Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 5036–5056
Computer Networks
www.elsevier.com/locate/comnet

Robust and efficient detection of DDoS attacks for large-scale internet

Kejie Lu [a], Dapeng Wu [b,*], Jieyan Fan [b], Sinisa Todorovic [c], Antonio Nucci [d]

[a] Department of Electrical and Computer Engineering at the University of Puerto Rico at Mayagüez, Mayagüez, PR 00681, United States
[b] Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, United States
[c] Computer Vision and Robotics Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States
[d] Narus, Inc., 500 Logue Avenue, Mountain View, CA 94043, United States

Received 15 November 2006; received in revised form 21 May 2007; accepted 31 August 2007
Available online 8 September 2007

Responsible Editor: R. Molva

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. Our framework is able to detect any source-address-spoofed DDoS attack, no matter whether it is a low-volume attack or a high-volume attack. 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.

Keywords: Distributed denial of service (DDoS) attacks; Detection; Machine learning; Spatial correlation

1. 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

* Corresponding author. Tel.: +1 352 392 4954; fax: +1 352 392 0044.
E-mail addresses: lukejie@uprm.edu (K. Lu), wu@ece.ufl.edu (D. Wu), fanjy@ufl.edu (J. Fan), sintod@vision.uiuc.edu (S. Todorovic).
URL: http://www.wu.ece.ufl.edu (D. Wu).

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.08.008

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 [3], spectral analysis [4,5], statistical methods [6–8]), and machine learning techniques [9] 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).

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. Our proposed approach is able to detect any type of DDoS attacks such as TCP SYN flood, TCP RST flood, UDP flood, ICMP flood, and DNS flood, as long as they use spoofed source IP addresses. 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 4.1.2. 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 and egress filtering, 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. Note that, although we use TCP SYN flood attacks during the experiments, our method can actually detect any source-address-spoofed DDoS attack.

The rest of the paper is organized as follows. In Section 2, we briefly overview the related work. Section 3 presents our framework for detecting DDoS attacks. In Section 4, we describe our feature extraction method. Section 5 presents our machine learning algorithm for DDoS attack detection. In Section 6, we discuss some important issues related to detection algorithms. Section 7 shows our simulation results and Section 8 concludes the paper.

2. Related work

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

2.1. Feature extraction

In [6], 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 an SYN packet and end with an 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 [14], 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 fool the system by generating 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. [7,15] 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. [7,15] that only consider new IP addresses, Ref. [16] 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 [14], the percentage of new IP addresses [7,15], and entropy of the IP address distribution [16] 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/Bloom-filter 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.

2.2. 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 [3], spectral analysis [4,5], statistical methods [6–8]), and machine learning techniques [9] can be used.

In [3], 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 [4,5], 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 was not designed to detect attacks having low-data-rate on a link, since attacks having low-data-rate may not cause large changes in the power 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 [6–8]. 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 [6–8] 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, which are inherent in real Internet.

In [9], 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.

A simple method, known as egress filtering, can be used to defend source-address-spoofed DDoS attacks by filtering unknown source IP addresses of all packets at edge routers. A more advanced method, called StackPi [17], inserts digital signature to IP packets to prevent source address spoofing. Unfortunately, both methods need to be deployed to the whole Internet to be effective. Any ISP not supporting these two methods will not be able to filter out potential attack packets. In addition, the high complexity incurred by these two methods may be unacceptable as they need to modify each IP packet at a router with these methods deployed. These limitations make these two methods impractical, if not impossible.

3. Framework for detecting DDoS attacks

In this section, we present our framework for detecting DDoS attacks.

Fig. 1 shows an ISP network architecture under our study, which consists of two types of IP routers,

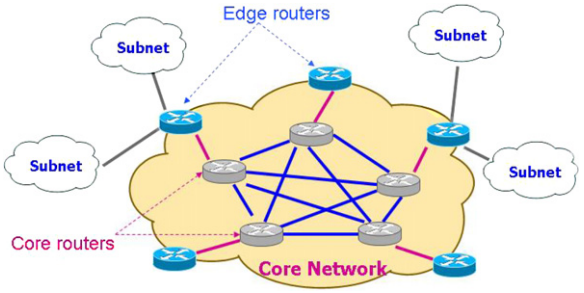


Fig. 1. An ISP network architecture.

namely, core routers and edge routers. Core routers interconnect with one another to form a high-speed core network. In contrast, edge routers are responsible for connecting subnets (i.e., customer networks or other ISP networks) with the core network. In this paper, a subnet can be either a customer network or an ISP network.

Given the ISP network architecture, we design a framework for detecting DDoS attacks. As shown in Fig. 2, our framework consists of three types of components: (1) traffic monitors, (2) local analyzers, and (3) a global analyzer, which are described as below.

3.1. Traffic monitor

A traffic monitor (represented by a filled oval in Fig. 2) is responsible for:

- scanning partial or all packets of a single unidirectional link;
- summarizing traffic characteristics;
- extracting simple features from the traffic characteristic;
- detect DDoS attacks based on simple online detection algorithm; and

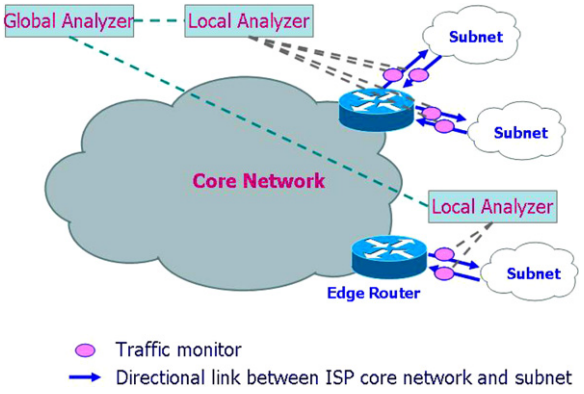


Fig. 2. Framework for detecting DDoS attacks.

- submit the summary of traffic information, simple feature data, and detection results to a local analyzer.

3.2. Local analyzer

A local analyzer is responsible for:

- extracting complicated features from traffic information obtained at a single edge router, and/or a few edge routers if they are co-located;
- detecting attacks with local information;
- submit the detection results, feature data, and traffic information (if necessary) to a global analyzer.

The local analyzer can utilize temporal correlation of traffic to generate feature data. Although a local analyzer may also have spatial correlated traffic information, for example, the local analyzer on top of Fig. 2, it is more appropriate to forward those information to a global analyzer because the global analyzer has the whole view of the network.

3.3. Global analyzer

A global analyzer is responsible for:

- extracting complicated features that requires global information, such as routing information, from traffic;
- analyzing feature data obtained from multiple local analyzers; and
- detecting anomalies with global information obtained from multiple edge routers.

The global analyzer utilizes both temporal correlation and spatial correlation of traffic. Here it is important to note that, some feature data must be obtained at the global analyzer if global information is required. For example, in Fig. 3, if the traffic from subnet A to server B passes through edge router X, and the traffic from server B to subnet A passes through edge router Y, then the 2D matching features between subnet A and server B shall be obtained at the global analyzer, which has the routing information of the ISP network.

Compared to existing network-based anti-DDoS systems, our framework does not require the upgrade of any router in the network.

Next, we describe feature extraction and detection algorithms in Sections 4 and 5, respectively.

