Robust and Efficient Detection of DDoS Attacks for Large-Scale Internet

Kejie Lu, Jieyan Fan, Dapeng Wu
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611

Sinisa Todorovic
Computer Vision and Robotics Lab
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Antonio Nucci
Narus, Inc.
500 Logue Avenue
Mountain View, CA 94043

***Abstract*— In recent years, distributed denial of service (DDoS) attacks have become a major security threat to Internet services. How to detect and defend against DDoS attacks is currently a hot topic in both industry and academia. In this paper, we propose a novel framework to robustly and efficiently detect DDoS attacks and identify attack packets. The key idea of our framework is to exploit spatial and temporal correlation of DDoS attack traffic. In this framework, we design a perimeter-based anti-DDoS system, in which traffic is analyzed only at the edge routers of an internet service provider (ISP) network. The novelties of our framework are 1) temporal-correlation based feature extraction and 2) spatial-correlation based detection. With these techniques, our scheme can accurately detect DDoS attacks and identify attack packets without modifying existing IP forwarding mechanisms at routers. Our simulation results show that the proposed framework can detect DDoS attacks even if the volume of attack traffic on each link is extremely small. Especially, for the same false alarm probability, our scheme has a detection probability of 0.97, while the existing scheme has a detection probability of 0.17, which demonstrates the superior performance of our scheme.**

I. Introduction

In recent years, *distributed denial of service* (DDoS) attacks have become a major security threat to Internet services. DDoS attacks can consume lots of resources of a server, making legitimate users unable to access the server. With the exponential increase of Internet-based e-business and e-commence, the damage caused by DDoS attacks is more severe than ever before. Therefore, how to defend against DDoS attacks and protect the access of legitimate users has attracted attention from both industry and academia. However, achieving this objective is not an easy task due to the difficulty in distinguishing attack traffic from normal traffic.

To address this problem, two types of anti-DDoS systems, i.e., host-based systems and network-based systems [1], [2], have been developed. Host-based systems are deployed on end-hosts. These systems typically use firewall and intrusion detection systems (IDS), and/or balance the load among multiple (geographically dispersed) servers to defend against DDoS attacks. The host-based approach can help protect the server system; but it may not be able to protect legitimate access to the server, because high-volume attack traffic may congest the incoming link to the server.

On the other hand, network-based anti-DDoS systems are deployed inside networks, e.g., on routers. Network-based anti-DDoS techniques can be classified into two categories: 1) detection/identification, and 2) defense. A detection/identification mechanism is responsible for detecting DDoS attacks and identifying attack packets or attack sources. To detect DDoS attacks, signal processing techniques (e.g., wavelet [4], spectral analysis [5], [6], statistical methods [14], [17], [19]), and machine learning techniques [8] can be used. To identify attack sources, IP traceback [10] is typically used. The IP traceback techniques can help contain the attack sources; but it requires large-scale deployment of the same IP traceback technique and needs modification of existing IP forwarding mechanisms (e.g, IP header processing), which may not be cost-effective.

To defend against DDoS attacks, traffic control mechanisms such as ingress filtering [11], route-based packet filtering [12], and rate limiting [13], are usually used. Ingress filters or packet filters [11], [12] can drop packets with spoofed source IP addresses that do not belong to the upstream networks; but their effectiveness depends on global deployment of these filters in the Internet; with a partial deployment, spoofing source IP addresses is possible. Rate limiter [13] are deployed at each link of certain designated routers; they indistinguishably drop some of the packets destined to a victim, when the victim is overwhelmed by (possible attack) traffic. In this way, the volume of attack traffic can be limited. Rate limiting is suitable for mitigating attacks having high-data-rate on a link; but it is not suitable for mitigating attacks having low-data-rate on a link, since attacks with low-data-rate on a link will not trigger rate limiting operation.

In this paper, we take a *network-based* approach and propose a novel framework to detect and identify DDoS attacks. The key idea of our framework is to exploit spatial and temporal correlation of DDoS attack traffic. In this framework, we design a perimeter-based anti-DDoS system, in which traffic is analyzed only at the edge routers of an Internet service provider (ISP) network. The anti-DDoS system consists of two major components: 1) feature extraction and 2) detection. Our feature extraction scheme exploits temporal correlation between the outgoing traffic and the incoming traffic on a link, which makes distinct features between normal and attack traffic; we call these features *2D matching features*, which

will be defined in Section III-D.2. The 2D matching features are used by the detection module. In detection, our machine learning algorithm is able to utilize the spatial correlation of DDoS attack traffic at different routers. Our system has the following advantages.

- Different from the existing network-based traffic control systems, our system is able to *accurately* detect attacks having low-data-rate on a link and *accurately* identify legitimate flows. Accurately detecting attacks having low-data-rate on a link is important because DDoS attackers are getting smarter and trying to hide their presence by launching attacks from (geographically dispersed) thousands of compromised machines, each of which generates low-rate attack traffic. Accurately identifying legitimate flows is critical in defense since it can help filter out attack traffic.
- Compared to IP traceback, our scheme can effectively detect attacks and identify attack packets without modifying existing IP forwarding mechanisms and without a large-scale upgrade to backbone routers.
- Our network-based system has an advantage over a host-based system, in that designated routers can throttle the attack traffic by forwarding packets from the identified legitimate flows only.

Simulation results show that the proposed framework can detect DDoS attacks even if the volume of attack traffic on each link is extremely small. Especially, for the same false alarm probability, our scheme has a detection probability of 0.97, while the existing scheme has a detection probability of 0.17, which demonstrates the superior performance of our scheme.

The rest of this paper is organized as follows. In Section II, we briefly overview the related work. Section III presents our framework for DDoS detection. In Section IV, we will develop a powerful machine learning algorithm that can analyze the spatial correlation of DDoS traffic feature. A test algorithm based on the machine learning algorithm will be discussed in Section V. Simulation results and discussions will be presented in Section VII, followed by the conclusions in Section VIII.

II. Related Work

In this section, we scan related work on feature extraction and detection.

A. Feature Extraction

In [14], Wang *et al.* proposed to detect TCP SYN flood by using the ratio of the number of TCP SYN packets to the number of TCP FIN and RST packets. Ideally, if there are no SYN flood attacks, this ratio will be close to 1 for a period that is sufficiently long, since most TCP sessions begin with a SYN packet and end with a FIN packet. However, one of the main difficulties of this scheme is that the duration of some TCP sessions can be very large, which means that the ratio may not be close to 1 for a short period. To overcome this problem, Wang *et al.* proposed to use the ratio of the number of SYN packets (on the incoming direction of a link) to the number of SYN/ACK packets on the outgoing direction of the link [15], since the time between the arrival of an SYN and the arrival of the corresponding SYN/ACK packet is related to the round-trip time of the connection, which has much less variation compared to the life-time of TCP sessions. Nevertheless, this ratio of SYN to SYN/ACK does not make distinct features between normal and attack traffic since DDoS attackers can generate attack traffic that makes the ratio of SYN to SYN/ACK close to 1, just like normal traffic. This results in poor performance on detecting DDoS attacks.

It is well known that most DDoS attacks use spoofed source IP addresses. Moreover, the number of packets from the same spoofed source IP address is relatively small, compared to the number of packets of a real session. Consequently, to generate a huge amount of attack traffic, a large number of spoofed source IP addresses need to be created. Based on this assumption, Peng *et al.* [16], [17] proposed to use the ratio of the number of new IP address to the total number of IP addresses to detect attacks with spoofed source IP addresses. In their scheme, a database is required to store the information of all IP addresses that appeared in a certain period, which means that the required memory size is very large for large-scale Internet. Hence, their scheme is not suitable for large-scale ISP networks.

Different from Ref. [16], [17] that only consider new IP addresses, Ref. [18] uses entropy of the IP address distribution as a feature for detection; but the complexity of calculating entropy for large-scale Internet can be extremely high.

To summarize, the ratio of SYN to SYN/ACK [15], the percentage of new IP addresses [16], [17], and entropy of the IP address distribution [18] have been used as features for detecting DDoS attacks. Features are important since they significantly affect the performance of detectors. The aforementioned existing features either do not lead to good performance of detectors, or require high storage/time complexity. To address these deficiencies, this paper proposes a hash-table based feature extraction scheme to efficiently extract so-called *2D matching features*, which makes distinct features between normal and attack traffic, thereby improving accuracy of detecting attacks.

