Hotspot-Based Traceback for Mobile Ad Hoc Networks

Yi-an Huang
yian@cc.gatech.edu

Wenke Lee
wenke@cc.gatech.edu

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

ABSTRACT

Traceback schemes are useful to identify the source of an attack. Existing traceback systems are not suitable for *Mobile Ad Hoc Networks* (MANET) because they rely on assumptions such as trustworthy routers and static route topology that do not hold in the ad hoc platform. In this paper, we propose a single-packet traceback solution that is extended from the hash-based traceback scheme [19] but not relying on these assumptions. In particular, our solution is fully distributed and resilient in the face of arbitrary number of collaborative adversaries.

In this paper, we develop a new technique, namely *Tagged Bloom Filters*, as an efficient means to store additional information associated with each incoming packet. The additional information can be used to accurately recover the attack path when an attack packet is queried in a traceback session. Based on this technique, we propose several distributed schemes, collectively called *Hotspot-Based Traceback* schemes, to defeat attacks under different security requirements. We present the protocol design, study possible security caveats and propose the corresponding countermeasures.

We present both theoretical and experimental results using ns-2 [8] simulations to show the effectiveness and efficiency of our approach.

Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: Security and protection

General Terms: Security

Keywords: Intrusion response, traceback, ad hoc networks

1. INTRODUCTION

Mobile ad hoc networks (MANET) consist of a group of wireless and mobile nodes where all nodes take part in forwarding packets for each other. MANET is useful in scenarios where infrastructure support is not available or cannot be relied upon [10].

Since MANET makes use of existing protocols such as TCP/IP, it suffers from many attacks in a similar way as the wired networks do, especially IP spoofing attacks. However, a number of MANET specific vulnerabilities make existing traceback schemes designed for the wired networks unsuitable. In particular, most of the techniques rely on strong assumptions such as trustworthy routers and static routes that are used by multiple packets in the same attack flow. While these assumptions are typically valid in wired networks, they generally do not hold in MANET.

The closest candidate that we can base our work on is the *Source Path Isolation Engine* (SPIE) proposed by Snoeren et al. [19], which only requires a single attack packet as evidence. The major problem with this scheme is its centralized design where a number of trusted global and regional servers to collect and process information are required. In a pure form of ad hoc networks, it is not always feasible to find nodes that can be fully trusted. In this paper, we propose a fully distributed design without this requirement. In addition, our protocol is able to reconstruct the attack path even when the topology has been changed from the time the path was actually used. The original SPIE protocol cannot address this because it relies on static network topology.

We further address the problem when the victim may have very limited resource to handle the necessary computation under a heavy *bandwidth consumption* attack by proposing three different protocols: *Investigator Directed*, *Volunteer Directed* and *Fast Filtering*. We argue that by performing differentiated response actions in different critical levels, we can provide the best trade-off in terms of resource consumption, usability and security.

The rest of the paper is organized as follows: Section 2 presents a formal definition of the problem and assumptions. Section 3 compares a number of existing traceback schemes. We address the SPIE framework in particular, as it is used as our base framework. Section 4 present the basic mechanisms that are used in our protocols. Section 5 discusses the three hotspot-based protocols. Section 6 shows experimental results where we show the effectiveness and efficiency of our protocols. Section 7 reviews the related work. The paper is concluded in Section 8.

2. PROBLEM STATEMENT

The original traceback problem defined in the wired network attempts to identify the true identity of the source of an attack packet. In this paper, we study how to identify at least one malicious node that involves in the attack that is being investigated. The original problem is much harder to be addressed in a highly vulnerable environment such as

MANET, because a malicious router may modify any packet it forwards in an arbitrary way and it is not always possible to reveal the original source based on the modified packet.

2.1 Environment

The MANET environment is generally considered as a peer network. In other words, no node should be trusted more than others. In some special cases, reliable servers may be available (an access point in a hybrid network may be such an example), but any of them becomes a single point of failure because the number of these reliable servers is typically much smaller than the number of peer mobile nodes. In general, a scalable protocol in MANET should not rely on the availability of any reliable nodes.

We assume that an Intrusion Detection System (IDS) agent, which may or may not reside on the victim host (for example, the cluster-based agent [12]), can detect intrusions on behalf of the victim. Therefore, the traceback can be triggered even if the victim itself is compromised. In this paper, we refer to the IDS agent that triggers a traceback session as the **investigator**.

We further assume that there may be multiple collaborative adversaries and any mobile node in an ad hoc network may be compromised by an adversary.

