Chapter 6

A Logging Based IP Traceback Mechanism

In the previous chapter, I discuss different packet marking schemes such as XORID and SEM which allow a packet to carry a mark which can be used at the victim end to discriminate spoofed packets from the benign packets. Such a defense system can only decide whether a received packet is spoofed or not, but can not reveal the actual source(s) of the attack packets. In this chapter, I discuss IP traceback mechanisms, where the goal is to find the actual source(s) of one or more attack packets irrespective of the SIP addresses carried by them. I propose and analyze a logging-based IP traceback mechanism, referred to as Singleton Flow Traceback(SFT), that can traceback to the source of a flow containing one or more packets. We consider the tuple $< S, D >$ as a flow, where S is the Source IP and D is the destination IP of the flow and all packets with the same $< S, D >$ pair belongs to the same flow. For logging, SFT requires significantly less storage (less than 8% of packet rate) than most of the existing single packet traceback mechanism in the literature. SFT guarantees zero false negative rate, i.e., if there exists a path $P = R_1, R_2....R_k$ of routers through which the packets of a flow $< S, D >$ has traversed, SFT can always reach R_1, i.e., the first router to which the source of the flow $< S, D >$ is connected with. Also, the expected value of false positive rate for a traceback query is very low (< 0.11).

6.1 Introduction

In the current Internet the most common but destructive threat is a DDoS attack. The most difficult part with a DDoS attack is to mitigate the attack even if an

attack is detected. An IDS can detect the presence of a DDoS attack by monitoring different traffic attributes such as traffic volume, entropy variance of the SIPs of the incoming packets or some flow level attributes like flow length and flow size of the incoming traffic. To mitigate a DDoS attack it is important to prevent the zombies (puppets of the attacker) from pushing more attack packets into the protected site or network. The presence of SIP spoofing allows a sender to fill the SIP field of the sent packets with any probable IP address. Hence at the victim side identification of the participating zombies based on the SIP of the incoming packets becomes void. Thus, it becomes important to have a traceback mechanism which can provide the true identity of the sender of one or more packets to the victim irrespective of the SIP of the examined packets.

To achieve this goal researchers have developed different approaches such as link testing schemes [102] ,[103], packet marking schemes [29],[74], packet logging schemes [138] and hybrid of packet marking and packet logging schemes [127],[100], [116]. Link testing approaches incur extra load to the network by generating extra packets as part of the traceback mechanism. Packet marking approaches use one or multiple less frequently used IP fields of a packet as the marking field. The most commonly used IP header fields for this purpose are 16 bit ID field, 3 bit fragmentation field and 13 bit fragmentation offset field. Thus, we see that the marking of a packet generally varies from 1 single bit to 32 bits based on the approach. In methods proposed in [29],[101] a border router inscribes its IP address into the 16 bit ID field of all its incoming packets. Since the border router's IP address does not fit in a single packet's ID field, it creates several segments of its IP address and selects a segment randomly to mark the current packet. When the victim receives enough packets it can construct the IP of the border router through which the packets entered into the Internet by assembling the received marks. Another packet marking approach is probabilistic packet marking (PPM). Schemes proposed in [74],[124] and [97] comes under this category. The main idea of PPM is that each intermediate router along a path probabilistically decides whether to mark a passing packet or not. On the receiving side, the victim has to collect enough number of packets carrying different marks to construct the entire path traveled by a set of packets. If the victim does not receive enough number of packet from a source it might not be able to reconstruct the path. Another disadvantage of such a scheme is that it imposes a computational burden on the victim side. In [26] and [27] each router along the path contributes a portion of the mark instead of completely overwriting it. Thus, when a packet reaches the destination the mark of the packet is used as an identification of the path traveled

by the packet. Two packets carrying the same mark is considered to travel the same path to reach the destination. However, due to the limited size of the marking field it is impossible to inscribe the entire path information into a packet. For this reason such approaches suffer from a high false positive rate in source identification. To achieve higher accuracy in traceback, researchers proposed logging packet specific information in the intermediate router. One of the pioneers of such approach is the work proposed in SPIE [138]. SPIE uses Bloom filters [139],[58] to manage the digests of the passing packets through a router. However, high storage requirement of SIPE in high speed router restricts its practicability. In an attempt to reduce the storage requirement of the log based approaches, hybrid of packet marking and packet logging techniques have been proposed in [127],[100],[116],[122],[123]. The main goal of such techniques is to reduce the logging frequency in an intermediate router. In such an approach some of the intermediate routers performs only marking operation and others perform both.

Investigating the above mentioned works from the literature we still find scope where improvements can be contributed. Most logging based techniques concentrate around facilitating a single packet traceback mechanism which accepts a copy of a specific packet or its transformation to reach to its actual source. However, in a situation like DDoS attack instead of following a single specific packet, following a particular flow originated from a claimed source, i.e., a group of packets carrying the same pair of $< S, D >$ suffices the need. Since the number of flows through a router at a time is many times less than the number of packets, the storage overhead at a router can be drastically reduced by logging individual flow information rather than packet information. Another issue which is needed to be handled in most of the hybrid and log based approach is the false negative caused by the refreshing of the log tables in the intermediate routers. Keeping these issues in mind, in this paper we propose a log based traceback system, referred to as Singleton Flow Traceback (SFT). The main emphasis of "Singleton Flow Traceback" is to traceback a flow up to its actual source even if it contains only one packet. The prime features of SFT are as follows:

- SFTs storage requirement is proportional to the number of flows rather than the number of packets through a router per unit time. Thus, SFT has many times less storage requirement than that of other single packet traceback approaches.

- We introduce a logging scheme with a rolling pair of bloom filters to guarantee zero false negative without needing the intermediate routers to synchronize

their log table refreshment event.

- In SFT a single traceback query can reveal all the sources which are communicating with the destination using the same SIP at the same time. This situation might occur in case of a DDoS attack where one or multiple zombies $z_1, z_2, ..z_k$ send attack traffic to a specific victim v with randomly spoofed SIPs. In that case, the victim might receive packets carrying a particular $< S, D >$ pair from multiple zombies. Under such situation, existing single packet traceback mechanisms can reach only up to one zombie based on the packet selected for traceback. However, in SFT, a single query can reveal all such zombies in parallel.

We provide detailed mathematical description to show how SFT attains zero false negative. Also, we provide both theoretical and practical analysis (using real Internet topology) to demonstrate low margin of false positive rate (less than 11% of the SFT queries return at least one false source along with the actual source) of SFT. In section 6.2, we provide our solution. Section 6.3 analyses the performance of SFT. In section 6.5 draws the conclusion.

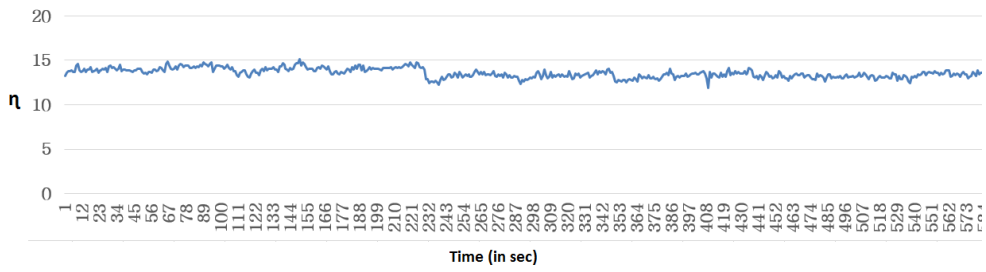
6.2 Singleton Flow Traceback(SFT)-The Proposed Traceback Mechanism

Most existing logging-based traceback mechanisms in the literature need a router to log the digests of the packets passing through it. Thus, the size of the log in a router is proportional to the packet rate through that router. In high speed link such approaches need several hundreds of Mega Bytes of storage to accommodate the log, which is not affordable. Instead of recording information against each packet, our method, referred to as SFT records information against each flow. Thus, the storage requirement in a router to log traceback information by SFT is significantly reduced by a factor η, where

