar	99.79%	99.83%	99.95%	96.28%	96.98%	96.06%	94.78%	96.38%	93.70%

long enough packet trace for performance evaluation. Instead, we build a dataset based on the pattern of the original dataset. There are 32 servers receiving SYN Flood in the original packet trace. Therefore, in the reconstructed dataset, we preserve the characteristic of the attack targeting 32 victims simultaneously, while also introducing target switching every 2 seconds to generate a dynamic high-rate carpet bombing attack.

- Low-rate HTTP (Low-rate): This scenario simulates a low-rate carpet bombing attack using HTTP-based attack traffic from CIC-IDS2018 (Hulk, GoldenEye, Slowloris, HOIC, LOIC) and a low-rate HTTP flood generated by duplicating randomly chosen low-rate HTTP flows from the benign dataset. We generate traffic with one attack at a time and switch the attack type every 5 seconds to simulate the attacker's behavior of switching traffic generation methods. We target 256 of 1024 victim IPs at the same time and change target IPs every 5 seconds.
- Synchronous Download (Low-rate): This scenario simulates a low-rate carpet bombing attack generated by requesting benign download services. We choose one download HTTP five-tuple flow from the benign dataset and duplicate it to generate malicious traffic. We target 256 of 1024 victim IPs at the same time and change target IPs every 5 seconds. This scenario is designed to emulate a situation where sophisticated adversaries craft attack traffic that closely mimics benign flows.

Baselines. We use state-of-the-art DDoS detection schemes as baselines for comparison. Each test scenario contains 120 seconds of packet trace. We train these schemes on the first 90 seconds of the dataset and test them with the latter 30 seconds.

- Lemon [15]: Sketch-based DDoS detection scheme. Lemon uses a hash-based per-flow measurement method to collect traffic features and detects DDoS based on these features. We set Lemon's detection threshold according to the average bandwidth of the test scenario used. We use the open-source Lemon system based on Mininet.
- Kitsune [32]: Machine learning based network intrusion detection scheme. Kitsune extracts traffic statistics and uses autoencoders to detect attacks. We train Kitsune with benign traffic and determine its detection threshold based on the loss on the training dataset. We use the open source Kitsune implementation.
- Flowlens [16]: Flowlens extracts the packet length and inter-packet delay distribution of each flow as traffic features. We use the random forest classifier as the detection model and Bayesian optimization to adjust Flowlens' hyperparameters to its optimal configuration. We reproduce Flowlens based on its paper.
- Whisper [18]: Whisper embeds the packet length, inter-packet delay, and packet type of each packet in a floating-point value. Then, Whisper applies the Fourier transform on the packet features sequence and uses an unsupervised clustering method to detect victim servers. We build a supervised version of Whisper with the same features and use a random forest classifier as the detection model.
- NetRadar [33]: we implement NetRadar in simulation environment. NetRadar takes both traffic and server-side features of 256 victim servers as input in each inference. We extract features every second. During the training process, we sample 256 different input sets (each contains features of 256 servers) every second and randomly erase some server-side features to train a robust detection model as introduced in §IV-B. During the test process, we divide the features of 1024 servers into 4 groups of 256 servers every second and give the detection results of each server with 4 inferences. For the flows identified as DDoS in the previous second, we erase the corresponding server-side features to simulate the runtime feature mismatch.

Metric. We report the carpet bombing detection performance of each scheme using packet-level Accuracy, Precision, and Recall. These metrics are calculated from the detection results of each packet, as these detection schemes use different feature extraction strategies and detection triggering mechanisms: NetRadar and Lemon trigger a detection process at a fixed time interval of one second, Flowlens and Whisper are triggered when the number of unclassified packets of a server reaches 160, and Kitsune analyzes every packet individually.

B. End-to-End Performance

In this section, we compare NetRadar with state-of-the-art DDoS detection schemes' performance in carpet bombing detection tasks in §VI-A. Results are shown in Table I.

NetRadar achieves comparable if not the best performance among the detection schemes in all carpet bombing detection settings. In the high-rate carpet bombing DDoS detection task, all solutions achieve an accuracy of over 99%. The reason is