B. Detection

Once features are extracted from the measurement data (obtained by traffic measurement devices), the features can be used in detecting DDoS attacks. To detect DDoS attacks, signal processing techniques (e.g., wavelet [4], spectral analysis [5], [6], statistical methods [14], [17], [19]), and machine learning techniques [8] can be used.

In [4], Kim *et al.* proposed to use wavelet to analyze traffic but it is not clear whether their scheme can detect attacks having low-data-rate on a link. In [5], [6], spectral analysis was used to detect DDoS attacks; it is assumed that high volume attack traffic causes significant changes in the power spectral density of traffic; but the technique may not be effective in detecting attacks having low-data-rate on a link, since attacks having low-data-rate may not cause large changes in the power

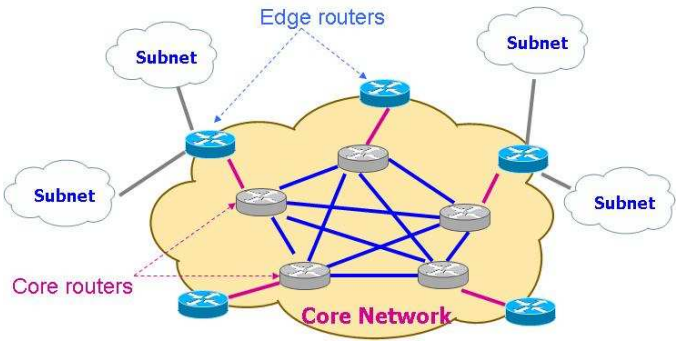


Fig. 1. A general structure of an ISP Network.

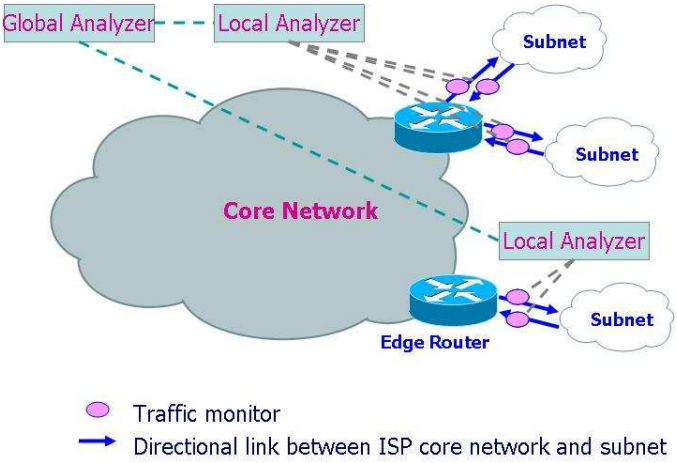


Fig. 2. System model.

spectral density of traffic. In this paper, we design a detection technique that addresses both high rate and low rate attacks.

A statistical method called change-point method was used to detect DDoS attacks [14], [17], [19]. Specifically, the change-point method is used to detect attack-induced abrupt changes in statistical patterns of traffic, compared to the "normal traffic pattern". However, the parameters of the existing change-point algorithms [14], [17], [19] are constant and preset *a priori* for a given traffic pattern; it is not clear how to dynamically adapt the values of these parameters when the traffic pattern changes. To address this problem, in this paper, we use machine learning techniques to make our scheme robust against time-varying traffic patterns.

In [8], Mukkamala and Sung employed a machine learning technique called support vector machine to detect DoS attacks; but their algorithm was not tested for networks with a large IP address space, which may significantly increase the time/storage complexity of detection algorithms. In this paper, we will test our machine learning algorithm for networks with a large IP address space.

III. Framework for DDoS Attack Detection

In this section, we propose a novel framework for detecting DDoS attacks. In this framework, we consider the following components.

- Network model: The network model defines the logical structure of an ISP network and defines where the DDoS detection system will be installed.
- Service model: The service model defines how an ISP and its customers can benefit from this DDoS detection system and how the DDoS detection system works from an administrative perspective.
- System model: The system model defines the components of the DDoS detection system.

A. Network Model

In our framework, we consider typical ISP networks, as illustrated in Fig. 1. We assume that the ISP network is composed of two types of IP routers: core routers and edge router. The core routers interconnect with one another to form a core network, which is responsible for exchange a huge amount of data in high data rates. By contrast, edge routers are responsible for connecting subnets with the core network and the data rate is relatively low.

B. Service Model

In our service model, the administrator of the ISP network can config the detection system to investigate traffic for different customers at the same time. For example, if a specific customer request to detect abnormal traffic toward its server, the detection system can be config such that only the packets with a specifical destination will be investigated. Meanwhile, another customer can request to detect traffic toward a whole subnet. In this manner, both the ISP and the customers can get benefit from the detection system.

C. System Model

In our framework, there are three types of modules in the system, as illustrated in Fig. 2.

- The traffic monitor is responsible for scanning incoming packets in real-time, summarizing traffic characteristics, and generating simple feature information. The summary of traffic information and simple feature data will be reported to the local analyzer.
- The local analyzer is responsible for generating more complicated feature data and for detecting attacks with local information. The detection results and feature data will be forwarded to the global analyzer.
- The global analyzer is responsible for analyzing feature data from multiple local analyzers and detecting abnormal situations through the machine learning algorithms, which will take into account the spatial and temporal correlation of traffic.

Compared to existing network-based anti-DDoS system, our framework does not require the upgrade of any routers in the network. In the rest of this subsection, we will focus on how to generate feature information and how to analyze these feature data.

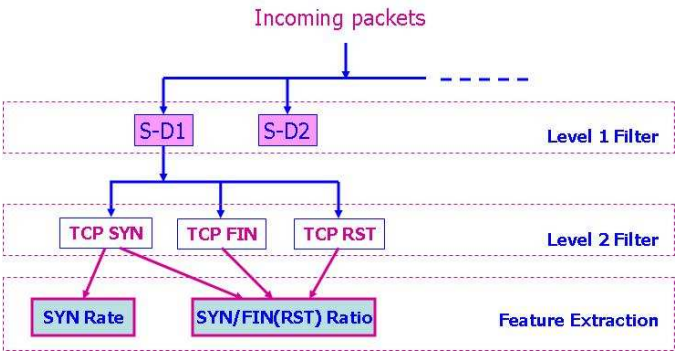


Fig. 3. An illustration of feature extraction.

D. Feature Generation

To extract feature data from traffic, we design a general architecture as shown in Fig. 3. In this architecture, there are basically three types of components: level-one filter, level-two filter, and feature extraction module. While the first two types of components are implemented in the traffic monitor, the feature extraction modules can be implemented in both the traffic monitor and the local analyzer, depending on what the feature is.

In Fig. 3, the level-one filters classify a packet based on the source-destination pair. Here we note that a source-destination pair is defined according to source IP, source network mask, destination IP, destination network mask. For example, we classify differently packets from 172.10.5.28 to 210.33.68.102 with network mask 255.255.255.255, from 172.10.x.x to 208.33.1.x with network mask 255.255.0.0 and 255.255.255.0, respectively. With this scheme, the proposed service model can be supported.

For packets with the same source-destination pair, the level-two filter further classifies packets into types. For example, TCP SYN packet and FIN packets. The matched packets will then be forwarded to one or more feature extraction modules. For example, in Fig. 3, the number of TCP SYN packets can be used to generate both the TCP SYN rate feature and the TCP SYN/FIN(RST) ratio feature. On the other hand, we also note that a certain feature may need the information from multiple level-two filters. For example, the SYN/FIN(RST) ratio feature requires the information from three filters, as illustrated in Fig. 3. Compared to the packet classification schemes in [14] and [17], our architecture is more general and efficient.

1) Feature Extraction Module in The Traffic Monitor: As we mentioned earlier, some features are generated within the traffic monitor. These features are generally simple and can be extracted by using the traffic information of a single uni-directional link.