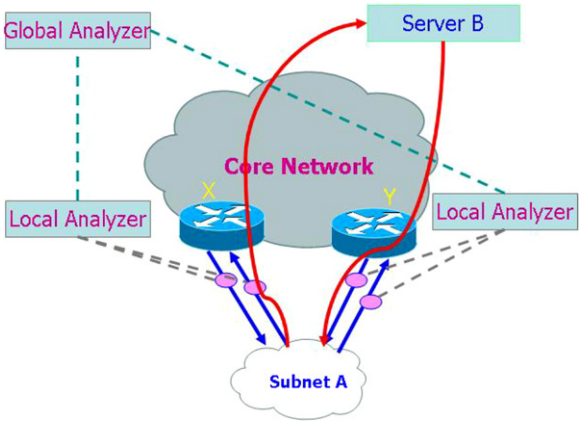


Fig. 3. An example of the asymmetric traffic.

4. Feature generation

To efficiently extract features from traffic, we design a three-level hierarchical structure shown in Fig. 4, where incoming packets are processed by level-one filters, then by level-two filters, and finally by (level-three) feature extraction modules. Level-one filters and level-two filters are placed in traffic monitors. A feature extraction module can be placed in either a traffic monitor or a local analyzer, depending on the type of the feature.

Level-one filters select a packet based on its source–destination pair, which is defined by the source IP address, the source network mask, the destination IP address, the destination network mask. For example, if we are interested in packets from 172.10.5.28 to 210.33.68.102, we can use 255.255.255.255 as both the source network mask and the destination network mask; if we are interested in packets from 172.10.x.x to 208.33.1.x, we can use 255.255.0.0 as the source network mask and 255.255.255.0 as the destination network mask. In this way, we can monitor an end-host or a subnet, giving much flexibility in configuring our detection framework. The output of a level-one filter is packets with the same source–destination pair, which are conveyed to level-two filters.

A level-two filter classifies the packets coming from level-one filters, based on the upper layer[1] data fields, e.g., TCP SYN or FIN. The packets of our interest will be forwarded to one or multiple feature extraction modules. For example, the number of TCP SYN packets can be used to generate both

[1] Here, the upper layer can be either Layer 4 or Layer 7.

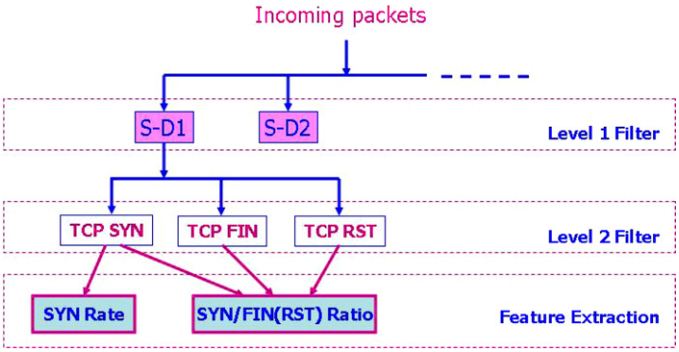


Fig. 4. Hierarchical structure for feature extraction.

the TCP SYN rate feature and the TCP SYN/FIN(RST) ratio feature; hence, TCP SYN packets are conveyed to both the TCP SYN rate module and the TCP SYN/FIN(RST) ratio module, as shown in Fig. 4. On the other hand, a feature module may need packets from multiple level-two filters. For example, the SYN/FIN(RST) ratio feature extraction requires packets from three filters, as shown in Fig. 4.

Compared to the packet classification schemes in [6,7], our hierarchical structure for feature extraction is more general and efficient.

Next, we describe (level-three) feature extraction modules and their implementations in Sections 4.1 and 4.2, respectively.

4.1. Feature extraction module

Similar to previous studies [6,7], we generate features in a discrete manner, i.e., our feature extraction module will generate a (feature) value or a vector at the end of each time slot. Here we define the duration of a time slot by T_s, which may vary for different features. Intuitively, a shorter T_s may reduce the detection delay, which is defined as the interval from the epoch when the attack starts to the epoch when the attack is detected; but a smaller T_s may increase the computational complexity, since the detection algorithm needs to analyze more feature data for the same time interval. On the other hand, if a feature is represented by a ratio, T_s must be sufficiently large to avoid division by zero. For example, if we want to use the SYN/FIN(RST) ratio as in [6] to detect TCP SYN flood, then T_s cannot be too small, because the number of FIN packets in a short period can be 0, which will result in a false alarm even if the number of SYN packets is not large.

Feature extraction can be done in a traffic monitor, a local analyzer, and a global analyzer, which we describe in Sections 4.1.1–4.1.3, respectively.

4.1.1. Feature extraction in a traffic monitor

As we mentioned earlier, some features are generated within a traffic monitor. These features are typically simple and can be extracted from the traffic on a single unidirectional link.

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

- Packet rate: defined by the number of packet arrivals in one time slot. This feature is simple but useful for detecting high-volume DoS and DDoS attacks. But it can hardly help detect low-volume attacks.
- Data rate: defined by the total number of bits of all packets that arrive in one time slot.
- SYN/FIN(RST) ratio[2]: defined by the ratio of the number of TCP SYN packets in one time slot to the number of FIN (and a portion of RST) packets in the same time slot.

4.1.2. Feature extraction in a local analyzer

Although a traffic monitor can generate simple features efficiently, these features may not be sufficient to detect attacks. In particular, the packet rate and data rate features may only be useful for detecting high-volume attacks; and SYN/FIN(RST) ratio has a large variation even for normal traffic and hence cannot help accurately distinguish normal network conditions from network anomalies. To improve detection accuracy, one can use a local analyzer to generate more sophisticated features, for

[2] How to obtain this ratio can be found in [14].

example, the SYN/SYN-ACK ratio proposed in [14] and the percentage of new IP addresses proposed in [7].

However, as discussed in Section 2.1, the existing features such as the SYN/SYN-ACK ratio [14] and the percentage of new IP addresses [7] either do not lead to good performance of detectors, or require high storage/time complexity. To address these deficiencies, we propose a new type of feature called *2D matching features*, which can make distinct features between normal and attack traffic, thereby improving accuracy of detecting attacks.

The motivation of proposing 2D matching features is the following. For most Internet applications, packets are generated from both hosts that are engaged in communication. Therefore, some information carried by packets on one direction shall match the corresponding information carried by packets on the other direction. For example, if station A communicates with station B through TCP, then we can observe packets 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 attacker generates source A on one direction, the response packet may not reach the link on the reverse direction (where a traffic monitor is placed) if the attacker spoofs its source IP address. Therefore, we can utilize this feature to detect DDoS attacks.

To facilitate the discussion, we need to specify 'keys' that contain the information that shall appear on both directions of a link. For instance, the key for TCP SYN packets on one direction can be defined by

$$\langle srcIP, dstIP, srcPort, dstPort, Seq\# \rangle$$

and the corresponding key for SYN-ACK packets on the opposite direction can be defined by

$$\langle dstIP, srcIP, dstPort, srcPort, Ack\# - 1 \rangle.$$

Intuitively, by examining whether a key matches the corresponding key on the opposite direction, we can detect the SYN flood or SYN-ACK flood attacks. Since the proposed feature is generated by matching the keys on two directions of a link, we call it *2D matching feature*.