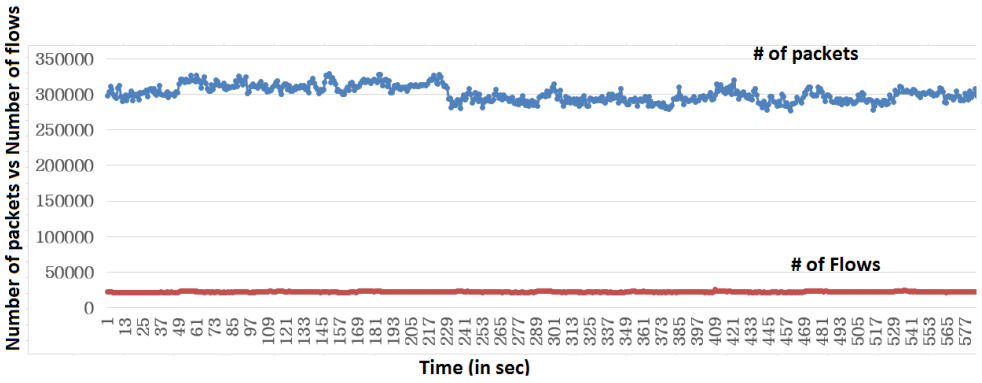
$$\eta = \frac{\#\ packets}{\#\ flows}$$

. Figure 6.1(a) shows η for CAIDA-2014 [84] high speed network traces. Figure 6.1(b) shows the number of $< S, D >$ pairs vs number of packets/unit time from the same trace. It can be seen from the figure that η is approximately 14, on average, i.e. number of digests needed to be stored by an SFT enabled router is less than 8% of the total packet rate through the router. Other than less storage requirement,

another advantage of SFT is that it requires comparatively less number of routers to log a specific flow to facilitate successful traceback of the flow. Let us consider an example as shown in Figure 6.2.



(a) η



(b) Number of $< S, D >$ pairs vs number of packets per second

Figure 6.1: Comparison of number of $< S, D >$ pairs vs number of packets per 10 minutes in CAIDA network trace.

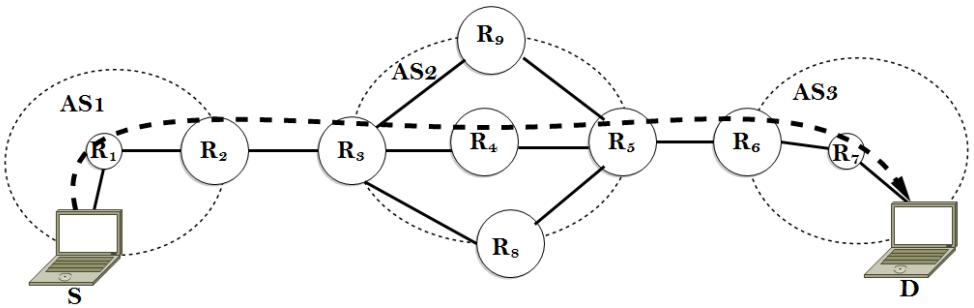


Figure 6.2: An example of transit flow

A source S from AS_1 is communicating a flow to the destination D in another autonomous system AS_3. AS_2 is the intermediate autonomous system traveled by the packets of the flow. R_3 is the router through which the flow entered into

AS_2 and R_5 is the router through which the flow exited the network. The flow is called a *transit flow* for AS_2. For any *transit flow* in an autonomous system both the source and destination are outside the autonomous system and thus, the autonomous system always have a pair of entry and exit edge routers through which the flow enters and exit the network. Autonomous system can record such *transit flow* only at the entry and exit edge routers by considering them as two consecutive hops in the path. The average number of hops in terms of routers and autonomous system between any source and destination in the Internet is approximately 15 and 3, respectively. Thus, in SFT, on an average a flow is needed to be logged in 5+2+5=12 nodes rather than 15 nodes. It is approximately 20% reduction in terms of the number of intermediate nodes to be visited to reach a source through a path by SFT.

We now discuss how an SFT enabled router logs the flow information to facilitate accurate traceback up to the sources of one or more flows. Table 6.1 gives a quick reference to some of the terminologies used to describe the paper.

Table 6.1: Short hand notations and their meanings

NAME	DESCRIPTION
AS	Autonomous System
EBGP	Stands for External BGP router. Two routers R_i and R_j are EBGP to each other if R_i and R_j belongs to two different AS
internal neighbor	R_i and R_j are *internal neighbor* to each other if both belongs to the same AS
SIP	Source IP address of a packet
DIP	Destination IP address of a packet
LT	LAN_TABLE
FT	FORWARD_TABLE

6.2.1 Flow Recording Process At Intermediate Routers By SFT

Each router maintains a pair of two fixed size tables , namely

1. LAN_TABLE (LT), which is used to log the flow information coming from the local LAN of the router.

2. FORWARD_TABLE (FT), which is used to log the flow information forwarded to it by its upstream routers.

In this chapter I use the term LOG to refer to both LT and FT collectively. Since the tables get filled up along with time, it is important to refresh the content of these tables periodically. However, such refreshing might lead to false negative if proper synchronization is not maintained among the routers. To overcome this problem, each SFT enabled router makes use of two LOGs, i.e., LOG[0] and LOG[1], which are refreshed alternatively at the end of a fixed time interval T. The refreshing of one router is completely independent from other routers. However, all have to agree with the duration T of the interval. Figure 6.3 shows a pictorial representation of the schema.

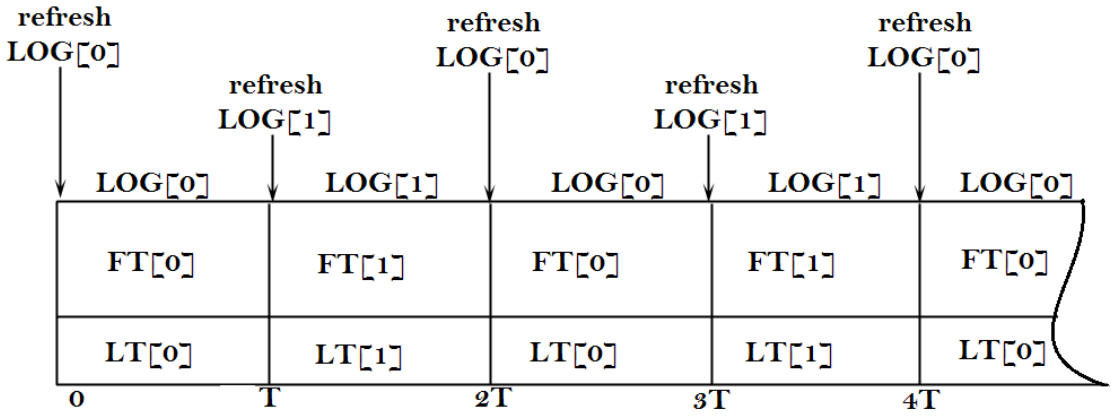


Figure 6.3: Demonstration of Lemma1

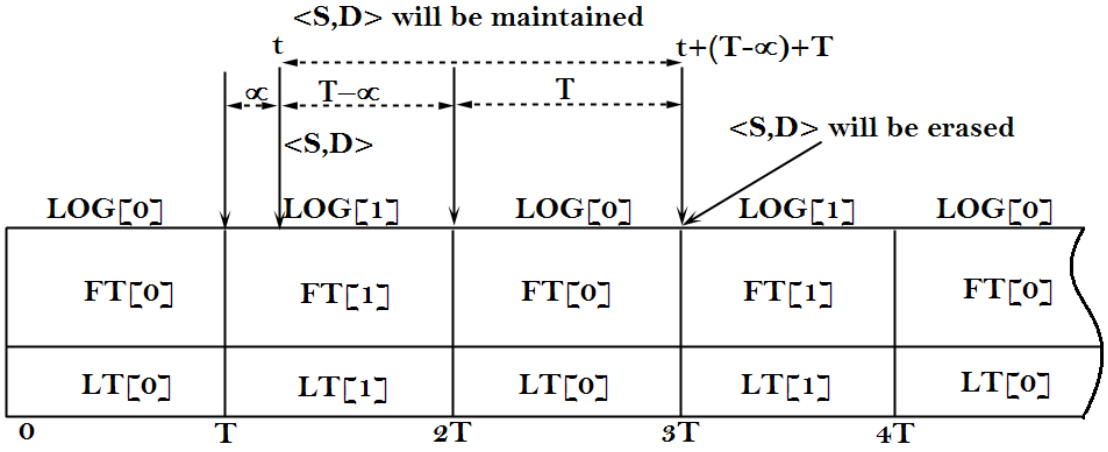


Figure 6.4: Demo

With such a scheme we have the following Lemma

Lemma 1. *If a flow traversed a router R at time t, the flow information will be available at R atleast upto time t+T. In other words every flow information through*

a router will be available at the router for atlest T time from the recording time of the flow in that router.