2.2 Secure Communication

In the protocols discussed in this paper, a broadcast authentication protocol is required as the underlying mechanism for the traceback protocols. There are several choices. The common choice utilizes public-key cryptography in a self-organized fashion [5]. Alternatively, we can use light-weighted symmetric cryptography based on one-way hash chains and time synchronization, such as TESLA [11] or μTESLA [15]. Both asymmetric and symmetric primitives have their advantages and disadvantages. In our protocols, we prefer public-key signatures. Since traceback is infrequent, it introduces little impact on the overall performance.

We assume that a pre-deployment key establishment protocol, such as [14], is in place so that key credentials are stored securely in every node during the system initialization stage. The key credentials may be revealed, but only when the node who holds the credential is compromised. In general, node compromise may require significant efforts. Therefore, we also assume that there will be no node being newly compromised during a traceback session.

Furthermore, we assume that traceback messages will eventually reach their destinations as long as the network is fully connected. First, a reliable transmission protocol, such as TCP, is used to prevent accidental packet loss due to wireless congestion or other transmission errors. Second, in order to mitigate the problem where packets may be intentionally dropped by intermediate malicious routers, a number of existing solutions can be considered. One possible solution is to explore the network for alternative uncompromised paths [10], if there are multiple paths available to the destination and the adversary cannot control all of them. Another approach is to require that any router that transmits an authenticated traceback message must ensure that the packet is forwarded by the next hop correctly, using either neighbor monitoring [16] or IDS based solutions. Since we use a reliable transmission protocol, a message being dropped by an intermediate router will result in retransmissions and thus monitoring techniques, for example, a statistics-based IDS [12], can be used to detect such an attack. Note that the source addresses of the traceback messages cannot be spoofed because they are always authenticated. This eliminates the need to run tracebacks recursively.

Note that we do not require normal traffic to be secured in a similar way as the traceback protocols does, because not all protocols or applications can afford the computational costs implied by the authentication schemes.

2.3 Goals

The best result from a traceback solution in the wired network is an attack path from the adversary site to the victim site. Many wired traceback schemes consider only one malicious attack source, and assume that intermediate routers are secure and reliable. In MANET, these assumptions do not hold and thus alternative approaches are needed. In addition, the "first-hop" problem also has to be addressed differently: the attack source may be aware of being traced, thus the best a traceback scheme can do is to identify the first (gateway) router that forwards the attack flow. In a typical wired environment, the adversary and the gateway router belongs to the same administrative domain where audit logs and other local measures often provide sufficient clues to identify the adversary. In MANET, there are no equivalent administrative domains with a physical boundary, thus making the problem much harder.

In this paper, we propose a different approach what we call **hotspot**-based traceback. A hotspot is a suspicious area where one or more unknown adversaries may reside or resided and it is covered by the transmission range of a particular node. The node itself may or may not be malicious. Once a hotspot is identified, offline or online investigation can be conducted there to identify the exact identity of the adversaries. Solutions ranging from neighbor monitoring [16], physical security [21, 6] to human intelligence may be used. Our protocols rely on these underlying mechanisms. It should be noted that there may be multiple hotspots detected in a single traceback. However, hotspot analysis is generally expensive, thus our goal is to find the hotspots as accurate as possible.

We present the related concepts and a more formal problem statement below.

2.4 Definitions

Formally, we define an **attack path** of a specific packet P as the transmission path from the *attack source* to the *victim*. A path does not contain loops, branches, or duplicated nodes.

We say a node (or router) is *malicious* or *compromised* if it can perform arbitrary actions that do not follow normal behavior. Otherwise, it is referred to as *well-behaving* or *non-malicious*. Let us define an **AP fragment** as a consecutive sequence of *well-behaving* routers within the *attack path*. The whole attack path can thus be seen as an interleaving sequence of compromised routers and AP fragments. We define an **observable AP fragment**, OAF, as an AP fragment where all routers compute the same digest[1] of packet P as the victim does. Since packet headers may be modified by compromised routers, not all AP fragments are observable. However, we know the last OAF which ends at the victim must be observable based on the definition of an OAF. We

[1]We will define the digest function in Section 3.1.

define this special OAF as the **essential attack path**, or EAP. Note that the victim itself may or may not be included in EAP, depending on whether it is well-behaving or not.

A hotspot-based traceback scheme is then expected to discover the well-behaving router(s) next to one of the malicious nodes as a hotspot. In addition, a malicious node not in the attack path may want to mislead the traceback investigation by providing false information. Our approach would identify such node directly as a hotspot, thus thwarting these types of attacks.