4.1.3. Feature extraction in a global analyzer

From the discussion above, we can observe that the 2D matching features can be extracted by a local analyzer if

1. the traffic is symmetric on a bi-directional link, i.e., if packets from host A to host B go through a link on one direction, then packets from host B to host A must go through the link on the opposite direction; or
2. the traffic is asymmetric but the local analyzer have all the traffic information from different links, which are necessary for the 2D matching feature.

In practice, if the above conditions do not exist in some scenarios (for example, in the scenario illustrated in Fig. 3), then the local analyzer can be set to forward the traffic information to a global analyzer. Consequently, the global analyzer will be responsible for extract the 2D matching features.

Next, we discuss how to implement feature extraction modules to extract the 2D matching features.

4.2. Implementation of 2D matching feature extraction

In this section, we describe the implementation at a traffic monitor and at a local analyzer in Sections 4.2.1 and 4.2.2, respectively. To simplify the discussion, we assume that the traffic is symmetric hereafter.

4.2.1. Implementation at a traffic monitor

To perform matching of the keys on two directions, traffic monitors on two directions of a link need to record some traffic information. To deal with high-speed data rates and a large number of source–destination pairs in large-scale networks, efficient traffic recording must be in place. To achieve this, we use a hash table to store the needed information.

To update an (Pempty or non-empty) hash table, the following procedure is conducted.

- For each packet that passes through a level-two filter, a hash function will be called to locate the corresponding record in the hash table. If the corresponding record is empty, then a key extracted from the packet will be stored in the record, and the counter of the record 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 increased 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 of the incoming packet will not be saved.
- At the end of each time slot, a traffic monitor sends all non-empty records to the corresponding local analyzer.

The performance of this implementation depends on three major factors:

- *Memory requirement*: Denote N_{ht} the maximum number of records that could be stored in a hash table; denote L_k the length (in bytes) of each record. Then the total memory requirement is $N_{ht} \times L_k$ (in bytes). Here L_k depends on the length of a key. For example, for the key of SYN packets, we need 16 bytes of memory to store the key and 4 bytes of memory to store the status information and the counter. Therefore, $L_k = 20$ bytes.
- *Processing speed*: The main time complexity is due to the calculation of the hash function and key comparing, both of which can be implemented efficiently in software or hardware (if necessary).
- *Data rate required for communication*: At the end of each slot, the record information needs to be transmitted from the traffic monitor to the local analyzer. In the worst case, the data rate requirement for the communication is

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

4.2.2. Implementation at a local analyzer

The major features to be extracted at a local analyzer are the number of unmatched keys and the percentage of unmatched keys. Other features, including the packet rate, can also be extracted at a local analyzer.

To achieve the functionality of key matching, we build two key databases: one for the incoming traffic and the other for the outgoing traffic. Both the incoming key database and the outgoing key database use a sliding window mechanism to update the database. Specifically, the incoming key database always stores the latest $R_f + 1$ slots of key records; e.g., if slot n is the current slot, the incoming key database stores key records from slot $n - R_f$ to slot n. The outgoing key database always stores the latest $R_b + R_f + 1$ slots of key records. For each slot, both the incoming key database and the outgoing key database can store at most N_{ht} key records.

Now, we describe how to update the two key databases. As mentioned in Section 4.2.1, a traffic monitor sends all non-empty records to the corresponding local analyzer, at the end of each time slot. At the end of slot n, upon receiving an array of key records from the incoming traffic monitor, a local analyzer updates the incoming key database by replacing the array of key records in slot $n - R_f - 1$ by the array of key records in slot n. For the outgoing key database, we use the data structure of the celebrated Bloom filter [18] for efficient key matching. A Bloom filter consists of (1) a bit-map for key records, and (2) multiple hash functions for storing key records in the bit-map and key searching (i.e., membership query) [18]. So, at the end of slot n, upon receiving an array of key records from the outgoing traffic monitor, the local analyzer updates the outgoing key database by Bloom filter's storing operation for the bit-map of slot n and replacing the bit-map of slot $n - R_b - R_f - 1$ by the bit-map of slot n.

Given the two key databases, we can do key matching. To check whether a key is matched or not, we need to consider the temporal correlation of packets. For example, in a normal procedure, a TCP SYN packet must be followed (in time) by a TCP SYN-ACK packet with the same key on the opposite direction of the link. Therefore, if we want to determine whether a TCP SYN is matched, we need to search for the corresponding TCP SYN-ACK in the later slots on the opposite direction of the link. On the other hand, if we want to determine whether a TCP SYN-ACK is matched, we need to search for the corresponding TCP SYN in the previous slots on the opposite direction of the link. Under this philosophy, we design three mechanisms for key matching as below.

Assume that the current slot is slot $n + R_f$ and we are searching for a key in the outgoing key database to match one intended key of slot n in the incoming key database. Then our three key matching mechanisms are

- Forward matching: search all the slots m ($n \leqslant m \leqslant n + R_f$) in the outgoing key database for the matching key. E.g., if the key in the incoming key database is TCP SYN, then we need to use forward matching to search for the corresponding TCP SYN-ACK.

- Backward matching: search all the slots m $(n - R_b \leqslant m \leqslant n)$ in the outgoing key database for the matching key. E.g., if the key in the incoming key database is TCP SYN-ACK, then we need to use backward matching to search for the corresponding TCP SYN.
- Forward and backward matching: search all the slots m $(n - R_b \leqslant m \leqslant n + R_f)$ in the outgoing key database for the matching key. E.g., if we want to check whether a source IP address appears as a destination IP address in the opposite direction of the link, then we use forward & backward matching to search for the matching destination IP address.

The search in the above matching mechanisms is efficient due to the use of Bloom filter. When there is an unmatch, some counters will be updated; e.g., when a TCP SYN is unmatched, the counter for unmatched TCP SYN packets will be increased appropriately.

The memory requirement for a local analyzer is as below:

- Memory size for an outgoing key database is $(R_b + R_f + 1) * (2^{(L_{bf}-3)})$ bytes, where L_{bf} is the length (in bits) of the index, which is the output of Bloom filter's hash functions.
- Memory size for an incoming key database is $(R_f + 1) * N_{ht} * L_k$ bytes.

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

Next, we present our machine learning algorithm for detecting DDoS attacks.

5. Machine learning algorithm for detection

In this section, we present our machine learning algorithm for detecting DDoS attacks. We organize this section as follows. Section 5.1 outlines our detection approach. In Section 5.2, we formulate the detection problem. Section 5.3 describes our machine learning algorithm.

5.1. Outline of our detection approach

To facilitate the discussion, we use the notion of network state. Specifically, the network is in the state of 'attack' when there are DDoS attacks, and in the state of 'normal' when there is no DDoS attack. In our approach, we assume that different network states have different statistics for some features. For example, the average number of TCP SYN packets destined to the victim in the case of TCP SYN flooding attacks differs from that in the normal network state.

Denote X the network state, and Y the traffic observed by traffic monitors. Since different network state induces different stochastic process of traffic, we employ the widely used graphic model representation [19] to depict this cause-effect relationship in Fig. 5.

We represent features of multiple source–destination pairs by a high-dimensional matrix F. An entry f in the mth row and nth column of this matrix represents a feature vector that characterizes the traffic from source edge router m to destination edge router n. The details on feature extraction have been explained in Section 4. 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 Fig. 6.