Proof. An SFT enabled router refreshes the LOGs alternately at the end of T time interval. Consider at some point of time t, a flow $< S, D >$ is recorded in the current LOG, as shown in Figure 6.4. Let α be the elapsed time from the most recent refresh time. In that case the information related to $< S, D >$ will be available in the LOG_TABLE until the table is refreshed again, i.e., upto $t + (T - \alpha) + T$ time. Since $T - \alpha >= 0$, we have

$$(T - \alpha) + T >= T$$

.

$$=> t + (T - \alpha) + T >= t + T$$

□

As mentioned above, SFT needs to record $transit\ flow$s only at the edge routers of the corresponding AS. To achieve this, we use the first two bits of the 16-bit ID field of the IP header to mark a packet P. A router takes the logging decision based on the mark of a packet. As soon as a packet P is received at an interface, a router processes the packet as shown in Figure 6.5. When a router receives a packet from its local LAN, it sets the packet mark as '10'. If the packet is received from an EBGP router, it's destination IP is checked. If the packet is destined for an external DIP then the mark field of the packet is set to '01', otherwise it is set to '11'. The mark field is not changed if the upstream router is a router from the same network. The marking of the incoming packets can be done independently at every incoming interface of a router. Before forwarding a packet to the next hop by an SFT enabled router, the mark of the packet is verified, as shown in Figure 6.6 to decide whether to log the packet or not. If a packet is being forwarded to an EBGP router, the flow information of the packet, i.e., $< SIP, DIP >$ is always recorded irrespective of the mark of the packet. On the other hand, if the packet is forwarded to a local router then the mark of the packet is checked. If the mark is '10', the flow information is recorded at the LT of the current router and the packet mark is changed to '11'. If the mark is '11' the current router always logs the corresponding flow information. If the packet mark is '01' then the current router logs the corresponding flow and resets the mark of the packet to '00' which prevents further logging of the flow upto the exit router of the intermediate network. The interpretations of the packet marks are mentioned in Table 6.2.

```
for each incoming packet P do
    if P is received from LAN then
        P.mark='10';
    end
    if P is received from EBGP then
        if P.DIP is external to the network then
            P.mark='01';
        else
            P.mark='11';
        end
    end
end
```

Figure 6.5: Incoming packet processing

```
for each outgoing packet P do
    if Outgoing link is to EBGP then
        log(< P.SIP, P.DIP >)in FT;
    else
        if P.mark=='11' then
            log(< P.SIP, P.DIP >) in FT;
        else if P.mark=='01' then
            log(< P.SIP, P.DIP >) in FT;
            P.mark='00';
        else if P.mark=='10' then
            log(< P.SIP, P.DIP >) in LT;
            P.mark='11';
    end
end
```

Figure 6.6: Outgoing packet processing

Table 6.2: Interpretation of the mark of an outgoing packet by a router

MARK	INTERPRETATION
11	Log the corresponding flow information in FT
01	Log the corresponding flow information in FT and reset mark as 00
10	Log the corresponding flow information in LT and reset mark as 11
00	Do not log

Figure 6.7 demonstrates how a flow (for simplicity we assume the flow is a single packet flow) is processed and logged along the path from a source-destination pair $< S, D >$. When the packet reaches R_1 from S, it's mark field is set to '10'. When forwarded by R_1, it logs the corresponding $< S, D >$ pair in its LT and sets the

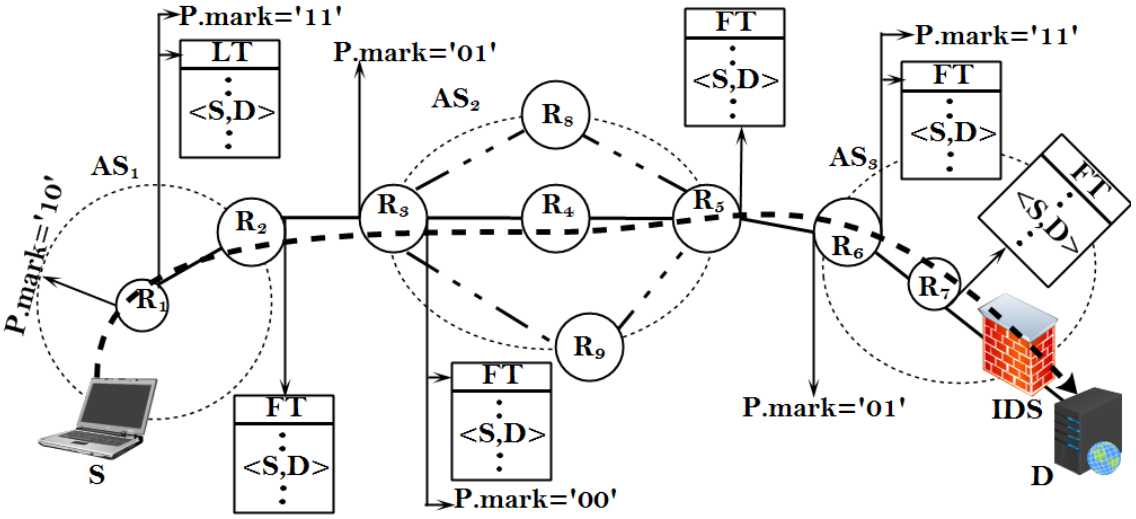


Figure 6.7: Demonstration of flow recording

mark as '11'. Since R_2 receives the packet from an *internal neighbor* R_1, it keeps the mark as it is. R_2 forwards the packet to an $EBGP$ router R_3 and hence logs $< S, D >$ in its FT. R_3 receives the packet from an $EBGP$ router and since the destination is outside the network, hence it sets the mark of the packet as '01'. When forwarded, R_3 logs $< S, D >$ in it's FT and sets the mark of the packet to '00'. R_4 neither changes the mark, as it is received from an *internal neighbor*, nor logs the flow, as its mark is '00'. R_5 does not change the mark since R_4 is an *internal neighbor* to it. But when forwarded to R_6, R_5 logs $< S, D >$ as R_6 is an $EBGP$ router to it. R_6 receives the packet from an $EBGP$ router and the destination is in the same network, hence, it marks the packet with '11'. Finally, when R_7 receives the packet from R_6 it does not change the mark and while forwarded, it logs $< S, D >$ in it's FT.

6.2.2 Tracing Back a Flow

In the previous section, we have seen how an SFT enabled router (here onwards we will refer to an SFT enabled router by simply a router, as we assume that all routers are SFT enabled to facilitate traceback) logs the flow information going through it. In this section we discuss the traceback mechanism to reach the source of a communication from the destination end of the communication. A traceback can be initiated by issuing a traceback query, referred to as *Query*, to the nearest router by an authorized IDS. A *Query* contains four fields. The significance of these fields are listed in Table 6.3.

When a router receives a *Query*, it processes it as shown in Figure 6.8. For

Table 6.3: The fields of a traceback *Query*

Field	Description
ID	A 32 bit number which uniquely identifies a Query
DIP	The destination IP of the flow being queried
SIPList	The list of SIPs whose corresponding flows with Query.DIP are being trace backed
Action	It specifies the action needed to be taken by a router when it identifies its LAN as the source for a traceback query.

each SIP S_i in $Query.SIPList$, the flow information $< S_i, Query.DIP >$ is checked against the LT and FT in parallel. If a flow is found, the corresponding Source IP i.e., S_i, is maintained in a list, referred to as $FOUND\_LIST$. After processing all S_is in $Query.SIPList$, if $FOUND\_LIST$ is not empty for LT, it implies that the SIPs in the $FOUND\_LIST$ are from the local LAN of the router. Thus, the router executes the action specified by $Query.Action$. On the other hand a non-empty $FOUND\_LIST$ in FT verification implies that the router has recorded communication from the sources in the $FOUND\_LIST$ to the destination $Query.DIP$. Since these flows might be forwarded to the router by any of its neighbors, the current router replaces $Query.SIPList$ by $FOUND\_LIST$ and forwards the $Query$ to all its neighbors. When we say 'neighbor', there might be two different kinds of neighbors to a router, such as

1. *Direct Neighbor*: Two routers R_1 and R_2 are direct neighbors of each other, if there is a direct communication link between R_1 and R_2.

