Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 2677–2700
Computer Networks
www.elsevier.com/locate/comnet

On deterministic packet marking

Andrey Belenky, Nirwan Ansari *

New Jersey Institute of Technology, Department of Electrical and Computer Engineering, 323 King Blvd., Newark, NJ 07102, United States

Received 23 June 2006; received in revised form 17 October 2006; accepted 25 November 2006
Available online 11 January 2007

Responsible Editor: Jelena Misic

Abstract

In this article, we present a novel approach to IP Traceback – deterministic packet marking (DPM).[1] DPM is based on marking all packets at ingress interfaces. DPM is scalable, simple to implement, and introduces no bandwidth and practically no processing overhead on the network equipment. It is capable of tracing thousands of simultaneous attackers during a DDoS attack. Given sufficient deployment on the Internet, DPM is capable of tracing back to the slaves responsible for DDoS attacks that involve reflectors. In DPM, most of the processing required for traceback is done at the victim. The traceback process can be performed post-mortem allowing for tracing the attacks that may not have been noticed initially, or the attacks which would deny service to the victim so that traceback is impossible in real time. The involvement of the Internet Service Providers (ISPs) is very limited, and changes to the infrastructure and operation required to deploy DPM are minimal. DPM is capable of performing the traceback without revealing topology of the providers' network, which is a desirable quality of a traceback method.

Keywords: Security; DDoS attacks; IP traceback

1. Introduction

In recent years, much interest and consideration have been paid to securing the Internet infrastructure that has become a medium for a broad range of transactions. A number of approaches to security have been proposed, each attempting to mitigate a specific set of concerns. The specific threat, which is the main focus of this article, is *anonymous attacks*. In anonymous attacks, the identity of attacker(s) is not immediately available to the victim since the Source Address (SA) field in the attack packets is spoofed. (Distributed) Denial of Service ((D)DoS) attacks are anonymous attacks that presently attract a lot of attention because there is no obvious way to prevent them or to trace them.

Currently, there are several ways of dealing with anonymous attacks. They include source address filtering, SYN Flood Protection, and implementing a BlackHole Router server. Source address filtering, introduced in [1], prevents packets with values of the SA field outside the preset appropriate range from entering the Internet. If deployed on every

* Corresponding author. Tel./fax: +1 973 596 3670.
E-mail address: Nirwan.Ansari@njit.edu (N. Ansari).
[1] Three U.S. Patent applications have been filed based on the content of this work.

1389-1286/$ - see front matter
doi:10.1016/j.comnet.2006.11.020

ingress interface, this would drastically reduce the number of anonymous packets in the Internet. Unfortunately, source address filtering incurs high overhead and administrative burden [2] and is ineffective, unless carried out on almost 100% of the ingress routers [3]. SYN Flood Protection monitors half-open TCP connections and does not allow more than a certain number of such connections to exist simultaneously. SYN Flood protection prevents only SYN Flood type (D)DoS attacks and is useless against other types of anonymous attacks. Finally, ISPs can determine the interface, where the DoS attack packets entered its network, by ''BlackHoling'' a router on its network, if the affected customer reports the attack [4]. This method works only for the backscatter DOS attacks, as discussed in [5]. It involves human interaction, must be performed while the attack is still in progress, and is limited to the boundaries of a given ISP. In summary, currently available methods for preventing anonymous attacks are insufficient for the Internet-wide prevention or traceback of attacks: SYN Flood prevention deals with a very limited set of attacks; access filtering is practically impossible to implement, since it requires close to 100% deployment; and BlackHole router traceback requires manual intervention and is limited to a single administrative domain.

While it may be simply impossible to prevent attackers from attempting to carry out an attack, it might be possible to lessen, or even completely mitigate, the effects of the attack by not allowing the packets to reach the victim(s). This is the proactive approach discussed in details in, for example, [6]. However, prevention of all attacks on the Internet is far from reality. When prevention fails, a mechanism of identifying the source(s) of the attack is needed to at least ensure accountability for these attacks. This is the motivation for designing IP Traceback techniques.

The rest of the paper is structured as follows. Section 2 presents a brief description of related works; Section 3 introduces the basic DPM approach; a modification for handling multiple simultaneous attackers is described in Section 4; Section 5 describes the modification to DPM to accommodate fragmented traffic; Section 6 describes various types of computer attacks and presents a unified traceback procedure for tracing all of them; Section 7 lists the items of future work and conclusions; Appendix A provides the performance analysis of DPM.

2. Related works

After several high-profile DDoS attacks on major US Web sites in early 2000, numerous IP traceback approaches have been suggested to identify the attacker(s) [7]. *IP Traceback* is defined in [6] as identifying a source of any packet on the Internet. Previously proposed methods can be categorized into four broad groups. Solutions in the first group, where most of the research has concentrated so far, rely on the routers in the network to send their identities to the destinations of certain packets passing through them, either by encoding this information directly in rarely used fields of the IP header, or by generating a new packet to the same destination [2,8–18].

In particular, the pioneering article on PPM [2] proposed marking packets with identities of routers along the path between the source and the destination. Subsequent works [11–13] improved on the performance on the basic PPM by, among others, enhanced encoding of marks, authenticating marks, and making the map of possible attacks available to the victim. Ref. [14] proposes a modification to the PPM that ensures that the probability of receiving the mark is equal to the original marking probability. Ref. [15] proposes a method of encoding a path identification by marking packets with path fingerprints. In [16], further improvements to PPM such as 1-bit distance encoding and construction of the map using marks, as opposed to *a priori* knowledge, are proposed. In [17], it was proposed to use relatively large, randomized messages to encode router information. In [18], it was proposed to encode the AS information in such a way that ensures the downstream router to check its correctness, thus reducing the impact of router subversion. Also belonging to the first groups are traceback methods that require routers to send an ICMP message containing the routers' identification information to the destination of every nth packet [8–10]. These techniques operate under the underlying assumption that the victim of a flood based attack will receive these ICMP messages and reconstruct the attack path.

Methods of the second group log some fields of every packet, or the digest of every packet on all the routers that these packets traverse [19–21]. During the traceback, all of the routers are polled and the path of a given packet is reconstructed by correlating the routers, which have stored the information about this packet. Further improvements where only a fraction of the packets are logged were

introduced in [22]. The solutions of this group are not easily scalable, have relatively high ISP involvement, and have no post-mortem traceback capabilities [7]. Solutions of the third group involve centralized management of the traceback process and changing the routing in the network with the tunneling to be able to identify the packets' origin [23–25]. The shortcomings of these schemes are high ISP involvement and high bandwidth and processing overhead associated with tunneling. The only solution in the final fourth group is controlled flooding described in [26]. Controlled flooding only works for DoS attacks. The attack path is determined while the attack is still in progress by systematically loading different links on the network and observing the effect on the victim. If increasing the load of a particular link results in the drop in the rate of the attack traffic, then this link is on the attack path. Controlled flooding is limited to tracing DoS attacks only, and it is manual. Also, it utilizes a questionable approach of essentially inducing DoS attacks for the purposes of traceback.

3. Basic DPM

The basic DPM is a packet marking algorithm, which was first introduced in [27]. This section provides the general principle behind DPM and discusses its most basic implementation.

3.1. Assumptions

The assumptions in this section were largely borrowed from [2]. The two key assumptions driving this effort are:

- an attacker may generate any packet; and
- routers are both CPU and memory limited.

3.2. DPM principle

As mentioned above, DPM is a packet marking algorithm. The 16-bit packet Identification (ID) field and 1-bit Reserved Flag (RF) in the IP header is used to mark packets. *Each* packet is marked when it enters the network. This mark remains unchanged for as long as the packet traverses the network. This automatically obviates the issue of mark spoofing which other marking schemes have to account for. The packet is marked by the interface closest to the source of the packet on the edge ingress router, as

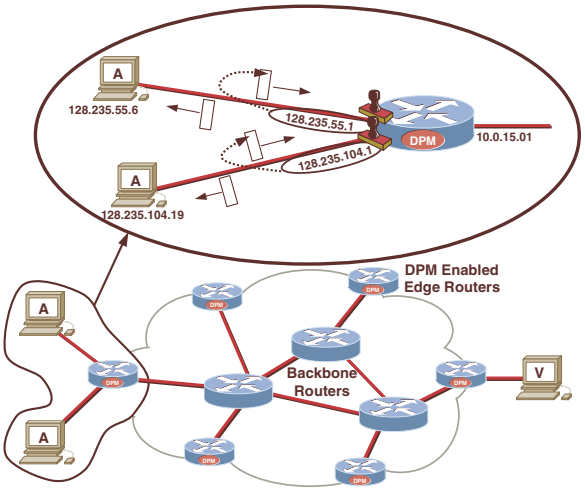

Fig. 1. Basic deterministic packet marking (DPM).

shown in Fig. 1. The routers with the engraved "DPM" signify the routers with DPM-enabled interfaces, and the rubber-stamps signify the interfaces on these routers that actually perform the marking. The mark contains the partial address information of this interface, and will be addressed later in Section 3.3. The interface makes a distinction between incoming and outgoing packets. Incoming packets are marked; outgoing packets are not marked. This ensures that the egress router will not overwrite the mark in a packet placed by an ingress router.

For illustrative purposes, assume that the Internet is a network with a single administration. In this case, only interfaces closest to the customers on the edge routers participate in packet marking. Once again, every incoming packet is marked by the ingress interface. Should an attacker attempt to spoof the mark in order to deceive the victim, this spoofed mark would be overwritten with a proper mark by the very first router the packet traverses.

Ref. [28] describes the methodology of deploying DPM on the Internet. Briefly, a continuous perimeter of DPM-enabled interfaces should be maintained. The deployment should start with the largest, tier-1, ISPs and expand in concentric circles until DPM is enabled on the ingress interfaces. When a tier-2 ISP enables DPM on its ingress interfaces, it must inform its upstream tier-1 ISP(s), which should disable DPM on the interfaces that connect to the tier-2 ISP. In general, when DPM is enabled on interfaces of an n-tier ISP, DPM must be disabled on the interface of its upstream $n-1$-tier ISP. These guidelines require collaboration and sharing information among ISPs. However, the

sharing of information occurs between ISPs, which already collaborate in sharing, among others, BGP routing information. If these DPM deployment guidelines are followed, then at any point in DPM deployment the DPM-enabled interfaces will form a continuous perimeter. However, if these guidelines are not followed, or if DPM is enabled on one or more interfaces inside the perimeter, the ability of the victim to traceback will be undermined.

3.3. Procedure

A 32-bit IP address needs to be passed to the victim. A total of 17 bits are available to pass this information: 16-bit ID field and 1-bit RF. Clearly, a single packet would not be enough to carry the whole IP address in the available 17 bits. Therefore, it takes at least two packets to transfer the entire IP address. The IP address is split into two segments, 16 bits each: segment 0 – bits 0 through 15, and segment 1 – bits 16 through 31. The marks are prepared in advance to decrease the per-packet processing. Each mark has two fields: Segment Number and Address bits. With equal probability, one of the two marks is inserted in the 17-bit field comprised of the ID field and RF of each incoming packet.

As described in [27], the victim maintains a table matching the source addresses to the ingress addresses. When a marked packet arrives at the victim, the victim checks if the table entry for the source address of this packet already exists, and if it does not exist, one is created. Then, it writes address bits of the segment into the corresponding bits of the ingress IP address value. After both segments corresponding to the same ingress address have arrived at the destination, the ingress address for the given source address becomes available to the victim. The details of the procedure are shown in Fig. 2.

