

TRACK: A Novel Approach for Defending Against

Distributed Denial-of-Service Attacks

Ruilian Chen^{*}, Jung-Min Park^{*}, and Randy Marchany[†]

^{*}Bradley Department of Electrical and Computer Engineering

[†]Virginia Tech Information Technology Security Office

Virginia Polytechnic Institute and State University

{rlchen, jungmin, marchany}@vt.edu

Abstract – This paper presents a novel countermeasure against Distributed Denial-of-Service (DDoS) attacks that we call the rouTer poRt mArking and paCkEt filtering (TRACK), which includes the functions of both IP traceback and packet filtering. TRACK is a comprehensive solution that is composed of two components: a router port marking module and a packet filtering module. The former is a novel packet marking scheme for IP traceback and the latter is a novel packet filtering scheme that utilizes the information gathered from the former component. The router port marking scheme marks packets by probabilistically writing a router interface’s port number, a locally unique 6-digit identifier, to the packets it transmits. After collecting the packets marked by each router in an attacking path, a victim machine can use the information contained in those packets to trace the attack back to its source (i.e., solve the “IP traceback” problem). In the packet filtering component, the information contained in the same packets are used to filter the malicious packets at the upstream routers (i.e., routers located in the direction towards the attackers), thus effectively mitigating attacks.

Because very little space is required to mark a port number, TRACK allows us to include attack signature information along with the port number within a single packet’s IP header. The resulting advantage is three fold: (1) a significantly less number of packets need to be collected to traceback the attack source compared to previous IP traceback schemes, (2) very little computation overhead is required in the traceback process, and (3) scalability: a large number of attackers (i.e., zombies) can be traced back efficiently. Because TRACK uses the router interface instead of the entire router as the “atomic unit” for IP traceback and packet filtering, it can accomplish these tasks with much finer granularity, which helps to lower the false positives. In the paper, we also show that TRACK supports gradual deployment .

Index terms – Distributed Denial-of-Service, IP Traceback, Packet Filtering, Network Security.

I. INTRODUCTION

As the Internet’s scale and complexity continue to grow, the lack of security mechanisms during its early deployment years has led to serious problems today. In the past few years, many forms of denial-of-service (DoS) and malware attacks have been documented and brought to our attention through the news media. Among such attacks, a distributed DoS (DDoS) attack is a particularly menacing threat that is very difficult to defend against. In a DDoS attack, the attacker gradually gains control over a large number of unsecured hosts, which are called *zombies*, as a prelude to the actual attack. The attacker then uses these zombies to synchronize attacks on a victim machine and overwhelms it by flooding it with packets. For this reason, DDoS attacks are sometimes referred to as “flooding” attacks.

Recognizing the importance of the problem, the research community has been making a great effort to find solutions for combating DDoS attacks. Unfortunately, finding effective solutions is a very challenging task. This is due to several reasons. First, the Internet is an open platform, and its value lies in its availability to everyone from everywhere. Integrating security mechanisms into such an open platform is very difficult. Second, in a DDoS attack, the number of zombie machines involved in an attack can reach several hundred or even several thousand. Mitigating the effects of such an attack is a daunting task. Third, IP source addresses are often forged (i.e., “IP spoofing”) to amplify DDoS attacks and hide the actual attack source. Although ingress filtering [Ferg98] is being deployed in many networks to prevent IP spoofing, there still exists a large number of subnets that have not implemented it, thus making IP spoofing still possible.

Because the network (or IP) layer is essentially responsible for forwarding DDoS attack traffic, most solutions to date have concentrated on making the IP layer more secure. These solutions can be divided into two categories: *IP traceback* and *attack mitigation*. IP

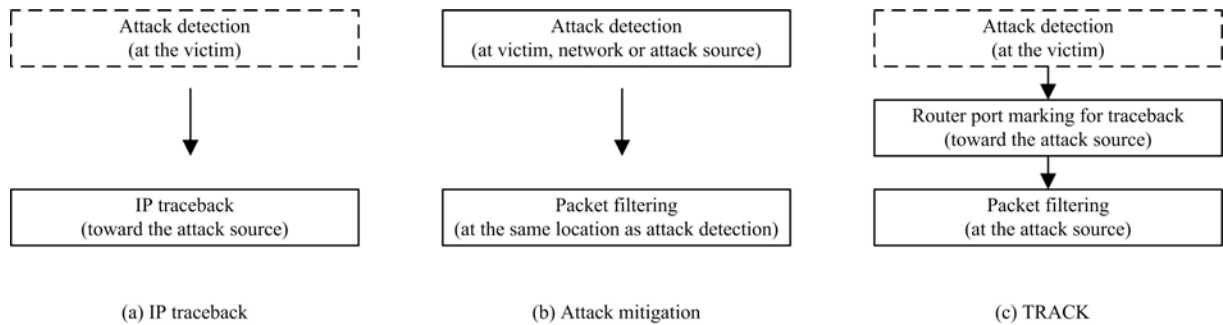


Figure 1: Conceptual Architectures for IP traceback, attack mitigation, and TRACK.

traceback techniques introduce a mechanism of storing routing path information of the packets that can later be used to trace the attack back to its source. Depending on where such information is stored, IP traceback can be classified into two categories: *probabilistic packet marking* [Dean02, Good02, Sava00, Song01] and *packet logging* [Li04, Snoe01]. In the former, the path information is stored in packets while in the latter, the information is stored in routers.

Figure 1(a) shows a simplified architecture of a typical IP traceback scheme. IP traceback schemes typically assume that there is an attack detection module installed in the victim that can recognize attacks and differentiate malicious packets from legitimate ones. Its core traceback module is responsible for tracing back the attacks to their sources (i.e., zombies). In the vast majority of packet marking schemes for traceback, packets are marked with the hash value of a router’s IP address. In other words, a router is used as the “atomic unit” of traceback. This approach ignores the fact that a router commonly has multiple interfaces. Our solution, TRACK, takes a totally different approach by treating each router’s interface as the atomic unit of traceback. There are numerous advantages to our approach, which we will discuss in detail later.

Attack mitigation techniques, compared to IP traceback, take a more active approach in thwarting DDoS attacks. In a typical attack mitigation scheme, a set of strategies are adopted to drop packets that are believed to be malicious (this is also known as *packet filtering*). Based on where the packets are filtered, mitigation schemes can be classified as victim based [Sung03, Thom03, Yaar03, Yaar04], network based [Maha02, Zhan03], or attack-source based [Mirk02, Papa03]. Attack mitigation schemes usually employ an attack detection module that extracts the characteristics of attacking packets, such as source IP addresses or marked IP header values [Sung03, Yaar03, Yaar04]. After the characteristics have been extracted, this information is used by the packet filtering module to filter out malicious packets. Figure 1(b) shows the simplified architecture of typical attack mitigation schemes. As

one can see, a typical system is composed of two modules that carry out the tasks of attack detection and packet filtering. Conventional schemes place the two modules in one of three possible locations: at the victim end (or victim network), in the network, or near the attack source (or attack source network). More importantly, the two modules are typically placed at the same location. Unfortunately, placing them in the same location is not desirable for optimal effectiveness. It is easy to see that attack detection can be most effectively done at the victim end where all the attack packets can be observed readily. In contrast, it is most effective to filter attack packets as close to the attack sources as possible. Note that in flooding attacks, not only do the malicious packets inundate the victim, but they also congest the victim’s neighborhood network [Maha02]. If one considers the above facts, it should be obvious that the optimal approach is to place the attack detection module at (or near) the victim and place the packet filtering module as close to the attack source as possible. For more details on this ideal paradigm for DDoS attack mitigation, see [Chan02].

In this paper, we propose a novel approach for combating DDoS attacks that we call *TRACK (rouTer poRt mArking and paCKet filtering)*. Figure 1(c) shows its architecture. Like previous IP traceback schemes, we assume the existence of an attack detection module located at the victim end. The router port marking traceback module of TRACK executes the functions of IP traceback. TRACK-enabled routers write router port numbers—locally unique router interface identifiers—to IP packets, which are then collected by the victim. Here, locally unique means that a port number is only unique within a single router. Information gathered from those packets can be used to reconstruct the path traversed by the malicious packets. The path information can be used not only for IP traceback but also for filtering attack traffic near its source. The packet filtering module, the second component of TRACK, filters attack traffic using the information generated in the router port marking module.

The major contribution of this paper is twofold: (1)

TRACK’s architecture essentially follows the ideal paradigm described in [Chan02] by separating the functions of attack detection and packet filtering, and executing them at optimal locations. (2) To the best of our knowledge, our traceback scheme is the first one that uses the router interface, instead of the router itself, as the atomic unit of traceback.¹ This approach has two noteworthy advantages over conventional techniques: reduction of traceback computation complexity and the capability to execute “fine-grained” filtering, which we will discuss in detail in the following paragraphs.