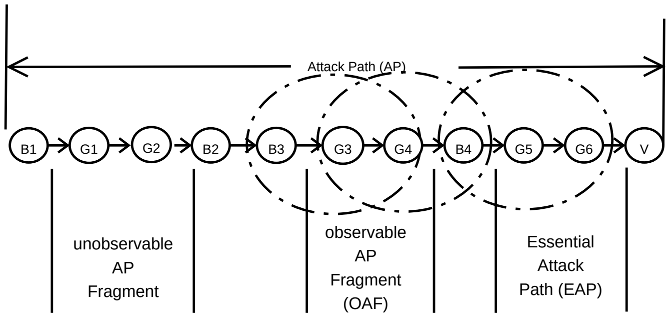


Figure 1: Attack Path

Figure 1 illustrates the example of an attack path from the attack source B_1 to the victim V. Assume all B nodes are malicious (bad) and all G nodes are well-behaving (good). Further assume B_3 alters the packet digest, while B_4 does not (it can still introduce other attacks as we will show later), then we can identify two observable AP fragments, including the EAP, as shown in Figure 1. The hotspot-based scheme may not identify the attack source B_1 because G_1 is not observable any more. But it would find at least some (if not all) of the three hotspots (shown in dash circles) centered at G_3, G_4 and G_5, respectively, because B_3 and B_4 are malicious.

To summarize, the traceback problem can be addressed with hotspot-based traceback by *finding at least one true hotspot and perform hotspot analysis on these hotspots.* In the rest of this paper, we focus on the problem of how to detect hotspots.

3. PREVIOUS PROTOCOLS

Many existing traceback protocols are not suitable for MANET. We illustrate two major problems. First, dynamic topology is a key characteristic of MANET in that the active route between any source-destination pair may vary from time to time. Packet-marking based traceback techniques [18, 20, 24], for example, assume that the same routing path is used by all packets within a single attack flow and thus cannot work effectively in MANET.

Another issue is the trustworthiness of intermediate routers that forward the attack packet. In wired networks, the majority of core backbone routers are well protected and can be assumed secure in most scenarios. Many traceback schemes rely on this assumption.

3.1 The Source Path Isolation Engine (SPIE)

Let us first review some of the basic elements in the SPIE [19] approach.

Infrastructure: In this framework, every SPIE-enabled router runs a *Data Generation Agent* (DGA) where a Bloom filter [3] based digest table is used to store the digests of all packets it forwards. The Bloom filter provides a space-efficient probabilistic membership testing structure, the details of which will be examined in Section 4.2. In addition, there are multiple *SPIE Collection and Reduction* (SCAR) agents where each SCAR agent is responsible for collecting results from all DGA within a network region. Finally, *SPIE Traceback Manager* (STM) controls the whole SPIE system. If an attack packet is detected, a traceback request that contains the digest of the attack packet is sent to STM, which in turn asks all SCARs in its domain to poll their respective DGAs for the relevant packet digest. An SCAR periodically collects the digest tables from its regional DGA agents. A partial attack path can thus be reconstructed by examining the DGA tables in the order they would have been queried if a reverse-path flooding were conducted. Eventually, the attack graph is fully reconstructed by combining the partial graphs from all SCARs.

Digest Input: In the SPIE framework, the digest is computed based on the hash of the 20-byte IP header plus the first 8 bytes of packet payload. Only non-mutable fields in the header are used. This excludes fields such as `Type of Service`, `TTL`, and `Checksum`. It is shown that the non-mutable fields can identify most packets uniquely with the collision rate less than 0.001% percent in a wide area network and 0.15% in a local area network [19]. It should be noted that valid IP packets might be transformed while traversing the network. Examples are fragmentations and ICMP messages. Packet transformation is addressed efficiently in SPIE with a *transform lookup table*, where the additional storage requirement is minimized.

The major problem to apply the SPIE scheme in MANET is that a central authority (STM) and a number of regional agents (SCAR) are required. We have asserted that the requirement for these hosts to be fully trusted is too strong for MANET. Instead, we propose a different distributed scheme, namely, *Hotspot-Based Traceback*.

4. BASIC MECHANISMS

4.1 Overview

First, let us assume that the investigator for a traceback session itself is well-behaving. This requirement will be relaxed later in Section 4.6.

Given this assumption, a natural alternative to the SPIE infrastructure is to use the investigator as the replacement of STM and all SCAR agents. The investigator first broadcasts a query that contains the digest of an attack packet and then collects responses from all routers that have previously forwarded the packet. The dynamic topology in MANET, however, makes the subsequent attack path reconstruction problem much harder. Without a global route topology, we cannot know the order of the routers in the original attack path without additional information. One possible solution is to require every router to remember the timestamp when a packet was forwarded, but comparing the timestamps provided by different routers requires (securely) synchronized clocks, which may be expensive.