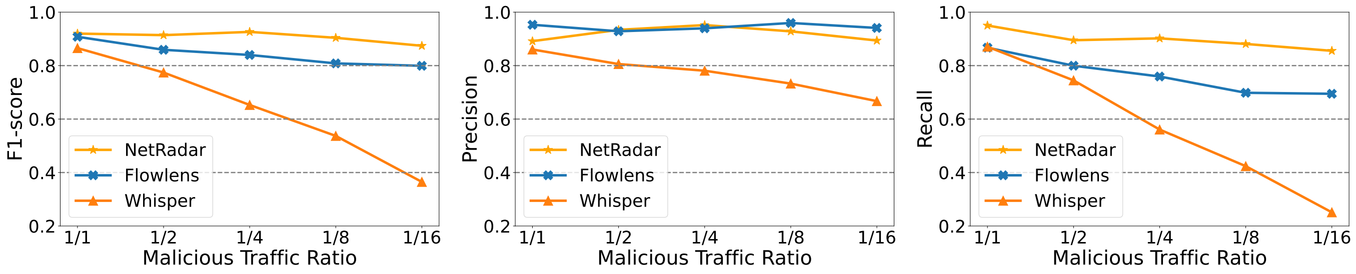

Fig. 8: Detection Robustness Test with Different Malicious Traffic Ratio Dataset

that the high-rate carpet bombing targets fewer victim servers, and the characteristics of each flow are still easily noticeable. Malicious traffic generated by volumetric DDoS scripts can be easily detected with the throughput anomaly. Especially in the real-world dataset we obtained, the attacker floods the victim network with a simple SYN Flood, which further reduces the detection difficulty. NetRadar achieves comparable performance with state-of-the-art DDoS detection schemes in high-rate carpet bombing DDoS detection.

In both low-rate carpet bombing DDoS detection tasks, NetRadar outperforms baselines. Lemon's performance significantly degrades in both low-rate scenarios, as its detection model is based on traffic statistics and identifies victim with a heavy-hitter logic. It is not suitable for low-rate carpet bombing detection where per-server malicious traffic volume is significantly reduced. Compared with Lemon, NetRadar achieves an accuracy improvement of 40.55% in the Synchronous Download test setting. In the first low-rate carpet bombing test setting, the decrease in individual attack flow throughout challenges the detection, resulting in a decrease in recall score for baselines. NetRadar achieves better performance by fully utilizing network-wide carpet bombing features with server-assisted detection and cross-server inbound traffic analysis mechanism. In the most challenging Synchronous Download test setting, baseline solutions cannot detect all the malicious traffic as we simulate an advanced attacker constructing a carpet bombing using only download traffic. Without the help of server-side features and the ability to perform cross-server analysis, baselines cannot distinguish benign downloads from the malicious ones. On the contrary, NetRadar identifies victim servers accurately based on the active resource size increasing synchronously on multiple servers.

C. Detection Robustness

In this section, we further evaluate NetRadar's detection robustness with extreme covert low-rate carpet bombing DDoS datasets. As we mentioned in §II-A, carpet bombing challenges existing DDoS detection schemes with concealed low-throughput malicious traffic. The lower the per-flow malicious traffic throughput, the more difficult it is for the detection schemes to identify the malicious behavior. As the proportion of malicious traffic to each server decreases, per-flow traffic features become less reliable in DDoS detection. To test whether NetRadar achieves higher detection robustness than the traditional per-flow DDoS detection scheme, we construct a set of carpet bombing detection tasks with different malicious traffic ratios. The test datasets are modified based on the Synchronous Download task in §VI-B. We test the malicious traffic throughput ratio of 1/1 to 1/16 by modifying the number of addresses being attacked simultaneously and the malicious throughput on each server. All schemes in comparison are retrained on the newly constructed dataset.

As Figure. 8 shows, NetRadar achieves the best performance among detection schemes. Even in the most challenging 1/16 ratio settings where the malicious traffic has a throughput less than 10% of the total traffic volume, NetRadar still achieves the F1-score of 0.874 and identifies 85% victim servers. Note that we report the F1-score instead of the accuracy in this experiment because adjusting the ratio of the malicious traffic will cause severe sample imbalance issues. A detector that only outputs benign labels can also achieve high accuracy in these test settings. For Whisper and Flowlens, detector performance decreases as the proportion of malicious traffic decreases, because malicious traffic has less impact on traffic features if its throughput is extremely low. For traditional per-flow traffic analysis schemes, it is difficult to identify all victim servers in this scenario, so we can observe that the recall score of these baseline solutions decreases significantly as the proportion of malicious traffic decreases. On the contrary, NetRadar uses the server-side features and analyzes the similarity of multiple features to enhance the detection performance. In extremely low-rate carpet bombing scenarios, these features can reveal the malicious property of spreading malicious traffic to multiple servers. Therefore, NetRadar maintains a higher recall score compared with baselines. It is worth mentioning that we find Flowlens achieves a better precision score in some test settings. However, Recall is a more important indicator in the carpet bombing detection scenario, especially in this detection robustness experiment featuring imbalanced positive and negative samples. NetRadar identifies more victim servers, thus better protecting benign services.

D. Deep Dive

In this section, we validate NetRadar's design and test hyperparameters in NetRadar with control experiments.