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

Since DDOS attacks are distributed, where numerous compromised computers (zombies) run similar (or even the same) programs during the same attack period, 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 Fig. 5 with yet another set of random variables, Z, that encode this correla-

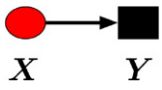


Fig. 5. Graphical representation of the generative process, in which the state of traffic generates the stochastic process of traffic.

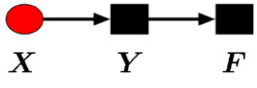


Fig. 6. The extended generative model including traffic-feature vectors.

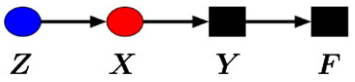


Fig. 7. The complete generative model including traffic data, traffic states, and correlation among traffic states.

tion, as depicted in Fig. 7. We anticipate that the introduction of Z may lead to an improved estimation of X.

5.2. Formulation of the detection problem

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 Fig. 8. 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 Fig. 8 is in fact the simplest possible generative model, since there are no lateral interdependencies among nodes. Consequently, the joint probability of the model reads

$$P(F,X) = \prod_i P(f_i|x_i)P(x_i). \tag{1}$$

From Eq. (1), we observe that the network-state estimation can be conducted for each traffic f_i independently, by using the *Maximum A Posteriori* (MAP) criterion given by

$$x_i^* = \arg\max_{x_i \in \{0,1\}} P(f_i|x_i)P(x_i). \tag{2}$$

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

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 Fig. 9. Here, in order to perform MAP estimation, we would need to marginalize a very complex prior distribution over network states X as

$$x_i^* = \arg\max_{x_i \in \{0,1\}} P(f_i|x_i) \sum_{\substack{j \\ j \neq i}} \sum_{x_j} P(x_1, x_2, \ldots, x_i, \ldots x_N). \tag{3}$$

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

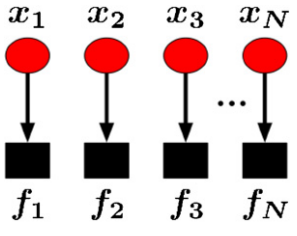


Fig. 8. The generative model that describes dependencies among traffic states and traffic-feature vectors.

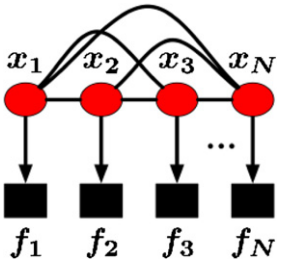


Fig. 9. Complex generative model, where the MAP estimation is intractable.

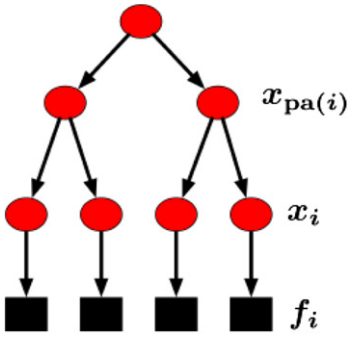


Fig. 10. The quad-tree model for one-dimensional data, where each parent has exactly two children; pa(i) denotes the parent of node i.

multiscale tree model, an example of which is depicted in Fig. 10. 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 Fig. 7, we graphically illustrate the dependencies between sets of variables F, X, and Z in the proposed multiscale model. From Fig. 7, the joint probability of the irregular tree is given by

$$P(F,X,Z,\Xi) = P(F|X)P(X|Z)P(Z|\Xi)P(\Xi). \quad (4)$$

From Eq. (4), 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 be NP-hard. We propose to solve the problem by using the Expectation–Maximization (EM) algorithm [19], 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 Fig. 11. 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 requirement 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 'attack' and 'normal' 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.

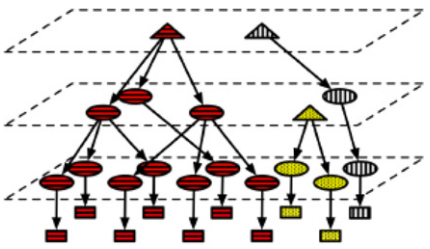


Fig. 11. The irregular tree consists of a forest of subtrees, each marked by distinct shading; round- and square-shaped nodes indicate hidden and observable variables, respectively; triangles indicate roots.

5.3. Machine learning algorithm for network-state estimation

5.3.1. Irregular tree

In this section we explain in greater detail the irregular-tree model for estimating the network states, given traffic 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 major tasks of this paper 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 [19], which is well known in the machine learning community.

An irregular tree is a directed acyclic graph with V nodes, organized in hierarchical levels, V^ℓ, $\ell = \{0, 1, \ldots, L\}$, 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 $|V^\ell| = |V^{\ell-1}|/4 = \cdots = |V^0|/4^\ell$. 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 $\ell+1$ 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 $i \in V^\ell$ and $j \in V^{\ell+1}$ 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 a network-state random variable, x_i, which can take values in a finite 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 $P(x_i|x_0, z_{i0} = 1) \triangleq P(x_i)$. The joint probability of all network-state variables $X = \{x_i\}$, $\forall i \in V$, is given by

$$P(X|Z) = \prod_{\ell=0}^{L} \prod_{i \in V^\ell} P(x_i|x_j, z_{ij} = 1). \tag{5}$$

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

$$P(F|X) = \prod_{i \in V^0} P(f_i|x_i), \tag{6}$$

$$P(f_i|x_i = c) = \sum_{g=1}^{G} \pi_c(g) N(f_i; \mu_c(g), \Sigma_c(g)), \tag{7}$$

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

Finally, we specify the connectivity distribution as

$$P(Z|\Xi) = \prod_{i,j \in V} P(z_{ij} = 1|\xi_{ij}) = \prod_{i,j \in V} N(\xi_{ij}; m, \Theta), \tag{8}$$

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 traffic of different source–destination pairs.

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 data F and Ξ.

5.3.2. Inference of the irregular tree

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

$$(\widehat{Z},\widehat{X}) = \arg\min_{Z,X} \mathbb{E}\{\mathscr{R}((Z,X),(Z',X'))|F,\Xi,\Omega\}, \tag{9}$$

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

$$\begin{aligned}\mathscr{R}((Z,X),(Z',X')) &= \mathscr{R}(X,X') + \mathscr{R}(Z,Z') \\ &= \sum_{\ell=0}^{L}\sum_{i\in V^\ell}[1-\delta(x_i - x_i\prime)] \\ &\quad + \sum_{\ell=0}^{L-1}\sum_{(i,j)\in V^\ell\times\{0,V^{\ell+1}\}}[1-\delta(z_{ij}-z'_{ij})],\end{aligned} \tag{10}$$

where $'$ stands for true values, and $\delta(\cdot)$ is the Kronecker delta function. From Eq. (10), the resulting Bayesian estimator of X is

$$\forall i \in V, \quad \hat{x}_i = \arg\max_{x_i\in C} P(x_i|Z,F,\Xi). \tag{11}$$

Next, given the constraints on connections in the irregular tree, discussed in Section 5.3.1, we derive that minimizing $\mathbb{E}\{\mathscr{R}(Z,Z')|F,\Xi,\Omega\}$ is equivalent to finding a set of optimal parents $\hat{j}$ such that

$$(\forall \ell)\ (\forall i \in V^\ell)\ (z_i \neq 0) \quad \hat{j} = \arg\max_{j\in\{0,V^{\ell+1}\}} P(z_{ij}=1|\xi_{ij}), \tag{12}$$

where $z_i \triangleq \sum_{k\in V^{\ell-1}} z_{ki}$, 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. (12) is an open problem in many research areas. We propose a stage-wise optimization, where, as we move upwards, starting from the leaf level $\ell = \{0,1,\ldots,L\}$, we include in the tree structure optimal parents at $V^{\ell+1}$ according to

$$(\forall i \in V^\ell)\ (\hat{z}_i \neq 0) \quad \hat{j} = \arg\max_{j\in\{0,V^{\ell+1}\}} P(z_{ij}=1|\xi_{ij}), \tag{13}$$

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.