Marking procedure at router R, edge interface A:
for $y = 0$ **to** 1
  $Marks[y]$.Seg_Num := y
  $Marks[y]$.A_bits := $A[y]$
for each incoming packet w
  let x be a random integer from [0,1]
  write $Marks[x]$ into w.Mark

Ingress address reconstruction procedure at V:
for each attack packet w
  $IngressTbl[w.\text{Mark.Seg\_Num}]$:= w.Mark.Seg_Num

Fig. 2. Pseudocode for the basic DPM.

For security reasons, passing the ingress IP address may be undesirable for ISPs for security reasons. DPM can perform as effectively if information of other types is passed. For example, it is possible to pass Autonomous System (AS) numbers instead of the ingress IP addresses. On one hand, passing an AS number is simpler because it is a 16-bit number. On the other hand, the ISP would need to either maintain internally or pass to the destinations some other information to determine the address of the interface closest to the attacker. For example, arbitrary 16-bit numbers that are unrelated to the IP addresses may be assigned to ingress interfaces and passed to the victim. Such marking would accomplish the same task of identifying the ingress interface without revealing its actual IP address, while transferring 32 bits of information.

Another approach is to alter the actual ingress address passed in the marks in a way that would allow the potential victim to identify the ISP, and in turn allow the ISP to identify the ingress interface. For example, the last byte of the ingress address inserted in the DPM marks may be changed in a predetermined way known only to the ISP. The victim would be able to determine the ISP based on the first three bytes, but not the actual ingress address of the marking interface, while the ISP would be able to actually determine the interface and take appropriate measures. Accordingly, DPM is relatively simple to modify such that details of the ISP topology are not revealed. For academic purposes, however, the rest of the article focuses on the actual IP address with the understanding that other information can be used in its place without loss of functionality.

4. Multiple attackers and IP source address inconsistency

The limitation of the basic DPM in handling certain types of DDoS stems from the fact that the victim associates segments of the ingress address with the source address of the attacker. However, because source addresses may be spoofed, the Basic DPM would be ineffective in many situations. For example, there are two illustrative situations when the reconstruction procedure of the basic DPM would fail. First, two hosts with the same SA may attack the victim. The ingress addresses corresponding to these two attackers are A_0 and A_1, respectively. The victim would receive four address segments: $A_0[0]$, $A_0[1]$, $A_1[0]$, and $A_1[1]$. The victim,

not being equipped to handle such an attack, would eventually reconstruct four permutations that are ultimately possible: $A_0[0].A_0[1]$, $A_0[0].A_1[1]$, $A_1[0].A_0[1]$, and $A_1[0].A_1[1]$, where '.' denotes concatenation. Only two of the four would be valid ingress address.

A typical benchmark for evaluation of the traceback methods for DDoS attacks is the *rate of false positives* or *false positive rate*. In the context of DPM, a false positive is defined as an incorrectly identified ingress address. The rate of false positives refers to the ratio of the incorrectly identified ingress addresses to the total number of identified ingress addresses. In the above example, the false positive rate is 50%. Clearly, the false positive rate would increase even further if the number of attackers, with the same SA, was even larger.

Second, (D)DoS attackers may simply change the source address field for every packet they send. The basic DPM is unable to reconstruct any valid ingress addresses because none of the entries in the *IngressTbl* would have a complete ingress address. The DPM methods described below builds on simplicity of the basic DPM while mitigating its deficiencies.

4.1. General principle of handling DDoS attacks

The general principle of handling (D)DoS attacks of these types is to rely *only* on the information transferred in the DPM mark [29]. The DPM mark can be used to not only transfer bits of the ingress address but also some other information. This additional information should enable the destination to determine which ingress address segments belong together to form a valid ingress address.

The reconstruction procedure utilizes the data structure called *Reconstruction Table* (*RecTbl*). The victim would first put the address segments in *RecTbl*, and then only after correctly identifying the ingress address, out of the many possible address segments permutations, would transfer it to *IngressTbl*.

4.2. Single-digest modification to DPM

The traceback method described in this section uses a hash function, $H(\cdot)$ to produce digests or hash values of the ingress address. It is assumed that the hash function is generally known to everyone, including all DPM-enabled interfaces, all destinations which intend to utilize DPM marks for traceback, and the attackers. The constraint of 17 bits still remains, and so inserting a longer digest in packets would result in fewer bits of the actual address transmitted in each mark, and consequently, the higher number of packets required for traceback.

4.2.1. Mark encoding

Recall that in the basic DPM, the ingress address was divided into two segments. In this single-digest modification, the ingress address is divided into k segments. Therefore, the mark contains $32/k$ bits of the address segment and $\log_2(k)$ bits required to identify a segment. The remaining bits are used for the digest. Regardless of which segment of the address is being sent to the victim, the digest portion of the mark always remains the same for a given DPM interface. This enables the victim to associate the segments of the ingress address with each other to reconstruct the complete address.

Fig. 3 shows the schematics of this approach. The DPM mark consists of three fields: a-bit address segment field, where $a = 32/k$, d-bit digest field, and s-bit segment number field, where $s = 17 - (a + d)$. Some padding may be required so that the address is split into segments of equal length.

At startup the DPM-enabled interface prepares k marks for all segments of the address. A d-bit hash value, or digest, of the ingress address is calculated only once and then inserted in the digest field of every mark. Each of the k marks have address bits set to a different segment of the ingress address. The segment number field is set to the appropriate value. These operations are shown to the left of

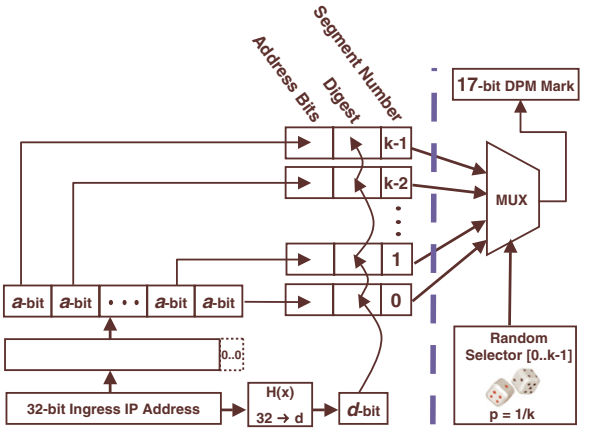

Fig. 3. Mark encoding for single-digest DDoS modification.

the bold dotted line in Fig. 3, and are performed only once. The processing required for every packet is limited to generating a small random number between 0 and $k-1$ and inserting the corresponding mark into the packet header.

4.2.2. Reconstruction by the victim

The reconstruction procedure of the single-digest method consists of two separate processes: Mark Recording and Ingress Address Recovery. The reason for separating these two tasks is that the attack packets may arrive at the destination faster than they can be analyzed. The mark recording process sets the appropriate bits in *RecTbl* to indicate which marks have arrived at the destination. Address recovery checks those bits, composes address segment permutations, and determines which ones are valid ingress addresses.

RecTbl is a 2^{17} bit structure, where every possible mark can be uniquely represented. It consists of 2^d areas, each area consists of k segments, and each segment consists of 2^a bits. Fig. 4 shows an example of *RecTbl*, where k, d, and a are 8, 10 and 4, respectively. When a marked packet arrives to the victim, the mark recording process sets the corresponding bit in the *RecTbl*. For an exemplary attacker, the ingress address can be possibly hashed into 2^d digest values. The digest is extracted from the mark and the area where the bit is to be set is determined. The segment number field in the mark indicates the segment in the *RecTbl* area, where the appropriate bit is to be set. Finally, the value of the address bits in the mark indicates the actual bit, which is set to '1'. This process is repeated for every incoming mark.

The address recovery process is a part of a larger traceback procedure, discussed below. It analyzes each area of the *RecTbl*. Once again, it runs independently from the mark recording process, thus allowing post-mortem traceback. The value of a bit in *RecTbl* indicates that the corresponding mark has arrived at the victim. For example, bit 12 in segment 3 of area 671 set to '1' means that there is an ingress address of interest, with digest of 671 having segment 3 equal to $'1100'_2$ as shown in Fig. 4. This segment has to be combined with other segments of this area in order to create permutations of segments. Once one or more permutations are created, hash function, $H(\cdot)$, is applied to each of them. If the result matches the area number, which is actually the digest embedded in the marks (in this example 671), the recovery process concludes that this permutation of segments is in fact a valid ingress address.

4.2.3. Performance analysis

In this section, we present the summary of the performance analysis for the single-digest modification. The two important metrics are: (1) the number of attackers, N, that this modified single-digest

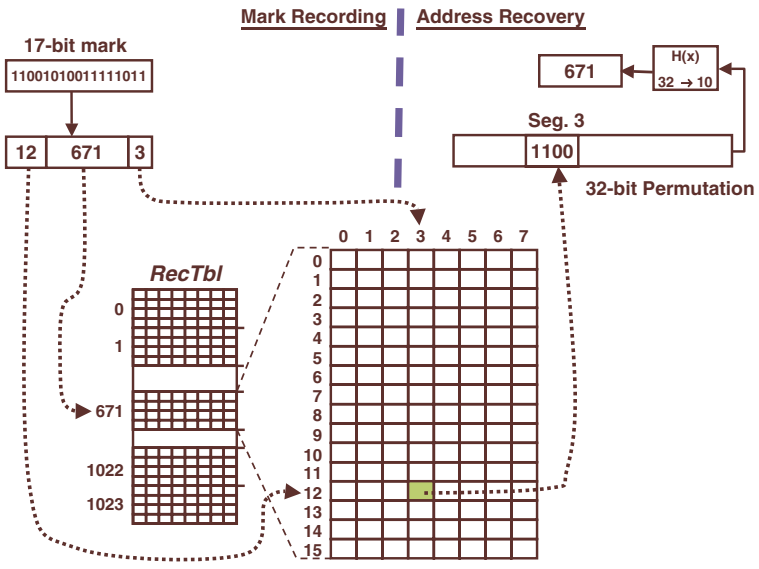

Fig. 4. *RecTbl* with $k=8$, $d=10$, $a=4$; mark recording; address recovery.

Table 1
Relationship between selected k and a, s, d, N_{MAX}, and $E[D]$ for the single-digest modification

k	a	s	d	N_{MAX}	$E[D]$
2	16	1	0	1	3
4	8	2	7	26	9
8	4	3	10	108	22
16	2	4	11	45	55
32	1	5	11	45	130

DMP technique can traceback with the false positive rate limited to 1%,[2] and (2) the expected number of datagrams, $E[D]$, required to be marked by a single DPM-enabled interface in order for the victim to be able to reconstruct its ingress. Table 1 presents this summary, showing the values of N_{MAX} and $E[D]$ for selected values of k. The derivations of these results is presented in detail in Appendix A.1.

4.3. Multiple digest DDoS modification to DPM

In the single-digest DPM described in Section 4.2, a single hash function, $H(\cdot)$, is used for identifying segments of ingress addresses. While this allows for identifying multiple ingress addresses of simultaneous attackers, this number is not sufficient for the real attacks. In this section, a modification, requiring a family of hash functions, is introduced.

4.3.1. Mark encoding

In this multiple digest DPM, a family of f hash functions, $H_0(\cdot)$ through $H_{f-1}(\cdot)$, is used to produce f digests of an ingress address. As in the single-digest method described in Section 4.2.1, an address segment and a segment number will be transferred in each mark. Instead of a single digest, however, one of the several digests produced by each of the f hash functions concatenated with the function identifier is embedded in the mark. The d-bit field, which was used solely for the digest in the single-digest scheme, is split into two fields: $\log_2(f)$-bit long field carrying the identifier of the hash function, and d-bit field with the digest itself.