The maximum number of interfaces that a router has is much smaller than the number of possible IP addresses. This means that far fewer bits are required to mark the traceback information on the packets. A common strategy used by previous schemes is to fragment an IP address (of a router) and other information into multiple parts and mark each individual part into a packet. Hash functions are often used in the fragmentation process. This is necessary because there is not enough room in the IPv4 header to fit all the traceback information, including the 32-bit address. The task of reassembling the original IP address is nontrivial. In fact, the computation cost involved in the reassembly process limits the number of attack sources that can be traced back—that is, the computation complexity rapidly increases as the number of attackers grows. This drawback is shared by the scheme proposed in [Sava00]. For example, the required traceback complexity in [Sava00] is $O(N_a^8)$ hash operations, where N_a is the number of attackers.

Our approach avoids the problems associated with reassembly by marking all traceback information, including the port number, in a single packet (or more precisely, in a single packet’s IP header). This is possible because only a few bits (six to be precise) are needed to mark a port number. Moreover, this approach has another advantage: since one packet is enough to carry all the information, TRACK needs to collect fewer packets than previous schemes.

Treating the router interface as the atomic unit of traceback also enables fine-grained filtering when attack traffic needs to be throttled. If the router is the atomic unit of traceback, all packets passing through a router must be throttled once the attack path has been traced to that router. In this approach, one cannot avoid throttling non-attack traffic passing through the same router. Such a problem can be avoided when the router interface is the atomic unit of traceback. TRACK-enabled routers have the capability to throttle traffic passing through

individual interfaces, thus making it unnecessary to throttle non-attack packets passing through other interfaces.

The remainder of this paper is organized as follows. Section II describes the router port marking and packet filtering scheme in detail. In Section III, we discuss issues related to gradual deployment. Practical considerations are discussed in Section IV, and we show our simulation results in Section V. An overview of related work is presented in Section VI. Finally, we conclude the paper and discuss future work in Section VII.

II. TRACK: THE ROUTER PORT MARKING AND PACKET FILTERING SCHEME

A. Terminology and Assumptions

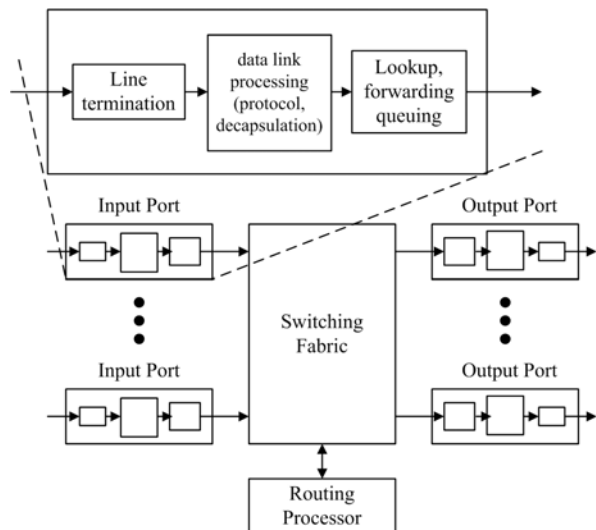


Figure 2: The typical structure of a commercial router.

We assume the following network environment. Every host, whether a client or a server, is connected to its local *border router*. Border routers are interconnected by *core routers*. A malicious client is called an *attacker*, and the server being attacked is the *victim*. Note that we use the term “attacker” to denote a node (or the person operating that machine) in which the attack packets are originating from. In DDoS attacks, these nodes are often zombies rather than the real attacker(s). In this paper, we do not address the problem of identifying the real attacker(s) controlling the zombies. It is assumed that routers cannot be compromised by attackers.

One study [Paxs97] has shown that the majority of Internet routes have fixed paths for that last up to several days. Furthermore, a recent study [Jin03] presented results showing that 95% of the routes observed had fewer than five observable daily changes.

¹ Belenky and Ansari [Bele02] proposed a similar idea, but their scheme made very limited use of this idea. To the best of our knowledge, the traceback module of TRACK is the first scheme to fully utilize the idea of using router interfaces as the atomic unit of traceback to execute a more “fine-grained” traceback.

Based on those findings, we make the reasonable assumption that every route from a client to a server is *stable* within our timeframe of interest. The stable route that a client uses to transmit packets to the destination is called the client’s *path*. Any hop or router that is on an attacker’s path is considered to be *infected*.

We will use the term *false negative* to denote an attacker that has escaped identification or filtering, and use the term *false positive* to represent a non-malicious (or “legitimate”) client that has been incorrectly identified as an attacker or has been throttled.

Table 1: List of terms used in the paper.

Terms	Descriptions
Attack path	A path that an attacker uses to attack a victim
Attacker	Source of the attack packets (i.e., a zombie)
False negative	An attacker that is incorrectly identified as a legitimate client
False positive	A legitimate client that is incorrectly identified as an attacker
Infected hop/router	When a hop/router is on an attack path
Legacy router	A router not supporting TRACK
Path	The stable route that a client uses to send packets to the destination
Port	The hardware interface of a router
Updated router	A TRACK-enabled router
Victim	The target of the DDoS attack

Our scheme requires that a few functions be added to Internet routers to make them TRACK-enabled. TRACK-enabled routers are called *updated routers*, and routers without such functions are called *legacy routers*. An updated router needs to support “port number marking”. The term *port* used in this paper represents a router’s hardware interface. We assume that every port is bidirectional, i.e., any input port is also an output port and vice versa. A router allocates a number to every port and can forward packets to a specified port number. Note that this number should not be confused with port numbers used to identify network applications. The port number is assigned in advance and locally unique. For illustration, Figure 2 shows the structure of a typical commercial router, which has multiple input ports and output ports connected by the switching fabric [Kuro04]. Note that every port can be either input or output port depending on the traffic direction. The routing processor computes the forwarding table that controls the switching logic. For many routers, the input port stores a shadow copy of the forwarding table so that most of the packet processing work is done there in a distributed fashion, instead of being concentrated at the processor in the router. We assume that the input port of an updated router can mark its port number to incoming packets and execute exclusive OR operations on certain fields of the packets’ IP headers. Given the fact that a conventional router routinely handles tasks such as

updating the *TTL* (Time-To-Live) and recomputing the checksum, the additional functions described above should be within the capabilities of most routers.

We assume the existence of an attack detection module installed at the victim, which is able to distinguish between legitimate and malicious packets. Note that previous IP traceback schemes have made similar assumptions. The terms that are used throughout the paper are summarized in Table 1.

B. Router Port Marking

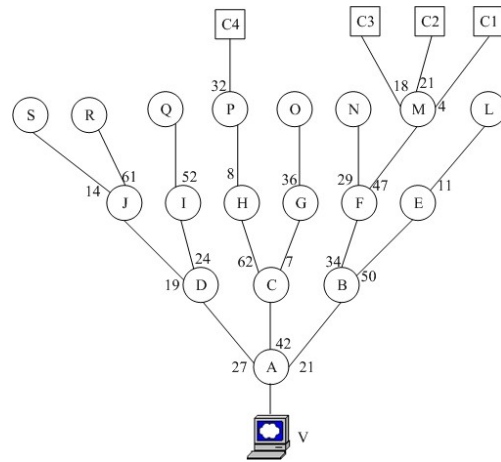


Figure 3: The upstream tree of victim (or server) V.

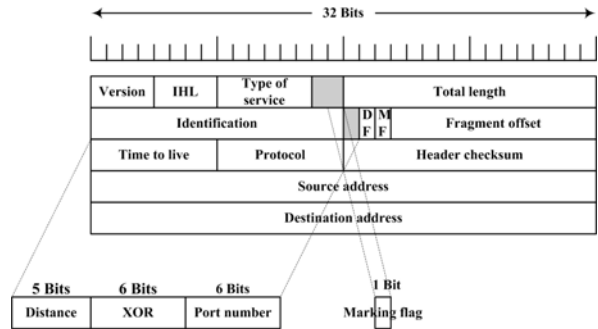


Figure 4: The marked fields in IP header

In the broader sense, our IP traceback module is similar to previous research, viz. the packet marking phase and the path reconstruction phase. But our approach in the two phases is fundamentally different from them.

The key difference between conventional IP traceback schemes and TRACK’s IP traceback method lies in the way that the two schemes uniquely identify an attack path. In the former, IP addresses are used, since they are globally unique identifiers. We take an entirely different approach. The basic principle of our scheme is very

straightforward: a string composed of locally-unique router input port numbers is a globally unique identifier of a path. Figure 3 shows an example. In the figure, an upstream tree of server V is shown. $C1$, $C2$ and $C3$ are three clients sharing the same border router M . The numbers on the links represent input port numbers. Suppose $C3$'s path to server V is the route $C3-M-F-B-A$. Then the string of port numbers 21-34-47-18 (starting from V) can be the unique identifier of $C3$ and its path². If $C3$ launches an attack on V , this information can be used by V to trace back the source of the attack.

In our router port marking scheme, we allocate six bits to the router port number so that it can be any number from 0 to 63. Besides the requirement of local uniqueness, the port numbers should also be allocated randomly to minimize collisions that may result in false positives. More discussions on this issue is given in Section II.C. Figure 4 shows the marking fields that are used in the scheme: *Distance*–5 bits, *XOR*–6 bits, *Port Number (PN)*–6 bits and *Marking Flag (MKF)*–1 bit. These fields replace the 16-bit Identification field, a reserved bit in the following field, and another reserved bit in the ToS (Type of Service) field. In Section IV, we give the rationales for why those fields were selected. The *Distance* field is used for recording hop-count related information. Its length is five bits because the current Internet has very few routes that span more than 32 hops [Jin03].