2. *Virtual neighbor*: Two routers R_1 and R_2 are virtual neighbors of each other, if both R_1 and R_2 are edge routers of the same AS.

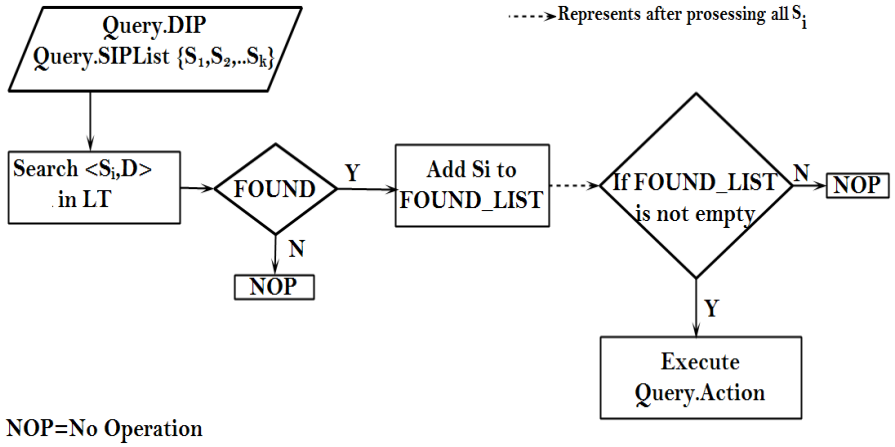
If the current router is an *internal router*, it will forward the $Query$ to all the routers to which it is directly connected with. On the other hand, if the current router is an edge outer, then along with all direct neighbors, the $Query$ will be forwarded to all other edge routers of the network.
Since the current router forwards $Query$ to all its neighbors, it might happen that a $Query$ back propagates, i.e., a router might receive a $Query$ back, that it just had forwarded to its neighbor. To overcome this problem the $Query.ID$ is used as an identifier to a traceback query. After processing a traceback query the corresponding $Query.ID$ is recorded by the current router. When a router receives a $Query$, it first checks if the $Query$ has been processed in the recent past. In that case the $Query$ is ignored by the router. We consider the size of the $Query.ID$ field as 32 bit. If there are k IDSs simultaneously issuing different $Query$s then the probabil-

ity of collision of two $Query.ID$ is $P(collision) = \frac{k}{2^{32}}$. Since $k << 2^{32}$, $P_{collision} \approx 0$.

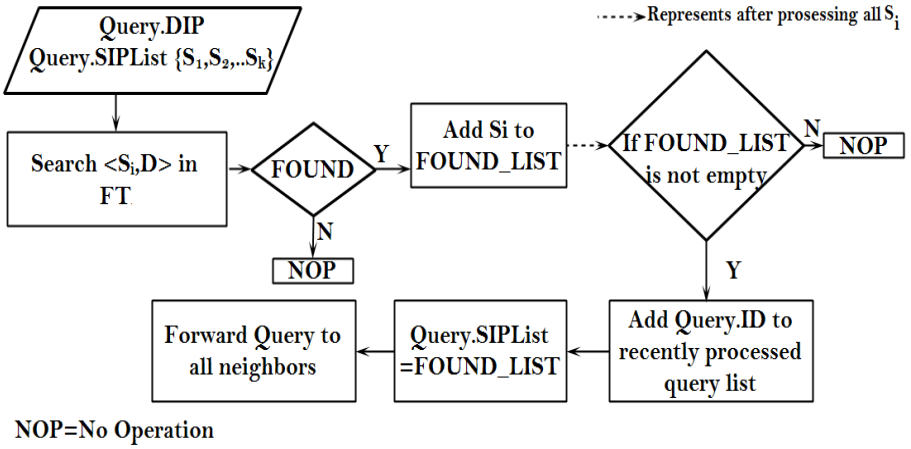
Figure 6.9 shows the propagation of a traceback query for the flow $< S, D >$. An IDS monitoring the traffic to the victim D, issues a traceback query at R_7. Since $< S, D >$ is logged at R_7, the query will be forwarded to R_6. R_6 forwards the query to R_5 as well as R_7. Since R_7 has already processed the query, it will not respond to the query received from R_6. However, R_5 will process the query and since the queried flow is recorded at R_5, the query will be forwarded to all the direct neighbors i.e., R_4 as well as virtual neighbors i.e., R_8,R_9 and R_3 of R_5. Since R_4,R_8 and R_9 has not recorded $< S, D >$, the query will be discarded at these routers with a very high probability. However, due to occurrence false positive, R_4,R_8 and R_9 might forward the query to their neighbors independent of each other. R_3 on the other hand will forward the query to R_2 as the queried flow $< S, D >$ is recorded at R_2. Finally R_1 will find a match in its LT for the queried flow and thus, identify its LAN as the source of the flow.

6.2.3 Implementation Details of the Log Tables

From the above discussion we see that during traceback the log tables maintained by a router need to answer only membership queries against an entry, i.e., they are used only to see if a particular $< S, D >$ pair is there in the table. No other information related to a flow $< S, D >$ is needed to be stored and retrieved. Due to this property, the log tables are implemented as Bloom filters[14]. A Bloom filter is a space-efficient but probabilistic data structure that supports membership queries against a set of elements. To represent and support membership queries for a set $X = \{x_1, x_2, ...x_n\}$ of n elements, a Bloom filter uses m bits and a set of k hash functions, $H\{h_1, h_2, , , , h_k\}$ with range $(1, 2, ...m)$. For theoretical analysis it is assumed that each hash function spreads the n items in the set uniformly over the range $(1, 2, ...m)$, independent of each other. For practical implementation we chose the hash functions from a family of universal hash function. For each element $x_j, j = 0, 1, ...n \in X$, $m[h_i(x_j)]$ is set to 1, where $i = 0, 1, ..k$. For a membership query of an element y, if $\exists h_l, l = 0, 1, ..k \in H$ such that $m[h_l(y)] = 0$, then the element y is certainly not there in the set. If $\forall h_l, l = 0, 1, ..k$ $m[h_l(y)] = 1$, then we assume that the element is a member of the set with a certain probability of being wrong, referred to as $false\ positive\ rate$. Let $X = \{x_1, x_2, ...x_n\}$ be the set of n elements, m be the number of bits in the Bloom filter, k is the number of hash function used by the Bloom filter and fpr is the maximum tolerable false



(a) Search in LOG_TABLE.LAN_TABLE



(b) Search in LOG_TABLE.FORWARD_TABLE

Figure 6.8: Traceback query processing at a router

positive rate of the Bloom filter then the following equations can be used to show the relationship among the parameters.

$$k = ln\frac{m}{n} \tag{6.1}$$

$$fpr = (0.6185)^{\frac{m}{n}} \tag{6.2}$$

From Equation (6.2), we see that the *false positive rate* of a Bloom filter decreases if bits per element i.e., $\frac{m}{n}$ increases. Also, from Equation (6.1) we see that the number of hash functions in a Bloom filter increases logarithmically with respect to $\frac{m}{n}$.