Fig. 5 illustrates the process of the mark encoding. The process is very similar to the one described in Section 4.2.1, but differs in that for every ingress address, not k, but $f \times k$ marks are created at startup. The DPM-enabled interface selects one of them for every packet. This does not affect the DPM-enabled interface per-packet overhead, however, because the per-packet processing remains limited to generating a small random number and overwriting the 17 bits in the header, just as for the basic DPM or single-digest modification.

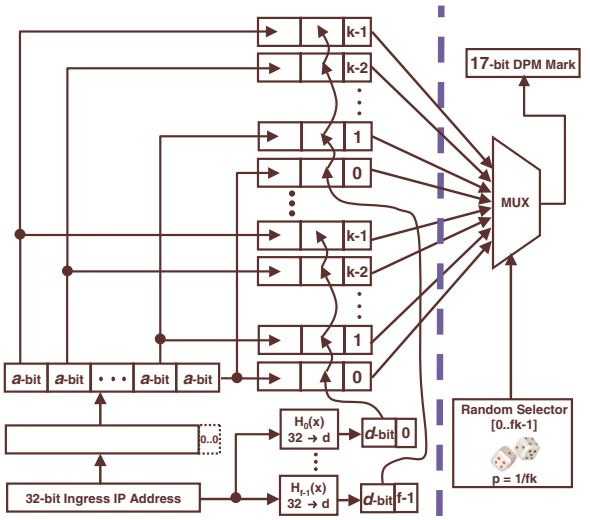

Fig. 5. Mark encoding for multiple digest DDoS modification.

4.3.2. Reconstruction by the destination

Reconstruction by the destination is also similar to that described in Section 4.2.2. The structure of *RecTbl* is changed slightly. To accommodate multiple digests, the *RecTbl* consists of f parts. Each of those parts has the structure identical to the *RecTbl* described in Section 4.2.2 (2^d areas, k segments in every area, and 2^a bits in every segment). The mark recording process first examines the hash function identifier field. Then it proceeds to the corresponding part of the *RecTbl*. Having identified the part in the *RecTbl*, the area, and the segment, the corresponding bit is set to '1', as in the single-digest modification.

The address recovery process, shown in Fig. 6, identifies the permutations that match the digest in areas of *Part*0 of *RecTbl*. Once a permutation is validated by comparing its digest obtained by applying $H_0(\cdot)$ to the area number, the rest of the hash functions, $H_1(\cdot)$ to $H_{f-1}(\cdot)$, are applied to it to produce $f-1$ digests. These digests are used to verify whether this permutation exists in other parts of *RecTbl*. The process then checks other areas of the

[2] False positives rate of 1% presents the theoretical boundary, typically used as a measure of performance for traceback algorithms. It is noted that lower numbers of N produce lower rate of false positives.

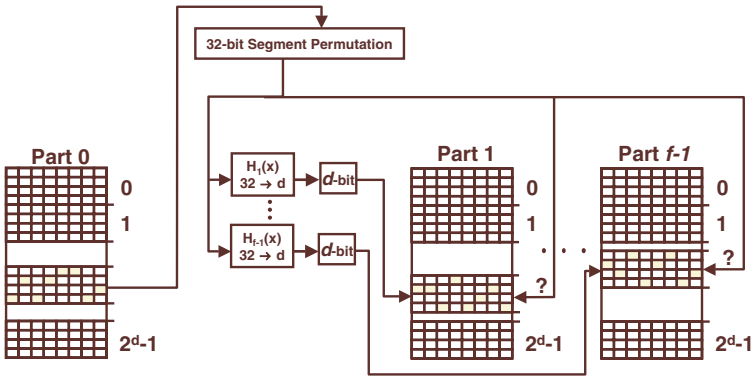

Fig. 6. Address recovery for multiple digest DDoS modification.

remaining parts for the permutation in question. If the permutation is present in the corresponding area of every part of the *RecTbl*, the process concludes that the permutation is a valid ingress address. Notice that the permutation does not have to be verified in every part. It is known that the digest obtained by applying $H_i(\cdot)$ to the permutation being checked will match the area number since the area was identified by this operation. Therefore, such verification would be redundant and always produce a positive outcome. The pseudocode in Fig. 7 provides the details of the mark encoding, mark recording, and address recovery processes.

4.3.3. Performance analysis

Table 2 provides the summary of multiple digest modification performance metrics. In particular, it shows the values of N_{MAX} and $E[D]$ for selected combinations of f, a, k, and d. The derivations of these values is presented in Appendix A.2.

As seen from Table 2, the multiple digest modification is able to reconstruct significantly more ingress addresses of simultaneous attackers than a single-digest modification without increasing $E[D]$ substantially. The details of performance analysis are provided in Appendix A.2.

5. Accommodating IP fragmentation

Fragmented traffic constitutes between 0.25% and 0.5% of the total IP traffic according to [2,30]. Though the amount of fragmented traffic is small, it still exists. The deterministic packet marking (DPM) methods, discussed so far, did not differentiate between fragmented and non-fragmented traffic. The ID Field, which is by design used for fragmentation, and RF of the IP header in every packet are completely replaced with one of $f \times k$ marks chosen at random. This prevents reconstruction of fragmented packets.

In this section, we explore why methods presented so far are a poor way to handle fragmented traffic, and present a modification to DPM to mitigate problems related to fragmentation.

5.1. IP fragmentation background and terminology

Fragmentation is a feature of Internet Protocol (IP) to enable transport of packets across the networks with different Maximum Transfer Unit (MTU). *Path MTU* is the smallest MTU of all links on the path from a source to a destination as described in [31]. When a packet enters the network, with MTU that is smaller than the packet length, the packet has to undergo a process of *fragmentation*.

Fig. 8 illustrates this process and introduces several important terms.[3] The *original datagram* is an IP datagram that is fragmented because its size exceeds the MTU of the next link. A *Packet Fragment*, or simply a *fragment*, refers to a packet containing a portion of the payload of the original datagram. While the terms *datagram* and *packet* are colloquially used as synonymous, here they refer to *original datagram* and *packet fragment*, respectively. A *fragment series*, or simply a *series*, is an ordered collection of fragments that results from a single original datagram.

When fragmentation occurs, each fragment becomes a valid IP packet. All fragments have their own IP headers. Most of the fields of the IP header

[3] The figure and the terminology used to describe different aspects of fragmentation is largely adopted from [30].

```
Marking procedure at router R, edge interface A:
 for z = 0 to f − 1
  Digest:= H_z(A)
  for y = 0 to k − 1
   Marks[z × k + y].Hash_num := z
   Marks[z × k + y].Digest := Digest
   Marks[z × k + y].Seg_Num := y
   Marks[z × k + y].A_bits := A[y]
 for each incoming packet w
  let x be a random integer from [0, f × k)
  write Marks[x] into w.Mark

Mark Recording procedure at victim V:
 for each attack packet w
   Part := w.Mark.Hash_num
   Area := w.Mark.Digest
   Seg := w.Mark.Seg_Num
   Bit := w.Mark.A_bits
   RecTbl[Part, Area, Seg, Bit] := '1'

Address Recovery procedure at victim V:
 for Area = 0 to 2^d − 1
  for Bit_0 = 0 to 2^a − 1
   if RecTbl[0, Area, 0, Bit_0] == '1' then
   ⋱
    if RecTbl[0, Area, k − 2, Bit_{k−2}] == '1' then
     for Bit_{k−1} = 0 to 2^a − 1
      if RecTbl[0, Area, k − 1, Bit_{k−1}] == '1' then
       Prm := Bit_0 . Bit_1 . .... Bit_{k−1}
       Digest := H_0(Prm)
       if Area == Digest then
        for Part = 0 to f − 1
         for Seg = 0 to k − 1
          if RecTbl[Part, H_Part(Prm), Seg, Bit_Seg] ≠ '1' then
           False flag := '1'
        if False flag ≠ '1' then
         Prm ⇒ IngressTbl
```

Fig. 7. Pseudocode for the modified multiple digest DPM algorithm.

Table 2
Relationship between f, k and a, s, d, N_{MAX}, and $E[D]$ for selected combinations for multiple digest modification

f	k	a	s	d	N_{MAX}	$E[D]$
4	8	4	3	8	2911	130
4	4	8	2	5	2296	55
8	4	8	2	4	2479	130

of the fragments are inherited from the original datagram IP header. The fields of interest are ID field, Flags, and Offset. ID field is copied from the original datagram to all the fragments. The Source Address (SA), Destination Address (DA), Protocol (P), and ID, are used by the destination to distinguish the fragments of different series [32,33]. The ID field of all fragments, which resulted from fragmenting a single datagram, must have their ID field in the IP header set to the same value for proper reassembly. More Fragments (MF) flag is set to '1' in every fragment except the last one. This flag indicates that more fragments follow. The last fragment has MF set to '0' to indicate that it is the last fragment in the series. Finally, the offset field of the IP header is set to the position of the data in the fragment with respect to the beginning of data in the original datagram. The offset is measured in the units of 8 bytes.

For successful reassembly, the destination has to acquire all of the fragments of the original datagram. A tuple (SA, DA, P, ID) is used to determine if the fragments belong to the same original datagram, MF is used to indicate the number of fragments, and Offset is used to determine the correct order of reassembly. Note that the fragments may arrive out of order, but reassembly will still be successful because the destination is able to determine

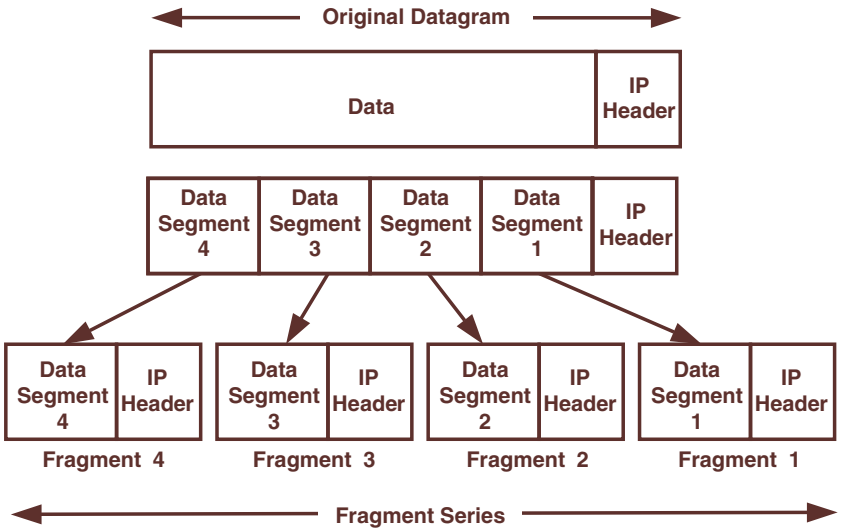

Fig. 8. IP fragmentation.

that a fragment belongs to a series, and its position relative to other fragments.

DPM uses the ID field to transfer the mark and that may cause reassembly failure.[4] We examine the effects of DPM on IP reassembly and then introduce a method to deal with these undesirable effects, and analyze the performance of the methods in terms of the probability of reassembly error.

5.2. Shortcomings of DPM related to fragmentation

Fragmentation can occur *upstream* or *downstream* relative to the point of marking according to [2]. These two situations have to be considered separately.

5.2.1. Upstream fragmentation

Upstream fragmentation is known to the DPM-enabled interface. The DPM-enabled interface can identify a packet to be a fragment by examining its MF and Offset.