In our packet marking scheme, a router's input port can work in one of two modes: *Active Probabilistic Marking Mode (APMM)* and *Passive Probabilistic Marking Mode (PPMM)*. A router is said to be in the (APMM) when it actively marks a packet. This means that the router:

1. Sets the packet's *MKF* field to 1;
2. Copies the least significant five digits of the *TTL* field to the distance field, and;
3. Copies the port number to both the port field and the *XOR* field.

When a router port is in PPMM and a packet's *MKF* field is not set to 1, then the port does nothing to the packet. However, if the packet's *MKF* field is set to 1, then the port computes the XOR (exclusive OR) of the port number and the value in the packet's *XOR* field, and write the result back in the *XOR* field. We assume that a router's port is in the APMM with a probability of p and in the PPMM with a probability of $(1 - p)$.

Note that the information marked by an upstream router

² We are aware that some clients are connected to a border router via a shared medium (or subnet), such as a LAN, instead of distinct ports. In that case, $C1$, $C2$ and $C3$ may be connected to the same port of M . Then the string of port numbers can be used to trace back up to M , but not any further than that. In section IV, we discuss possible solutions to this problem.

can be overwritten by downstream routers. When the victim receives a packet, its fields reveal the following information: the *MKF*, *Distance*, and *PN* fields were marked by the nearest upstream router that was in APMM when processing the packet, and the *XOR* field value is the XOR value taken over all the routers' port numbers from that nearest upstream router to V 's border router.

We illustrate our port marking scheme with an example. Again let us refer to $C3$'s path in Figure 3. Suppose an attacker $C3$ sends a packet to server V with an initial *TTL* set to 255. Moreover, suppose that M is in APMM, and F , B , and A are all in PPMM when forwarding the packet. With these assumptions, we can see that M will set *MKF* to 1, copy the least significant five digits of the packet's current *TTL* at M (which is 254) to the *Distance* field, and copy the input port number 18 (010010) to the *XOR* and *PN* fields. On the other hand, F , B , and A will update the *XOR* field only and not modify the other fields. For example, F will compute the XOR of its port number (47) and the packet's *XOR* field value and write the result in the *XOR* field. When V receives the packet, it first checks to see whether the *MKF* is set to 1. The packet reveals the following information: *Distance* field is 30 (which is the least significant five bits of 254), the last 5 digit of *TTL* is 27, *XOR* field value is $2(=18 \oplus 47 \oplus 34 \oplus 21)$, and the *PN* value is 18. Since the difference between the *Distance* field value and the least significant five digits of *TTL* is three, V can infer that the packet was actively marked by a router three hops away. Hence, V records the *XOR* and *PN* field values associated with hop count $d = 3$. (In other words, hop count d is computed as $d = Distance - TTL_5$, where TTL_5 denotes the least significant five bits of *TTL*.) After a sufficient amount of time, V will eventually receive packets marked by other routers in APMM (such as F , B or A). After accumulating a sufficient number of these packets, V can organize the information contained in these packets into a table sorted by the hop count. We call such a table a *trace table*. A trace table showing $C3$'s attack path is shown in Table 2.

Table 2: A trace table containing $C3$'s path

Router closest to V in APMM *	Hop Count : d	Port Number: $PN(d)$	XOR: $XOR(d)$
A	0	21 [010101]**	21 [010101]
B	1	34 [100010]	55 [110111] ($\oplus 34 = 21$)
F	2	47 [100111]	16 [010000] ($\oplus 47 = 55$)
M	3	18 [010010]	02 [000010] ($\oplus 18 = 16$)

* This column is for illustration purposes. This information is not directly revealed by the packets collected.

**The binary strings inside [...] represent the binary equivalents.

Using this trace table, V needs to first group the records (i.e., rows) based on the paths they belong to, beginning with hop 0. The grouping is done by verifying whether

for any two consecutive hop count records satisfy

$$XOR(d+1) \oplus PN(d+1) = XOR(d), \quad (1)$$

where the numbers inside () represent the hop count. If the above equation is satisfied, then the two rows are grouped together and associated with the same path. This process is repeated for all the rows in a sequential manner. It is easy to see that in this example, the four records can be grouped to the same path. This means that the corresponding port number sequence 21-34-47-18 is recovered, which represents $C3$'s path.

This method can also be used to differentiate multiple paths. Suppose that another attacker $C4$ sends packets to V via $C4-P-H-C-A-V$ in addition to $C3$ (see Figure 3). The resulting trace table that V would create is shown in Table 3. The first thing that V notices is the fact that there are two hop 0 records, which implies that there are at least two attack paths. The two attack paths can be separated by iteratively verifying (1) starting with $d=0$ and incrementing d by one for each iteration. The verification is done for every row. The string of PN values of the records that satisfy (1) is the desired port number sequence. For example, in Table 3, the four records in the shaded rows represent $C3$'s path 21-34-47-18. Similarly, the non-shaded records represent $C4$'s path 42-62-08-32.

Table 3: A trace table containing $C3$'s and $C4$'s paths

Router closest to V in APMM	Hop Count: d	Port Number: $PN(d)$	XOR: $XOR(d)$
A	0	21 [010101]	21 [010101]
A	0	42 [101010]	42 [101010]
B	1	34 [100010]	55 [110111] ($\oplus 34 = 21$)
C	1	62 [111110]	20 [010100] ($\oplus 62 = 42$)
F	2	47 [100111]	16 [010000] ($\oplus 47 = 55$)
H	2	08 [001000]	28 [011100] ($\oplus 08 = 20$)
M	3	18 [010010]	02 [000010] ($\oplus 18 = 16$)
P	3	32 [100000]	60 [111100] ($\oplus 32 = 28$)

The maximum number of records in a trace table, defined as N , is a function of the number of attackers N_a and their hop counts away from the victim. If we denote H_i as the hop count of the i -th attacker from a victim's border router, then N 's upper bound can be computed as:

$$N = \sum_{i=1}^{N_a} H_i \quad (2)$$

Because multiple attack paths can share the same upstream links with the same XOR value, the actual number of records in a trace table will be less than (2). Since it is most likely that H_i is less than 32 [Jin03], we can assume that N is less than $32N_a$, which is linear to the number of attackers.

In all probabilistic packet marking schemes, the total

number of packets that need to be collected for traceback is dependent on many factors, including: the number of attackers, the number of hops from the victim to the attackers, the number of packets needed to uniquely identify the atomic unit of traceback, and the active marking probability of all routers. Our port marking scheme requires far less number of packets than previous schemes because it only requires one packet to uniquely identify a router's interface, which is the atomic unit of traceback used by our scheme. An in-depth analysis about the number of packets needed for IP traceback is given in [Park01].

C. The Path Reconstruction Process

At this point, V knows the port number sequence, but does not know the actual physical path corresponding to that sequence. Therefore, a path reconstruction process is needed. In the path reconstruction process, V needs to send a few query packets to the upstream routers that contain collected marking information. This is done hop by hop in a manner similar to "pushback" [Ioan02].

Although V is capable of recovering the port number sequences of the paths using the trace table, we propose to distribute this task to the upstream routers to improve the accuracy of traceback. In this section, we describe a path reconstruction process in which the routers recover the port number sequences in a distributive manner. Such a task is well within the capabilities of most routers since recovering the port number sequences only requires XOR operations and since the path reconstruction process is expected to be executed infrequently.

To start the path reconstruction process, the victim V first needs to create a trace table and then include it in a "query packet" that is sent to V 's border router. We assume that V creates a trace table of multiple rows in increasing order of d , where each row has three fields: d ($0 \leq d \leq 31$), $PN(d)$, and $XOR(d)$. One record (i.e., row) is 17 bits in length, but for convenience, suppose that three bytes are allocated for it. Since a single IP packet can hold up to 65516 bytes, we can include up to 21838 records per query packet ($65516/3 = 21838$). Here, 65516 bytes is the size of the maximum allowed payload of a TCP segment inside an IP packet. Since at most 32 records are needed to trace back one attacker, one packet can hold the traceback information of at least 682 attackers ($21838/32 = 682$). Recall that we have assumed that no path in the network is longer than 32 hops. If there are more than 21838 records that need to be sent, then multiple query packets are used to transmit a trace table. For example, to trace back a 10000-attacker DDoS attack, fifteen packets are enough for the trace table. For the convenience of our discussions, we assume that one query packet is sufficient to hold the trace table. After the victim creates