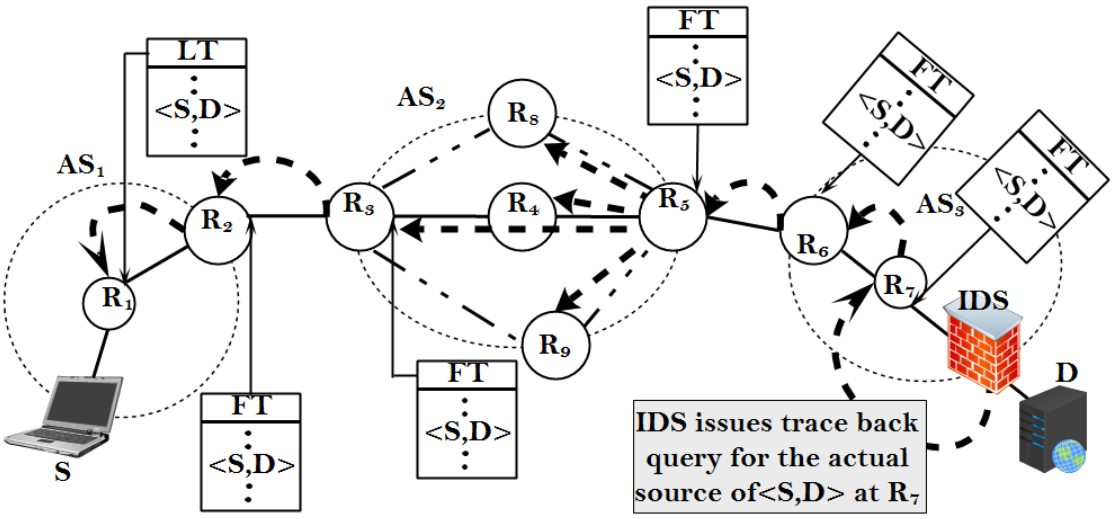


Figure 6.9: Demonstration of traceback mechanism

6.3 Analysis of SFT

The efficiency of a traceback mechanism can be evaluated based on false positive rate, false negative rate, storage requirement at intermediate routers, computational overhead at intermediate routers and the traceback time to reach the source.

6.3.1 Analysis of False Negative Rate of SFT

We use Bloom filter to store and query the flow information in a router. Bloom filters have the useful property of *zero false negative rate*, i.e., if $x_i \in X$ then a membership query for x_i in X will always return a 'positive', irrespective of the values and the number of other elements in the set. From the discussion in section 6.3.2, we know that a traceback query propagates from the current router to its upstream routers (i.e. ,both direct and virtual neighbors of the current router) only when the current router returns a 'positive' for the queried flow $< S, D >$ in their FT. Also, to confirm the source, the first router from the source must return a 'positive' in its LT. Thus, to reach R_1 from R_k along the path $\{R_1, R_2, ...R_k\}$ all routers $R_i, i = k, k-1, ..1$ must return 'positive' for the queried flow $< S, D >$. This will be always true, only when all the routers $R_i, i = k, k-1, ..1$, contains the entry for $< S, D >$ in their corresponding LOGs, when they are being queried, according to the *zero false negative rate* property of Bloom filters. According to Lemma 1, a router always keeps a flow information at least for the time interval T, from the most recent recording time of the flow in the router. Thus, for successful retrieval of the source of a flow $< S, D >$ it is important that the query reaches the

source router, i.e., R_1 within T time from the most recent recording time of the flow in that router. To meet this requirement the following equation must be true

$$ppt + DT + qpt \leq T \tag{6.3}$$

Where, DT is the time difference between the time at which the traceback query is issued by the destination of the flow and the time at which the flow reached the destination (shown in 6.10), ppt is the propagation time of the flow from R_1 to R_k and qpt is the propagation time of query from R_k to R_1

Since both ppt and qpt are typically in milliseconds and DT is in minutes, we can consider DT as a safe upper bound for $ppt + qpt$, i.e., we can consider $ppt + qpt \leq DT$. Thus we can write

$$DT + DT \leq T$$

$$=> 2 \times DT \leq T$$

$$=> T \geq 2 \times DT$$

From this we state the following lemma.

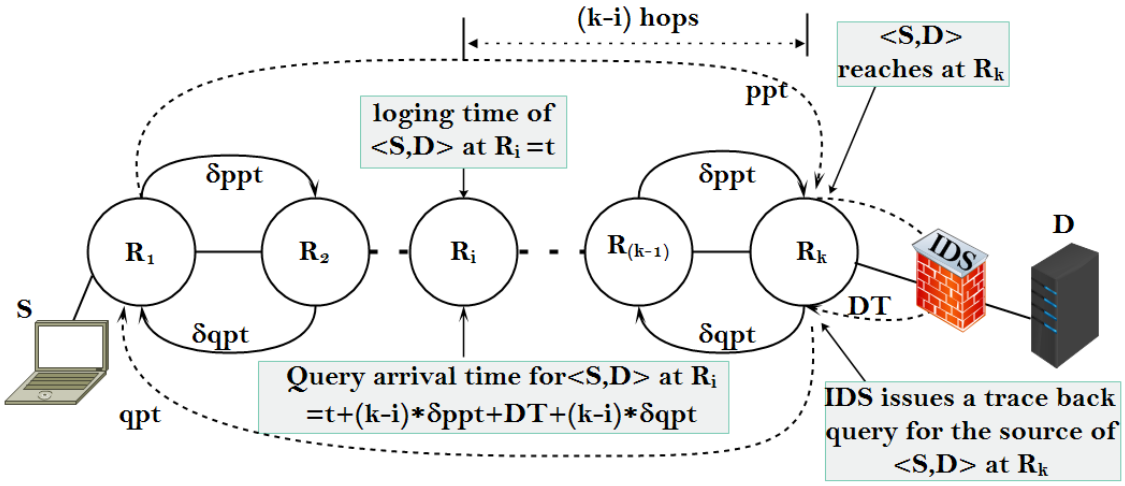


Figure 6.10: Demonstration of Lemma2

Lemma 2. *For any flow $< S, D >$ traversing the path $\{R_1, R_2, ...R_k\}$, the first router R_1 can always be reached (traced back), if $T > 2 \times DT$ assuming $ppt + qpt << DT$, where T is the minimum guaranteed time interval of the routers (as mentioned in Lemma 1)*

Proof. Say t is the time at which R_i recorded the queried flow $< S, D >$ in its LOG. Then the flow will reach and will be recorded at R_k at time $t + (k - i) \times \delta ppt$,

δppt is the flow propagation delay for one hop. Given, the traceback query for the flow is issued at R_k within $\frac{T}{2}$ time of its arrival, hence, the query will reach R_i at $t + (k-i) \times \delta ppt + \frac{T}{2} + (k-i) \times \delta qpt$, where δqpt query propagation delay for one hop. Figure 6.10 depicts the fact pictorially. Thus, the time difference between the recording of the flow and the arrival of the traceback query at router R_i is

$$\delta t_i = (k-i) \times \delta ppt + \frac{T}{2} + (k-i) \times \delta qpt$$

Since $i \geq 1$ *and* $i \leq k, (k-i) \times \delta ppt \leq (k-1) \times \delta ppt \leq ppt$ and $(k-i) \times \delta qpt \leq (k-1) \times \delta qpt \leq qpt$, we can write

$$\delta t_i = (k-i) \times \delta ppt + \frac{T}{2} + (k-i) \times \delta qpt \leq ppt + \frac{T}{2} + qpt$$

Agin, $qpt + ppt << DT$ and $DT \leq \frac{T}{2}$. Which implies

$$=> \delta t_i \leq DT + \frac{T}{2}$$

$$=> \delta t_i \leq \frac{T}{2} + \frac{T}{2}$$

$$=> \delta t_i \leq T$$

The traceback query will reach all the routers along the sequence $\{R_k, R_{k-1}...R_1\}$ within T time from the recording time t_i of the queried flow in that router. Since a router always keeps a flow information at least for duration T from its recording time, due to *zero false negative* of the Bloom filters, each $R_i, i = k, k-1, ...2$ will return 'positive' for the the queried flow in its FT and will forward the query to R_{i-1}. On the other hand, R_1 will return a 'positive' from it's LT, i.e, R_1 will identify the source of the queried flow as its local LAN. □

It can be safely assumed that a real-time IDS can detect and pinpoint the sources of the malicious traffic within a few minutes, i.e., less than 5 minutes, from the starting of the attack. Thus if $T \geq 10$ minutes then Lemma 1 assures *zero false negative*, i.e. if a flow $< S, D >$ traversing through the sequence of k routers $\{R_1, R_2...R_k\}$ is traced back within a short time interval after its arrival at the destination, then SFT will always be able to reach to the source router, i.e., R_1. Each router can set its T to any value ≥ 10 minutes independent of one another.