In case of upstream fragmentation, a datagram is fragmented by a router or a host before it reaches the DPM-enabled interface. When a series of fragments of the original datagram reaches the DPM-enabled interface, the ID and RF fields of all the fragments is replaced with one of the $f \times k$ marks *picked at random*. This will cause fragments to have different ID fields when they arrive to the destination. Fragments with different ID fields will be considered to be parts of different datagrams by the destination. The reassembly will eventually timeout since the destination will never get all the fragments necessary for the reassembly of what it considers to be several different series. The probability of all fragments in a series of even 2 fragments having the same ID field after marking is $\frac{1}{fk}$; for a series of three packets, $\frac{1}{fk^2}$, etc. For $f \times k = 16$, the probability of a series consisting of 2 fragments being correctly reassembled is 6.25%, for a series of 3 fragments – 0.4%, etc. Clearly, the rate of reassembly errors caused by upstream fragmentation is unacceptable. The ability of DPM-enabled interface to recognize upstream fragmentation allows for a different strategy for marking these packets described in Section 5.3.

5.2.2. Downstream fragmentation

Downstream fragmentation is unknown to DPM. The DPM-enabled interface has no knowledge if the marked datagrams, are fragmented anywhere along the path. Therefore, the datagrams that will be fragmented after the marking, cannot be treated differently from the traffic which is not fragmented.

Luckily, fragmentation downstream from the DPM-enabled interface does not cause any problems for reassembly. The router that performs fragmentation, simply copies the content of the ID field of the original datagram into every fragment. The value of RF is also copied into every fragment as

[4] Other IP traceback schemes based on packet marking also face the same problem, but most of them simply ignore the issue.

specified in [34]. At the destination, reassembly will be successful since the ID field will be the same for every fragment in the series. The fact that the ID field has been set by DPM, and the original value set by the host has been replaced, is unknown to the destination, and is irrelevant for the purposes of reassembly.

5.3. Fragment-persistent DPM

In this section we introduce a modification to the DPM marking procedure that eliminates most potential reassembly failures associated with upstream fragmentation. The simplest modification is discussed first, followed by additional changes resulting in the fragmentation-friendly marking procedure.

5.3.1. Fundamentals of handling upstream fragmentation with DPM

For proper reassembly, all fragments of the original datagram must have the same ID field. The basic DPM marks packets randomly chosen among $f \times k$ marks. This randomness must be suspended when processing fragments. In order to accomplish this task, DPM has to keep track of the fragments, which pass through. If a certain mark has been inserted in the first fragment that DPM-enabled interface encounters (which does not have to be the fragment with offset 0), then the same mark must be inserted in the rest of the fragments of this series. The information about which mark is used for which series has to be stored in a table, called *FragTbl*, at the DPM-enabled interface and checked every time a new fragment arrives. To identify fragments that belong to the same original datagram, DPM should check if the four-field tuple (SA, DA, P, ID) is the same as any other it marked within the maximum reassembly timeout of 120 s.

5.3.2. Dealing with infinite series

Assuming that an attacker can generate any packet, it is possible that he will utilize *artificial fragmentation* to force the DPM-enabled interface to send only a single mark to the destination. Artificial fragmentation is sending packets with MF Flag set to '1' or non-zero offset while fragmentation is not required for the proper reason of the datagram exceeding the MTU size of a link. With artificial fragmentation, the attacker may generate infinitely many packets with the same SA, DA, P, and ID fields, which would look like fragments of one very long series to the DPM-enabled interface or the destination. This is known as an *infinite series*. Only when the destination would attempt to reassemble the datagram, it would realize that fragments are invalid, but for success of (D)DoS attacks it would be enough that the invalid packets occupy the resources of the victim. In this situation, the victim will never recover the full ingress address, because only a single mark would be available.

To remedy this situation, another simple modification in addition to the fragment persistence is introduced. The modification is based on the observation based on real Internet traffic that the longest series is 44-fragments [30]. DPM should recognize the fact that if the number of fragments in the series exceeds 44, it is, in all likelihood, an attack, or a result of an error. In either case, such traffic is not expected to be properly reassembled. So, after DPM has persistently marked 44 fragments of a single series with the same mark, subsequent fragments from the same series are marked randomly. According to this modification, the *FragTbl*, which DPM has to keep for fragments, where the value of ID+RF corresponding to (SA, DA, P, ID) is kept also keeps a counter that is incremented for every fragment when a given tuple is encountered. Once this counter exceeds 44, marking persistence is suspended and marking at random is resumed.

5.3.3. Practical compromise

The modification described in Section 5.3.2 accommodates all of the valid fragmented traffic. However, artificial fragmentation may still be used by the attacker to generate bogus 44-fragment series directed to the victim. This allows the attacker to increase the expected number of packets required to be marked by a DPM-enabled interface for the victim to reconstruct its ingress address, $E[Pkt]$, by the factor of 44. It is possible to modify the procedure outlined in Section 5.3.2 to significantly reduce this factor with minimal trade-off.

According to [30], about 99% of series are only 2 or 3 fragments long. This fact may be taken in consideration when resuming randomness. Accordingly, if the randomness in selecting a mark is resumed after only 3 fragments have passed through the DPM-enabled interface, 99% of fragmented datagrams will be unaffected and will reassemble successfully at the destination. To the attacker, suspending randomness for only 3 fragments makes sending series longer than 3 fragments to the victim totally pointless. For example, sending a series of 45

fragments will result in 3 fragments marked with the same mark, and the remaining 42 fragments marked randomly. The marks will be picked at random 43 times. Assuming $f \times k = 16$, approximately 15 different marks will be sent to the victim, according to the classical occupancy problem discussed in [35]. The same number of packets may be sent to the victim if the attacker sends 15 series, 3 fragments each. All 3 fragments in every series are marked with the same mark. Therefore, random mark will be picked only 15 times, resulting in approximately 10 different marks sent to the victim. Clearly, sending series of 3 fragments to the victim, becomes the most sensible option for the attacker. While this approach takes care of all 2- and 3-fragment series, which is 99% of all fragmented traffic, the remaining 1% of valid series that contain more than 3 fragments, practically speaking, will never get reassembled at the destination.

The compromised approach presented in this section accommodates these longer series. According to this approach, when the DPM-enabled interface encounters the first (not necessarily with offset equal to 0) fragment in a series, it determines if the randomness will be suspended for 3 fragments or for 44 fragments for this series. The probability p, with which the randomness is suspended for 44 fragments, should be selected in such a way that would provide no advantage to the attacker in sending series longer than 3 fragments.

Sending series of more than 44 fragments results in no advantage to the attacker in either case, because randomly selected marks are inserted in the fragments after the 44th one in both cases. However, the attacker may send series of exactly 44 fragments hoping that the number of marks sent in a series of 44 packets is less than in a series of 3. If the attacker generates a 44-fragment series, only a single mark is inserted in all the fragments, with probability p. Alternatively, only 3 first fragments would have the same mark, and the rest 41 fragments would have randomly picked marks with probability $1 - p$. Accordingly, 42 randomly selected marks are transferred to the victim in the fragments of this series with probability $(1 - p)$.

We would like to find the value of p that would make it impractical for the attacker to send series of longer than 3 fragments. Denote D as the number of datagrams being sent. In case of sending series of 3 fragments, the expected number of times marks are randomly picked (different from the number of marks acquired by the victim) is D, and the number of packets sent to the victim is $3D$. In case of sending 44 fragment series, the expected number of randomly chosen marks is $D(42(1-p)+p)$, and the number of packets sent to the victim is $44D$. The ratio of number of packets to the number of generated marks is called a fragmentation coefficient C. For these two possible ways of using artificial fragmentation, C must be the same, to ensure that the attacker gains no advantage in sending long series

$$\frac{44D}{D(42(1-p)+p)} = \frac{3D}{D},$$

$$\frac{44}{42(1-p)+p} = 3.$$

Solving for p results in the value of $\frac{2}{3}$. It is important that the number of datagrams sent by the given host does not affect the value of p. This means that DPM can suspend randomness in mark selection for 44 fragments in 2 out of every 3 datagrams. Approximately 33.3% of the datagrams fragmented into more than 3 fragments upstream would still fail to reassemble at the destination. However, considering that the fragmented traffic is only 0.5% of the overall traffic, only about 0.017% of the overall traffic would be affected. The pseudocode of the encoding procedure reflecting the practical compromise is depicted in Fig. 9. Processing at the victim is not affected by fragmentation accommodating modification.

```
Marking procedure at router R, edge interface A:
 for z = 0 to f − 1
  Digest:= H_z(A)
  for y = 0 to k − 1
   Marks[z × k + y].Hash_num := z
   Marks[z × k + y].Digest := Digest
   Marks[z × k + y].Seg_Num := y
   Marks[z × k + y].A_bits := A[y]
 for each incoming packet w
  let x be a random integer from [0, f × k)
   if w.MF == '1' OR w.offset ≠ 0 then
    if FragTbl[SA, DA, P, ID] == NIL then
     create FragTbl[SA, DA, P, ID]
     let v be a random integer from set {3, 44, 44}
     FragTbl[SA, DA, P, ID].Mark_num := x
     FragTbl[SA, DA, P, ID].counter := 1
     FragTbl[SA, DA, P, ID].Suspend := v
    else
     if (FragTbl[SA, DA, P, ID].counter <
     ↪    < FragTbl[SA, DA, P, ID].Suspend) then
      x := FragTbl[SA, DA, P, ID].Mark_num
      FragTbl[SA, DA, P, ID].counter++
  write Marks[x] into w.Mark
```

Fig. 9. Pseudocode for the fragment-persistent DPM.

5.4. Size of the FragTbl

In this section, the size of memory required for the *FragTbl* is discussed. This is an important issues because this memory overhead is actually incurred by the routers, and as mentioned earlier the ISPs involvement for traceback methods should be minimal. The amount of memory required for the *FragTbl* depends on the interface speed and will therefore vary for different interfaces.

The size of *FragTbl* is directly proportional to the rate of the DPM-enabled interface, R. The interfaces with the higher rate are able to process more packets per second. According to [30], approximately 0.5% of IP packets are fragmented. For every series 12 bytes (4-Byte SA, 4-Byte DA, 2-Byte ID, 1-Byte P, 4-bit *fk* value, and 1-bit required to store two values of threshold for the number of fragments to resume randomness) are allocated in the *FragTbl* and every entry should be held in the *FragTbl* for 120 s. Keeping the entry longer than 120 s is impractical, because the time of the reassembly process at the destination is 120 s [36]. We conservatively consider the average packet size to be 1000 bits [19]. The recent traffic measurement studies suggest that the average packet size, however, is closer to 400–600 Bytes [37,38]. It follows then, that the size of the *FragTbl* in Bytes is given by

$$\frac{R\ \text{bits/s} \times 120\ \text{s} \times 0.005 \times 12\ \text{Bytes}}{1000\ \text{bits}} = 0.0072R\ \text{MBytes}.$$

Table 3 summarizes memory requirements of *FragTbl* for various commonly used interfaces. The interfaces, which are likely to be on the edges of even a large ISP would not require more than 100 MB of RAM.

Table 3
Common interfaces, their rates and estimated *FragTbl* sizes

Type	Rate	*FragTbl*
OC-768	40 Gb/s	288 MB
OC-192	10 Gb/s	72 MB
10GigE	10 Gb/s	72 MB
OC-48	2.5 Gb/s	18 MB
GigE	1 Gb/s	7.2 MB
OC-12	622 Mb/s	4.5 MB
OC-3	155 Mb/s	1.12 MB
FastE	100 Mb/s	0.72 MB
OC-1	51.84 Mb/s	0.37 MB
DS3	44.736 Mb/s	0.33 MB
DS1	1.544 Mb/s	11 KB
DS0	64 Kb/s	<1 KB

5.5. Alternative methods of handling fragmentation

Given the small overall percentage of the fragmented traffic on the Internet, it becomes questionable whether it is a useful feature of just an additional exploit available to the attacker. While the practical compromise described above accommodates most of the fragmented traffic, it increases $E[Pkt]$ by a factor of 3. In this sub-section, we briefly introduce some alternatives that may be more practical in the actual deployment.