Ablation test. To fully understand the performance gain of NetRadar's designs on carpet bombing detection accuracy, we conduct the ablation test on NetRadar. We build a baseline

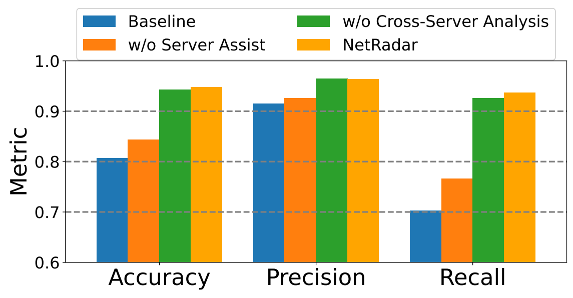

Fig. 9: Ablation Test on NetRadar

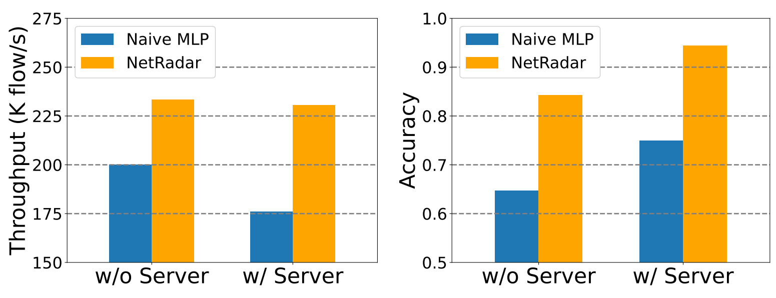

Fig. 10: Cross-Server Analysis Efficiency

model that takes only traffic features from one server as input. The results of the baseline model represent the detection accuracy of the traditional per-flow traffic analysis solution. We also build a per-flow NetRadar without cross-server analysis and a traffic-feature-only NetRadar without server-assisted detection. The baseline model and per-flow NetRadar use a standard multi-layer perceptron (MLP) model. We input only traffic features into traffic-feature-only NetRadar and leave the other parameters unchanged to NetRadar. Four solutions are evaluated in the Synchronous Download traffic dataset in §VI-A, and results are shown in Figure. 9.

Compared with the baseline per-flow traffic analysis model, NetRadar improves carpet bombing detection accuracy by 14.08% in total. Intuitively, the performance gain by server-assisted detection is greater, which is related to the characteristics of the Synchronous Download test scenario, where the attacker requests large file downloads to generate malicious traffic. Malicious behavior can be easily identified based on the active resource size feature provided by victim servers. On the other hand, for traffic-feature-only solutions, malicious flows have no obvious difference from normal download flows in terms of statistical traffic features. Therefore, the baseline model and traffic-feature-only NetRadar are relatively ineffective in finding all victim servers and thus achieve a lower recall score. Besides, solutions with cross-server analysis perform better compared with their per-flow version. Traffic-feature-only NetRadar achieves an accuracy improvement of about 4% over the baseline model because cross-server analysis enables the model to capture the synchronous increase in throughput on multiple servers. The improvement from per-flow NetRadar to NetRadar is relatively smaller as the server-assisted detection has achieved a high enough accuracy boost.

Cross-Server Analysis Efficiency. As we analyzed in §V-A, the straightforward way to implement a cross-server analysis model is to build a large neural network that directly takes features of multiple servers as input. NetRadar designs a more scalable solution with the permutation-equivariant model structure. To test how this design improves model efficiency, we build a standard MLP model that takes features from all servers as input and outputs corresponding detection results. We use the largest possible model size for our GPU to ensure the best performance of the naive MLP model. And the traffic trace is consistent with the Synchronous Download in §VI-A. For detection throughput, we run model inference 10,000 times and calculate the time consumed for each detection result on each server. We report performance in terms of the number of flows processed per second. Results are shown in Figure. 10.

NetRadar performs better than the naive MLP model in detection accuracy and detection throughput. In terms of accuracy, NetRadar achieves great improvement compared with naive MLP, especially in the scenario without server assistance, which shows that NetRadar's model performs more accurate cross-server analysis. Although the neural network of the naive model is larger, NetRadar still achieves better detection accuracy because NetRadar takes advantage of carpet bombing detection task characteristics that features of multiple servers can be treated as an unordered set, as we analyzed in §V-A. As for detection throughput, it is clear that the NetRadar is more efficient, achieving at most 31% more processing throughput than the baseline model. To handle multiple inputs, the baseline model is forced to increase the model size, while some parts of the NetRadar model are shared among multiple inputs, and therefore NetRadar requires fewer computational resources. Besides, it is worth mentioning that the naive MLP model takes features of all victim servers in a fixed order in this test. We have tried to use the same inputs (features of 256 victim servers in random order for each inference) as NetRadar in the naive MLP model, but we failed to train an effective detection model. The naive MLP model has difficulty in handling input order swaps. We argue that inputting all features into the model like this is an impractical solution, as the number of servers in large networks far exceeds 1024, and therefore an even larger model is needed to handle these features, resulting in an unacceptable computational overhead.