Similar to previous studies [14], [17], we consider that the feature data are generated in a discrete manner. In other words, the feature extraction module will report a value (or a vector) at the end of each time slot. Here we define the duration of the time slot as T_s, which may vary for different features. Intuitively, a shorter T_s may reduce the detection delay, which is defined as the duration from the time the attack starts to the time epoch that the attack is detected. On the other hand, a smaller T_s may also increase the computational complexity, since the detection algorithm need to analyze the feature data slot by slot. Moreover, the T_s must be sufficient large if the feature is represented by ratio. For example, if we want to use the SYN/FIN(RST) ratio [14] to detect SYN flood, then the T_s cannot be too small, because the number of FIN packets in a short period can be 0, which will result in false alarm even if the number of SYN packets is not large.

In our framework, the traffic monitor can generate the following features:

- Packet rate: the number of packet arrivals in this time slot. This feature is simple but useful to detect high volume DoS and DDoS attacks. It can hardly detect low volume attacks.
- Data rate: the total number of bits of all packets arrived in this time slot.
- SYN/FIN(RST) ratio: the ratio of the number of SYN packets arrived in this time slot and the number of FIN (and a portion of RST) packets arrived in this time slot[1].

2) Feature Extraction Module in The Local Analyzer: Although the traffic monitor can generate simple features efficiently, these features may not be sufficient to detect various attacks. In particular, the packet rate and data rate features may only be useful for high volume attacks; while the performance of SYN/FIN(RST) ratio depends on the duration of TCP sessions, which has a large variation. To improve the performance of detection algorithm, we can use the local analyzer to generate more sophisticated features, for example, the SYN/SYN-ACK ratio proposed in [15] and the new IP ratio proposed in [17].

- SYN/SYN-ACK ratio: the ratio of the number of SYN packets arrived in one direction of a bi-directional link and the number of SYN-ACK packets arrived in the opposite direction of the bi-directional link in one time slot. This scheme can only be deployed at the location that is close to the attacker and can only detect SYN attack with spoofed source IP address.
- New IP ratio: the number of new IP addresses appeared in this time slot versus the total number of IP addresses appeared in this time slot. The feature can only be used to detect attacks with spoofed address and requires a huge history IP address database.

Since the local analyzer can have the traffic information on both directions of a link, we can design a general and more efficient feature extraction module to take into consideration the temporal correlation of traffic information on both directions of the link. The reason is as the following.

For most IP applications, packets are generated from both ends. Therefore, the information carried by packets on one direction shall match with the information carried by packets on the other direction. For example, if station A communicates with station B through TCP, then we can observe packets

[1]Detail discussion on how to get the ratio can be found in [15].

with source A and destination B on one direction, and we can also observe packet with source B and destination A on the opposite direction. On the other hand, if a DDoS attack generate source A on one direction, the response may not appear if the source IP address is spoofed. Therefore, we can utilize this feature to detect a set of DDoS attacks.

We facilitate the discussion, we define a *key* and the information that shall appear on both direction of a link. For instance, we can define the key for TCP SYN packet as:

$$< srcIP, dstIP, srcPort, dstPort, Seq\# >$$

on the opposite direction, we can define the corresponding key for SYN-ACK packet as:

$$< dstIP, srcIP, dstPort, srcPort, Ack\# - 1 >$$

Intuitively, by examining whether a key has matched correspondence on the other direction, we can detect the SYN flood or SYN-ACK flood attacks. Since the proposed features are generated by matching the key information on two directions of a link, we name it *2D matching* features[2]. In the next subsection, we will discuss how to implement the 2D matching features in our framework.

E. Implementation of 2D Matching Features

1) Traffic Monitor: To implement the 2D matching features, the traffic monitors on the two directions of a link must record statistical traffic information in each time slot. In our test bed, we use hash table to store the required information.

The procedure in traffic monitor is as the following.

- For each packet that passed through the level-two filter, a certain hash function will be called to locate the record. If the record is empty, then the key will be stored and a counter will be set to one; if the record is not empty and the key of the packet and the key of the record are identical, then the counter will be added by one; if the record is not empty and the keys are different, then a collision occurs. The collision can be either resolved by some avoidance techniques such as linear probing and rehashing, or can be simply ignored which means that the key information of the incoming packet will not be saved.
- At the end of each time slot, the monitor will send all non-empty records to the local analyzer.

The performance of this implementation depends on two major factors:

- Memory requirement:
  Let the total record in the hash table be N_{ht} and let the length of each record be L_k bytes, then the total memory requirement is $N_{ht} \times L_k$. Here L_k is many depending on the length of the key. For example, for the key for SYN packets, we need 16 bytes to store the key and 4 bytes to store the status information and the counter. Therefore, $L_k = 20$ bytes.
- Processing speed:
  The main time complexity is due to the hash function and key comparing, both of which can be implemented efficiently either through software or hardware (if necessary).
- Communication:
  If the local analyzer and the monitor are not in the same device, then the record information will be transmitted from the monitor to the local analyzer at the end of each slot. In the worse scenario, the data rate requirement is

$$\frac{8 \times N_{ht} \times L_k}{T_s} \quad (b/s)$$

2) Local Analyzer: The main feature data of the 2D matching feature are the number of unmatched and the ratio of unmatched keys. Note that other feature data, including the packet rate, can also be collected. To check whether a key in matched or not, we need to consider the temporal correlation of packets. For example, a SYN packet must followed by a SYN-ACK packet with the same key. Consequently, if we want to determine whether the SYN is matched, we need to investigate the SYN-ACK in the later slots. On the contrary, we need to scan SYN in previous slots to check if a SYN-ACK packet is valid.

In our framework, we design a general mechanism to realize these requirements. We define R_b as the maximum number of slots we want to check backward, and we let R_f be the maximum number of slots we want to check forward. Suppose the current time slot is n, then we will check if a key exists on the opposite direction by investigating the key database in slot m ($\forall n - R_b \leq m \leq n + R_f$). In the previous examples, to check the key of SYN, we can set $R_b = 0$ because the SYN-ACK must appear after SYN; and we can set $R_f > 0$ depending the duration of time slot (i.e. T_s) and the average round trip time of the TCP session. On the other hand, for the key of SYN-ACK, we can set $R_b > 0$ and $R_f = 0$.

To efficiently perform 2D matching, we utilize Bloom Filter for the key database on the opposite direction. Consequently, the memory requirement for the local analyzer is as the following:

- Memory for each Bloom Filter: $2^{(L_{bf}-3)}$ Bytes, where L_{bf} is the size of bloom filter.
- Memory for all bloom filters:

$$(R_b + R_f + 1) * (2^{(L_{bf}-3)}) \quad Bytes.$$

- Memory for all records: $(R_f + 1) * N_{ht} * L_k$ Bytes.

The minimum detection delay is upper bounded by $(R_f + 1) \times T_s$.

F. Detection Algorithms

In both the local analyzer and the global analyzer, certain detection algorithms must be used to detect DDoS attacks. In practice, two simple algorithms are generally used: 1)

[2]In this paper, we assume that the traffic is symmetric on a link, which means that packets from host A to B and packets from B to A can be observed on the opposite directions of a link. In practice, this condition is generally valid for the edge links. It is important to note that, even if the traffic is not symmetric on a link, the key matching can still be conducted if the network topology is known.

threshold-based algorithm and 2) change-point algorithm. Both of them can use the feature data provided by the feature extraction modules to detect abnormalities. However, we note that an important issue has largely been ignored in the literature, that is, how to set the parameter of these algorithms.

Moreover, existing schemes do not take into account the spatial correlation of attacks, which is nature to all DDoS attacks. In the rest of this subsection, we will elaborate on how to determine the parameters of the above two algorithms. We will then introduce a novel machine learning algorithm that can analyze the spatial correlation of feature data.

1) Performance Metrics: To evaluate the performance of different detection algorithms as well as the efficiency of various features, we will use the *Receiver Operating Characteristic* (ROC) method, which has never been used in the previous studies [14], [15], [17], [19].

To understand the usage of ROC, we first define the following parameters

- N_f denotes the number of slots in which the detection algorithm declares an attack given that there are no attacks in these slots;
- N_n denotes the number of slots in which there are no attacks;
- N_d denotes the number of slots in which the detection algorithm declares an attack given that there are attacks in these slots;
- N_a denotes the number of slots in which there are attacks.