the query packet (in which $TTL=1$), it is sent to the 0-hop count router (i.e., V 's border router). The 0-hop router processes the query packet and forwards it to the 1-hop router (i.e., the next router in the attack path in the direction of the attacker), which processes the packet and forwards it to the 2-hop router and so on. This process-and-forward procedure is repeated until the query packet reaches the *attacker's* border router. This router processes the query packet further and then returns it to V . In the following paragraph, we describe in detail how the query packet is handled in the process-and-forward procedure.

```

The victim  $V$  sends the trace table  $T$  ( $d_m = 0$ ) in a query packet to its
border router;
FOR EACH router  $R$  that receives  $T$  {
  IF ( $R$  is a border router ) and ( $PN_i(d_m)$  is a valid port of  $R$ ) {
     $R$  attaches its IP address to  $T$ ;
     $R$  sends  $T$  back to  $V$  and drops  $T$ ;
    return();          // reconstruction process is finished
  }
  ELSE {
    FOR  $i = 1$  to  $n_{d_m}$  {          // going through each record in  $T$ 
                                // with hop count being  $d_m$ 
      IF ( $PN_i(d_m)$  is a valid port of  $R$ )
        AND (there exists hop  $d_m+1$  records with
               $XOR_j(d_m+1) \oplus PN_j(d_m+1) = XOR_i(d_m)$ ) {
           $T' = T$ ;
           $R$  truncates records of  $T'$  where ( $(d = d_m)$  OR
            ( $d = d_m + 1$ ) AND
            ( $XOR_j(d_m+1) \oplus PN_j(d_m+1) \neq XOR_i(d_m)$ ));
          // now in  $T'$ ,  $d_m = d_m + 1$ 
           $R$  attaches its IP address to  $T'$ ;
           $R$  sends updated  $T'$  as  $T$  to the neighbor router
            connected to the port numbered  $PN_i(d_m)$ ;
        }
      }
    }
     $R$  drops  $T$ ;
  }
}
    
```

Figure 5: The pseudo-code of distributed path reconstruction process

When a router receives a query packet, it first scans the trace table (inside it) and reads the least hop count value d_m in the table. We denote n_{d_m} as the number of records with hop count $d = d_m$. Then the router goes through each hop count $d = d_m$ record to check if *both* of the following two conditions hold:

1. $PN_i(d_m)$ ($1 \leq i \leq n_{d_m}$) is a valid local port;
2. There exists some hop count d_m+1 record(s) that satisfies $XOR_j(d_m+1) \oplus PN_j(d_m+1) = XOR_i(d_m)$ ($1 \leq i \leq n_{d_m}$, $1 \leq j \leq n_{d_m+1}$).

If the conditions are not satisfied, the router repeats the same procedure with the hop count $d_m + 1$ record. Note that the router must repeat the procedure for *all* records corresponding to hop count d_m before moving on to hop count $d_m + 1$. The router continues on until a record that satisfies the two conditions are found. When the record is found, the router makes a copy of the trace table and deletes a portion of the trace table from the query packet.

The rows that are deleted include all rows that correspond to hop count d_m and rows that correspond to hop count d_m+1 in which $XOR_j(d_m+1) \oplus PN_j(d_m+1) \neq XOR_i(d_m)$ is satisfied. Next, the router attaches its own IP address to the end of the truncated query packet. Finally, the modified query packet's TTL is set to 1, and is sent out to the port identified by $PN_i(d_m)$.

The procedure described above is repeated in all of the routers along the attack path until the query packet reaches the attacker's border router. This router checks whether port $PN_i(d_m)$ actually exists (Here, d_m denotes the least hop count value of the truncated table.) and then attaches its IP address to the query packet before sending it back to the victim. The packet that the victim receives contains the IP addresses of the routers along the attack path. This concludes the path reconstruction process, which is described as a pseudo-code in Figure 5.

The computation complexity of the path reconstruction process is $O(d \cdot N_a^2)$, where d (≤ 32) is the assumed maximum hop count from the attackers to the victim. The worst case occurs when there are N_a records with hop count d_m and N_a records with hop count d_m+1 in the trace table, and therefore every pair records need a XOR operation, resulting a total of N_a^2 operations. Since the computation is distributed among the upstream routers, the actual complexity at each router is upper bounded by N_a^2 6-bit XOR operations. If we assume that there are 10,000 attackers and that a 6-bit XOR operation takes one clock cycle to execute, then it will take a router with a 1GHz processor as little as 0.1 sec. to finish its share of the path reconstruction computation. Therefore, TRACK is scalable in terms of computation complexity of path reconstruction.

D. Analysis of False Negative and False Positive

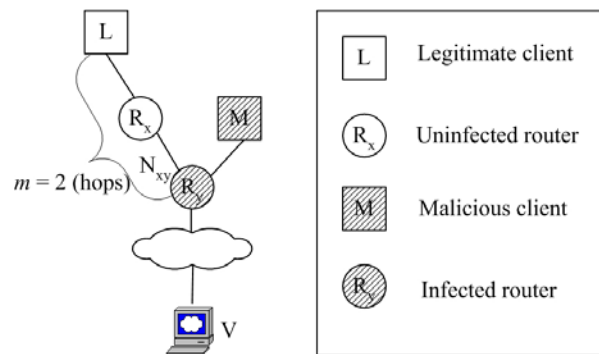


Figure 6: Illustration of false positive analysis

Based on the previous discussions, we can infer that if the attack packets are all collected and if their information is included in the trace table, all attack paths can be traced back. In other words, there are no

false negatives (see the Appendix for a proof). Unfortunately, the same cannot be said of false positives. We use Figure 6 for illustration. In the figure, L is a legitimate client, M is a malicious client, R_y is an infected router, and R_x is an uninfected router. Since the route reconstruction process is processed in a hop-by-hop manner, for L to be a false positive, R_x has to be recognized as an infected router and receive a query packet containing the trace table. Since R_y is an infected router, it must receive the trace table. R_y will forward trace table to R_x only if there exists a record in the trace table with PN value equal to N_{xy} . Meanwhile, the fact that this record is still in the table implies that when R_y 's immediate downstream router processes the trace table, (1) holds for this record. Therefore, for R_x to be identified as an infected router, the hop count d , PN and XOR field values actively marked by N_{xy} must collide with a record in the trace table, which was generated by an attacker. For attackers to maximize the probability of creating false positives, they must be far away from the victim as possible (in terms of hops) and generate packets that cause the maximum number of unique combinations of PN and XOR field values per hop count in the trace table. Since 12-bits are allocated for these two fields, there are $2^{12} = 4096$ combinations that are possible per hop count. This is why randomly allocating port numbers using uniform distribution can reduce false positives. When port numbers are uniformly distributed, the type of collisions described above is less likely to occur.

Based on the above observation, we can derive the probability of an arbitrary uninfected router having an immediate downstream router that is recognized as an infected router. If we assume that all attack paths come from 32-hops away, then this probability is:

$$P_r = \frac{\sum_{k=1}^{\min(N_a, S_b)} k \binom{S_b}{k} \left(\frac{1}{S_b}\right)^{N_a} M(k)}{S_b} \quad (3)$$

where $S_b = 4096$ is the number of 12-bit combinations, N_a denotes the number of attackers (thus the number of their attack paths), and

$$M(k) = \begin{cases} 1, & k = 1, \\ k^{N_a} - \sum_{j=1}^{k-1} \binom{k}{j} M(j), & 2 \leq k \leq 4096. \end{cases} \quad (4)$$

A proof for (3) can be found in the Appendix. With the same reasoning, we can see that if L 's border router R_x is recognized as infected, then the false positive probability of a legitimate client L can also be expressed by (3). Since different routers allocate port numbers independently, if a legitimate client's path has m hops that are not infected, then its false positive probability will be given by:

$$P_{FP} = P_r^m \quad (5)$$

The curve shape of (3) converges very fast to N_a/S_b as S_b

increases. Figure 7 shows the approximation of $P_{FP}(N_a)$ that is generated by scaling the function with $S_b = 50$.

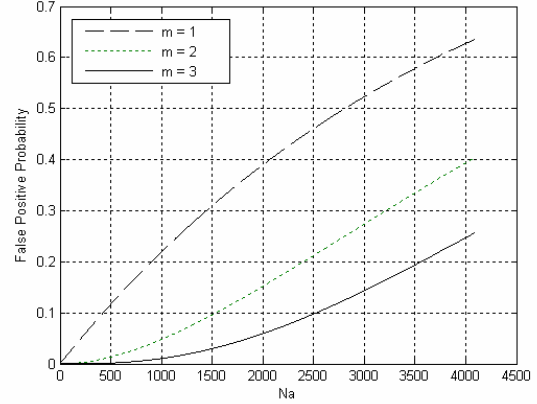


Figure 7: The expected false positive of router port marking scheme

E. Practical Concerns and Enhancement

The router port marking scheme that we have described in Section II.B allocates six bits for representing a port number. We call that scheme the *basic port marking* scheme. In practice, the degree of a router, which is the total number of all working interfaces of the router, may exceed what can be represented by six bits (i.e., $2^6 = 64$). In the Skitter project [Skit04], a statistical study of the Internet topology was carried out. In the study, 192,244 routers were traced. According to the study, the average and maximum router degrees were found to be 6.34 and 1071, respectively. The Skitter project also reveals that 98.5% of the traced routers have degrees less than or equal to 64. This implies that our basic port marking scheme cannot be implemented in a small percentage of the routers. To support routers with more than 64 interfaces, here we propose an extend version of our basic port marking scheme that we call the *advanced port marking* scheme.