Inference Subnet Size. The performance gain of cross-server analysis comes from analyzing features from multiple servers simultaneously. Therefore, the number of traffic features the detection model takes as input in each inference affects detection accuracy. However, an excessively large size may lead to unacceptable costs in model inference and training. We test this parameter with the Synchronous Download dataset in §VI-A. To better demonstrate the performance difference, we use traffic-feature-only NetRadar in this test and change its inference subnet size from 4 to 512 servers. We modify the number of features that the group function takes as input,

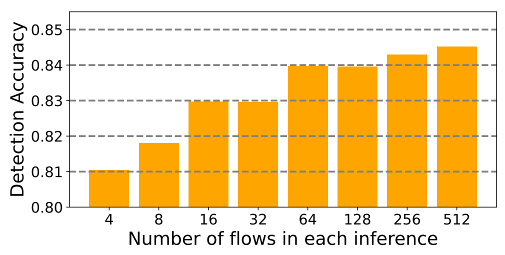

Fig. 11: The Impact of Inference Subnet Size

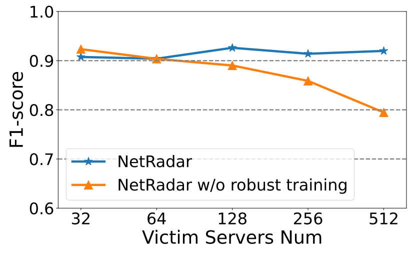

Fig. 12: Random Erase Test

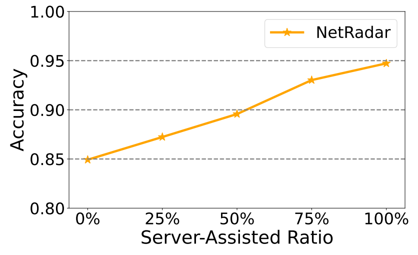

Fig. 13: Incremental Deploy Test

while leaving the other model parameters unchanged.

Results are shown in Figure. 11. In general, a larger inference subnet size leads to better detection accuracy as the model analyzes the similarities of more flows and is less susceptible to errors caused by local fluctuations. However, training a model with larger inference subnet sizes takes a significantly longer time. In this test, the training time of the 512-server version NetRadar is about 3 times that of the 256-server version. Therefore, based on the comprehensive consideration of model accuracy and training efficiency, we use 256 as NetRadar's default inference subnet size in the experiments. Besides, analyzing a relatively small number of servers at a time, such as 64 servers, achieves comparable performance with larger inference subnet sizes in our test setting. Using such a small inference subnet size can further improve the detection throughput of NetRadar if needed.

TABLE II: Group Function Comparison

Group Function	DeepSets	PointNet	NetRadar
Accuracy	86.86%	85.98%	94.78%

Group Function Design. As we mentioned in §V-B, prior works using permutation-equivariant model structure [29], [30] focus on overall properties rather than individual similarities in their group function *f*, which is not suitable for carpet bombing detection tasks. To demonstrate that, we compare NetRadar's performance with DeepSets [34] and PointNet [29]. Noted that we remove the input transformation in PointNet (specifically, transforming point position in input vector while leaving the rest unchanged) as it is not suitable for traffic and host features. Apart from the model, we keep all other parts of the baseline identical to NetRadar. Test dataset is consistent with the Synchronous Download in §VI-A.

As shown in Table II, NetRadar performs better than DeepSets and PointNet. The group function of NetRadar is specifically designed for individual similarity analysis. NetRadar retains features for each flow in global feature embedding, subsequent classifier network can analyze the similarity between current flow embedding and sorted all flow embeddings. Therefore, NetRadar can easily capture the similarity between DDoS traffic on multiple hosts and identify carpet bombing based on that. On the contrary, traditional schemes discard the per-flow features and use the overall properties of the victim network as global feature embedding, which is less effective as individual similarities matter more than overall properties in carpet bombing detection.

Random Erase Test. NetRadar uses the random erasing mechanism to train a robust model that achieves stable detection no matter whether traffic to server systems is processed by mitigation devices or not (§IV-B). To test the performance gain of this design, we build a NetRadar without random erasing and compare it with the original NetRadar. We use the dataset in §VI-C where the num of the victim servers is changed from 32 to 512. NetRadar without random erasing is trained with original traffic and server-side features without the random erasing mechanism introduced in §IV-B. During the test process, we simulate the runtime feature change by erasing server-side features of victim servers according to previous second model detection result.

The results are shown in Figure. 12. NetRadar achieves a stable detection F1-score of over 0.9 in all test settings. The model without random erasing achieves a lower F1-score as the number of victim servers increases. In scenarios with a large number of victim servers, the number of servers that are affected by the DDoS scrubbing process will increase. These affected servers can only extract server-side features of benign traffic, leading to the runtime feature mismatch between server-collected features and gateway-collected features. The model without random erasing is trained on the complete traffic and server-side features, so it is not robust to the server-side feature changes caused by the mitigation devices, which causes the detection results to fluctuate repeatedly. In the 512 victim servers test, NetRadar achieves a recall of 95% while NetRadar without random erasing achieves 67%.