We now define the false alarm probability as N_f/N_n and define the detection probability as N_d/N_a. Consequently, given the same feature data, we will get different pairs of false alarm probability and detection probability, when we vary the parameters of the detection algorithm,

2) Threshold Based Algorithm: To utilize the threshold-based algorithm, we only need to set up a threshold for each feature and an attack will be declare if a the feature value exceeds the threshold in any time slot. When we varying the threshold, we will get a ROC curve since the threshold is the only parameter and a smaller threshold will lead to a smaller false alarm probability and large detection probability.

3) Change-point algorithm: In the literature, a simple change-point algorithm — non-parametric *Cumulative Summation* (CUSUM) algorithm — has been widely used [14], [15], [17], [19]. However, existing works have not provided a comprehensive study on how to choose the parameter of CUSUM. Furthermore, these studies only consider the change from normal state to abnormal state, which means that the false alarm can be very large after the end of an attack. To conduct the discussion, we define the following parameters used in CUSUM:

- X_n denotes the observed variable at the end of time slot n.
- $\overline{X}_n$ denotes the expectation of X_n in normal states.
- $\overline{X}_a$ denotes the expectation of X_n in abnormal states. Without losing generality, here we assume that $\overline{X}_n < \overline{X}_a$

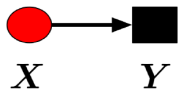


Fig. 4. Graphical representation of the generative process in which the state of traffic "generates" the underlying stochastic process of traffic data.

- Y_n denotes the adjusted variable, which is defined as

$$Y_n = X_n - \alpha,$$

where α is a parameter such that $\overline{X}_n < \alpha < \overline{X}_a$.

Now define parameter S_n as

$$\begin{cases} S_n = 0 & n = 0 \\ S_n = \max(0, S_n - 1) & n > 0 \end{cases} \quad (1)$$

In the CUSUM algorithm, the state is normal if S_n is smaller than a threshold H, otherwise the state is abnormal.

From the discussion above, we note that there are two parameters need to set, namely, α and H. However, we can see that these two parameters are correlated with each other. To overcome this problem, we shall use another parameter, i.e. the detection delay, denoted as D. According to the change-point theory, we have

$$\frac{D}{H} \to \frac{1}{(\overline{X}_a - \overline{X}_n) - |\overline{X}_n - \alpha|} = \frac{1}{\overline{X}_a - \alpha}. \quad (2)$$

From Eq. (2) we can derive

$$H = D \times (\overline{X}_a - \alpha). \quad (3)$$

Consequently, we can vary parameter D and α to get the performance of the CUSUM algorithm[3].

To reduce the false alarm problem of a single CUSUM algorithm (denoted as single-CUSUM), we develop a dual-CUSUM algorithm. In this algorithm, one CUSUM will be used in the normal state to detect the change from normal to abnormal state as soon as possible, while another CUSUM is responsible for detect the state change from abnormal to normal with a minimum delay. The setting of parameters for dual-CUSUM is similar to that of the single-CUSUM algorithm.

4) Machine Learning Algorithm:

IV. Outline of Our Approach

In our approach, we assume that the network states "under attack" or "no attack" generate statistically different traffic. For example, the traffic with the flooding type of attack differs in volume from the traffic in state "no attack". The network state is denoted as X, and traffic data as Y. Graphically, we represent this generating process as depicted in Figure 4.

We propose to represent network-related data (i.e., features) as a matrix F of origin-destination pairs. An entry f in this matrix in the m-th row and n-th column represents a feature vector that characterizes network traffic from source m to destination n. The details on feature extraction are

[3]In [14] and [17], $\alpha = (\overline{X}_a - \overline{X}_n)/2$; thus only the detection delay is needed.

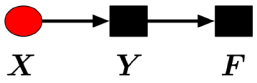


Fig. 5. The extended generative model including traffic feature vectors.

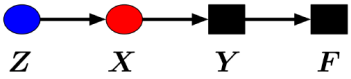


Fig. 6. The complete generative model including traffic data, traffic states, and correlation among traffic states.

explained in Section XXX. Most importantly, in selection of the optimal features, we seek for the most discriminative statistical properties of the corresponding origin-destination traffic. Since features are extracted from traffic data Y we extend the above model as illustrated in Figure 5.

Our goal is to estimate X given Y and F, that is to estimate state x_i of each source-destination traffic pair i, characterized by the i-th entry f_i in the traffic matrix F. Possible values that x_i can take, in our two-class classification problem, are $\{0, 1\}$.

Since DDOS attacks are distributed, where numerous *worms* and *zombies* run similar (or even the same) programs at the same time, we hypothesize that the traffic of several origin-destination pairs is characterized by the same statistical properties. Therefore, it seems reasonable to account for correlation among traffic pairs. Therein lies the novelty of our approach, as we augment the model in Figure 4 with yet another set of random variables, Z, that encode this correlation, as depicted in Figure 6. We anticipate that the introduction of Z may lead to an improved estimation of X.

V. Problem Formulation

Once F is extracted, we assume that F represents well data Y, such that we can operate only over lower-dimensional F, and in this manner reduce computational load. Now, we formulate the network-state estimation as a machine-learning problem, where to each f we assign label x, which takes vales in the predefined set of classes $x \in \{0, 1\}$. The graphical representation of the outlined generative model is depicted in Figure 7. In the graph, nodes represent random variables (vectors), and the connections represent statistical dependencies among random variables. Since feature vectors are measurable, we call them observable random vectors, and depict them as rectangular-shaped nodes. The states of observable vectors need to be estimated; therefore, we call them hidden variables, and depict them as round-shaped nodes. The model in Figure 7 is in fact the simplest possible generative model, since there are no lateral interdependencies among nodes. Consequently, the joint probability of the model reads

$$P(F, X) = \prod_i P(f_i|x_i)P(x_i) \ . \tag{4}$$

From Eq. (4), we observe that the network state estimation can be conducted for each traffic f_i independently, by using

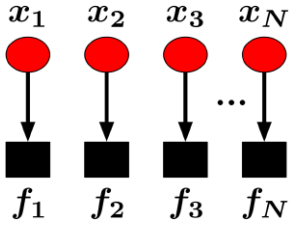


Fig. 7. The generative model that describes dependencies among traffic states and traffic feature vectors.

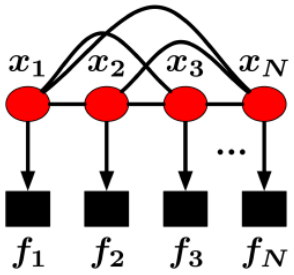


Fig. 8. Complex generative model, where the MAP estimation is intractable.

the *Maximum A Posteriori* (MAP) criterion given by

$$x_i^* = \arg \max_{x_i \in \{0,1\}} P(f_i|x_i)P(x_i) \ . \tag{5}$$

To this end, it is necessary to learn likelihood $P(f_i|x_i)$ and prior $P(x_i)$ in the training process off-line.

We extract yet another network feature – the correlation, ξ_{mn}, between given traffic m and n over a predefined time interval. Consequently, in addition to matrix F, we have the correlation matrix, Ξ, as network features.

The reason for computing Ξ stems from our goal to account for interdependencies among traffic pairs in the network. We speculate that this auxiliary information may lead to a more accurate estimation. However, if we introduced additional connections in the previous model, representing statistical dependencies among all the nodes, we would arrive at computationally intractable model in Figure 8. Here, in order to perform MAP estimation, we would need to marginalize a very complex prior distribution over network states X as

$$x_i^* = \arg \max_{x_i \in \{0,1\}} P(f_i|x_i) \sum_{\substack{j \\ j \neq i}} \sum_{x_j} P(x_1, x_2, ..., x_i, ...x_N) \ . \tag{6}$$

Since the marginalization has to be done for every node, such a model for a large network would not be suitable.