The advanced port marking scheme uses twelve bits for port number allocation, which can represent up to 4,096 ports. This number is large enough to support all routers, including today's most advanced routers which have a large number of interfaces. For instance, Cisco 12816, the first terabit router, supports a maximum of 1,740 ports [Cisc04], which can be supported by the advanced port marking scheme.

In the advanced port marking scheme, a router separates the 12-bit port number into two 6-bit numbers and marks a given packet using those two numbers. In the marking process, the router acts like two "virtual" routers marking the same packet independently. Each virtual router marks the 6-bit number on the packet using the basic port marking scheme. The decisions to mark the two numbers are independent and both happen

with probability p . Let us denote the four marking fields of an incoming packet Pkt as $Pkt.MKF$, $Pkt.Distance$, $Pkt.XOR$, and $Pkt.PN$. Also, let the port number's least significant six bits and the most significant six bits be represented by p_1 and p_2 , respectively. The pseudocode for the advanced port marking scheme is given in Figure 8. We assume that the least significant six bits are marked first. Note that the least significant six bits are not actively marked with a probability p , but rather with a probability of $p(1-p)$, because those bits can be overwritten by the most significant six bits that are marked actively later with a probability of p .

```

FOR EACH packet  $Pkt$ {
    select random number  $r$  from  $[0,1)$ ;
    IF ( $r < p(1-p)$ ) { // at a probability of  $p(1-p)$  to enter
         $Pkt.MKF = 1$ ;
         $Pkt.Distance = (Pkt.TTL - 1) \bmod 32$ ;
         $Pkt.XOR = Pkt.XOR \oplus p_1 \oplus p_2$ ;
         $Pkt.PN = p_1$ ;
    }
    ELSEIF ( $r < p(2-p)$ ) { // at a probability of
         $Pkt.MKF = 1$ ; //  $p(2-p) - p(1-p) = p$  to enter
         $Pkt.Distance = (Pkt.TTL - 2) \bmod 32$ ;
         $Pkt.XOR = Pkt.XOR \oplus p_2$ ;
         $Pkt.PN = p_2$ ;
    }
    ELSEIF ( $Pkt.MKF == 1$ ) {
         $Pkt.XOR = Pkt.XOR \oplus p_1 \oplus p_2$ ;
    }
     $Pkt.TTL = Pkt.TTL - 2$ ;
}
    
```

Figure 8: Packet marking process for routers with degree greater than 64

F. The Packet Filtering Scheme

As mentioned previously, in addition to the traceback component, TRACK also includes a packet filtering component. In this section, we describe how the packet filtering module mitigates DDoS attacks by using the information gathered by the traceback module.

In the path reconstruction process that we discussed in Section II.C, the query packets are pushed back on to upstream routers, in a manner similar to pushback [Ioan02], so that the attack path can be traced using the information written on those query packets. The same query packets can also be used to effectively throttle attack traffic. When the border router of an attacker receives a query packet, it can immediately start throttling attack packets by dropping packets that are destined for the victim and received from “infected” input port(s). In essence, we are killing two birds with one stone—the same query packets that are used for traceback serve as the control packets to throttle malicious traffic.

One can readily see that the effectiveness of the packet filtering scheme is entirely dependent on the accuracy

of the IP traceback module. Therefore, the packet filtering scheme incurs no false negatives and incurs a false positive probability expressed by (5).

III. CONSIDERATION ABOUT GRADUAL DEPLOYMENT

In the above discussions, we have implicitly assumed that all the routers are capable of port marking and is able to forward packets to a specified port. However, it is impractical to assume that any scheme can be implemented simultaneously in all the routers. The coexistence of updated routers and legacy routers must be considered. Recognizing this problem, in this section, we introduce techniques to gradually deploy TRACK.

A. The Neighbor-Discovery Handshake Protocol

One method to support gradual deployment is to require an updated router to maintain a record of all updated routers in its neighborhood so that in the path reconstruction process, a query packet can be sent to those routers. If we assume that updated routers are deployed in the Internet so that every H -hop path has at least one such router, then updated routers can learn their H -hop neighbors cooperatively with the following handshake protocol. When an updated router $A1$ forwards packets, it randomly extracts the destination address from them and sends *Hello* packets to those destination addresses with *TTL* field set to 1. This *Hello* packet serves as a request for a handshake procedure. If $A1$'s immediate router $A2$ that receives this packet is an updated router, then $A2$ will reply with a *Hello-Response* packet to $A1$ with *TTL* = 1. After receiving the *Hello-Response* packet, $A1$ associates the port that received this packet with $A2$ and includes the port number and $A2$'s IP address in one of the 1-hop entries of its *updated router list*. Then, $A1$ sends a *Hello-Confirm* packet to $A2$ so that $A2$ can include $A1$ in one of its 1-hop entries in the same manner. If $A1$ does not receive a response after sending the first *Hello* packet, it sends another *Hello* packet with the same destination address but, this time, sets the *TTL* to be H . When another updated router $A3$ (which must be at least 2 hops away from $A1$) receives the packet, it reads the *TTL* value (before subtraction) from the packet and infers that $A1$ is d ($= H - TTL + 1$) hops away from it. Then, $A3$ replies with a *Hello-Response* packet to $A1$ by setting the packet's *TTL* field to d . Note that the *Hello* packet received from $A1$ is not forwarded any further by $A3$. Upon receipt of the packet, $A1$ adds $A3$'s IP address and the port number that received the packet to its d -hop entry and sends a *Hello-Confirm* packet to $A3$, which adds $A1$ in its d -hop entry. This “three-way handshake” procedure is repeated for every randomly selected destination address. Entries in the updated router list may need to be updated when changes in network topology occur. The protocol that we have described

above enables an updated router to keep track of its neighboring updated routers' locations by enabling it to record a neighbor's hop distance and the corresponding outgoing port number.

We expect that executing such a protocol will not be taxing on the network or the routers themselves. Recall that most Internet routes are stable for a relatively long period (i.e., hours or even days). Thus, sampling a few hundred packets' destination addresses within the span of a few hours should be enough to create the updated router lists. Furthermore, the protocol can be arranged to be executed when the network traffic levels are low, thus further reducing its burden on the routers and the network.

B. *Trackback and Packet Filtering to Support Gradual Deployment*

By executing the neighbor-discovery handshake protocol periodically, every updated router can create an updated router list that contains the IP addresses of the H -hop neighbors, their distance (hop count), and the corresponding outgoing port numbers. Using this information, the attack path can be traced back, but the algorithm that was presented in II.C for path reconstruction will not work in its present form and needs to be modified. However, the router port marking scheme remains the same.

In the path reconstruction algorithm presented in Section II, when a query packet is received, a router that is d hops away from the victim is only required to check the hop count ($d+1$) records in the trace table to see whether $XOR_j(d+1) \oplus PN_j(d+1) = XOR_i(d)$. This procedure has to be changed when only a subset of the routers along a path are updated routers. Now, we need to assume that an updated router may need to forward a query packet to other updated routers H -hops away. In this case, the updated router that receives the query packet needs to check the trace table (in the query packet) to find a hop count d record that satisfies two conditions:

1. $PN_k(d)$ is a local port number;
2. $XOR_k(d+d') \oplus PN_k(d+d') = XOR_i(d)$, ($1 \leq k \leq n_{d+d'}$), where d' is the distance to a neighboring updated router in the updated router list.

All hop count d records are tried before moving on to a hop count $d + 1$ record. When such a record is found, the router makes a copy of the trace table and deletes all rows of the trace table that correspond to hop count $d+d' - 1$ and rows that correspond to hop count $d+d'$ in which $XOR_k(d+d') \oplus PN_k(d+d') \neq XOR_i(d)$ is satisfied. Next, the router attaches its IP address to the end of the truncated query packet and then sent to every neighboring updated router that is d' hops away.

If the query packet is successfully delivered to the

attacker's border router, then this router sends back the query packet back to the victim in the same manner as described in Section II.C. However, the query packet may not reach the attacker's border router and the process-and-forwarding process might halt at mid-stream because there are no more updated routers located upstream. In such a case, the most upstream updated router that has received the query packet acts as though it was the attacker's border router, i.e., it sends back the query packet to the victim and initiates packet filtering.

Obviously, when the network does not consist entirely of updated routers, the effectiveness of both the traceback and packet filtering modules will decrease. In Section V, we look into this issue further using simulation results.

IV. DISCUSSION AND ANALYSIS

A. *Supporting IP Fragments*

To mark IP packets with information required in our router port marking scheme, we use the 16-bit identification field, one reserved bit coming immediately after the *Identification* field, and another reserved bit in the *ToS* field.

Like our scheme, the IP traceback scheme proposed by Savage et al. [Sava00] also used the *Identification* field to encode traceback information. The problem with this approach is that fragmented IP packets can no longer be handled by the receiving host. The information in the *Identification* field is needed to reassemble the IP fragments. Savage *et al.* discussed possible methods to handle IP fragmentation, but failed to provide a concrete solution.