Incremental Deployment Test. NetRadar enhances carpet bombing detection performance with server-side features feedback. However, full-scale deployment across all servers may not be feasible in practical scenarios. Therefore, we test the performance of NetRadar when some servers are not available for server-side features feedback. We use the Synchronous Download dataset in §VI-A and randomly remove server-side features from 0% to 100%. Specifically, we select a group of servers, remove their server-side features, and pad the feature vectors with 0 to the original length. The detection model and training process remain unchanged. Note that this preprocessing method does not conflict with runtime feature change, and the server-side features are still affected by the real-time defence strategy during the simulation.

Results are shown in Figure. 13. In general, the detection accuracy of NetRadar is positively correlated with the proportion of server-side features feedback deployment. NetRadar trains the detection model with the random erasing mechanism, which also has a positive effect on this test, as the model learns to make decisions based on traffic features. The results of this test demonstrate that NetRadar supports incremental deployment. Partial-deployed server-side features feedback mechanism can enhance the detection performance. NetRadar user can choose a suitable deployment scale based on detection performance gain and corresponding deployment cost.

VII. Discussion

Features Synchronization. Real-time detection solutions must make decisions before the suspicious flows end. While analyzing complete flow features enables ideal classification accuracy, it also means that subsequent defense processes (such as DDoS mitigation) will be significantly delayed. Therefore, most solutions use a fixed number of packets as a trigger for starting the detection process [16]–[18]. However, this packet-trigger method is not suitable for cross-server inbound traffic analysis. Servers in a victim network receive traffic at varying throughputs, causing features to be generated at different rates across servers. When these asynchronous features from multiple servers are fed into the detection model for cross-server analysis, the temporal misalignment may significantly degrade detection performance.

In NetRadar, we use a synchronized time interval for feature extraction, with detection triggered at each interval. We refresh the features of each server and trigger the DDoS detection process every second to ensure temporal alignment of the input features. While this synchronization enables effective cross-server analysis, it introduces a detection latency where the system identifies victim servers based on features from the previous second. This delay creates a vulnerability window where carpet bombing attacks with a sub-second target switching interval can evade detection. We plan to optimize the detection delay by adaptively adjusting the feature extraction time interval according to network status in future work.

Server Scheduling. NetRadar currently uses a static server scheduling configuration when the server feedback mechanism is not fully deployed. While evaluation results in the incremental deployment test demonstrate that, with this simple scheduling strategy, NetRadar's performance is positively correlated with the number of feedback-enabled servers, the static host scheduling strategy is far from perfect.

An ideal host scheduling strategy needs to comprehensively consider the service similarity, changes in attacks, and real-time service load to dynamically adjust the configuration at runtime. Such an adaptive scheduling strategy can further optimize the detection accuracy of the system while reducing the transmission and computational overhead associated with the server feedback mechanism. However, developing this optimized scheduling strategy requires detailed knowledge of specific victim network topologies and service characteristics. Currently, we simulate the behavior of NetRadar for preliminary test and leave this for future work.

Real-world Deployment. Although NetRadar employs simulation experiments for initial validation, its design prioritizes real-world deployability. Firstly, the system eliminates the need for complex traffic feature extraction or traffic decryption, significantly simplifying monitoring system design. Required traffic features rely only on packet header information and are queried at a low frequency, which can be easily implemented on modern programmable data planes [35] at line rate. The simplicity of features also allows NetRadar to easily adapt to the network with multiple access points, where features from multiple gateways can be aggregated to achieve comprehensive traffic analysis. Secondly, server-side features can be acquired through established methods—such as eBPF-based monitoring on the servers themselves or cloud-based solutions like DeepFlow [36]. These approaches are capable of collecting server-side features at a similar or even finer granularity to support NetRadar's deployment.

VIII. Related Works

New Variant of DDoS. As a long-standing threat to the Internet, many new variants of DDoS have appeared in recent years. Pulse-wave DDoS targets defense systems' response time by generating malicious traffic with a periodic on-off pattern. A recent solution addresses this attack by deprioritizing malicious traffic on programmable switch [37]. Carpet bombing DDoS also requires low-latency defense mechanisms, as attackers continuously shift target servers within the victim network. Link-flooding attacks (LFAs) target network links rather than servers, potentially cutting off the Internet connection to selected victim network [19], [23]–[25]. Similarly, link-flooding attacks use low-rate normal-looking traffic to evade DDoS detection. However, launching an LFA is far more costly than launching a carpet bombing attack, as it necessitates the collection and analysis of routing information to construct a link map based on network topology in advance. On the contrary, the carpet bombing attacker only needs to acquire the victim's IP addresses to launch an effective attack.