6.3.2 Analysis of False Positive Rate of SFT

Another desirable property of a traceback mechanism is *zero false positive*. Let's first define what is a *false positive* in SFT.

Definition 1. For a particular flow $< S, D >$, traversing through a sequence of routers $\{R_1, R_2, ...R_k\}$, if SFT returns atleast one router R_x such that $R_x \neq R_1$ as the source, then it called a $false\ positive$.

We will now get an estimation of the $false\ positive\ rate$ (fpr) i.e., the average number of $false\ positive$s returned by SFT for one traceback query. As mentioned in section 6.3.2, when a router receives a traceback query, it checks the queried flow in its LT and FT simultaneously. A $false\ positive$ in LT will cause a $false\ positive$ in SFT. Due to $false\ positive$, the current router identifies its LAN as the source of the queried flow. Although we can not make Bloom filters deterministic, but we certainly can set fpr of a Bloom filter to a very low value (such as ≤ 0.0001) to make it almost deterministic. From Equation (6.2) and (6.1), a Bloom filter with fpr=0.0001 will need $< 20N$ bits of memory and ≤ 3 hash function executions per search. If we consider N to be $1,00,000$ then the amount of memory needed will be around $2MB$. Since LT records only those flows coming from the LAN of the corresponding router, N can be expected to be even smaller than $1,00,000$. On the other hand a $false\ positive$ in FT will cause unnecessary forwarding of the traceback query to all neighbors of the current router, which will increase the network load as well as the fpr of SFT. The FT might have to store hundreds of millions of flows within $2 \times T$ time, specially when the router is a back bone router. Figure 6.1 shows the number of distinct flows per ten minutes in CAIDA's[84] high speed network traces. Thus, to assure a low false positive rate in FT, the memory requirement will be high. Table 6.4 mentions the memory requirement for different fpr of the Bloom filter, considering $N = 10^7$, representing a sufficiently large number of flows.

Table 6.4: FPR vs Memory needed to store $N = 10^7$ flow information for Bloom filter

fpr	MEMORY in MB
0.5	14
0.2	33
0.1	50
0.01	100
0.001	150
0.0001	200

We see that a low fpr (≤ 0.01) Bloom filter needs hundreds of MBs of memory in a router, which might not be available. However, a moderate fpr (< 0.10) Bloom filter can be achieved by using less than 50 MB of memory. To estimate the fpr of SFT, first we estimate the average number of routers, to which a single traceback

query propagates. For theoretical analysis we consider the network topology as a rooted tree, where the root of the tree is considered as the destination. The sources can be connected to any node in the tree. We assume each internal node has exactly $(d+1)$ neighbors, where d is the number of children of the node and 1 for its parent. Let us assume that all the nodes set the same fpr for their Bloom filters. Say, R_x is a router which never recorded the queried flow. In that case expected number of nodes, which will be visited by a traceback query issued from R_x is given by

$$\sum_{i=0}^{i=h} d^i \times (fpr)^i$$

$$= \sum_{i=0}^{i=h} (d \times fpr)^i$$

where h is the height of the tree, or the diameter of the Internet. Consider k is the average number of nodes along a path traversed by a flow $< S, D >$. Since from each node the query will always propagate to all d neighbors, the expected value of the number of nodes visited by a single traceback query is given by

$$E(visited\ nodes) = k \times d \times \sum_{i=0}^{i=h} (d \times fpr)^i \tag{6.4}$$

It implies the expected value of fpr of SFT traceback query is as follows

$$E(fpr) = E(visited\ nodes) \times (fpr\ of\ the\ LT) \tag{6.5}$$

From Equation (6.4) we see that, if the product $d \times (fpr) > 1$, the expected value of the number of nodes visited by a single query in SFT will grow rapidly. Table 6.5 reports $E(visited\ nodes)$ for different combinations of d and fpr. The value of k is assumed to be 15, as it is the average number of nodes between any $< S, D >$ pairs in the Internet.

Table 6.5: $E(visited\ nodes)$ per SFT traceback query

	fpr=0.5	fpr=0.4	fpr=0.3	fpr=0.2	fpr=0.1	fpr=0.01	fpr=0.001
d=2	1250.00	249.06	125.00	83.33	62.50	51.02	50.10
d=3	*	*	696.16	187.50	107.14	77.32	75.23
d=4	*	*	*	498.11	166.67	104.17	100.40
d=5	*	*	*	3125.00	250.00	131.58	125.63
d=6	*	*	*	*	375.00	159.57	150.91
d=7	*	*	*	*	583.26	188.17	176.23
d=8	*	*	*	*	996.22	217.39	201.61

* values $> 10,000$

The fpr of the LT is normally kept at a very low value(< 0.0001). Considering d=3, which is the average degree of an Internet router (Figure 6.14), and fpr of the FT as 0.2, from Equation (6.5), we get the expected value of fpr in SFT as

$$E(fpr) = 498.11 \times 0.0001$$

$$= 0.0498$$

$$< 0.05$$

In other words, 5% of the SFT queries will return atleast one false source.

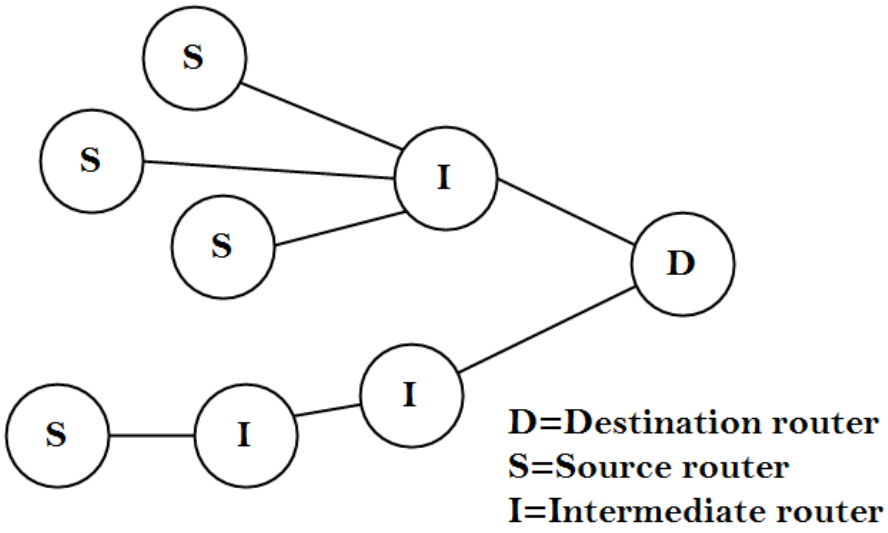
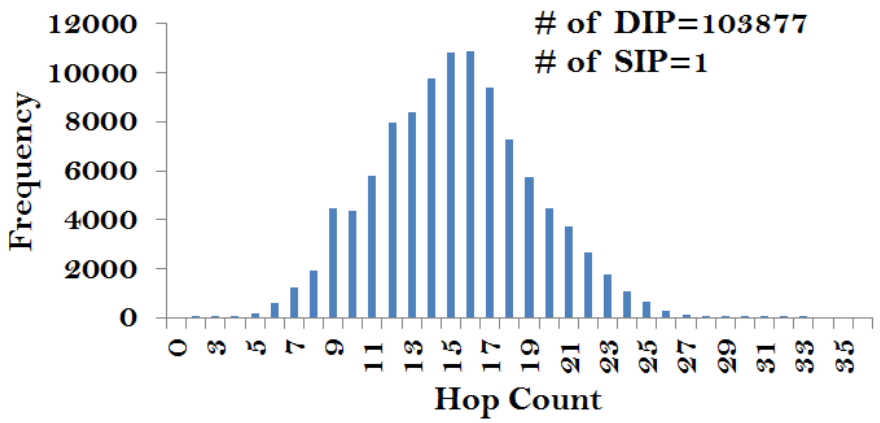