We propose a novel method of supporting IP fragments while encoding IP packets with traceback information. We propose using the *TTL* field combined with the *Distance* field and the *MKF* flag as an alternative for the *Identification* field. In most modern operating systems, one value chosen from a set of fixed values consisting of 30, 32, 60, 64, 128, and 255 is used to set the *TTL* value [Jin03]. This means that some of the packets arriving from a client will have the same *TTL* value. We assume that every router treats all the fragments of a packet in the same way. As before, we are assuming that all routes are stable, which means that the fragments of a packet will be traversing the same path to their destination. We will later describe how the routers can be modified to support this assumption. If the fragments are not marked by updated routers (i.e., *MKF* is set to zero), then the *Identification* field remains unchanged and can be used to reassemble IP fragments. However, when the fragments are marked by an APMM router, then additional marking is required in addition to

traceback information marking. Specifically, an APMM router needs to add a value of k to the *TTL* field of each fragment. The value k starts at zero and should be incremented by one for every unique source-destination address pair of the fragments. If adding k causes the value of *TTL* to overflow, then k is reset to zero. Then, the APMM router copies the least significant five bits of the new *TTL* value to the *Distance* field.

When the server receives the fragments, it can group fragments belonging to the same packet by following a two-step process:

1. Check the difference between the *TTL* and *Distance* fields to group fragments according the APMM router that marked them.
2. Differentiate the different set of fragments processed by the same APMM router by using the *TTL* values.

Since the fragment offset field is not used in the router port marking process, its value can be used to sort the fragments corresponding to a single packet. This scheme can differentiate up to $(255 \cdot 32) \cdot 2^5 = 7136$ fragmented packets, which should be sufficient for most applications. The actual number of fragmented packets that can be processed is larger since the unmarked fragments can be differentiated by using the Identification field.

To support our assumption that every router treats all the fragments of a packet in the same way, we need to add additional functionalities to updated routers. Specifically, every router has to maintain a k -counter and several sets of recent (several tens of seconds) fragment marking records, one for every source-destination pair of a fragment. It should be noted that fragmented packets accounts for less than 0.25% of Internet traffic, and the fragments that cannot be reassembled can be recovered by high level protocol retransmissions

B. Consideration of Local Area Networks

In router port marking, we have assumed that every port is only connected to one neighbor router so that its input port number corresponds to a unique upstream router. However, in practice, a router port can possibly be connected to multiple routers, such as in a LAN. In such a case, the proposed reconstruction process will not work properly. To overcome this problem, we propose a variation of our port marking scheme. The concept is straightforward: map MAC addresses to 6-bit identifiers, then use these six bits as the *PN* field value for packet marking. If a router port only hears frames coming from a single MAC address, it simply marks its port number to incoming packets. If multiple MAC addresses are heard, the port maps each address to a unique 6-bit identifier and marks packets from different addresses with the respective 6-bit identifiers. If there are more than 64 MAC addresses that are detected, then a 12-bit

identifier will be used and the advanced port marking scheme is invoked.

V. SIMULATION

A. Simulation Environment and Topology Model

We adopt two different network topologies for our simulations. The first topology was chosen from the Skitter Internet map [Skit04]. Based on the provided undirected link data, we first chose a router with a degree of six as the victim's border router, and then randomly chose several distinct routes originating from it. The second network topology is an n -degree complete tree model. In the n -degree complete tree model, we model the victim's upstream routers as a subset of a complete tree rooted by its border router, with every router having n adjacent routers or hosts (i.e., one parent router and $(n - 1)$ children routers or hosts. Recall that in the Skitter project, the average router degree in the Internet was found to be 6.34. In our simulations, we use $n = 6$ and $n = 8$.

In [Jin03], experiments have shown that a typical hop count distribution from clients to a server can be regarded as a Gaussian distribution, with a mean of 16.5 and a standard deviation of 4. In our simulation experiments for both network topologies, we used this distribution for generating the routes' hop counts. The upper bound and lower bound of the hop counts were set to 32 and 1, respectively.

B. The Performance of TRACK

An important performance indicator for IP traceback is the number of packets that need to be collected for reconstructing attack paths. Figures 9 and 10 show the 95th percentiles for the number of packets required to reconstruct attack paths when marking probabilities are set to 4% and 1%, respectively. To better evaluate TRACK's effectiveness, we repeated the same experiments with *fragment marking schemes* [Sava00, Song01]. In the simulations, we fixed the number of fragments to eight, which is a typical value used in such schemes. Every point in the figures represent the 95th percentile value out of 1000 independent experiments. We can see from the figures that the number of packets needed by TRACK is an order of magnitude less than that required by existing packet marking schemes. This result is expected since TRACK only needs to collect one packet from each router along an attack path, whereas fragment marking schemes require eight fragments. Note that some fragment marking schemes can be adjusted to support less number of fragments [Dean02, Song01], such as 5 or 4 fragments per router, at the cost of increased false positives. One can readily see that TRACK would require less number of packet even if those schemes made such adjustments.

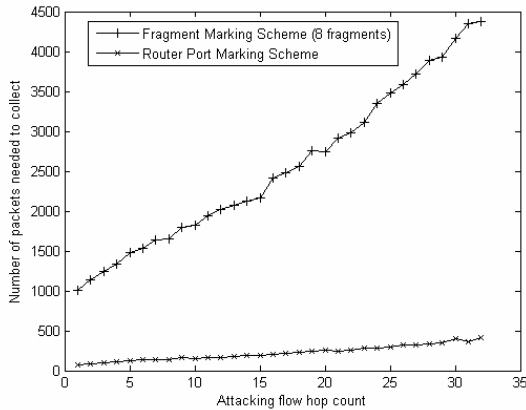


Figure 9: Number of packets required for reconstruction ($p = 4\%$)

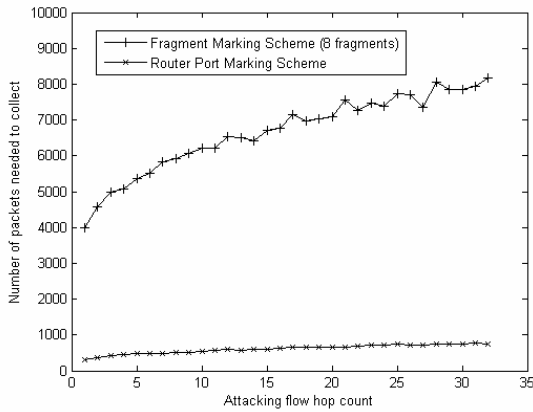


Figure 10: Number of packets required for reconstruction ($p = 1\%$)

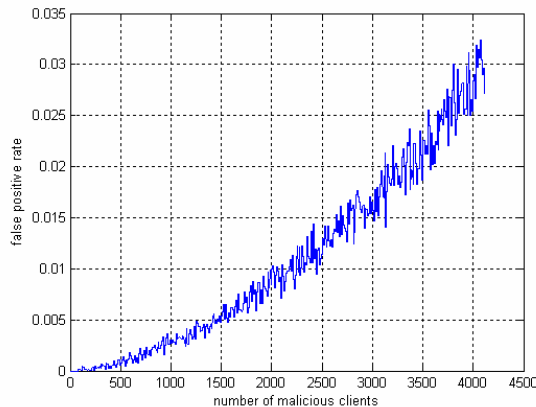


Figure 11: False positive rate on Skitter map

Our packet filtering module uses the information gathered from the traceback module for filtering attack traffic. Hence, the two modules both have zero false negatives and their false positive rates can be shown by the same simulation experiment. Figure 11 shows the

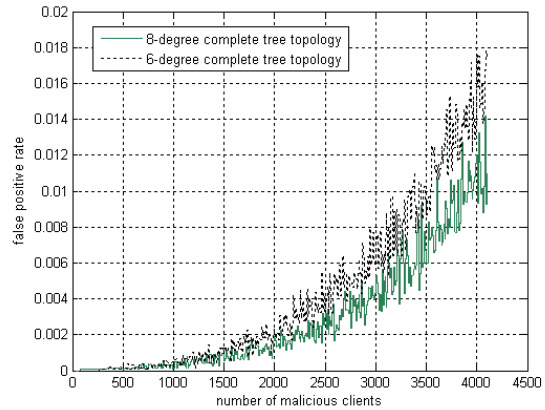


Figure 12: False positive rate on 6-degree and 8-degree complete tree model

false positive rate on a Skitter map topology, and Figure 12 shows the same experiment on 6-degree and 8-degree complete tree model topologies. In each simulation, we fix the total number of client to 5000, and vary the number of randomly chosen attackers from 1 to 4101 in increments of 10. The number of routers in each topology is 59025, 55098, and 59413 for the Skitter map, 6-degree tree topology, and 8-degree tree topology, respectively. The false positive rate is the value of the number of legitimate clients that are mistakenly recognized as attackers by TRACK. All data shown in the figures are the average of three independent experiments.