Eqs. (11) and (13) suggest that it is possible to solve for $(\widehat{Z},\widehat{X})$ in a recursive procedure until some

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 $\ell = 0,1,\ldots,L$

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

(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** $\left|\frac{P(F|\hat{X}) - P(F|X(t-1))}{P(F|X(t-1))}\right| < \varepsilon$ for N consecutive iteration steps.

Fig. 12. Inference of the irregular tree.

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. (11); then, substituting $X(t)$ in Eq. (13) 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. (11) and (13) requires simultaneous optimization of $\widehat{X}$ and $\widehat{Z}$, note that we infer the irregular tree stage-wise, which may yield suboptimal solutions. From our experience, though, the algorithm recovers from stationary points for sufficiently large N. The overall inference algorithm is summarized in Fig. 12.

Steps 2 and 7 in the algorithm can be interpreted as inference of $\widehat{X}$ 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 Fig. 13. 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, $F_{d(i)}$

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** $\forall i \in V^1, V^2 ..., V^L$,

- $P(x_i|F_{d(i)}) \propto P(x_i) \prod_{c\in c(i)} \sum_{x_c} \dfrac{P(x_c|F_{d(c)})P(x_c|x_i)}{P(x_c)}$,
- $P(x_i,x_j|F_{d(i)}) = P(x_i|x_j)P(x_j)P(x_i|F_{d(i)})/P(x_i)$,

↓ **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** $\forall i \in V^{L-1}, V^{L-2} ..., V^0$,

- $P(x_i|F) = \sum_{x_j} \dfrac{P(x_i,x_j|F_{d(i)})}{\sum_{x_i} P(x_i,x_j|F_{d(i)})} P(x_j|F)$,
- $\hat{x}_i = \arg\max_{x_i} P(x_i|F)$

Fig. 13. Steps 2 and 7 in Fig. 12: Belief Propagation for the tree.

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.

6. Discussion on detection algorithms

The aforementioned machine learning algorithm is used in a global analyzer to exploit both spatial and temporal correlation of traffic. In a local analyzer, if detection of DDoS attacks must be conducted, one can employ (1) the threshold-based algorithm (to be defined in Sections 6.2) and 2 the change-point algorithm [6–8]. Both algorithms can use the feature data provided by the feature extraction modules in our framework. However, we note that an important issue has largely been ignored in the literature, that is, how to set the parameters of these algorithms.

We organize this section as below. Section 6.1 presents the performance metrics for detection algorithms. We describe how to set the parameters of the threshold-based algorithm and the change-point algorithm in Sections 6.2 and 6.3, respectively.

6.1. Performance metrics

A typical method to quantify the performance of detection algorithms is to use so-called *Receiver Operating Characteristics* (ROC) curve [20, p. 107].

To obtain an ROC curve, we need the following quantities

- N_f: the number of false alarms, i.e., the number of slots in which the detection algorithm declares 'attack' given that no attack actually happens in these slots;
- N_n: the number of slots in which no attack happens;
- N_d: the number of slots in which the detection algorithm declares 'attack' given that attacks actually happen in these slots;
- N_a: the number of slots in which attacks happen.

The false alarm probability and the detection probability of the detection algorithm can be estimated by N_f/N_n and N_d/N_a, respectively. By varying the detection threshold, we can obtain different pairs of false alarm probability and detection probability, which give the ROC curve [20, p. 107]. In this paper, we will use the ROC curve to compare the performance of different detection algorithms.

6.2. Threshold-based algorithm

The idea of the threshold-based algorithm is that if the feature value exceeds a preset threshold, declare 'attack'; otherwise, declare 'normal'. Note that the detection operation is conducted in each slot. By varying the threshold for the feature value, we can obtain different pairs of false alarm probability and detection probability, resulting in the ROC curve. Given the ROC curve and the desired false alarm probability, one can determine the value of the threshold for detection operation. This is the method to set the parameter of the threshold-based algorithm.

6.3. Change-point algorithm

In the literature, a simple change-point algorithm—non-parametric *Cumulative Summation* (CUSUM) algorithm—has been widely used [6–8,14]. However, existing works have not provided a comprehensive study on how to set the parameters of CUSUM. Furthermore, these studies only consider the change from normal state to abnormal state, which means that the number of false alarms can be very large after attacks end. To facilitate the discussion, we define the following parameters used in CUSUM:

- X_n denotes the observed traffic variable at the end of 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 $\overline{X}_n < \overline{X}_a$.
- 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 variable S_n by

$$\begin{cases} S_n = 0 & n = 0, \\ S_n = \max(0, S_{n-1} + Y_n) & n > 0. \end{cases} \tag{14}$$

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

From the discussion above, we note that two parameters, i.e., α and H, need to be determined. However, we cannot uniquely determine these two parameters. To overcome this problem, we shall introduce another parameter, i.e., the detection delay, denoted as D. According to the change-point theory, we have

$$\frac{D}{H} \to \frac{1}{(\overline{X}_a - \overline{X}_n) - |\overline{X}_n - \alpha|} = \frac{1}{\overline{X}_a - \alpha}. \quad (15)$$

From Eq. (15), we can obtain

$$H = D \times (\overline{X}_a - \alpha). \quad (16)$$

Hence, once D and α are given, we can determine H through Eq. (16).[3] Given α and H, we can use the CUSUM algorithm to detect network anomaly.

We notice that the existing CUSUM algorithms [6–8] only consider one change, i.e., from the normal state to the abnormal state. In practice, this approach may lead to a large number of false alarms after the end of attacks. To mitigate the false alarm problem of the existing algorithms, which we call single-CUSUM algorithms, we develop a dual-CUSUM algorithm. In this algorithm, one CUSUM will be used to detect the change from the normal to the abnormal state, while another CUSUM is responsible for detecting the change from the abnormal to the normal state. The method of setting parameters for dual-CUSUM is similar to the method described in this section.

7. Simulation results

In this section, we evaluate the performance of the proposed schemes through simulation. Due to the space limit, in this paper, we only consider TCP SYN attacks with spoofed source IP addresses.

7.1. Experiment setting

In our study, we develop a testbed to (1) extract various feature information, and (2) analyze feature data by our machine learning algorithm and CUSUM algorithms. Next, we describe the setting for networks, traffic traces, and feature extraction used in our experiments.

[3] In [6,7], $\alpha = (\overline{X}_a - \overline{X}_n)/2$; thus only the detection delay is needed.