Collaborative DDoS Defence. Given the distributed nature of DDoS attacks, numerous studies have adopted multi-device collaborative defense strategies. DefCOM [38] organizes existing defense devices into a defensive overlay network to enable coordinated detection and mitigation within ISP networks. Sketch-based methods [12], [15], [23] typically employ distributed data collection and centralized processing to achieve network-wide collaborative detection, generally applied in ISP networks. Meanwhile, approaches like SENSS [39] and IETF DOTS [40] facilitate collaborative mitigation by allowing victim networks to request defense services from remote mitigation service providers. Unlike most existing solutions deployed in ISP networks that primarily target simple volumetric DDoS, NetRadar enables collaboration between victim servers and the gateway within the victim network, achieving accurate carpet bombing attack detection — a task that is challenging for the gateway to perform independently.

IX. Conclusion

This paper tackles detecting carpet bombing attacks, with a particular focus on those that generate malicious traffic through application-layer attack methods. We propose NetRadar, a DDoS detector that achieves accurate and robust carpet bombing detection. NetRadar introduces server-assisted detection and cross-server inbound traffic analysis in carpet bombing detection with a simple server-gateway cooperation architecture. Our evaluation with real-world and simulated traces shows that NetRadar is capable of detecting both high-rate and low-rate carpet bombing attacks with high accuracy. Moreover, NetRadar achieves robust detection even in extreme covert low-rate application-layer carpet bombing scenarios.

References

[1] China Telecom, China Unicom, Baidu Security, Nexusguard, China Mobile, Tsinghua University and Huawei, "Global DDoS Attack Status and Trend Analysis in 2023," https://e.huawei.com/eu/material/networking/security/0c561b8fd2d342999cd402bcecf6d452, accessed in 2024.

[2] CORERO and Juniper Networks, "2023 DDoS Threat Intelligence Report," https://www.juniper.net/content/dam/www/assets/analyst-reports/us/en/2023/corero-ddos-threat-intelligence-report.pdf, accessed in 2024.

[3] NETSCOUT, "Carpet-Bombing A Shotgun Approach to DDoS Attacks in 2024," https://www.netscout.com/blog/asert/carpet-bombing, accessed in 2024.

[4] T. Heinrich, C. A. Maziero, N. C. Will, and R. R. Obelheiro, "How DRDoS attacks vary across the globe?" in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022.

[5] T. Heinrich, R. R. Obelheiro, and C. A. Maziero, "New kids on the DRDoS block: Characterizing multiprotocol and carpet bombing attacks," in *International Conference on Passive and Active Network Measurement*, 2021.

[6] Q. Mao, D. Makita, M. van Eeten, K. Yoshioka, and T. Matsumoto, "Characteristics comparison between carpet bombing-type and single target drdos attacks observed by honeypot," *Journal of Information Processing*, 2024.

[7] NSFOCUS, "A Deep Dive into DDoS Carpet-Bombing Attacks," https://nsfocusglobal.com/a-deep-dive-into-ddos-carpet-bombing-attacks/, accessed in 2025.

[8] Nokia, "Threat Intelligence Report 2025," https://www.nokia.com/asset/215118/, accessed in 2025.

[9] K. Singh, P. Singh, and K. Kumar, "Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges," in *Computers & Security*, 2017.

[10] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," in *IEEE Communications Surveys & Tutorials*, 2013.

[11] A. Praseed and P. S. Thilagam, "DDoS Attacks at the Application Layer: Challenges and Research Perspectives for Safeguarding Web Applications," in *IEEE Communications Surveys & Tutorials*, 2019.

[12] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[13] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *Network and Distributed Systems Security (NDSS) Symposium*, 2020.

[14] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[15] W. Wu, Z. Li, X. Liu, Z. Wang, H. Pan, G. Zhang, and G. Xie, "Lemon: Network-Wide DDoS Detection with Routing-Oblivious Per-Flow Measurement," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

[16] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications." in *Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[17] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[18] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime Robust Malicious Traffic Detection via Frequency Domain Analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[19] M. S. Kang, S. B. Lee, and V. D. Gligor, "The Crossfire Attack," in *2013 IEEE Symposium on Security and Privacy (SP)*, 2013.

[20] H. Griffioen, K. Oosthoek, P. van der Knaap, and C. Doerr, "Scan, test, execute: Adversarial tactics in amplification DDoS attacks," in *Proc. of CCS*, 2021.

[21] Alibaba, "Alibaba Cloud Anti-DDoS," https://www.alibabacloud.com/product/ddos, accessed in 2024.

[22] Tencent, "Tencent Cloud Anti-DDoS," https://www.tencentcloud.com/products/ddos-pro, accessed in 2024.