First, DPM-enabled interfaces may simply drop all fragmented traffic. While being a crude alternative, it may actually eliminate applications that still rely on IP fragmentation. As a result, hosts will not need to have fragmentation and reassembly modules, which would completely obviate the need for looking into DF, MF, and Offset fields, which in turn would allow the use of those fields for other purposes, such as transfer of traceback-related information, as proposed for example in [18].

Second, DPM may not mark fragmented traffic. This creates an open invitation for attackers to use artificially fragmented traffic for DDoS attacks. To prevent such attacks, routers on the Internet, as well as hosts, may implement a flow control mechanism that would give the lowest priority to the fragmented traffic and drop fragmented traffic in case of a flooding. This alternative ensures that non-attack fragmented traffic would be properly reassembled at the destination while decreasing the possibility of a DDoS attack using the fragmented traffic.

6. Traceback

In this section, we discuss different types of attacks and introduce the traceback for various types of attacks. We then discuss the conditions for traceability for each type.

6.1. Types of cyber attacks

In [39], cyber attacks are divided into four classes: reconnaissance, informational, access, and denial of service. The first three classes can be combined into one in the context of the DPM traceback and can be called *intrusions*. An important characteristic of an intrusion is that the attacker is interested in receiving some information from the victim. The attacker is thus forced to use a stable

IP address in order to receive the replies from the victim, and the traceback of such attacks is trivial.

Denial-of-service attacks have become very popular recently. Currently, there is no complete comprehensive defense against these attacks, which is why the traceback of these attacks becomes even more important. The common goal of all denial-of-service attacks is to render the victim unable to provide services to the customers. This is usually accomplished by exhausting physical or logical resources on the victim's servers and networks or the ISP uplink.

A required feature of a DDoS attack is a collection of *slaves*. The slaves are the hosts on the Internet that the attacker has compromised by using common vulnerabilities and bugs in the operating systems [40]. When the attacker compromises a host and gains full or partial control, he installs special software for sending packets, called a flood server on the host, thus making it a slave. Since the attacker controls the slaves, it is possible to have the slaves generate any packet, in particular, packets with spoofed SA and artificially fragmented datagrams. The DDoS attack may also involve *reflectors*, which are uncompromised hosts with opened services (such as www), which are used to reflect the traffic from the slaves to the victim. A typical reflection works as follows. The slaves spoof the source address in the packet directed to the reflectors with the victim's IP address. As a result, the victim is being flooded by replies, which were originated by reflectors in response to the requests. Note that the attacker has no control over the reflectors, and therefore the reflectors can only generate valid packets. That is, the SA field will have the reflector's IP address, and the artificial fragmentation cannot be orchestrated by the reflectors.

In the most general case of a DDoS attack, called a mixed DDoS attack in this article, the attacker may instruct the slaves to flood the victim directly *and* send packets to reflectors, from which packets are reflected to the victim. In Fig. 10, slave **S** sends packets 1 and 2 to reflectors **R1** and **R2**, respectively. The SA of these packets are spoofed with the address of the victim **V**. The generated replies 1 and 2 are then directed to **V**. Note that SA fields of the replies 1 and 2 are not spoofed and contain valid source addresses of **R1** and **R2**. Also, **S** sends packet 3, with a spoofed SA for some random value **R3**, directly to **V**. The fact that **S** sends the packets to both **V** and **R**'s constitutes a mixed DDoS. A major attack would involve hundreds or even thousands of slaves and up to a million of reflectors [40].

There are well known special cases of the mixed DDoS attacks. One special case is a reflector-based DDoS attack, which occurs when slaves send packets only to reflectors, and the victim is flooded by the replies from the reflectors only. Another special case is a slave-based DDoS attack, which occurs when slaves send packets only to the victim. The reflectors are not engaged in a slave-based attack. A special case of a slave-based DDoS attack is a

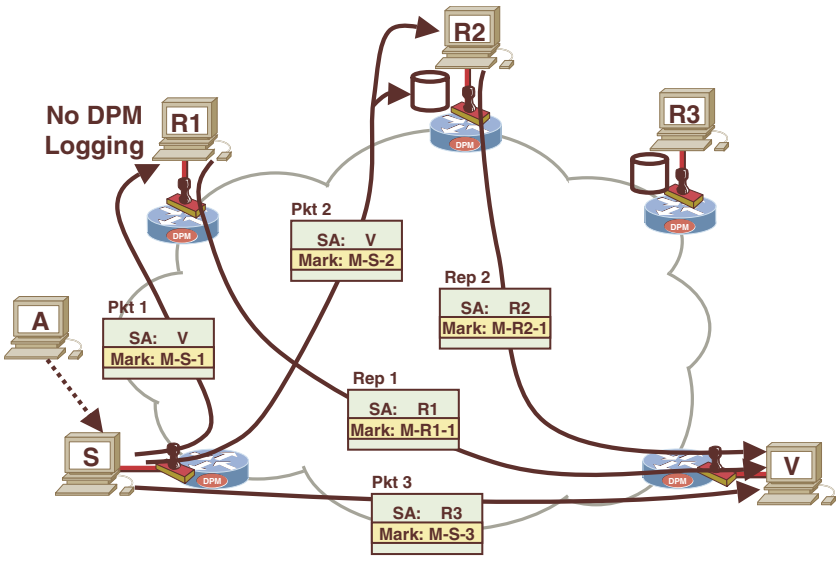

Fig. 10. Composition of (D)DoS attacks.

DoS attack, which occurs when only a single slave participates in a slave-based attack.

6.2. Traceback data structures

As a stream of packets enters the victim's network, the SA field and the mark of *every* packet must be logged. Logging is already performed by most Web servers. Logging accomplishes two tasks: (1) it enables tracing slaves from reflectors as discussed in Section 6.4, and (2) it enables post-mortem traceback. During traceback, the DPM Traceback Procedure creates an instance of a *TraceTbl* that consists of *RecTbl*s. Each *RecTbl* is associated with a source address (SA) of one of the attack packets. In addition, there is a *StatTbl*, a data structure identical to *TraceTbl*, that is associated with the *TraceTbl* and is used for analysis of the marks. Finally, there is a common *RecTbl*, where final address reconstruction occurs. The ingress addresses may be recovered in *StatTbl* or the common *RecTbl*, and so they can be copied to *IngressTbl* from either data structure. Fig. 11 illustrates the data structures involved.

6.3. Traceback procedure

A single procedure must be able to handle all types of attacks discussed in Section 6.1. Every attack packet that arrives to the victim has a mark. The victim sets the appropriate bit in the *RecTbl* associated with the SA address of the attack packet in the *TraceTbl* to '1', as described in Section 4.

At predetermined time intervals or once the attack is over, the content of the *TraceTbl* is copied into *StatTbl*, which is used for the statistical analysis of the marks. The recording of marks is not performed in *StatTbl*, although while the *StatTbl* is analyzed, the incoming marks continue to be recorded in the *TraceTbl*. The *StatTbl* is used only to analyze the SAs and associated *RecTbl*s.

It is beneficial to examine how receiving marked packets from a reflector affects a corresponding *RecTbl*. A single area of every part would have exactly k bits set to '1'. Furthermore, none of the bits set to '1' can be within the same segment. We define a *proper RecTbl* to be a *RecTbl* that has exactly one bit set to '1' for every segment. A proper *RecTbl* therefore resembles a *RecTbl* that would result for a SA, when a reflector sends packets to the victim. It is entirely possible, of course, for a slave to spoof the SA field in its packet directed to the victim in such a way that would result in a proper *RecTbl*. Therefore, the victim cannot automatically assume that marks in a proper *RecTbl* are from a reflector's ingress address. The analysis of a slave being able to fake a proper *RecTbl* is provided below.

The procedure relies on the observation that a reflector sending packets to the victim results in

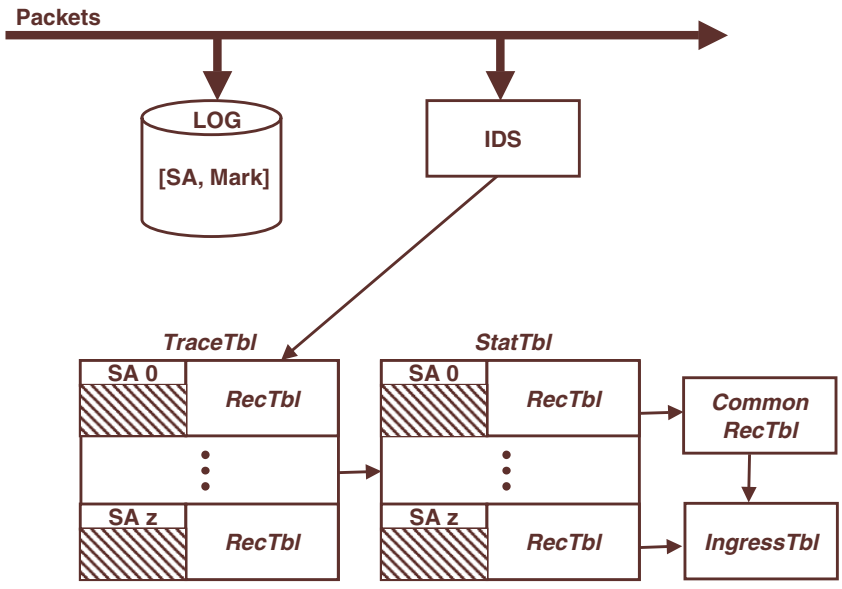

Fig. 11. DPM traceback data structures.

creation of a proper *RecTbl* (provided it sends enough packets) due to the fact that the reflector cannot change the SA-mark association. An attacker may want to pollute proper *RecTbl*s to make reconstruction more difficult. For example, the attacker may pollute the proper *RecTbl* by directing its slaves to send packets with the SA of the reflector directly to the victim. By doing so, however, the attacker increases the number of marks associated with the slaves ingress addresses to be sent to the victim.

The procedure applies the address recovery process on every individual *RecTbl* in the *StatTble*. Any ingress addresses, which are reconstructed, are stored in the *IngressTbl*. Moreover, if the *RecTbl* is proper, it is removed from the *StatTbl*. Refer to Sections 6.4.1 and 6.4.2 for further discussion on proper *RecTbl*s and mark deletion.

After applying the address recovery process to the individual *RecTbl*s, the victim has determined the addresses of reflectors, and possibly some slaves that have not effectively spoofed their SAs, so that all k marks from one slave happened to be associated with a single SA.

The number of potential marks from slaves' ingress addresses identified by the victim should not exceed $\overline{M}_{SL}$. $\overline{M}_{SL}$ is the expected number of marks, which the victim would collect if attacked by N_{MAX} attackers simultaneously, and is given by the equation below

$$\overline{M}_{SL} = f \times 2^d \times k \left[2^a - 2^a \left(1 - \frac{1}{2^a} \right)^{\frac{N}{E[H]}} \right],$$

where $E[H]$ is given by Eq. (1) in Appendix A. For $f = 4$ and $k = 4$, $\overline{M}_{SL}$ is 32,038.[5] Keeping the number of marks analyzed by the victim under $\overline{M}_{SL}$ ensures that the rate of false positives will not exceed 1%.