7.1.1. Network

In our experiment, we assume that the ISP network consists of a core network, a victim network, and 16 edge routers that connect to 16 subnets, as illustrated in Fig. 14. At each edge router, two monitors are placed to measure the inbound and outbound traffic between a subnet and the victim network, respectively. To simplify the notation, we denote link i ($i = 1, 2, \ldots, 16$) as the virtual connection between subnet i and the victim network. Moreover, we define the inbound direction of link i as the direction from subnet i to the victim network; and the outbound direction of link i as the direction from the victim network to subnet i.

7.1.2. Traffic

A link may carry traffic with no attack (called background traffic) or traffic with TCP SYN attacks.

For the background traffic, we use the trace data provided by Auckland University [21]. This data set contains packet header information of the real traffic between the Internet and Auckland University. The connection is OC-3 (155 Mb/s) for both directions. Since we do not have real data traces obtained from 16 different links, we use the real traffic trace measured on one link (between the Internet and Auckland University) to create traffic traces for 16 different links. Specifically, we use traffic traces of different days to represent traffic traces of different links. So we use a traffic trace of 16 days to represent traffic traces of 16 different links. That is, the traffic from the Internet to Auckland University in day i ($i = 1, 2, \ldots, 16$) corresponds to the inbound traffic of link i; the traffic from Auckland University to the Internet in day i corresponds to the outbound traffic of link i.

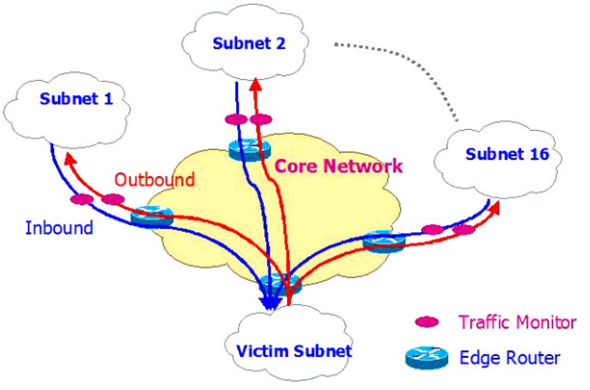


Fig. 14. Experiment network.

To simulate TCP SYN attacks, we first generate a random number N_{link}; then we randomly select N_{link} links from the 16 links; for each of the selected N_{link} links, we randomly add TCP SYN packets with spoofed source IP addresses into the background traffic of that link. The average packet rate of TCP SYN attack traffic on each of the selected N_{link} links, is 1% of the total packet rate on the link. The attacks on each of the selected N_{link} links are launched in almost the same period so that we have synchronized DDoS attacks across the selected N_{link} links. Since the attack traffic on each link is low (just 1%), we effectively simulate low-volume attack traffic. Note that, in this paper, we use TCP SYN flood attack as an example to demonstrate the performance of our DDoS detection method. Actually, it is able to detect any type of DDoS attacks, as long as it uses spoofed source IP addresses.

7.1.3. Feature

To detect distributed TCP SYN attacks, we use the 2D features described in Section 4.1.2, i.e., the number of unmatched SYN packets in one time slot. Note that the number of SYN packets toward the victim subnet can also be reported by our feature extraction modules. So, both the number of SYN packets and the number of unmatched SYN packets can be used in our analysis. The parameter setting of our feature extraction module is listed in Table 1.

Given the parameters in Table 1, we can derive that the memory requirement for each traffic monitor is

$$N_{ht} \times L_k = 320 \text{ Kbyte},$$

the memory requirement for each local analyzer is

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

and the minimum detection delay is upper bounded by

$$(R_f + 1) * T_s = 100 \text{ s}.$$

Table 1
Parameter setting for feature extraction

Parameter	Setting
T_s	10 s
R_f	9
R_b	0
N_{ht}	$2^{14} = 16K$
L_{bf}	16
L_k	20 bytes

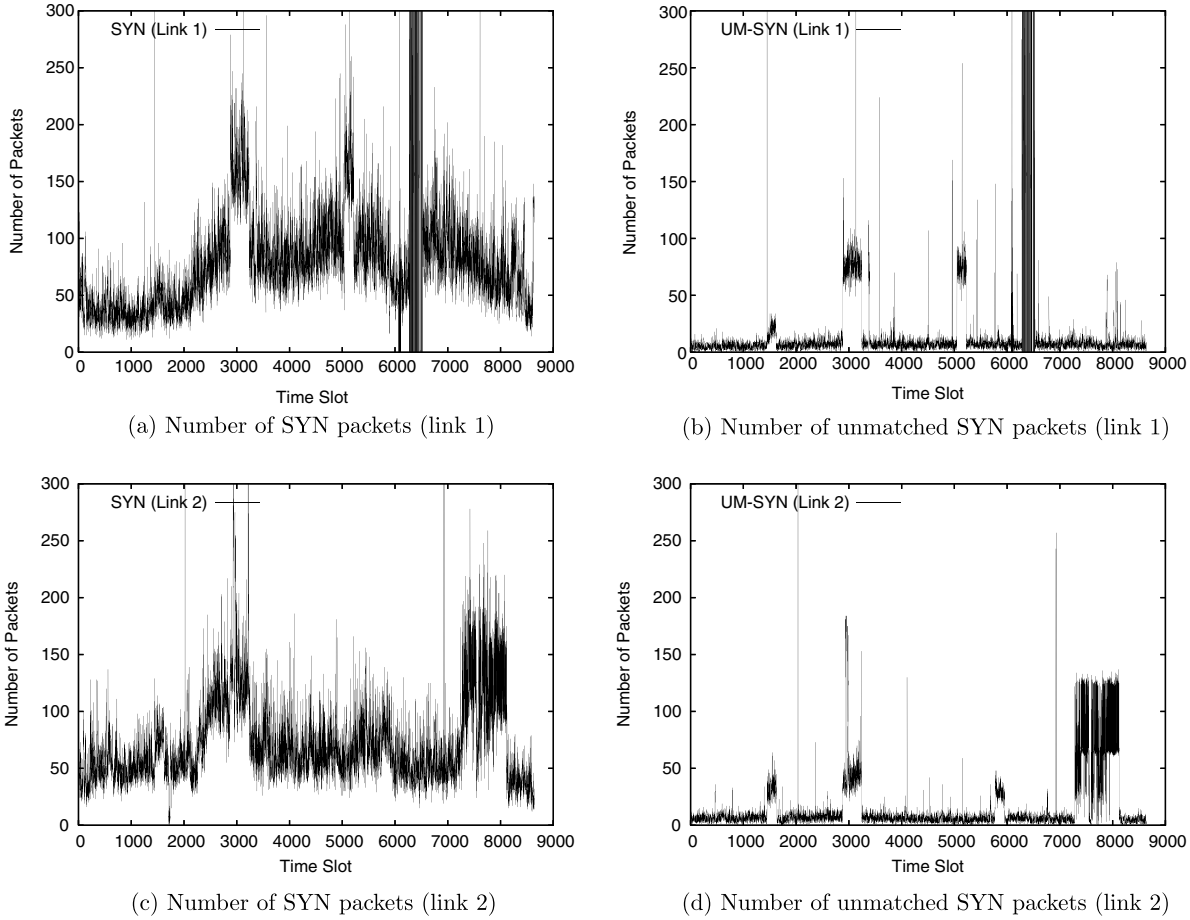
Fig. 15 shows the features of two links, specifically, the number of SYN packet arrivals and the number of unmatched SYN packet arrivals during a slot, where the duration of a slot is 10 seconds. For these two links, we launch (synchronized) attacks on both link 1 and link 2, during two periods, i.e., Slot 1400–1600 and Slot 2800–3200. In addition, asynchronous attacks are launched during Slot 5000–5200 on link 1 and during Slot 5700–5900 on link 2.