Therefore, the most challenging task in network-state estimation lies in choosing a suitable statistical model for specifying the joint probability distribution over hidden and observable random variables, since this choice conditions the MAP estimation. Our goal is to preserve one-step connections, also known as Markovian dependencies, since they provide for a tractable MAP estimation. But, at the same time, we would like to account for dependencies among as many nodes as possible. To balance these two opposing goals, we propose to use a multiscale tree model, an example of which is depicted

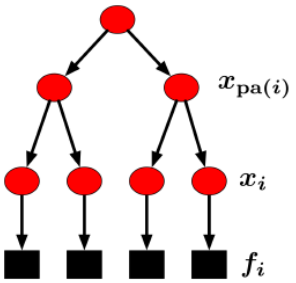


Fig. 9. The quad-tree model for one-dimensional data, where each parent has exactly two children; pa(i) denotes the parent of node i.

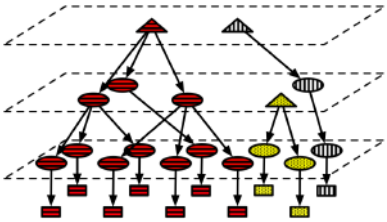


Fig. 10. The irregular tree consists of a forest of subtrees, each marked by distinct shading; round- and square-shaped nodes indicate hidden and observable variables, respectively; triangles indicate roots.

in Figure 9. In the tree, the initial set of traffic states X is augmented with so called parents, grandparents and further up until the root node. That is, hidden variables are organized in levels, where nodes at the finest scale represent states of each source-destination traffic, while nodes at higher levels represent the state of a group of traffic-feature vectors. We assume that hidden variables at higher levels can also take values in $\{0, 1\}$. Connections are allowed only between nodes that belong to adjacent levels in the model. As such, we do not account *explicitly* for all possible dependencies among traffic states, but *implicitly* through parent nodes. In this manner, we preserve Markovian dependencies, while correlating all the nodes.

The next question is how to determine which parent-child pairs should be connected. One possibility is to choose the fixed-structure tree, where, for example, each parent has exactly four children. This model is the well-known Hidden Markov Tree (a.k.a. quad-tree) used for image modeling. However, it has been reported that due to the fixed structure, quad-trees yield "blocky" estimates. Therefore, we propose to use irregular-structure tree (or short irregular tree), where connections between parent and children nodes are not fixed, but rather estimated on a given data.

The novelty of our approach to multiscale statistical modeling is that we introduce connectivity variables, z_{ij}, which regulate if there is a connection between child i and parent j. The connectivity variables, Z, are hidden, and need to be estimated, based on the correlation matrix Ξ. In Figure 6, we graphically illustrate the dependencies between sets of variables F, X, and Z in the proposed multiscale model. From Figure 6, the joint probability of the irregular tree is given by

$$P(F, X, Z, \Xi) = P(F|X)P(X|Z)P(Z|\Xi)P(\Xi)\ . \quad (7)$$

From Eq. (7), we observe that the irregular tree defines the distribution over connections between nodes, and the distribution over node classes (i.e., traffic states). Thus, for a given network data, we need to estimate these two distributions simultaneously. The problem of estimating the optimal model's topology and its distributions is known to NP-hard. We propose to solve the problem by using the Expectation-Maximization (EM) algorithm, which is a stage-wise optimization algorithm, guaranteed to increase the likelihood of the model in each iteration step. Because of the Markovian connections through scales, irregular trees are characterized by very fast inference algorithms, which makes them attractive tools for applications with stringent real-time constraints. The inference of model structure and distributions from the given data produces a hierarchical model depicted in Figure 10. From the figure, we observe that the model structure adapts to encode underlying statistical processes in the data. It consists of a forest of subtrees, since there is no constraint that there be only one root as in quad-trees.

For the Bayesian approach that we propose, it is necessary to learn the parameters of the model through training. To this end, it is necessary to prepare representative examples of the network data containing "under attack" and "no attack" states. Once the parameters of the irregular tree are learned, we are in a position to estimate the state of a given unseen traffic, by computing the posterior distribution of each node state. The probabilistic approach that we propose allows us to easily obtain the ROC curve of our classifier, that is, to test false alarm and detection rate in a principled manner.

VI. Network-State Estimation Algorithm

A. Irregular Tree

In this section we explain in grater detail the irregular-tree model for estimating the traffic states, given network data. In order to fully characterize the irregular tree (and any graphical model, for that matter), it is necessary to learn both the graph topology (structure) and the parameters of transition probabilities between connected nodes from training data. Usually, for this purpose, one maximizes the likelihood of the model over training data, while at the same time minimizing the complexity of model structure. Current methods are successful at learning both the structure and parameters from *complete* data. Unfortunately, when the data is *incomplete* (i.e., some random variables are *hidden*), optimizing both the structure and parameters becomes NP-hard.

One of the principal research topics of this proposal is a solution to the NP-hard problem of model-structure estimation. In our approach, we use a variant of the Expectation-Maximization (EM) algorithm, to facilitate efficient search over large number of candidate structures. In particular, the EM procedure iteratively improves its current choice of parameters by using the following two steps. In Expectation step, current parameters are used for computing the expected value of all the statistics needed to evaluate the current structure.

That is, the missing data (hidden variables) are completed by their mean values. In Maximization step, we replace current parameters with those that maximize the likelihood over the complete data. This second step is essentially equivalent to learning model structure and parameters from complete data, and, hence, can be done efficiently by using the *Belief Propagation* algorithm.

An irregular tree is a directed acyclic graph with V nodes, organized in hierarchical levels, V^ℓ, ℓ=$\{0,1,...,L\}$, where V^0 denotes the leaf level. The layout of nodes is identical to that of the quad-tree, such that the number of nodes at level ℓ can be computed as $|V^\ell|$=$|V^{\ell-1}|/4$=...=$|V^0|/4^\ell$. Connections are established under the constraint that a node at level ℓ can become a root or it can connect only to the nodes at the next ℓ+1 level. The network connectivity is represented by a random matrix, Z, where entry z_{ij} is an indicator random variable, such that z_{ij}=1 if i∈V^ℓ and j∈$V^{\ell+1}$ are connected. Z contains an additional zero ("root") column, where entries z_{i0}=1 if i is a root node.

Each node i is characterized by an image-class random variable, x_i, which can take values in a finite class set C. In our case, $C = \{0,1\}$. For the given Z, the label x_i of node i is conditioned on x_j of its parent j, and is given by conditional probability tables $P(x_i|x_j, z_{ij}$=1). For roots i, we have $P(x_i|x_0, z_{i0}$=1)≜$P(x_i)$. The joint probability of all image-class variables X=$\{x_i\}$, $\forall i$∈V, is given by

$$P(X|Z)=\prod_{\ell=0}^{L}\prod_{i\in V^\ell} P(x_i|x_j, z_{ij}=1). \quad (8)$$

Next, leaf nodes are characterized by observable random variables F=$\{f_i\}$, $\forall i$∈V^0. We assume that observables f_i are conditionally independent given the corresponding x_i:

$$P(F|X) \;=\; \prod_{i\in V^0} P(f_i|x_i)\;, \quad (9)$$

$$P(f_i|x_i=c) \;=\; \sum_{g=1}^{G} \pi_c(g) N(f_i; \mu_c(g), \Sigma_c(g))\;, \quad (10)$$

where $P(f_i|x_i$=$c)$, c∈C, is modeled as a mixture of Gaussians. The Gaussian-mixture parameters can be grouped in θ=$\{\pi_c(g), \mu_c(g), \Sigma_c(g), G_c\}$, $\forall c$∈C.

Finally, we specify the connectivity distribution as

$$P(Z|\Xi) = \prod_{i,j\in V} P(z_{ij}=1|\xi_{ij}) = \prod_{i,j\in V} N(\xi_{ij}; m, \Theta)\;, \quad (11)$$

where ξ_{ij} is the observable correlation between child i and parent j, and m and Θ are the mean and variance of z_{ij}. Note that a higher-level node represents network data at the corresponding coarse scale. Therefore, ξ_{ij} should be suitably extracted as a network feature to represent correlation between groups of traffic. For example, ξ_{ij} may represent the average correlation of traffic i with the traffic of four neighboring nodes whose parent is j. Other strategies are also possible. The details of how to extract $\Xi = \{\xi_{ij}\}$ are discussed in Section XXXX.

The irregular tree is fully characterized by the joint prior $P(F, X, Z|\Xi) = P(F|X)P(X|Z)P(Z|\Xi)$. The introduced parameters of the model can be grouped in the parameter set Ω. In the next section we explain how to infer the "best" configuration of Z and X from the observed image data F and Ξ.