Depending on the attack profile, some marks that remained in the *StatTbl* at this point may be from the slaves' ingress addresses while others may be from the reflectors' ingress addresses stored in improper *RecTbl*s. A certain number of these remaining marks should be selected to be copied to the common *RecTbl*. The total number of marks in the common *RecTbl* should not exceed $\overline{M}_{SL}$.

[5] This result was also supported by the following simulation. N_{MAX} 32-bit numbers were generated and divided into segments; the corresponding marks were created; the number of unique marks M_{SL} was calculated; $\overline{M}_{SL}$ was computed by averaging M_{SL} over multiple experiments.

We define a *mark occurrence* as the number of *RecTbl*s in which that mark appears. In other words, if a given mark arrived multiple times in the packets with the same SA, only one occurrence is counted. However, if a mark appears in packets with different SAs, then such mark has the number of occurrences corresponding to the number of the unique SA's. For its final analysis, the victim selects marks with the highest number of occurrences to be copied to the common *RecTbl*. Assuming that the marks are distributed uniformly in the interval of $[0, 2^{17})$, the only reason for a mark to have a higher number of occurrence is that a slave has sent packets to the victim with different SAs. This leads to the situation when the same marks appear in the *RecTbl*s associated with different SAs, and thus its number of occurrence is higher than for other marks. Therefore, marks with a higher number of occurrence are more likely to be from the slaves' ingress addresses.

Once $\overline{M}_{SL}$ most frequently occurred marks are copied to the common *RecTbl*, the address recovery process is applied to the common *RecTbl* and the ingress addresses are reconstructed. The reconstructed addresses are ingress addresses of slaves that sent packets to the victim directly, but cannot be reflectors addresses because they would have been constructed in the corresponding *RecTbl*s in the *StatTbl*, or in the alternative if they would not have been reconstructed, the marks of those ingress addresses would not be most frequently occurring.

6.4. Tracing slaves from reflectors

As discussed above, potential victims log attack packets. Information from the logs kept on reflectors may be used by the victim to collect marks with ingress addresses of slaves.

Recall that the attacker engages a reflector in the attack by sending a packet to the reflector from a slave with the SA spoofed for the address of the victim. The reply to this packet, whatever it might be, is directed to the victim. Even though the attacker may change the SA of the packets sent by slaves, the DPM marks cannot be changed. Therefore, the marks of the packets to a reflector with the SA of the victim may be used to reconstruct the ingress address of the slave(s), which sent the packets to this reflector. The marks obtained from reflectors may be used in the traceback procedure to identify the ingress addresses of slaves.

The protocol of obtaining the logs is beyond the scope of this work. It should be noted that the protocol itself may be exploited by the attackers. To mitigate this, the protocol may implement a new or presently existing security mechanism such as IPSec. Also, in this article the authors would like to emphasize the principles of being able to trace the slaves from the victim by obtaining marks from reflectors, not the actual implementation.

A given reflector may have DPM logging enabled or disabled. When the victim makes a request to the reflector for the marks from the logs, the victim's address and the approximate time of packet arrival are supplied to the reflector. Three responses are possible:

- **Error (or no response)**, if the logging is not enabled on a given reflector;
- **Positive response**, with the list of marks matching the specified parameters returned to the requesting victim; and
- **Negative response**, if the logging on the reflector is enabled, but none of the logged entries matched the specified parameters.

Fig. 12 shows three reflectors. **R2** and **R3** have DPM logging enabled and **R1** does not. When **V** performs the traceback, the addresses of **R1**, **R2**, **R3** will be available to **V**. At this point, **V** has no knowledge that the packet with the spoofed SA of **R3** was sent from slaves. **V** sends log requests to each of the received addresses. Fig. 12 illustrates three possible responses to the log requests. When the log request is sent to **R1**, error (or no response at all – depending on the implementation) is returned, since the logging was not enabled. **R2** had logging enabled and had a record of the packet with the SA of **V**. The marks (in this case only one) are sent to **V**. **R3** also has logging enabled; however, it did not receive any packets with the SA of **V**, and so the response is negative. Both positive and negative responses are useful to the victim as will be seen in Section 6.3.

The traceback procedure may be modified as follows to take advantage of reflectors' logs. First, the traceback procedure makes log requests to potential reflectors. SAs which have *RecTbl*s associated with them would be used to determine addresses of those reflectors. In case of a positive response from a reflector, the victim obtains a list of marks associated with the ingress addresses of one or more slaves. It is certain that those marks are from the ingress interface of one or more slaves since they came in the packets that had the victim's SA, and so the marks received in the response are copied to the common *RecTbl*. A reflector could perform the traceback based on these marks by itself. However, the number of attack packets, which it would receive may not be enough to recognize the attack, and even if the attack was recognized, the marks, which a single reflector would obtain, may not be enough for the traceback. In particular, the attacker may send only a few packets to a reflector from all its slaves, thus preventing the reflector from identifying slaves by itself.

In case of a negative response, the traceback procedure on the victim concludes that every mark in

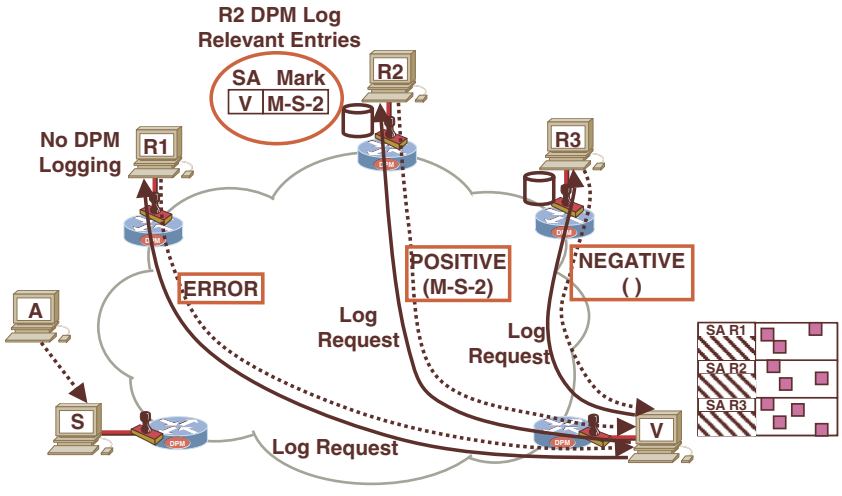

Fig. 12. Illustration of reflector log requests and responses.

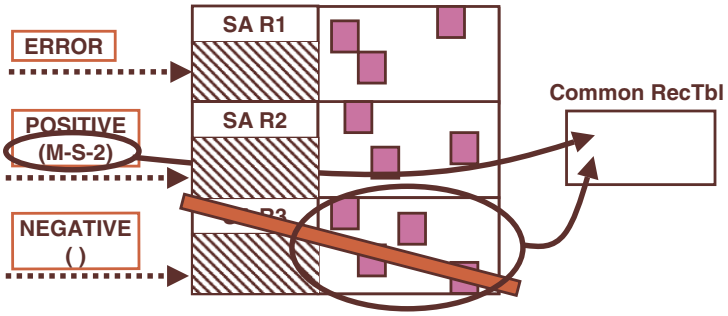

Fig. 13. The victim's processing of log responses.

the *RecTbl* associated with that SA came from the ingress interface of one or more slaves. A negative response means that the reflector indicates that it did not send packets to the victim. Therefore, the marks associated with the reflector's SA must be from slaves. These marks should also be copied to the common *RecTbl*. The *RecTbl*s associated with spoofed SAs should be removed from the *StatTbl* and should not be considered for further analysis. Only in the case of no response, the procedure cannot make any determination as to the nature of the marks and has to move on to the next SA in the *StatTbl*. These alternatives and the corresponding actions are depicted in Fig. 13.

Finally, the marks with the highest number of occurrence are copied to the common *RecTbl* to bring the number of marks in it to $\overline{M}_{SL}$, and the reconstruction process is executed. A formal description of the procedure is presented in Fig. 14.

Traceback procedure at victim V:
for each SA in $StatTbl$
send $Log\_Request(SA, V, time \pm \delta)$
if $Log\_Response \neq NIL$ **then**
read $Marks[\,]$ of $Log\ Response$
record $Marks[\,] \Rightarrow com.\ RecTbl$
if $Log\_Response == NIL$ **then**
read $Marks[\,]$ of $SA.RecTbl$
record $Marks[\,] \Rightarrow com.\ RecTbl$
delete $SA.RecTbl$ from $StatTbl$
for each SA in $StatTbl$
run $Address\_Recovery(SA.RecTbl)$
if $proper(SA.RecTbl) == TRUE$ **then**
delete $SA.RecTbl$ from $StatTbl$
for each SA in $StatTbl$
read $Marks[\,]$ of $SA.RecTbl$
for each $Mark$ in $Marks[]$
$Occ\_Structure[Mark]$.Value := $Mark$
$Occ\_Structure[Mark]$.Occurrence ++
Num_Select := $\overline{M}_{SL} - n(com.\ RecTbl)$
sort $Occ\_Structure[]$ by Occurrence
for x := 0 **to** Num_Select
record $Occ\_Structure[x].Value \Rightarrow com.\ RecTbl$
run $Address\_Recovery(com.\ RecTbl)$

Fig. 14. Pseudocode for the traceback procedure.

6.4.1. On proper RecTbls and hiding of marks

In this section, a concept of a proper *RecTbl* is analyzed. Recall that one of the steps of the procedure discussed above is to remove the proper *RecTbl* from the *StatTbl*. As a part of the mixed DDoS, the attacker may attempt to send a packet from a slave with the SA of the reflector that sent enough packets for the victim to collect a complete set of marks to reconstruct its ingress interface. The attacker may try to make a mark in its packet the same as the one already received by the victim from a reflector. The attacker may attempt to select a slave in such a way that one or more marks of slaves ingress addresses are the same as one or more reflectors marks, thus having the slaves marks deleted once the victim establishes that the *RecTbl* is proper.

Intuitively, the chances of this situation occurring are very low. The attacker may find a few slaves' ingress addresses with one or even two segments matching the corresponding segments of the ingress address of one of the reflectors. The probability that at least one of the digests is the same multiplied by the probability that this digest and that segment are picked in the only mark sent to the victim is very low. Accordingly, such attempts by the attacker are not an effective way to hide the slaves marks.

6.4.2. On deleting marks from RecTbl

If the victim undergoes a mixed attack, the attacker could instruct the slaves to send packets with the SA of the reflectors, thus making sure that the marks will be recorded in *RecTbl* associated with at least some SAs of reflectors. By having additional marks in those *RecTbl*s, the attacker ensures that the marks from reflectors do not get deleted from the *StatTbl*. Why the marks which were used in

reconstruction of a valid ingress address are not removed in the presence of other marks in the *RecTbl*? If the attacker came into possession of many slaves, it is possible to select reflectors in such a way that at least one out of $f \times k$ marks inserted by the DPM interface of the reflector is the same as one of the marks inserted by the DPM interface of the slave. The attacker may instruct a slave to send packets to the victim with the SA of reflectors. If the slave does not send enough packets so that the victim collects all the marks in the *RecTbl* associated with the SA of the reflector and the ingress address of that slave is reconstructed, then the ingress address of the slave will never be reconstructed if one of its marks is deleted. Therefore, the marks may be deleted from the *StatTbl* only if there is a high degree of certainty that only the marks from a single reflector's DPM interface are recorded in the given *RecTbl*, which occurs when the *RecTbl* is proper.

6.5. Conditions for traceable and untraceable attacks

In this section, we analyze the attacks that can and cannot be traced back with the procedure described in Section 6.3. All attacks are analyzed from the victim traceback procedure's point of view. Let S be the number of slaves involved in the attack, L be the fraction of hosts on the Internet with enabled DPM logging, and C be the fragmentation coefficient.