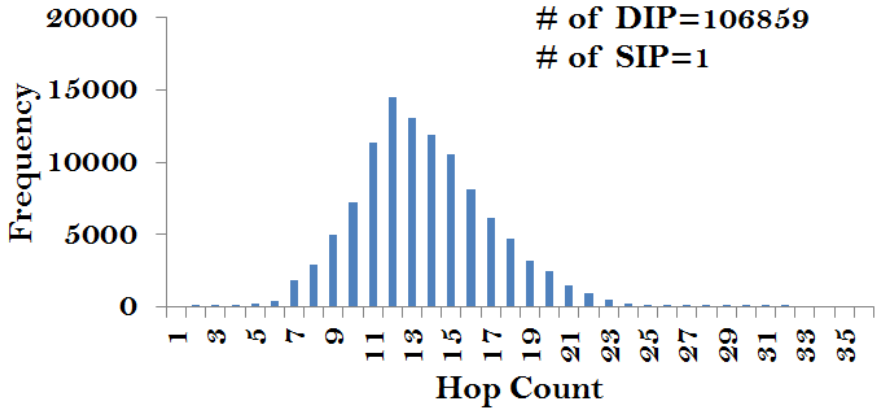
Figure 6.11: Pictorial representation of preprocessed skitter topology

7.4.2.1 fpr Analysis in Real Internet Topology Dataset

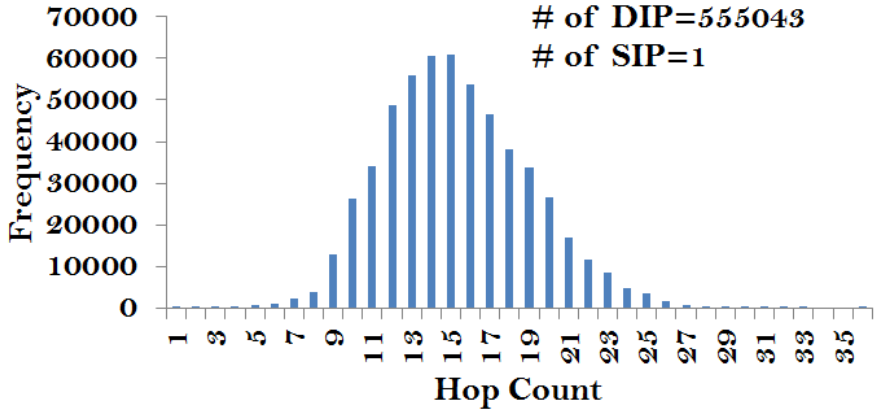
To evaluate the fpr of SFT we use CAIDA's LNK0304[64] Internet topology dataset. Skitter maintains trace route informations from selected monitoring points to different destinations. The datasets contain both complete as well as incomplete paths. Also for a single destination there are multiple paths in some of the datasets. For our experiments we consider only the complete paths. In case of multiple paths to the same destination from the monitoring point, we chose the longest path sequence. Thus, after preprocessing we get a tree, where the root of the tree is the destination and the leafs are the sources. Out of skitter's 25 monitoring points, we chose the topology generated from m-root, f-root and mwest monitoring points, in our experiments. However, topology related to other monitoring points also exhibit almost same characteristics.



(a) m-root

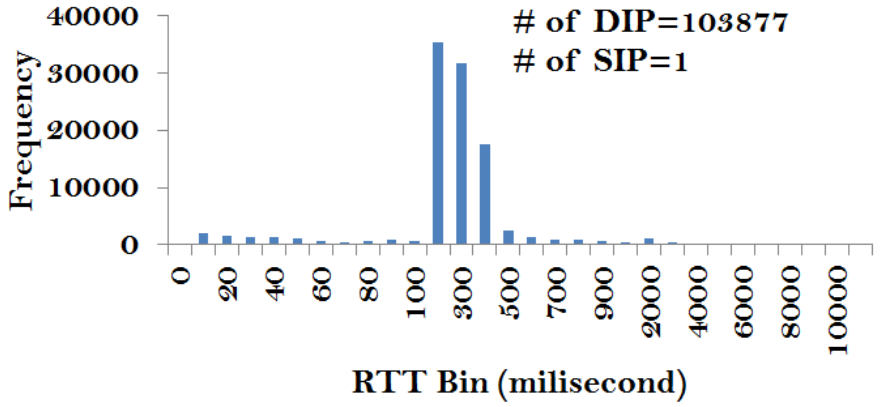


(b) f-root

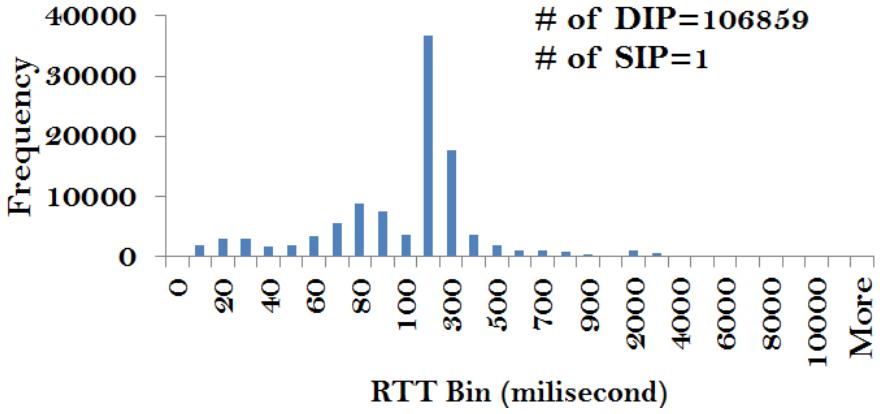


(c) mwest

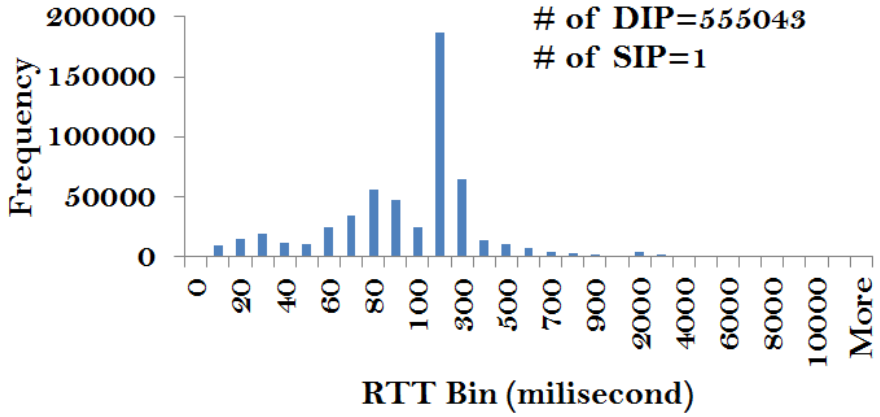
Figure 6.12: hopcount distribution of SKITTER dataset