[23] H. Zhou, S. Hong, Y. Liu, X. Luo, W. Li, and G. Gu, "Mew: Enabling Large-Scale and Dynamic Link-Flooding Defenses on Programmable Switches," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[24] J. Xing, W. Wu, and A. Chen, "Ripple: A Programmable, Decentralized Link-Flooding Defense Against Adaptive Adversaries," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[25] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis," *IEEE Transactions on Information Forensics and Security*, 2018.

[26] Cisco, "Cisco IOS NetFlow," https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html, accessed in 2023.

[27] Cloudflare, "4.2 Tbps of bad packets and a whole lot more: Cloudflare's Q3 DDoS report," https://blog.cloudflare.com/ddos-threat-report-for-2024-q3/, accessed in 2024.

[28] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[29] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proc. of CVPR*, 2017.

[30] M. Wang, S. Huang, Y. Cui, W. Wang, and Z. Liu, "Learning Buffer Management Policies for Shared Memory Switches," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 2022.

[31] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.

[32] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *Network and Distributed Systems Security (NDSS) Symposium*, 2018.

[33] NetRadar, https://github.com/Neptune17/NetRadar-opensource.git.

[34] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," *Advances in neural information processing systems*, vol. 30, 2017.

[35] Intel, "Intel Tofino," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, accessed in 2023.

[36] J. Shen, H. Zhang, Y. Xiang, X. Shi, X. Li, Y. Shen, Z. Zhang, Y. Wu, X. Yin, J. Wang *et al.*, "Network-centric distributed tracing with deepflow: Troubleshooting your microservices in zero code," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023.

[37] A. G. Alcoz, M. Strohmeier, V. Lenders, and L. Vanbever, "Aggregate-based congestion control for pulse-wave DDoS defense," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022.

[38] G. Oikonomou, J. Mirkovic, P. Reiher, and M. Robinson, "A framework for a collaborative DDoS defense," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, 2006.

[39] S. Ramanathan, J. Mirkovic, M. Yu, and Y. Zhang, "SENSS against volumetric DDoS attacks," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018.

[40] F. Andreasen, N. Teague, and R. Compton, "RFC 8811: DDoS Open Threat Signaling (DOTS) Architecture," 2020.

Appendix A
Artifact Appendix

In this section, we provide information about obtaining the source code and dataset of NetRadar. This artifact is designed to reproduce the main experimental results presented in the Evaluation section of the paper on a standard GPU-equipped machine. Researchers should be able to reproduce results related to End to End Performance (TABLE I), Detection Robustness (Figure 8), Ablation Test (Figure 9), and Cross-Server Analysis Efficiency (Figure 10) using this artifact. Additionaly, we also provide the flexibility to customize the carpet bombing attack generation process, allowing researchers to explore various carpet bombing attack scenarios.

A. Description & Requirements

1) How to access: The source code of NetRadar (namely `netradar_opensource.tar.gz`), including the config files needed to reproduce the main experiments in Evaluation section is available on hotcrp.com. This artifact is available on Zenodo (https://doi.org/10.5281/zenodo.17582526).

2) Hardware dependencies: A GPU-equipped machine is required to run this artifact (tested on NVIDIA GTX 1080, more advanced models are preferable). Furthermore, as the experiments involve generating multiple carpet bombing attack traffic test datasets, it is recommended to have at least 200 GB of available disk space (excluding the original dataset).

3) Software dependencies: This artifact run on Linux system and utilizes libpcap for feature extraction, scikit-learn and PyTorch for training machine learning models (tested on Ubuntu 22.04 with libpcap-dev 1.10.1, scikit-learn 1.7.2 and torch 2.7.1). Detailed descriptions of the software dependencies are provided in the `README.md` file in the package.

4) Benchmarks: NetRadar utilizes the CICIDS2017, CICIDS2018 dataset and a real-world dataset from a top-tier cloud service provider in Evaluation. We set up multiple carpet bombing attack detection tasks based on these datasets. For baseline comparisons, NetRadar is evaluated against Flowlens [30] and Whisper [31]. We implement these baselines according to the specifications in their respective publications. Furthermore, we implement ablated versions of NetRadar with specific components removed and naive MLP carpet bombing detectors to conduct ablation tests and related experiments.

B. Artifact Installation & Configuration

Please install the corresponding software packages according to the requirements listed in the `README.md`. Note that the version numbers specified in the `README.md` reflect our testing environment; newer software versions (and more advanced hardware) should also be compatible. The CIC-IDS{2017,2018} dataset is not included in the artifact and needs to be downloaded from its official website. Please store it in directory according to the `README.md`.

C. Experiment Workflow

The experiment workflow is organized as follows:

1) Step 1: Dataset Preprocessing: The raw pcap files from the CIC-IDS{2017,2018} dataset are partitioned into individual HTTP flow captures. URLs and flow lengths are subsequently extracted to serve as server-side feature inputs for NetRadar.