We define *marginally traceable* attacks as the attacks during which the number of packets received by the victim, is less than the expected number of packets required for traceback, $E[Pkt]$. Yet, that number may be enough to collect all the marks necessary for traceback. We also define *untraceable* attacks as the attacks which can never be traced. The difference between the two is that while *marginally traceable* can still be traced with the probability of success of below 50%, the *untraceable* attacks cannot possibly be traced. For the marginally traceable attacks with multiple hosts involved, such as DDoS attacks, the victim may be able to trace back to some of the hosts involved in the attack.

6.5.1. Intrusions

The intrusion cannot comprise packets with the spoofed SA, and so theoretically even a single packet would be enough to perform the traceback. However, if an intruder engages in some elaborate scheme in which he spoofs his address and is still capable to get the desired information, the DPM tracing procedure would have to be used. As mentioned above, artificial fragmentation would not provide any benefit to the attacker in terms of obstructing the traceback process, and therefore is not likely to be used. The expected number of packets required for the traceback $E[Pkt]$ is then equal to $E[D]$. Therefore, the marginally traceable intrusion with the spoofed SA would be the one which contains up to $E[D]-1$ packets. The untraceable attack must consist of no more than $f \times k - 1$ packets.

6.5.2. DoS attacks

The DoS attacks come from a single source, most likely with spoofed SAs in the attack packets, since the attacker is not interested in the replies from the victim. The number of marks required to be received would be $f \times k$ as described in Section 4. The artificial fragmentation could be used by the attacker to be able to send more packets before the traceback becomes possible. In order for the victim to be able to trace the ingress addresses of the slaves participating in a DoS attack, $C \times E[D]$ packets must be received as mentioned in Section 5. Sending less datagrams than $E[D]$ and $f \times k$ would produce marginally traceable and untraceable attacks, respectively. The respective number of packets would be $C \times (E[D]-1)$ and $C \times (f \times k - 1)$.

6.5.3. Slave-based DDoS attacks

A slave-based DDoS attack can be thought of as a number of DoS attacks toward a single victim executed simultaneously. The expected number of packets required to be able to trace any of the slaves' ingress addresses is $C \times E[D]$, as discussed in Section 6.5.2. Therefore, the marginally traceable slave-based DDoS attack may consist of up to $S \times (C \times (E[D]-1)$ packets. An untraceable attack must consist of no more than $S \times (C \times (f \times k - 1))$ packets because in order for the whole attack to be untraceable, every slave must be untraceable.

6.5.4. Reflector-based DDoS attacks

The reflector-based DDoS attack currently causes the most concern. The reflectors would be identified in the initial stage of the DPM tracing procedure. Identifying the reflectors is not the goal of the traceback, however. The reflectors are just the innocent servers with opened services used by the attacker to generate traffic to the victim. The number of packets that the collection of slaves may send in order to remain marginally traceable or untraceable depends on the fraction of hosts

which perform DPM logging, and would be $\frac{S\times(E[D]-1)}{L}$ and $\frac{S\times(f\times k-1)}{L}$, respectively.

6.5.5. Mixed DDoS attacks

If L is sufficiently large, then using reflectors becomes detrimental to the attacker's goal to have his slaves untraceable. The slave-based attack allows the attacker to send more packets while remaining marginally traceable or untraceable. If, on the other hand, very few hosts on the Internet implement DPM logging, and L is small, then the reflector-based attack allows the attacker to attack the victim with more packets while keeping slaves marginally traceable or untraceable. Note that reflector-based DDoS and slave-based DDoS attacks are both special cases of mixed DDoS attacks. The number of packets that allow the attacker to wage marginally traceable attack is

$$\max\left\{\frac{S\times(E[D]-1)}{L}, S\times C\times(E[D]-1)\right\},$$

and the number of packets for the largest untraceable attack is

$$\max\left\{\frac{S\times(f\times k-1)}{L}, S\times C\times(f\times k-1)\right\}.$$

Two observations can be made. First, we observe that the number of reflectors is irrelevant for traceability of the attack. Notice that it does not appear in any of the expressions. Second, since $C=3$, about 1/3 of hosts on the Internet must have DPM logging enabled so that it becomes detrimental, in terms of the number of packets required for traceability, to the attacker to engage reflectors. Table 4 summarizes the findings of this section.

6.6. Storage requirements

16KByte (2^{17} bits) of storage has to be allocated for every new SA involved in the attack. This number should be doubled since all *RecTbl*s are copied to the *StatTbl* for analysis. If millions of reflectors and slaves are involved in the attack, the storage requirements may be large. This may be an issue if the storage facilities are not properly sized. Currently the storage is a commodity and there are TByte hard drives available commercially [41]. The organizations interested in DPM should plan the storage capacity accordingly.

7. Conclusions and future work

In this article, we have presented DPM – a novel traceback method which is secure, scalable and capable of tracing back most types of attacks. Furthermore, it provides the mechanism to tracing slaves in various DDoS attacks. Unlike other marking-based schemes, DPM can accommodate valid fragmented traffic. Furthermore, unlike other traceback methods, DPM facilitates tracing slaves from reflectors.

As the work on DPM continues, we plan to investigate effects on traceability of lossy conditions on effectiveness of DPM. In particular, during a (D)DoS attack a significant number of packets are lost. Such loss may skew the distribution of marks received by the destination, and therefore may adversely affect the value of $E[Pkt]$. Additionally, DPM is sensitive to router subversion because it is critical that marks, once placed in packets, are not overwritten. We plan to investigate the effects of subversion of one or more routers in the various Internet locations. Finally, it may be particularly useful to integrate the traceback with IDS. Possible ways of such integration and their benefits are planned to be addressed in the future.

Appendix A. Performance analysis

A.1. Single hash function approach analysis

In this section, the number of attackers, N, that the single-digest modification can traceback with

Table 4
Maximum number of packets for marginally traceable and untraceable attacks

Type of the attack	Marginally traceable	Untraceable
Instruction	$E[D]-1$	$f\times k-1$
DoS attack	$C\times(E[D]-1)$	$C\times(f\times k-1)$
Slave-based DDoS	$S\times C\times(E[D]-1)$	$S\times C\times(f\times k-1)$
Reflector-based DDoS	$\frac{S\times C\times(E[D]-1)}{L}$	$\frac{S\times C\times(f\times k-1)}{L}$
Mixed DDoS	$\max\left(\frac{S\times C\times(E[D]-1)}{L}\cdot S\times C\times(E[D]-1)\right)$	$\max\left(\frac{S\times C\times(f\times k-1)}{L}\cdot S\times C\times(f\times k-1)\right)$

the false positive rate limited to 1% is evaluated. Let us examine the origin of false positives. If there is only one ingress address with a given digest, there will be no false positives; however, as N increases, the chance of the digest repeated for another address also increases. The expected number of digests for a certain number of N can be thought of as the expected number of the faces turning up on a 2^d-sided die after N throws. This is a special case of a classical occupancy problem [35]. The expected number of different digests, $E[H]$, is

$$E[H] = 2^d - 2^d\left(1 - \frac{1}{2^d}\right)^N.$$

Therefore, the rate of false positives is 0 for the values of N, for which the expected number of digests, $E[H]$, equals to N, since every ingress address will have a unique digest, assuming $H(\cdot)$ is a good hash function.

Since there may be more than one address having the same digest, each segment associated with a given digest has a certain number of unique patterns. For example, if two addresses have the same digest, segment 0 in the area of the *RecTbl* corresponding to this digest could have either one or two bits set to '1'. If respective segment 0 of these two addresses is the same, then there would be only one bit set to '1', and if segment 0 of the first address is different from segment 0 of the second address, then two bits will be set to '1'. The expected number of unique patterns that a segment assumes can also be thought of as the expected number of the faces turning up on a 2^a-sided die after N_d throws [35], where N_d is the number of ingress addresses having the same digest. The expected number of unique patterns the segment will assume is given by

$$2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{N_d},$$

for those areas, which have segments of more than one ingress addresses, and 1 for those which have segments of only a single ingress address. The expected number of all permutations of address segments for a given digest is

$$\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{N_d}\right]^k.$$

Recall that after a permutation of segments is obtained, the hash function $H(\cdot)$ is applied to the permutation, and if the result does not match the original digest, the permutation is considered to be false. The expected number of permutations that result in a given digest for a given area of the *RecTbl* is

$$\frac{\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{N_d}\right]^k}{2^d}.$$

The number of false positives for a given area would be the total number of permutations, less the number of valid ingress addresses, which match the digest. For this modification, just a few areas, which have segments of more than one ingress addresses, will produce more than $0.01N$ of false positives. We assume that for all those areas $N_d = 2$. The number of such areas is given by $N - E[H]$, and the number of valid ingress addresses with segments in those areas is $2(N - E[H])$. The number of false positives is given by

$$\frac{(N - E[H])\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^2\right]^k - 2(N - E[H])}{2^d}. \quad (1)$$

This number has to be less than 1% of N. Therefore, Eq. (1) has to be set to be less or equal to $0.01N$, and solved for N. Recall that a, d, and $E[H]$ can be expressed in terms of k. Accordingly, the maximum N, N_{MAX}, which would satisfy this inequality, is difficult to express in terms of k. However, it is possible to find N_{MAX} by substitution. Table 5 (which is a copy of Table 1 reproduced here for convenience) provides the values of N_{MAX} for selected k.

Another important consideration is the expected number of datagrams required for reconstruction. This number is related to k, the number of segments into which the ingress address is split. The larger the value of k, the more different packets it would be required for the victim to receive in order to reconstruct the ingress address. The expected number of datagrams, $E[D]$, required to be marked by a single DPM-enabled interface in order for the victim to be able to reconstruct its ingress address is given by the Coupon Collector problem [35]:

Table 5
Relationship between selected k and a, s, d, N_{MAX}, and $E[D]$ for the single-digest modification

k	a	s	d	N_{MAX}	$E[D]$
2	16	1	0	1	3
4	8	2	7	26	9
8	4	3	10	108	22
16	2	4	11	45	55
32	1	5	11	45	130

$$E[D] = k\left(\frac{1}{k} + \frac{1}{k-1} + \cdots + 1\right).$$

Table 5 provides the value of $E[D]$ for selected values of k.

A.2. Multiple hash function approach analysis

The purpose of the analysis of this further modified method remains the same: to determine N_{MAX}, the maximum number of simultaneous attackers that can be traced back with the false positive rate not exceeding 1%. For the multiple digest method, the number of false positives in one area of *RecTbl* may be higher than in a single-digest modification because the same false positive has to appear in the appropriate areas of all other parts of *RecTbl* for a given false positive to be identified as an ingress address.