The results show that TRACK incurs very low false positive rates. When more than 4000 malicious attackers are present, the false positive rate is around 4% for the Skitter map. For 6-degree and 8-degree complete tree topologies, the results are even better. It is interesting to note that TRACK performs better in the 8-degree tree topology than in the 6-degree tree topology. The reason is that if the topology is more complex, then this implies that there is larger number of ports per router. This in turn, reduces the probability of collisions in the trace table that can cause false positives. Thus, we can assume that TRACK's performance will further improve in more complex topologies.

C. The Impact of Legacy Routers

We have simulated TRACK when a portion of the network routers are legacy routers. In the simulation, we assumed that the victim's border router is an updated router and H is 32 in the handshake protocol. Figures 12 and 13 show the simulation results of packet filtering on the Skitter Internet map and on the 6-degree complete tree topology, respectively. In the simulations, we varied the percentage of updated routers starting from 20% all the way up to 100% in increments of 20%. Except for

the victim's border router, each router in the simulation is randomly updated at the specified probability. Therefore, updated routers are uniformly distributed in the network. In the simulations, the total number of clients was fixed at 2000, out of which the number of attackers were varied from 1 to 1501 in increments of 10. Each point in the figures is the average of three independent experiments.

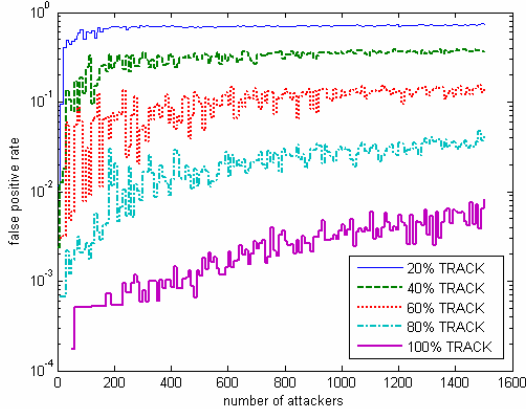


Figure 13: False positive rate on Skitter map with gradual deployment

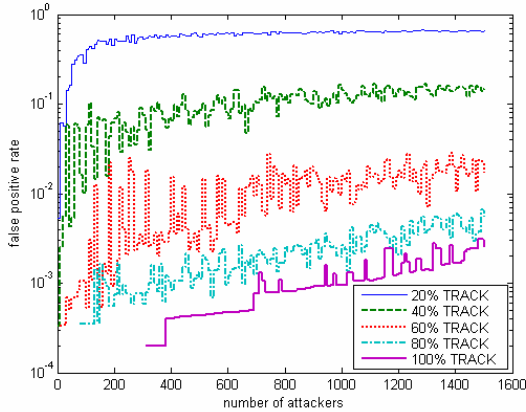


Figure 14: False positive rate on 6-degree complete tree model with gradual deployment

From the results, we can see that TRACK performs relatively well even when only a proportion of the routers are updated routers.

In both figures, we can observe that when the ratio between updated routers and legacy routers is 20/80, the false positive rate reaches 0.7 when there are approximately 1500 attackers. However, the false positive rate drops quickly when the ratio is increased. For instance, in Figures 13 and 14, when the ratio is 60/40, the false positive rates drop to 0.12 and 0.02, respectively when there are 1500 attackers.

VI. RELATED RESEARCH

There has been extensive research dealing with DDoS attacks. As is mentioned in section I, the most relevant schemes seek for making network layer more secure, which falls into IP traceback schemes and attack mitigation schemes.

The objective of IP traceback schemes is to find the origin of attack packets, or malicious clients. They can be further classified those of probabilistic packet marking and packet logging. The schemes presented in [Dean02, Good02, Sava00, Song01] are all packet marking schemes that probabilistically mark each packet with partial path information. By receiving a significant number of packets, the victim can construct the attack paths. Stefan Savage *et al.* [Sava00] proposes the very first packet marking scheme that adopts probabilistic packet marking for IP traceback, but its computation complexity of path reconstruction for multiple attackers is too high ($O(n^8)$, n is the number of attackers) to be practical. In [Song01] this problem is solved by assuming the pre-knowledge of upstream router map of the victim, which itself, however, is non-trivial. An algebraic method for traceback presented in [Dean02] significantly reduces the computation complexity of path reconstruction. However, it requires collecting considerably more packets for path reconstruction. Michael Goodrich [Good02] presents a scheme using large checksum cords to link message fragments, and the cords serve both as associative addresses and data integrity verifiers. The idea of checksum cords is similar to the XOR field in TRACK, therefore it also obviates the complex computation on matching multiple fragments. However, in [Good02] 12-bit additional space is used for cords and virtually nothing is left for other data coding. As a result, 8-bit ToS field is also used for the scheme and even more fragments are needed to be collected.

Packet logging schemes [Li04, Snoe01], on the other hand, only need one single packet to trace one attacker. To make it possible, each router needs to store packet digests in the form of Bloom filters. In [Snoe01] the idea of packet logging is first proposed. Then in [Li04] its scalability in terms of high link speed is improved by increasing the correlation of packets that are logged. The common problem of packet logging schemes lies in the great overhead imposed on routers. For instance, even with optimized technique in [Li04], a router on a duplex OC-192 link still need to compute 8 million hash function per second, and store 5GB data per hour.

Attack mitigation schemes try to actively mitigate DoS attacks by packet filtering instead of locating their sources. Based their different locations to filter packet, they fall into three categories. The first one is victim based filtering [Sung03, Thom03, Yaar03, Yaar04].

NetBouncer is proposed in [Thom03], whose basic idea is to maintain a legitimate client's list at the victim end. If a client is in the list, it is accepted. Otherwise, it is challenged to test its reliability. There is also a controller to prevent the legitimate clients from overwhelming the bandwidth. The packet filtering schemes in [Sung03, Yaar03, Yaar04] are all victim (of victim neighborhood) based and rely on packet marking. Minh Sung and Jun Xu [Sung03] adopt the way using probabilistic packet marking that either inserts an edge (the link connecting two routers) marking for the IP traceback scheme or a router marking identifying the router. Then the victim can try to reconstruct the attack path and estimate whether forthcoming packets lie on the attack path and whether to filter them or not. In [Yaar03] a deterministic packet marking scheme is proposed for packet filtering. It uses the packet's TTL field as an index into the IP Identification field where a router should add its marks. The victim filters packets with a specific marking if the proportion of malicious packets received with the same marking exceeds a pre-defined threshold. In [Yaar04] a mechanism is presented to differentiate the privileged flows with unprivileged ones. The assumption is made that privileged flows never drop packet in favor of non-privileged ones so that the normal flow will not be disrupted by DDoS attack as long as it has privilege over the attacking flow. The way to realize the mechanism relies on a handshake session between the client and server to set up a privileged channel. Every intermediate router joins in the marking of packet and the marking sequence will later be used for authenticated proof of privileged flow. This sequence varies by time to avoid replay attack.

The second category is network based filtering [Maha02, Zhan03]. In [Maha02] a framework of aggregate detection and control is proposed, which consists of two components. The aggregate-based congestion control (ACC) module detects local network congestion and limits high-bandwidth traffic. If a router cannot control an aggregate by itself, pushback module is activated to ask its adjacent upstream routers to rate-limit the aggregate. A DDoS defense system presented in [Zhan03] builds protection at the transport layer of the Internet. It proposes to upgrade the border routers in Autonomous Systems to "Hardened Routers" so that they can support encryption, signing, verifying and dropping packets that they route. The hardened routers are also assumed to have the capability to detect any possible DDoS attacks in order to respond to them. The first and the last hardened router in a route encrypts and decrypts a packet respectively so that the last router knows the location of the first router when attack is detected. Then based on the information decrypted by the last router, it can tell which the attack path is and inform the first router to filter the packet near the source.

The last category is attack source based filtering [Mirk02, Papa03]. D-WARD [Mirk02] is a DDoS defense system deployed at source-end networks that autonomously detects and stops attacks originating from these networks. It makes the edge router monitor the flow from and to the network to which it provides access. It is suggested that bi-directional flows are monitored to TCP layer so that the TCP sessions can be traced. And the flow ratio between a client and a server is made statistics and a threshold that differentiate attack flow and compliant flow. Based on the threshold, filtering is executed in the router at a TCP session level. COSSACK [Papa03] is a system built on all edge networks' border routers, with the core software system called watchdog. The border routers are assumed to have both functions of ingress and egress attack detection. Also, they support ingress IP check to prevent IP spoofing. The other critical assumptions made include the existence of attack signature, the capability of border routers to filter packets based on the signature, and the connection availability between watchdogs. With all these support from lower level, the application layer watchdog can multicast attack notification from the victim side to the source side and suppress DDoS around the source. Though all of these packet filtering schemes are different from one another and have respective advantages and disadvantages, they without exception only support filtering packets at the same location as attack detection. As is discussed in Section I, this is not desirable and TRACK works in the opposite more ideal way.