(a) m-root

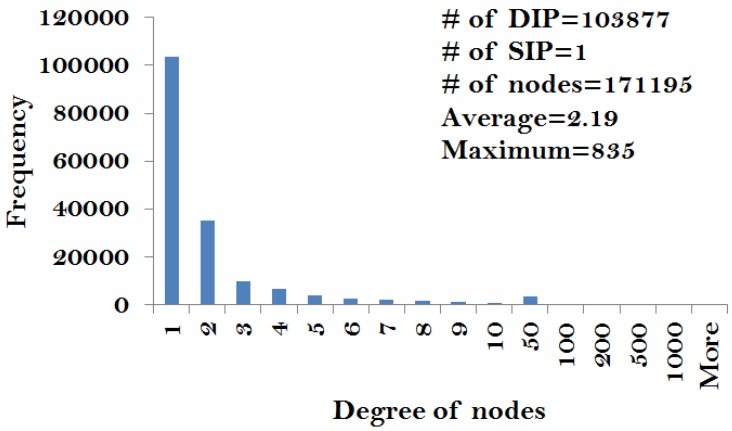


(b) f-root

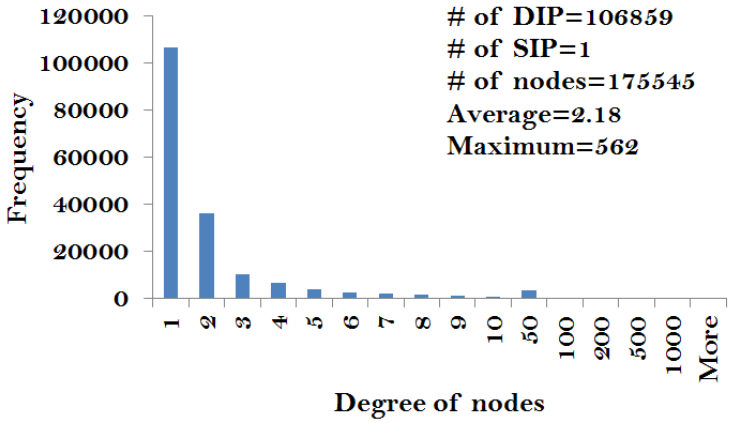


(c) mwest

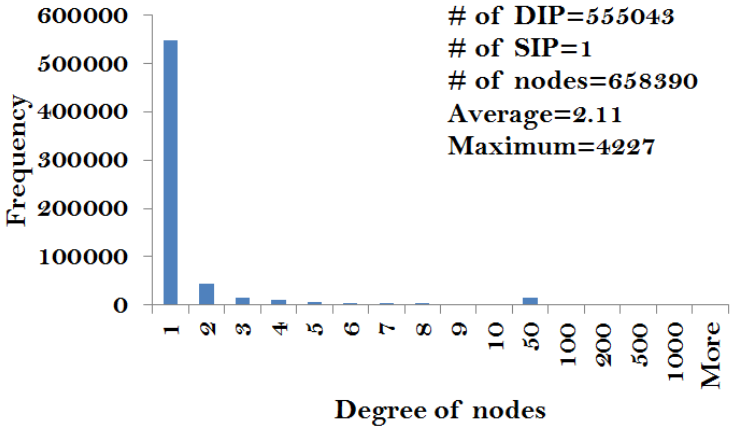
Figure 6.13: RTT distribution of SKITTER dataset



(a) m-root

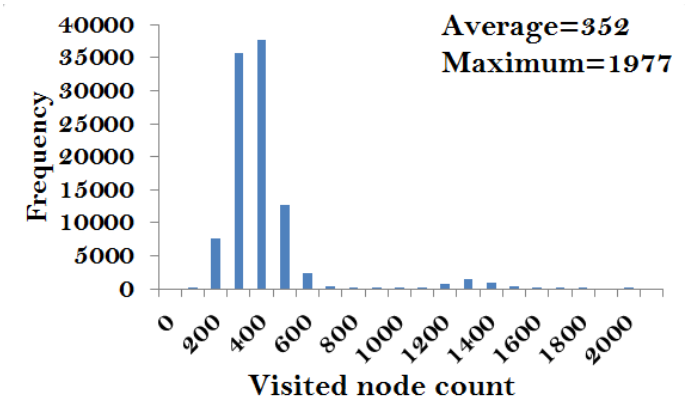


(b) f-root

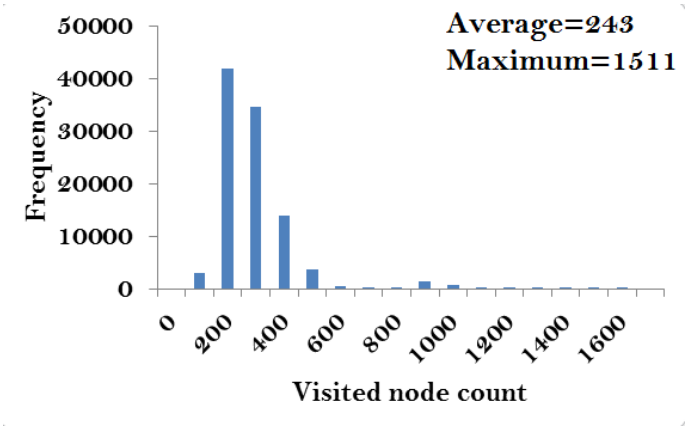


(c) mwest

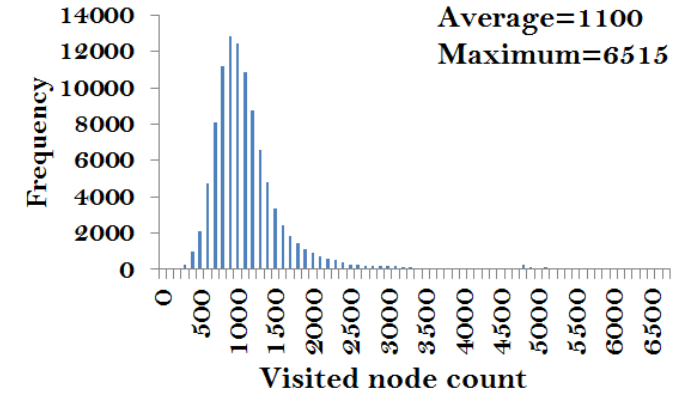
Figure 6.14: degree distribution of SKITTER dataset



(a) m-root



(b) f-root



(c) mwest

Figure 6.15: Distribution of visited nodes by a traceback query in SKITTER dataset

Figure 6.12 shows the hop count distribution from different sources to a specific monitoring point. We see that the average hop count is approximately 15.

Figure 6.13 shows the RTT distribution of the probe messages from different sources to a three monitoring points. We see that the in most of the cases RTT is below 500 millisecond. This observation validates our assumption of $(ppt + qpt)$ to be very small compare to the typical real time detection time of 5 minutes in lemma 2.

Figure 6.14 shows skitter graph nodes degree distribution corresponding to three different monitoring points. We see that the average degree of a node is < 3.0. However, the maximum degree of the mwest topology(4227) is comparatively larger that that of m-root(835) and f-root(562) topologies.

Figure 6.15 shows the distribution of the visited nodes by a single traceback query corresponding to the topologies obtained from three monitoring points. The $false\ positive\ rate$ of the Bloom filters is set to 0.10. The average number of visited nodes in mwest, m-root and f-root monitoring points are 1100,352 and 243 respectively. Comparing with Figure 6.14, we see that mwest topology contains a set of routers with large degree. When a traceback query goes through such a high degree router, the query is spread to all the neighbors, causing the query to visit more number of routers. From equation (6.5) we see that in all the cases, we get the expected value of $false\ positive\ rate$ in SFT as

$$E(fpr) = 1100 \times 0.0001$$

$$= 0.11$$

I.e. On average 11% of the SFT traceback queries will return at least one false source of the queried flow. From Table 6.4 we can see that on a high speed link, (like CAIDA), a Bloom filter with 0.1 false positive rate will need $\leq 50MB$ of storage to facilitate SFT.

6.3.3 Computational Overhead At Intermediate Routers

As we discussed in section 6.3.1, a router marks a packet as soon as it is received at an interface. Based on the mark, a router decides to log a packet. The marking operation time is comparatively negligible than the log operation time. In SFT a log operation includes the execution of k hash functions, where each hash function typically includes one or more multiplication and division operations based on their type. Since the execution of one hash function is completely independent of the

other $k-1$ hash functions, parallel executions of these hash functions are possible. Also the hardware implementation of the Bloom filters allows the logging of a packet at line speed. In our implementation of Bloom filter, the hash functions are implemented as universal hash functions.

6.3.4 Traceback Time

The traceback time depends on *(i)* the number of intermediate routers to be visited by a query to reach a source along a path and *(ii)* the complexity of searching at each router. The average hop count between any source and destination in the Internet is approximately 15. Thus, a traceback query needs to traverse on average approximately 15 intermediate routers in sequence to reach from destination to the source. Each router has to check at most 2 LOG_TABLEs each containing a pair of Bloom filters. Thus, SFT offers a near real time traceback of the sources.

6.3.5 Storage Requirement

As mentioned in section 6.3, the storage requirement of an SFT enabled router is less than 8% of the packet rate through the router. Other similar approaches like HIT and PPIT require 50% and 39% of the packet rate.

6.4 Discussion

In this chapter, I present a log based IP traceback system with comparatively less storage requirement than earlier approaches. The proposed approach achieves zero false negative without the need of synchronization among the routers. Also, I demonstrate the low margin of false positive rate (< 0.11) of our proposed technique both theoretically and on real Internet topology.