B. Inference of the Irregular Tree

The standard Bayesian formulation of the inference problem consists in minimizing the expectation of some cost function $\mathcal{R}$, given the data

$$(\hat{Z}, \hat{X}) = \arg\min_{Z,X} \mathbb{E}\{\mathcal{R}((Z,X),(Z',X'))|F,\Xi,\Omega\}\;, \quad (12)$$

where $\mathcal{R}$ penalizes the discrepancy between the estimated configuration (Z, X) and the true one (Z', X'). We propose the following cost function:

$$\begin{aligned}&\mathcal{R}((Z,X),(Z',X')) = \mathcal{R}(X,X') + \mathcal{R}(Z,Z') = \\ &=\textstyle\sum_{\ell=0}^{L}\sum_{i\in V^\ell}[1-\delta(x_i-x'_i)] + \\ &\quad\textstyle\sum_{\ell=0}^{L-1}\sum_{(i,j)\in V^\ell\times\{0,V^{\ell+1}\}}[1-\delta(z_{ij}-z'_{ij})],\end{aligned} \quad (13)$$

where $'$ stands for true values, and $\delta(\cdot)$ is the Kronecker delta function. From Eq. (13), the resulting Bayesian estimator of X is

$$\forall i\in V,\quad \hat{x}_i = \arg\max_{x_i\in C} P(x_i|Z,F,\Xi). \quad (14)$$

Next, given the constraints on connections in the irregular tree, discussed in Section VI-A, we derive that minimizing $\mathbb{E}\{\mathcal{R}(Z,Z')|F,\Xi,\Omega\}$ is equivalent to finding a set of optimal parents $\hat{j}$ such that

$$(\forall\ell)(\forall i\in V^\ell)(z_i\neq 0)\;\hat{j} = \arg\max_{j\in\{0,V^{\ell+1}\}} P(z_{ij}=1|\xi_{ij}), \quad (15)$$

where z_i≜$\sum_{k\in V^{\ell-1}} z_{ki}$, and $z_i\neq$0 represents the event "node i has children", that is, "node i is included in the irregular-tree structure." The global solution to Eq. (15) is an open problem in many research areas. We propose a stage-wise optimization, where, as we move upwards, starting from the leaf level ℓ=$\{0,1,...,L\}$, we include in the tree structure optimal parents at $V^{\ell+1}$ according to

$$(\forall i\in V^\ell)(\hat{z}_i\neq 0)\;\hat{j} = \arg\max_{j\in\{0,V^{\ell+1}\}} P(z_{ij}=1|\xi_{ij}), \quad (16)$$

where $\hat{z}_i\neq$0 denotes an estimate that i has already been included in the tree structure when optimizing the previous level V^ℓ.

Equations (14) and (16) suggest that it is possible to solve for $(\hat{Z}, \hat{X})$ in a recursive procedure until some convergence criterion is met. Thus, in a recursive step t, we first assume that estimate $Z(t-1)$ of the previous step $t-1$ is known and then derive estimate $X(t)$ using Eq. (14); then, substituting $X(t)$ in Eq. (16) we derive estimate $Z(t)$. We consider the algorithm converged if $P(F, X|Z)$ does not vary more than some threshold ε for N consecutive iteration steps t, where ε and N are subject to specific application requirements. Although, the optimization given by Eqs. (14) and (16) requires simultaneous optimization of $\hat{X}$ and $\hat{Z}$, note that we infer the irregular tree stage-wise, which may yield sub-optimal solutions. From our experience, though, the algorithm recovers from stationary points for sufficiently large N. The overall inference algorithm is summarized in Figure 11.

Inference Algorithm

(1) $t=0$; initialize irregular-tree structure $Z(0)$ to quad-tree;
(2) Belief Propagation: compute $\forall i \in V,\ \hat{x}_i(0) = \arg\max_{x_i \in C} P(x_i|Z(0), F, \Xi)$
(3) **repeat**
  (4) Expection-Maximization Algorithm
    repeat
      Expectation: $Q_t(\Omega, \Omega') = \mathbb{E}[\log P(Z, \hat{X}(t), F|\Xi, \Omega)|\hat{X}(t), F, \Xi, \Omega]$
      Maximization: $\hat{\Omega}' = \max_{\Omega'} Q_t(\Omega, \Omega')$
      Assign: $\Omega \leftarrow \hat{\Omega}'$
    until convergence
  (5) $t = t+1$;
  (6) Bayesian estimation of structure:
    Compute in bottom-up pass for $\ell = 0, 1, ..., L$
    $(\forall i \in V^{\ell})(\hat{z}_i \neq 0)\ \hat{j} = \arg\max_{j \in \{0, V^{\ell+1}\}} P(z_{ij}=1|\xi_{ij})$;
  (7) Belief Propagation: compute $\forall i \in V,\ x_i(t) = \arg\max_{x_i \in C} P(x_i|Z(t), F, \Xi)$;
  (8) $\hat{X} = X(t)$; $\hat{Z} = Z(t)$;
(9) **until** $|\frac{P(F|\hat{X}) - P(F|X(t-1))}{P(F|X(t-1))}| < \varepsilon$ for N consecutive iteration steps.

Fig. 11. Inference of the irregular tree.

Belief Propagation for the Tree

↓ **Preliminary downward pass:** $\forall i \in V^{L-1}, V^{L-2}, ..., V^0$,
• $P(x_i) = \sum_{x_j} P(x_i|x_j)P(x_j)$,
↑ **Bottom-up pass:**
■ **Initialize leaf nodes:** $\forall i \in V^0$,
  • $P(x_i|f_i) \propto P(f_i|x_i)P(x_i)$,
  • $P(x_i, x_j|f_i) = P(x_i|x_j)P(x_j)P(x_i|f_i)/P(x_i)$,
▲ **compute upward** $\forall i \in V^1, V^2 ..., V^L$,
  • $P(x_i|F_{d(i)}) \propto P(x_i) \prod_{c \in c(i)} \sum_{x_c} \frac{P(x_c|F_{d(c)})P(x_c|x_i)}{P(x_c)}$,
  • $P(x_i, x_j|F_{d(i)}) = P(x_i|x_j)P(x_j)P(x_i|F_{d(i)})/P(x_i)$,
↓ **Top-down pass:**
■ **Initialize root:** $i \in V^L$,
  • $P(x_i|F) = P(x_i|F_{d(i)})$,
  • $\hat{x}_i = \arg\max_{x_i} P(x_i|F)$,
▼ **compute downward** $\forall i \in V^{L-1}, V^{L-2} ..., V^0$,
  • $P(x_i|F) = \sum_{x_j} \frac{P(x_i, x_j|F_{d(i)})}{\sum_{x_i} P(x_i, x_j|F_{d(i)})} P(x_j|F)$,
  • $\hat{x}_i = \arg\max_{x_i} P(x_i|F)$

Fig. 12. Steps 2 and 7 in Figure 11: Belief Propagation for the tree.

Steps 2 and 7 in the algorithm can be interpreted as inference of $\hat{X}$ given F for a fixed-structure tree. In particular, for step 2, where the initial structure is the quad-tree, we can use the standard inference on quad-trees, where, essentially, belief messages are propagated in only two sweeps up and down the tree. For step 7, the irregular tree represents a forest of subtrees, which also have fixed, though irregular, structure; therefore, we can use the very same tree-inference algorithm for each of the subtrees. This algorithm is called Belief Propagation, which we present in Figure 12. In the figure, we simplify notation as $P(x_i|Z, F, \Xi) \rightarrow P(x_i|F)$ and $P(x_i|x_j, Z) \rightarrow P(x_i|x_j)$. Also, we denote with $c(i)$ children of i, and with $d(i)$ the set of all the descendants down the tree of node i including i itself. Thus, $F_{d(i)}$ denotes a set of all observables down the subtree whose root is i. Also, for computing $P(x_i|F_{d(i)})$, in the bottom-up pass, $\propto$ means that equality holds up to a multiplicative constant that does not depend on x_i.