Recall, from Section A.1, that the expected number of permutations in a given digest is given by

$$\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{N_d}\right]^k,$$

where N_d is the number of ingress addresses having this digest. Since in the multiple digest method, unlike the single-digest method, the number of ingress addresses with the same digest will be more than 2, the following analysis is more suitable. The number of ingress addresses having the same digest is $\frac{N}{E[H]}$. The number of permutations having the same digest is then given by

$$\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{\frac{N}{E[H]}}\right]^k.$$

The number of false positives associated with this digest is

$$\frac{\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{\frac{N}{E[H]}}\right]^k - N}{2^d}.$$

The number of false positives in a single part (for example *Part*0) is given by

$$\frac{E[H]}{2^d}\left(\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{\frac{N}{E[H]}}\right]^k - N\right).$$

For large values of N, $E[H]$ approaches 2^d, and thus $\frac{E[H]}{2^d} = 1$. So the number of false positives in *Part*0 can therefore be approximated by

$$\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{\frac{N}{E[H]}}\right]^k - N.$$

According to the multiple digest method, once the permutation is identified as a possible ingress address in *Part*0, the remaining digests are calculated. Since we assume uniform distribution of addresses, any permutation is as likely to appear as any other. The probability of a random permutation to appear is $\frac{1}{2^{32}}$. The probability that the given permutation, which is a false positive, occurs in the appropriate area of *Part*1 is

$$\frac{\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{\frac{N}{E[H]}}\right]^k}{2^{32}}.$$

Note that this expression is not divided by 2^d because if the permutation in question is present in the identified areas of all other parts, it must match the appropriate digest per discussion at the end of Section 4.3.2. The probability of a given permutation having occurred in the appropriate areas of all parts of *RecTbl* is

$$\left[\frac{\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{\frac{N}{E[H]}}\right]^k}{2^{32}}\right]^{f-1}.$$

Multiplying this expression by the number of false positives in *Part*0 results in the number of false positives, after areas matching the digests 1 through $f-1$ in all remaining parts of the *RecTbl* were checked. This is the total number of false positives for the *RecTbl*. Setting it not to exceed $\frac{N}{100}$ results in the following inequality:

$$\frac{\left\{\left[2^a - 2^a\left(1 - \frac{1}{2^a}\right)^{\frac{N}{E[H]}}\right]^k\right\}^f}{2^{32(f-1)}} \leqslant \frac{N}{100}.$$

Recall once again that a, d, and $E[H]$ can be expressed in terms of k. So, the whole inequality can be expressed in terms of k and f. Similar to the single-digest modification, N_{MAX} can be found by substitution.

The expected number of datagrams required to reconstruct the ingress address is now given by

$$E[D] = f \times k\left(\frac{1}{f \times k} + \frac{1}{f \times k - 1} + \cdots + 1\right).$$

Table 6 (which is a copy of Table 2 reproduced here for convenience) provides the values of N_{MAX} and $E[D]$ for selected combinations of f, a, k, and d.

Table 6
Relationship between f, k and a, s, d, N_{MAX}, and $E[D]$ for selected combinations for multiple digest modification

f	k	a	s	d	N_{MAX}	$E[D]$
4	8	4	3	8	2911	130
4	4	8	2	5	2296	55
8	4	8	2	4	2479	130

As seen from Table 6, the multiple digest modification is able to reconstruct more ingress addresses of simultaneous attackers than a single-digest modification without increasing $E[D]$.

References

[1] P. Ferguson, D. Senie, Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing, RFC 2827, May 2000.

[2] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Network support for IP traceback, IEEE/ACM Trans. Networking 9 (3) (2001) 226–237.

[3] K. Park, H. Lee, On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets, in: Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, August, ACM Press, New York, NY, USA, 2001, pp. 15–26.

[4] C. Morrow, B. Gemberling, How to track a DoS attack (nanog post) [Online]. <http://www.secsup.org/Tracking/>.

[5] D. Moore, G.M. Voelker, S. Savage, Inferring Internet denial of service activity, in: Proc. 10th USENIX Security Symposium, 2001, pp. 9–22.

[6] R.K.C. Chang, Defending against flooding-based distributed denial-of-service attacks: a tutorial, IEEE Commun. Mag. 40 (10) (2002) 42–51.

[7] A. Belenky, N. Ansari, On IP traceback, IEEE Commun. Mag. 41 (7) (2003) 142–153.

[8] S.M. Bellovin, ICMP traceback messages, IETF Draft, March 2000.

[9] S.F. Wu, L. Zhang, D. Massey, A. Mankin, Intention-driven ICMP trace-back, IETF Draft, February 2001 [Online]. <http://www.silicondefense.com/research/itrex/archive/tracing-papers/draft-wu-itrace-intention-00.txt>.

[10] S.F. Wu, L. Zhang, D. Massey, A. Mankin, On design and evaluation of intention-driven ICMP traceback, in: Proc. 10th Inter. Conf. on Computer Comm. and Networks, October 2001, pp. 159–165.

[11] D. Dean, M. Franklin, A. Stubblefield, An algebraic approach to ip traceback, ACM Trans. Inform. Syst. Security (TISSEC) 5 (2) (2002) 119–137.

[12] D.X. Song, A. Perrig, Advanced and authenticated marking schemes for IP traceback, in: Proc. INFOCOM 2001, April 2001, vol. 2, pp. 878–886.

[13] T.W. Doeppner, P.N. Klein, A. Koyfman, Using router stamping to identify the source of IP packets, in: Proc. of 7th ACM Inter. Conf. on Computer Comm. and Networks, November, ACM Press, New York, NY, USA, 2000, pp. 184–189.

[14] Y.K. Tseng, H.H.C.W.S. Hsieh, Probabilistic packet parking with non-preemptive compensation, IEEE Commun. Lett. 8 (6) (2004) 359–361.

[15] A. Yaar, A. Perrig, D. Song, Pi: a path identification mechanism to defend against DDoS attacks, in: Proc. 2003 Symposium on Security and Privacy, 2003, May 2003, pp. 93–107.

[16] A. Yaar, A. Perrig, D. Song, FIT: fast Internet traceback, in: INFOCOM 2005, March 2005, vol. 2, pp. 1395–1406.

[17] M.T. Goodrich, Efficient packet marking for large-scale IP traceback, in: CCS'02: Proc. of the 9th ACM Conf. on Comp. and Comm. Sec., November 2002, pp. 117–126.

[18] Z. Gao, N. Ansari, A practical and robust inter-domain marking scheme for IP traceback, Computer Networks 51 (3) (2007) 732–750.

[19] A.C. Snoeren et al., Single-packet IP traceback, IEEE/ACM Trans. Networking 10 (6) (2002) 721–734.

[20] T. Baba, S. Matsuda, Tracing network attacks to their sources, IEEE Internet Comput. 6 (2) (2002) 20–26.

[21] S. Matsuda, T. Baba, A. Hayakawa, T. Nakamura, Design and implementation of unauthorized access tracing system, in: Proc. of the 2002 Symposium on Applications and the Internet, 2002 (SAINT 2002), January/February 2002, pp. 74–81.

[22] J. Li, M. Sung, J. Xu, L. Li, Large-scale IP traceback in high-speed Internet: practical techniques and theoretical foundation, in: Proc. 2004 IEEE Symposium on Security and Privacy, May 2004, pp. 115–129.

[23] R. Stone, Centertrack: an IP overlay network for tracking DoS floods, in: Proc. of 9th USENIX Security Symposium, August 2000.

[24] H. Chang et al., Deciduous: decentralized source identification for network-based intrusions, in: Proc. of 6th IFIP/IEEE International Symposium on Integrated Net. Management, May 1999, pp. 701–714.

[25] H. Chang et al., Design and implementation of a real-time decentralized source identification system for untrusted ip packets, in: Proc. of the DARPA Information Survivability Conference & Exposition, January 2000, vol. 2, pp. 100–111.

[26] H. Burch, B. Cheswick, Tracing anonymous packets to their approximate source, in: Proc. of 2000 USENIX LISA Conference, December 2000, pp. 319–327.

[27] A. Belenky, N. Ansari, IP traceback with deterministic packet marking, IEEE Commun. Lett. 7 (4) (2003) 162–164.

[28] A. Belenky, IP Traceback with Deterministic Packet Marking (DPM), Ph.D. dissertation, NJIT, Newark, August 2003.

[29] A. Belenky, N. Ansari, Tracing multiple attackers with deterministic packet marking (DPM), in: Proc. of IEEE PacRim, August 2003, vol. 1, pp. 49–52.

[30] C. Shannon, D. Moore, K.C. Claffy, Beyond folklore: observations on fragmented traffic, IEEE/ACM Trans. Networking 10 (6) (2002) 709–720.

[31] J. Mogul, S. Deering, Path MTU discovery, RFC 1191, November 1990.

[32] J. Postel, Internet protocol, RFC 791, September 1981.

[33] D.D. Clark, IP datagram reassembly algorithms, RFC 815, July 1982.

[34] F. Baker, Requirements for IP version 4 routers, RFC 1812, June 1995.

[35] W. Feller, An Introduction to Probability Theory and Its Applications, John Wiley & Sons, Inc., 1968.

[36] R. Braden, Requirements for Internet hosts – communication layers, RFC 1122, October 1989.
[37] S. McCreary, C.K. Claffy, Trends in wide area IP traffic patterns, in: ITC Specialist Seminar, CAIDA, 2000.
[38] S. Bhattacharyya, C. Diot, J. Jetcheva, Pop-level and access-link-level traffic dynamics in a tier-1 POP, in: Proc. of the First ACM SIGCOMM Workshop on Internet Measurement Workshop, ACM Press, New York, NY, USA, 2001, pp. 39–53.
[39] E. Carter, Cisco Secure Intrusion Detection Systems, 1st ed., Cisco Press, Indianapolis, IN, 2001.
[40] V. Paxson, An analysis of using reflectors for distributed denial-of-service attacks, Comput. Commun. Rev. 31 (3) (2001) 38–47.
[41] IBM TotalStorage Product Guide, IBM Systems Group, October 2006 [Online]. <http://www.ibm.com/totalstorage>.

Andrey Belenky received the B.S.Comp.E. degree (summa cum laude) and M.S. in Telecommunication Networks from Polytechnic University, Brooklyn, NY in 1998, He has recently received the Ph.D. degree from New Jersey Institute of Technology (NJIT), Newark, NJ. The main focus of his research was Distributed Denial of Service Attacks and IP Traceback. Prior to receiving the Ph.D. degree, he worked in Telcordia Technologies (formerly Bellcore) as a network engineer.

Nirwan Ansari received the B.S.E.E. degree (summa cum laude) from the New Jersey Institute of Technology (NJIT), Newark, NJ, in 1982, the M.S.E.E. degree from the University of Michigan, Ann Arbor, MI, in 1983, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1988.

He joined the Department of Electrical and Computer Engineering, NJIT, as an Assistant Professor in 1988, and has been a Full Professor since 1997. He is also the Associate Dean for Research and Graduate Studies at the Newark College of Engineering at NJIT. He authored *Computational Intelligence for Optimization* (Kluwer, 1997) with E.S.H. Hou, and co-edited *Neural Networks in Telecommunications* (Kluwer, 1994) with B. Yuhas. He is a Senior Technical Editor of the *IEEE Communications Magazine*, and also serves on the editorial board of *Computer Communications*, the *ETRI Journal*, and the *Journal of Computing and Information Technology*. His current research focuses on various aspects of broadband networks and multimedia communications. He has also contributed 100 refereed journal articles, plus numerous conference papers and book chapters.

He initiated (as the General Chair) the First IEEE International Conference on Information Technology: Research and Education (ITRE2003), was instrumental, while serving as its Chapter Chair, in rejuvenating the North Jersey Chapter of the IEEE Communications Society which received the 1996 Chapter of the Year Award and a 2003 Chapter Achievement Award, served as Chair of the IEEE North Jersey Section and in the IEEE Region 1 Board of Governors during 2001–2002, and has been serving in various IEEE committees such as Vice-Chair of COMSOC Technical Committee on Ad Hoc and Sensor Networks, and (TPC) Chair/Vice-chair of several conferences. He was the 1998 recipient of the NJIT Excellence Teaching Award in Graduate Instruction, and a 1999 IEEE Region 1 Award. He is frequently invited to deliver keynote addresses, tutorials, and talks. He has been selected as an IEEE Communications Society Distinguished Lecturer (2006–2007).