VII. CONCLUSION

This paper presented a novel countermeasure against DDoS attacks that we call TRACK, which includes the functions of both IP traceback and packet filtering. TRACK is a comprehensive solution that is composed of two components: a router port marking module and a packet filtering module. The former is a novel packet marking scheme for IP traceback and the latter is a novel packet filtering scheme that utilizes the information gathered from the former component. The router port marking scheme marks packets by probabilistically writing a router interface's port number, a locally unique 6-digit identifier, to the packets it transmits. After collecting the packets marked by each router in an attacking path, a victim machine can use the information contained in those packets to trace the attack back to its source. In the packet filtering component, the information contained in the same packets are used to filter the malicious packets at the upstream routers (i.e., routers located in the direction towards the attackers), thus effectively mitigating attacks. Our simulation results show that TRACK has several advantageous features, which include: requires low communication and computation overhead, incurs

zero false negatives and low false positive rates, and is capable of supporting gradual deployment. We are currently extending the basic concepts of TRACK to design a more effective solution. The following paragraphs briefly describe our plans for future work.

To avoid being located, attackers may modify marking fields, i.e., set *MFK* and mark the packets with random data in the *Distance*, *XOR* and *PN* fields. There won't be much impact if the attacker is far away from the victim, because most of the packets will be marked again by intermediate routers. However, if the attacker is close to the victim, such an attack may severely hamper the path reconstruction process. This problem is shared by all IP traceback schemes that employ probabilistic packet marking. We are currently working on techniques to address this problem.

We have shown that TRACK is an effective solution for IP traceback. However, we have not considered the more difficult problem of attack traceback. In attack traceback, the objective is to identify the real attacker that ordered the zombies to launch the DDoS attack in question. We are investigating techniques to integrate the functions of TRACK with network monitoring tools like Netflow [Stev00] to design an effective attack traceback scheme.

REFERENCES

- [Bele03] A. Belenky and N. Ansari. "IP Traceback with Deterministic Packet Marking." *IEEE Communications Letters*, Vol. 7(4), Apr. 2003, pp. 162 – 164.
- [Chan02] R.K.C. Chang. "Defending against Flooding -based Distributed Denial-of-service Attacks: a Tutorial." *IEEE Communications Magazine*, Vol. 40, Issue: 10, Oct. 2002, pp. 42 – 51.
- [Cisc04] Cisco 12816 Data Sheet. Available at: http://www.cisco.com/en/US/products/hw/routers/ps167/products_data_sheet09186a00801df20d.html.
- [Dean02] D. Dean, M. Franklin, and A. Stubblefield. "An Algebraic Approach to IP Traceback." *ACM Transactions on Information and System Security (TISSEC)*, Vol. 5(2), May 2002, pp. 119-137.
- [Ferg98] P. Ferguson and D. Senie. "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing." *RFC 2267*, Jan. 1998.
- [Good02] M. T. Goodrich. "Efficient Packet Marking for Large Scale IP Traceback." In *Proceedings of the 9th ACM conference on Computer and communications security (CCS)*, Nov. 2002, pp. 117 – 126.
- [Ioan02] J. Ioannidis and S. M. Bellovin. "Implementing Pushback: Router-Based Defense Against DDoS Attacks." In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Feb. 2002.
- [Jin03] C. Jin, H. Wang, and K. G. Shin. "Hop-Count Filtering: An Effective Defense Against Spoofed DoS Traffic." In *Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS)*, Oct. 2003, pp. 30-41.
- [Kuro04] J. Kurose and K. Ross. *Computer Networking: a Top-down Approach Featuring the Internet*. Addison Wesley, 3rd edition, 2004.
- [Li04] J. Li, M. Sung, J. Xu, and L. Li. "Large-scale IP Traceback in High-speed Internet: Practical Techniques and Theoretical Foundation." In *Proceedings of 2004 IEEE Symposium on Security and Privacy*. May 2004, pp. 115 – 129.
- [Maha02] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. "Controlling High Bandwidth Aggregates in the Network." *Computer Communications Review*, 32(3), Jul. 2002, pp. 62-73.
- [Mirk02] J. Mirkovic, G. Prier, and P. Reiher. "Attacking DDoS at the Source." In *10th IEEE International Conference on Network Protocols*, Nov. 2002, pp. 312 – 321.
- [Papa03] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan. "Cossack: Coordinated Suppression of Simultaneous Attacks." In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Vol. 1, Apr. 2003, pp. 2 – 13.
- [Park01] K. Park and H. Lee. "On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack." In *Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Vol. 1, Apr. 2001, pp.338 – 347.
- [Paxs97] V. Paxson. "End-to-end Routing Behavior in the Internet." *IEEE/ACM Transactions on Networking*, 5(5), Oct. 1997.
- [Sava00] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. "Practical Network Support for IP Traceback." In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, Aug. 2000, pp.295-306.
- [Skit04] CAIDA's Skitter project web page. Available at: <http://www.caida.org/tools/measurement/skitter/index.xml>.
- [Snoe01] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. "Hash-based IP Traceback." In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, Aug. 2001, pp. 3 – 14.
- [Song01] D. X. Song, A. Perrig. "Advanced and authenticated marking schemes for IP traceback." In *Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Vol. 2, Apr. 2001, pp. 878 – 886.
- [Stev00] R. Steve, F. Mark, and Ron Luman, "The OSU Flow-tools Package and CISCO Netflow Logs," In *Proceedings of the Fourteenth Systems Administration Conference*, Dec. 2000, pp. 291–304.
- [Sung03] M. Sung, J. Xu. "IP Traceback-based Intelligent Packet Filtering: a Novel Technique for Defending against Internet DDoS Attacks." *IEEE Transactions on Parallel and Distributed Systems*, Vol.14(9), Sep. 2003, pp. 861 – 872.
- [Thom03] R. Thomas, B. Mark, T. Johnson, and J. Croall. "NetBouncer: Client-legitimacy-based High-performance

- DDoS Filtering.” In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Vol. 1, Apr. 2003, pp. 14 – 25.
- [Yaar03] A. Yaar, A. Perrig, and D. Song. “Pi: a Path Identification Mechanism to Defend against DDoS Attacks.” In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003, pp. 93 – 107.
- [Yaar04] A. Yaar, A. Perrig, and D. Song. “SIFF: a Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks.” In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, May 2004, pp. 130 – 143.
- [Zhan03] S. Zhang and P. Dasgupta. “Denying Denial of Service Attacks: a Router Based Solution.” In *Proceedings of the 2003 International Conference on Internet Computing*, Jun. 2003.

APPENDIX

Theorem I: In route port marking, if the victim can detect malicious packets by 100% and collect a specific attacker’s packets actively marked by every intermediate router, then the attacker can be traced back. (Zero false negative)

Proof: assuming the attacker is d hop away from the victim’s border router, its path is denoted as a string of input port number: $PN(0), PN(1), \dots, PN(d)$, and the XOR value of the packet actively marked by a i ($i = 0, 1, \dots, d$) hop away router is $XOR(i)$, then under the above condition, what the victim must have in the trace table is these records: $\{0, PN(0), XOR(0)\}, \{1, PN(1), XOR(1)\}, \dots, \{d, PN(d), XOR(d)\}$, where

$$XOR(i) = PN(0) \oplus PN(1) \oplus \dots \oplus PN(i)$$

Since

$$\begin{aligned} XOR(i) \oplus PN(i) &= PN(0) \oplus PN(1) \oplus \dots \oplus PN(i) \oplus PN(i) \\ &= PN(0) \oplus PN(1) \oplus \dots \oplus PN(i-1) = XOR(i-1) \end{aligned}$$

This equation is just what the algorithm introduced in II.B and II.C uses to associate different records into paths. Therefore, we know that $PN(0), PN(1), \dots, PN(d)$ must be ascribed to the same path, which is right the attacker’s path.

Theorem II: In route port marking, if all attackers come from 32-hop away, and if an uninfected router or a legitimate client has its immediate downstream router recognized as infected, then the probability the router being recognized as infected or the client becoming a false positive can be expressed by (3).

Proof: From the analysis in Section II.C, it can be inferred that N_a attackers try to fill S_b combinations in a uniformly random way. Assuming $M(k)$ to be the number of ways to fill a given k combinations with N_a attackers, it is easy to know that there is only 1 way to fill 1 combination, in which case all N_a attackers fall into it, so we have:

$$M(1) = 1$$

for $k > 1$, we have:

$$M(k) = k^{N_a} - \sum_{j=1}^{k-1} \binom{k}{j} M(j)$$

where the first item stands for the ways to fill equal to or less than k combinations, and the subtracted sum is that of the ways to fill less than k combinations.

Therefore, the probability that N_a attackers fill exactly k out of the total S_b combinations is:

$$P_k = \binom{S_b}{k} \frac{M(k)}{S_b^{N_a}}$$

Considering the fact that S_b is the maximum number of combinations the attackers can fill, we can compute the expectation of the number of combinations that are filled as:

$$\bar{N} = \sum_{k=1}^{\min(N_a, S_b)} k P_k$$

Consequently a random combination falls into that covered by attackers, namely the probability an infected router being recognized as infected or a legitimate client becoming a false positive whose immediate downstream router is recognized as infected, is:

$$P_r = \frac{\bar{N}}{S_b}$$

which is the same as (3).