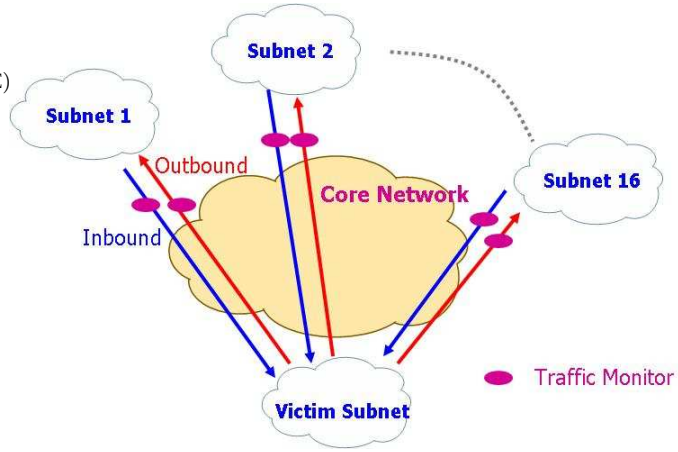


Fig. 13. Experiment Network

VII. Simulation Results

In this section, we evaluate the performance of the proposed framework through simulation. Due to the space limit, in this paper, we only consider the SYN flood attack with spoofed source address.

A. Experiments Setting

1) Network: In our experiment, we assume that the ISP network consists of a core network, a victim network, and 16 subnets, as illustrated in Fig. 13. To simplify the notation, we denote link i as the virtual connection between subnet i and the victim network. Moreover, we define the inbound of link i as the direction from subnet i to the victim network; and the outbound of link i as the direction from the victim network to subnet i.

2) Traffic: For the background traffic, we use the trace data provided by Auckland University [21]. This data set consists of packet header information of traffic between the Internet and Auckland University. The connection is OC-3 ($155Mb/s$) for both directions. In our study, we use the traffic of a single day as the traffic between a specific subnet and the victim subnet, where the inbound traffic is the traffic from the Internet to Auckland University and the outbound traffic is the traffic from the Auckland University to the Internet. Therefore, 16 days of trace are used to represent 16 different links.

To simulate the low volume distributed SYN flood attacks, we randomly add SYN packets with spoofed source IP addresses into the background. The average SYN attack packet rate is 1% of the total packet rate at the same period. For DDoS attacks, the duration of attacks on several links (not necessarily all links) are synchronized.

3) Feature: To detect distributed SYN flood attacks, we use the 2D feature we discussed earlier, i.e. the unmatched SYN packets in one time slot. Note that the number of SYN packets toward the victim subnet can also be reported. Consequently, both the number of SYN packets and the number of unmatched SYN packets will be used in our analysis. The setting of feature extraction module is listed in Table. I.

TABLE I

SETTING OF FEATURE EXTRACTION

Parameter	Setting
T_s	10 sec
R_f	9
R_b	0
N_{ht}	$2^{14} = 16K$
L_{bf}	16

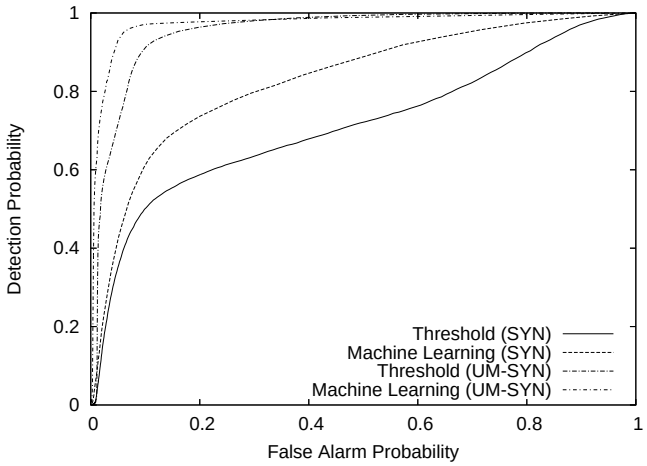


Fig. 15. Performance of threshold-based and machine learning algorithms with different feature data

Since the length of key for the unmatched SYN is $L_k = 20$ bytes, the memory requirement for each monitor is

$$N_{ht} \times L_k = 320 \ KB.$$

In the local analyzer, the memory requirement can be calculated as

$$(R_b + R_f + 1) \times 2^{L_{bf}-3} + (R_f + 1) * N_{ht} * L_k = 3.28 \ MB.$$

Since we choose $R_f = 9$, the minimum detection delay is bounded by

$$(R_f + 1) * T_s = 100 \ sec.$$

Fig. 14 shows the feature data from two links. For these two links, we add synchronized attacks during $1400 - 1600$ and $2800 - 3200$ slots. In addition, asynchronous attacks are also added in $5000 - 5200$ and $5700 - 5900$ slots, for link 1 and link 2, respectively.

From Fig. 14 we can further observe that the feature data are rather noisy, especially for the number of SYN packets. From Fig. 14 (a) and (c) we can hardly distinguish the low volume attacks from background traffic. On the other hand, the attack traffic can be much easily found, when we use the unmatched SYN packets as the feature. Although there are still many anomalies in Fig. 14 (b) and (d), we notice that the anomalies are generally not synchronized.

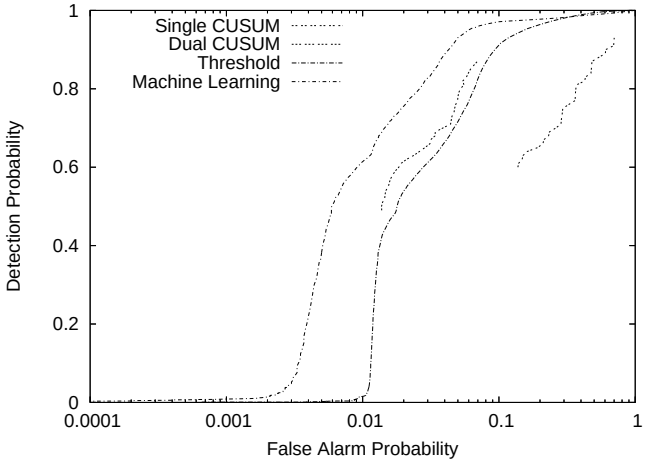


Fig. 16. Performance of four detection algorithms

B. Performance Comparison

Fig. 15 compares the ROC of threshold-based and machine learning algorithms with two different features: the number of SYN packets and the number of unmatched SYN packets. We can first observe that, with the same detection algorithm, using the number of unmatched SYN packets can significantly improve the ROC. In other words, given the same false alarm probability, the detection probability is much higher when we use the unmatched SYN as feature.

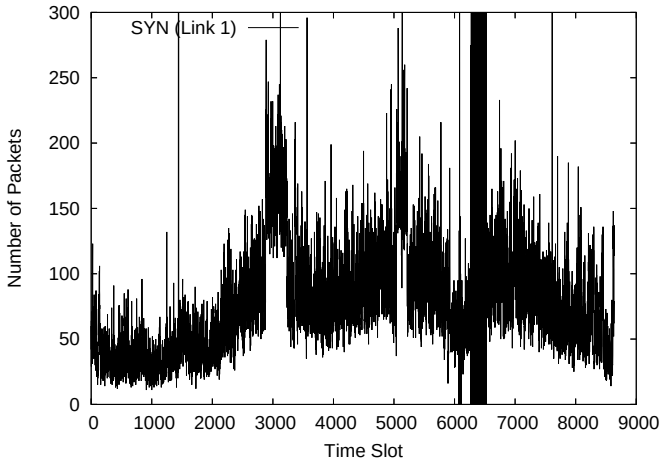
Another important result from Fig. 15 is that the machine learning algorithm can significantly improve the ROC given the same feature data, which validate our expectation.