From Fig. 15, it can be observed that the features are rather noisy, especially for the feature of the number of SYN packets. From Fig. 15a and c, we can hardly distinguish the slots under the low-volume synchronized attacks from the slots without attacks (by visual inspection). In comparison, it is much easier to identify the slots under the synchronized attacks (by visual inspection) when the number of unmatched SYN packets is used as the feature (see Slot 1400–1600 and Slot 2800–3200 in Fig. 15b and d).

7.2. Performance comparison

Table 2 compares the performance of different schemes, where the benchmark is the scheme in [6], i.e., the CUSUM scheme with SYN/FIN ratio as the feature; for the benchmark scheme, we use the same parameter setting as that in [6]; we compare the benchmark with CUSUM and our machine learning algorithm under different features. To make fair comparison, we make the false alarm probability of each scheme almost the same and compare the detection probability. From Table 2, it can be seen that, the benchmark scheme ('SYN/FIN ratio' + CUSUM) performs very poorly in detecting low-volume DDoS attacks. In contrast, a CUSUM algorithm with the number of SYN packets or the number of unmatched SYN packets as the feature can achieve much higher detection probability. More importantly, our machine learning algorithm can significantly outperform CUSUM, given the same feature data, no matter whether the feature is the number of SYN packets or the number of unmatched SYN packets.

Fig. 16 compares the receiver operating characteristics (ROC) curve of the threshold-based scheme described in Section 6.2 and our machine learning algorithm under two different features, i.e., the number of SYN packets (denoted by 'SYN') and the number of unmatched SYN packets (denoted



(a) Number of SYN packets (link 1)

(b) Number of unmatched SYN packets (link 1)

(c) Number of SYN packets (link 2)

(d) Number of unmatched SYN packets (link 2)

Fig. 15. Feature data.

Table 2
Performance of different schemes

Feature	Detection algorithm	Detection probability	False alarm probability
SYN/FIN ratio [6]	CUSUM	0.174	0.129
SYN	CUSUM	0.52	0.129
SYN	Machine learning	0.6563	0.1228
Unmatched SYN	CUSUM	0.69	0.130
Unmatched SYN	Machine learning	0.9726	0.1146

by 'UM-SYN'). We observe that, for the same detection algorithm, using the number of unmatched SYN packets can significantly improve the ROC performance, compared to using the number of SYN packets. In other words, given the same false alarm probability, the detection probability is much higher when using the number of unmatched SYN as feature.

Another important observation from Fig. 16 is that given the same feature data, our machine learning algorithm can (significantly) improve the ROC, compared to the threshold-based scheme; e.g., for the same false alarm probability of 0.05, our machine learning algorithm achieves a detection probability of 0.93, while the threshold-based scheme only achieves a detection probability of 0.72. This is due to the fact that our machine learning algorithm exploits the spatial correlation among traffic on multiple links, while the threshold-based scheme only uses the traffic on one link.

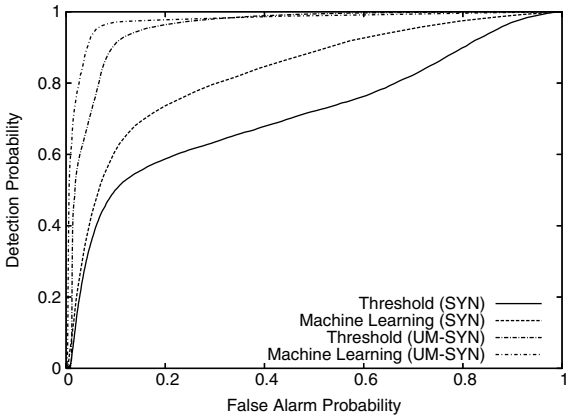


Fig. 16. Performance of threshold-based and machine learning algorithms with different feature data.

In Fig. 17, we compare the ROC performance of four detection algorithms (the threshold-based, the single-CUSUM, the dual-CUSUM described in Section 6.3, and our machine learning algorithm) under the same feature, i.e., the number of unmatched SYN packets. For the single-CUSUM and the dual-CUSUM algorithm, the detection delay D is chosen from 1 to 10 and the parameter α_i of link i is determined by

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

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

As shown in Fig. 17, the ROC performance of our machine learning algorithm is the best among all the algorithms. We also see that the dual-CUSUM out-performs the simple threshold-based algorithm and the single-CUSUM algorithm has the worst ROC performance.

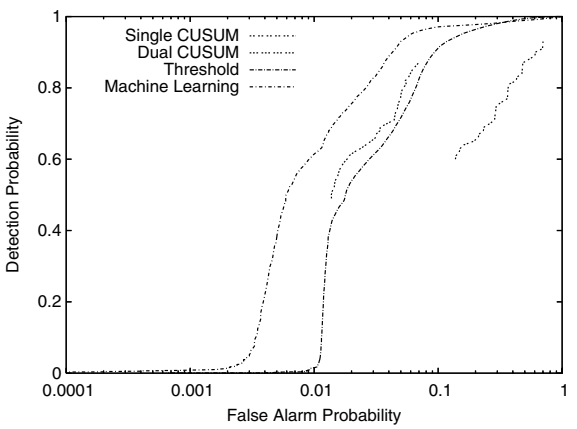


Fig. 17. Performance of four detection algorithms.

7.3. Discussion

We would like to point out that, besides detecting low-volume attacks, our machine learning algorithm is also able to detect high volume attacks, the results of which are not shown here due to the space limit. The machine learning algorithm is shown to be robust under realistic time-varying traffic patterns such as the Auckland data traffic [21]. We tested our machine learning algorithms for a large IP address space, i.e., the IP address space can be the whole IP address space for the Internet.

8. Conclusion

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 ISP network. The originalities of our framework are temporal-correlation based feature extraction and spatial-correlation based detection. Our feature extraction is novel in two aspects, i.e., (1) unique 2D matching features that make distinct features between normal and attack traffic, thereby improving accuracy of detecting attacks, and (2) efficient implementation based on hash-table/Bloom-filter. Our detection scheme has the following nice properties:

- In addition to detecting attacks having high-data-rate on a link, our scheme is also capable of accurately detecting attacks having low-data-rate on a link. This is due to exploitation of spatial correlation of DDoS attack traffic.
- Our scheme is robust against time-varying traffic patterns, owing to powerful machine learning techniques.
- Our scheme can be deployed in large-scale high-speed networks, thanks to use of hash-table and Bloom-filter.

With the proposed techniques, our scheme can effectively 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.

Our future work will focus on designing defense mechanisms against DDoS attacks. The detection framework proposed in this paper can facilitate defense against DDoS attacks in the following way. Since our framework can identify (1) matched IP addresses/flows, and (2) unmatched IP addresses/flows. If a DDoS attack is detected, the defense mechanism (a simple packet filter based on IP addresses) only allows matched flows to pass while blocking unmatched flows. During a DDoS attack period, new legitimate users will not be allowed, i.e., packets from new legitimate users will be dropped, which is a limitation of this defense mechanism. But such a defense mechanism can protect frequent (old) users, which are, from the business perspective, typically more important than new users.