2) Step 2: Test Dataset Construction: Carpet bombing attack traffic is generated according to specific experiment configurations. This process involves replicating attack traffic to multiple victim IPs, with periodic rotation of target victim IP sets to emulate carpet bombing attack behavior. Parameters such as number of victim IPs and frequency of IP rotation are configured to set up multiple attack detection tasks.

3) Step 3: Performance Evaluation: Using the constructed test datasets, the detection performance of NetRadar and baseline methods is evaluated through feature extraction, model training, and inference phases to simulate the detection performance of each solution in carpet bombing detection tasks.

It is worth noting that Step 1 is a one-time procedure. We organize Steps 2&3 as multiple experiment configurations in this artifact (End to End Performance for example). Each configuration corresponds to the generation of test datasets and the subsequent performance evaluation of the relevant solutions for one task in Evaluation.

D. Major Claims

- (C1): NetRadar achieves better detection accuracy compared with state-of-the-art DDoS detection methods on both high-rate and low-rate carpet bombing DDoS detection. This is proven by the experiment (E1) whose results are reported in Table I.
- (C2): NetRadar achieves robust detection even in extreme covert low-rate carpet bombing attack detection tasks. This is proven by the experiments (E2) whose results are reported in Figure 8.
- (C3): The performance gains of NetRadar are attributed to its two key mechanisms: Server-Assisted Detection and Cross-Server Inbound Traffic Analysis (proven by experiments (E3) whose results are reported in Figure 9). The Cross-Server Inbound Traffic Analysis component demonstrates superior scalability and detection accuracy compared to naive MLP-based approach (proven by experiments (E4) whose results are reported in Figure 10).

E. Evaluation

This artifact organizes the experiments into one dataset preprocessing script and four distinct experiment configurations (corresponding to the Experiments below), all of experiments can be executed automatically through a single script.

[Prepare] Firstly, please perform dataset preprocessing before running experiments:

```
$ sh scripts/compile_cpp_datacooking.sh
$ sh scripts/run_cicids2017_preprocess.sh
$ sh scripts/run_cicids2018_preprocess.sh
$ sh scripts/run_malicious_pcap_preprocess.sh
```

[Execute] Then, run all experiments proposed in this artifact through a single script as follows:

```
$ sh autorun.sh
```

Individual experiments can also be executed by running the corresponding commands in the `autorun.sh` script.

1) Experiment (E1): [End to End Performance]: This experiments evaluates the end-to-end detection performance of NetRadar and baseline methods in carpet bombing attack detection tasks. The performance of three distinct solutions was evaluated across three different scenarios. Detailed configurations for these experiments are specified in the **Experiment Setup** of the Evaluation section. The corresponding configs and reference results are located in `configs/table_1_end_to_end`.

[Results] Compare evaluated results of NetRadar, Whisper and Flowlens in `autorun_cooked.log` with Table I.

2) Experiment (E2): [Detection Robustness]: This experiments evaluates detection performance of NetRadar and baseline methods on extreme covert low-rate carpet bombing DDoS dataset. We build multiple modified Synchronous Download test datasets with the malicious traffic throughput ratio of 1/1 to 1/16 to test detection robustness of NetRadar. The corresponding configs and reference results are located in `configs/figure_8_detection_robustness`.

[Results] Compare evaluated result figures of F1-score, precision, and recall with Figure 8 in paper.

3) Experiment (E3): [Ablation Test]: This experiments evaluates detection performance of NetRadar and its ablated versions with specific components removed to demonstrate the performance gain of NetRadar's designs. The corresponding configs and reference results are located in `configs/figure_9_ablation_test`.

[Results] Compare evaluated figure with Figure 9 in paper.

4) Experiment (E4): [Cross-Server Analysis Efficiency]: We compare straightforward cross-server analysis model (naive MLP) with NetRadar on detection accuracy and detection throughput to prove that NetRadar is more efficient in handling cross-server traffic analysis. The corresponding configs and reference results are located in `configs/figure_10_efficiency`.

[Results] Compare evaluated figure with Figure 10 in paper.

F. Customization

All configurable parameters in this artifact are consolidated within a JSON configuration files, including settings for carpet bombing attack generation, baseline, and training hyperparameters (`configs/config_0.json` in End to End Performance config directory for example). This section focuses specifically on customizing the carpet bombing attack generation configuration.

The following parameters for simulating carpet bombing attacks can be modified through the configuration file:

- Total number of victim network IP addresses
- Number of IPs targeted simultaneously by attacker
- Frequency at which attackers switch targeted IP addresses
- Attack traffic used by attacker (pcap files)

Change these parameters in the JSON configuration file and run the corresponding experiment script. The system automatically reads these JSON configurations to dynamically construct corresponding traffic, enabling flexible emulation of diverse carpet bombing attack behaviors.