In Fig. 16 we compare the performance of four detection algorithms with the same feature, i.e. the number of unmatched SYN packets. We first note that, since there are more than one parameters for the single-CUSUM and dual-CUSUM algorithms, the ROC of each of these two algorithms is actually an area instead of a curve. In particular, we plot all points for the single CUSUM algorithm and the contour for the dual-CUSUM algorithm, since there are too many points within this area. For these two algorithms, the detection delay D is chosen from 1 to 10 and the adjust parameter can be chosen from

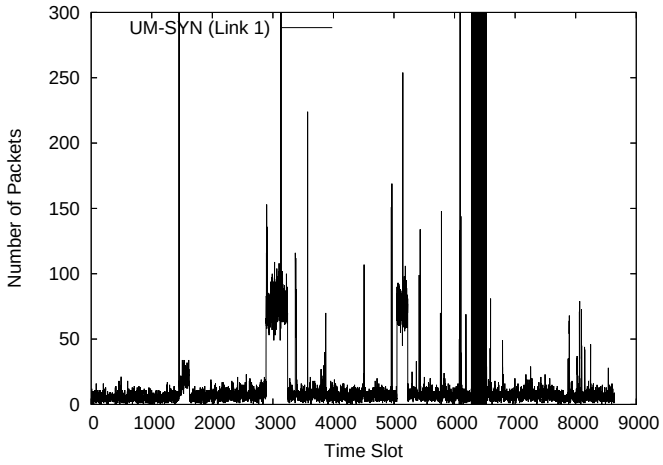
$$\alpha_i = (A_{attack} - A_{normal}) \times \frac{i}{17}, \quad \forall \ 1 \le i \le 16,$$

where A_{attack} and A_{normal} are the average number of unmatched SYN packets in attack and normal conditions, respectively.

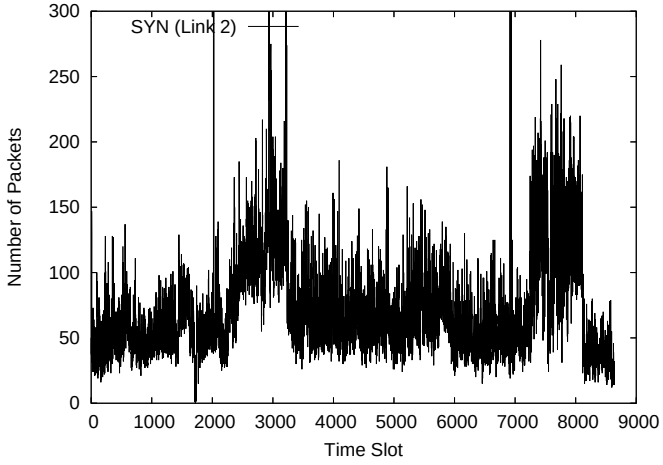
The results in Fig. 16 show that, the performance of the machine learning algorithm is the best amongst all algorithms. We can also see that dual-CUSUM can out-perform the simple threshold-based algorithm with appropriate setting of parameters. Finally, the results reveal that the single-CUSUM algorithm shall not be used in practice since its performance is the worst.



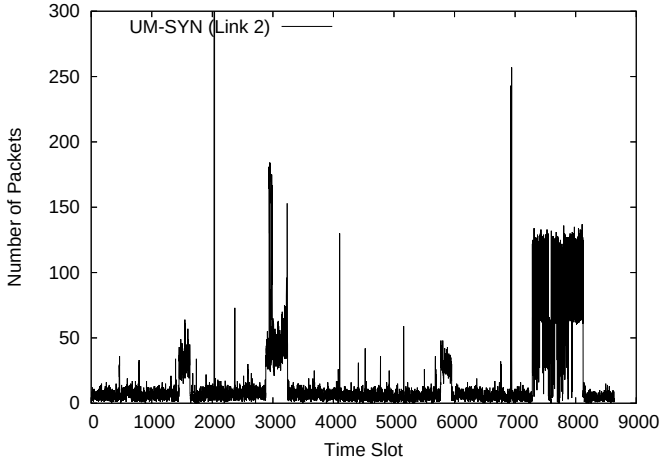
(a) Number of SYN packets (link 1).



(b) Number of unmatched SYN (UM-SYN) packets (link 1).



(c) Number of SYN packets (link 2).



(d) Number of unmatched SYN (UM-SYN) packets (link 2).

Fig. 14. Feature data

VIII. Conclusion

In this paper, we propose a novel framework to efficiently and accurately detect DDoS attacks and identify attack packets. to facilitate defense against DDoS attacks.

Future work: identify 1) matched IP addresses, 2) unmatched IP addresses/flows Let valid flows pass while blocking unmatched flows. New legitimate users will suffer during attack period. But our scheme can protect old users, which is better than the existing schemes, e.g., [13].

In this paper, we address how to efficiently detect various DDoS attacks for large scale Internet. The key novelties of our framework are 1) the temporal correlation of traffic flows on the two directions of a single link is exploited; 2) the spatial correlation of DDoS attack traffic at different locations is taken into account; and 3) powerful machine learning algorithms are used. With these techniques, our scheme can effectively detect and identify attack sources without modifying existing IP forwarding mechanisms and without a global upgrade to Internet backbone routers. More importantly, our framework can detect synchronized DDoS attacks even if the volume of attack traffic is extremely small at the location that is close to the attack source.

Acknowledgement

The authors would like to thank James Greco for conducting some of the simulations in this work.

References

[1] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communications Review*, vol. 34, no. 2, p. 39, 2004.

[2] L.-C. Chen, T. A. Longstaffv, and K. M. Carley, "Characterization of defense mechanisms against distributed denial of service attacks," 2003, submitted to Computers & Security.

[3] M. Crovella and E. Kolaczyk, "Graph wavelets for spatial traffic analysis," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 2003.

[4] S. S. Kim, A. L. N. Reddy, and M. Vannucci, "Detecting traffic anomalies using discrete wavelet transform," in *Proceedings of International Conference on Information Networking (ICOIN)*, vol. III, Busan, Korea, Feburary 2004, pp. 1375–1384.

[5] C.-M. Cheng, H. T. Kung, and K.-S. Tan, "Use of spectral analysis in defense against dos attacks," in *Proceedings of IEEE GLOBECOM 2002*, Taipei, Taiwan, November 2002.

[6] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, August 2003.
[7] L. L. Scharf, *Statistical signal processing: detection, estimation, and time series analysis*. Addison Wesley, 1991.
[8] S. Mukkamala and A. H. Sung, "Detecting denial of service attacks using support vector machines," in *Proceedings of IEEE International Conference on Fuzzy Systems*, May 2003.
[9] D.-R. Tsai, W.-P. Tai, and C.-F. Chang, "A hybrid intelligent intrusion detection system to recognize novel attacks," in *Proceedings of IEEE 37th Annual 2003 International Carnahan Conference on Security Technology*, October 2003.
[10] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for ip traceback," in *Proc. of ACM SIGCOMM'2000*, Aug. 2000.
[11] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attackswhich employ ip source address spoofing," *IETF, RFC 2267*,, Janurary 1998.
[12] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets," in *Proceedings of ACM SIGCOMM*, August 2001.
[13] S. Chen and Q. Song, "Perimeter-based defense against high bandwidth DDoS attacks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 6, pp. 526–537, June 2005.
[14] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *Proc. IEEE INFOCOM'2002*, New York City, NY, June 2002, pp. 1530–1539.
[15] ——, "Change-point monitoring for the detection of dos attacks," *IEEE Transactions on Dependable and Secure Computing*, no. 4, pp. 193–208, Oct. 2004.
[16] T. Peng, C. Leckie, and K. Ramamohanarao, "Protection from distributed denial of service attacks using history-based IP filtering," in *Proc. IEEE ICC 2003*, Anchorage, AK, May 2003, pp. 482–486.
[17] ——, "Detecting distributed denial of service attacks using source IP address monitoring," Department of Computer Science and Software Engineering, The University of Melbourne, Tech. Rep., 2002. [Online]. Available: http://www.cs.mu.oz.au/~tpeng
[18] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to ddos attack detection and response," in *Proceedings of DARPA Information Survivability Conference and Exposition*, vol. 1, April 2003, pp. 303–314.
[19] R. B. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "A novel approach to detection of "denial-of-service" attacks via adaptive sequential and batch-sequential change-point detection methods," in *Proc. IEEE Workshop on Information Assurance and Security*, West Point, NY, June 2001, pp. 220–226.
[20] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proc. of ACM SIGCOMM'2004*, Aug. 2004.
[21] "Auckland-IV trace data," 2001. [Online]. Available: http://wand.cs.waikato.ac.nz/wand/wits/auck/4/