Acknowledgement

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

References

[1] J. Mirkovic, P. Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, ACM SIGCOMM Computer Communications Review 34 (2) (2004) 39.
[2] L.-C. Chen, T.A. Longstaffv, K.M. Carley, Characterization of defense mechanisms against distributed denial of service attacks, Computers & Security 23 (8) (2004) 665–678.
[3] S.S. Kim, A.L.N. Reddy, M. Vannucci, Detecting traffic anomalies using discrete wavelet transform, in: Proceedings of International Conference on Information Networking (ICOIN), Busan, Korea, 2004, vol. III, pp. 1375–1384.
[4] C.-M. Cheng, H.T. Kung, K.-S. Tan, Use of spectral analysis in defense against dos attacks, in: Proceedings of IEEE GLOBECOM 2002, Taipei, Taiwan, 2002.
[5] A. Hussain, J. Heidemann, C. Papadopoulos, A framework for classifying denial of service attacks, in: Proceedings of ACM SIGCOMM, Karlsruhe, Germany, 2003.
[6] H. Wang, D. Zhang, K.G. Shin, Detecting SYN flooding attacks, in: Proceedings of IEEE INFOCOM'2002, New York City, NY, 2002, pp. 1530–1539.
[7] T. Peng, C. Leckie, K. Ramamohanarao, Detecting distributed denial of service attacks using source IP address monitoring, Tech. Rep., Department of Computer Science and Software Engineering, The University of Melbourne, 2002. URL <http://www.cs.mu.oz.au/tpeng>.
[8] R.B. Blazek, H. Kim, B. Rozovskii, A. Tartakovsky, A novel approach to detection of "denial-of-service attacks via adaptive sequential and batch- sequential change-point detection methods, in: Proceedings of IEEE Workshop on Information Assurance and Security, West Point, NY, 2001, pp. 220–226.
[9] S. Mukkamala, A.H. Sung, Detecting denial of service attacks using support vector machines, in: Proceedings of IEEE International Conference on Fuzzy Systems, 2003.
[10] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Practical network support for ip traceback, in: Proceedings of ACM SIGCOMM'2000, 2000.
[11] P. Ferguson, D. Senie, Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing, IETF, RFC 2267.
[12] K. Park, H. Lee, On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets, in: Proceedings of ACM SIGCOMM, 2001.
[13] S. Chen, Q. Song, Perimeter-based defense against high bandwidth DDoS attacks, IEEE Trans. Parallel Distrib. Syst. 16 (6) (2005) 526–537.
[14] H. Wang, D. Zhang, K.G. Shin, Change-point monitoring for the detection of dos attacks, IEEE Transactions on Dependable and Secure Computing (4) (2004) 193–208.
[15] T. Peng, C. Leckie, K. Ramamohanarao, Protection from distributed denial of service attacks using history-based IP filtering, in: Proceedings of IEEE ICC 2003, Anchorage, AK, 2003, pp. 482–486.
[16] L. Feinstein, D. Schnackenberg, R. Balupari, D. Kindred, Statistical approaches to DDoS attack detection and response, in: Proceedings of DARPA Information Survivability Conference and Exposition, vol. 1, 2003, pp. 303–314.
[17] A. Yaar, A. Perrig, D. Song, Stackpi: New packet marking and filtering mechanisms for ddos and ip spoofing defense, Tech. Rep., Carnegie Mellon University, 2003.
[18] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (7) (1970) 422–426.
[19] B.J. Frey, Graphical Models for Machine Learning and Digital Communication, MIT Press, Cambridge, MA, 1998.
[20] L.L. Scharf, Statistical Signal Processing: Detection, Estimation, and Time Series Analysis, Addison Wesley, 1991.
[21] Auckland-IV trace data, 2001. URL <http://wand.cs.waikato.ac.nz/wand/wits/auck/4/>.

Kejie Lu received the B.S. and M.S. degrees in telecommunications engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1994 and 1997, respectively. He received the Ph.D. degree in electrical engineering from the University of Texas at Dallas in 2003.

In 2004 and 2005, he was a Postdoctoral Research Associate in the Department of Electrical and Computer Engineering, University of Florida. Currently, he is an Assistant Professor in the Department of Electrical and Computer Engineering, University of Puerto Rico at Mayagüez. His research interests include architecture and protocols design for computer

and communication networks, performance analysis, network security, and wireless communications.

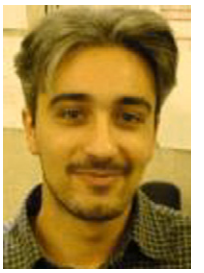
Dapeng Oliver Wu received B.E. in Electrical Engineering from Huazhong University of Science and Technology, Wuhan, China, in 1990, M.E. in Electrical Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1997, and Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, PA, in 2003.

Since August 2003, he has been with Electrical and Computer Engineering Department at University of Florida, Gainesville, FL, as an Assistant Professor. His research interests are in the areas of networking, communications, multimedia, signal processing, and information and network security. He received the IEEE Circuits and Systems for Video Technology (CSVT) Transactions Best Paper Award for Year 2001, and the Best Paper Award in International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QShine) 2006.

Currently, he serves as the Editor-in-Chief of Journal of Advances in Multimedia, and an Associate Editor for IEEE Transactions on Wireless Communications, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Transactions on Vehicular Technology, and International Journal of Ad Hoc and Ubiquitous Computing. He is also a guest-editor for IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Cross-layer Optimized Wireless Multimedia Communications. He served as Program Chair for IEEE/ACM First International Workshop on Broadband Wireless Services and Applications (BroadWISE 2004); and as a technical program committee member of over 30 conferences. He is Vice Chair of Mobile and wireless multimedia Interest Group (MobIG), Technical Committee on Multimedia Communications, IEEE Communications Society. He is a member of the Best Paper Award Committee, Technical Committee on Multimedia Communications, IEEE Communications Society.

Jieyan Fan received B.E. and M.E. in Electrical Engineering from Shanghai Jiaotong University, Shanghai, China, in 2001 and 2004, respectively. He is pursuing Ph.D. degree in Electrical and Computer Engineering from University of Florida, Gainesville, FL. His research interests are in the areas of network security, network traffic analysis, and pattern classification.

Sinisa Todorovic received his B.S. degree in electrical engineering at the University of Belgrade, Serbia, in 1994. From 1994 until 2001, he worked as a software engineer in the communications industry. He earned his M.S. and Ph.D. degrees in electrical and computer engineering at the University of Florida, in 2002, and 2005, respectively. Since 2005, he holds the position of Postdoctoral Research Associate in the Beckman Institute, University of Illinois at Urbana-Champaign. His main research interests concern computer vision and machine learning, with current focus on statistical image modeling for object/scene recognition. He has published approximately 20 journal and refereed conference papers.



Antonio Nucci received the Dr. Ing Degree in Electronics Engineering in 1998 and the Ph.D. degree in telecommunications engineering in 2002, both from the Politecnico di Torino, Turin, Italy. In September 2001 he joined the Sprint Advanced Technology Laboratories, Burlingame, CA, where he was a principal member of technical staff in the IP research group until February 2005. He currently holds a CTO position at Narus Inc. His research interests include traffic measurement, characterization and analysis, performance evaluation, traffic engineering, security and network design. He is a IEEE senior member.