Instead, we require each router to record two measures: the TTL value observed from the forwarding IP packet (Section 4.3) and the neighbor list, i.e., the nodes within its own transmission range (Section 4.4). An attack graph can then be constructed by finding all edges between two neighboring nodes, where two nodes become neighbors if they have a TTL difference exactly by one and/or they are in the neighbor lists of each other. Finally, the hotspots can be detected based on the constructed attack graph. We note that both TTL and the neighbor list may be inaccurate due to the existence of malicious nodes and other factors, but it does not prevent us from using them effectively to detect true hotspots. We illustrate the *Attack Graph Construction Algorithm* and *Hotspot Detection Algorithm* in Sections 4.5 and 4.6, respectively.

4.2 Tagged Bloom Filter

The TTL value must be stored along with every packet a router forwards. Maintaining a separate lookup table has a huge storage requirement, and thus defeats the purpose of a Bloom filter. In this subsection, we present an extension to the basic Bloom filter to address this problem.

A *Bloom Filter* [3] is used in SPIE to provide efficient probabilistic membership testing. The basic structure consists of a bit table with m bits and k independent hash functions that each maps an input value to an index into the table. Initially, all bits are set to zeroes. When an element x is inserted, the hashes of x produce k indices and the corresponding bits are set. Later, if the membership of an element y is queried, we compute the hashes of y and assert that y is a valid member if and only if all k bits at these indices are set.

The hash functions must be chosen randomly from a universal hash family so that the hash results are uniformly and independently distributed. Although not required in SPIE, we expect the hash functions to have the property of second preimage resistance (i.e., it is hard to find a different element that has the same hash of a known element). Without this requirement, a compromised node can fabricate an attack packet that is indistinguishable from a non-attack packet, which complicates our design. In practice, we generate a series of k hash functions using the keyed hash function HMAC-SHA1 with k different random keys. We use the HMAC function because it is popular and implemented in many crypto libraries. A more efficient hash function, such as UMAC [2], may also be used.

We propose an extension of the basic Bloom filter where an additional *tag* associated with an element can be stored. It is called a *Tagged Bloom Filter* or TBF. It can be implemented with only minimal modification to the underlying data structure: instead of a single bit, multiple bits are stored in each table entry[2]. Let us assume c bits are necessary for each tag. Our algorithm, however, allows only $2^c - 2$ valid values $\in [0, 2^c - 3]$. $2^c - 2$ and $2^c - 1$ have special meanings, which are hereby referred to as the **invalid tag** and the **empty tag** respectively.

Initially, all entries are initialized to the *empty tag* (which is equivalent to set 1s on all bits. Note that this is different from the basic Bloom filter). The insert operation for an element with tag t examines the k entries corresponding to the element: if an entry contains the *empty tag*, it is changed to t; otherwise it is changed to the *invalid tag* to indicate that a conflict has occurred. Finally, a membership test operation returns either the *empty tag* if at least some of the k entries are empty, the *invalid tag* if all entries contain conflicts, or the smallest tag value from all valid entries. We illustrate these operations in Figure 2.

[2]The *Counting Bloom Filter* presented in [9] also uses multiple-bit entries, but it serves a different purpose and its operations are different.

```
insert(element, $t$) {
  $\forall\ i \in$ [1, $k$] {
    if TBF[$h_i$(element)] = $2^c - 1$
      TBF[$h_i$(element)] $\Leftarrow$ $t$
    else
      TBF[$h_i$(element)] $\Leftarrow$ $2^c - 2$
    end
  }
}

test(element) {
  if $\exists$ TBF[$h_i$(element)] = $2^c - 1$
    return $2^c - 1$
  else
    return $\min_{i \in [1,k]}$(TBF[$h_i$(element)])
  end
}
```

Figure 2: Tagged Bloom Filter Operations

It should be noted that a TBF always has zero false negative rate, i.e., query for any element that has been inserted will always be successful. However, an effective TBF must ensure both the *invalid tag rate* and *false positive rate*, the probabilities when an *invalid tag* is returned and when all k entries are set but the element was never inserted, are sufficiently small. After n items are inserted, the probability that a particular entry is not used by any of the nk hashes is $(1-1/m)^{nk}$. Hence, the invalid tag rate can be computed as

$$\mu = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{-kn/m})^k$$

Similar analysis can show that the ϵ, the false positive rate, is equal to μ. We summarize ϵ and μ in Table 1.

Table 1: Tagged Bloom Filter: Probability Matrix

	returned tag		
	t	$2^c - 2$ (invalid)	$2^c - 1$ (empty)
never inserted	ϵ		$1 - \epsilon$
inserted with tag t	$1 - \mu$	μ	0

We observe that ϵ (or μ) is minimized when $k = \frac{m}{n} \ln 2$, which yields $\epsilon_{min} = \left(\frac{1}{2}\right)^{\frac{m}{n} \ln 2} \approx 0.6185^{\frac{m}{n}}$. The storage requirement can then be computed as cm, or $cnk/\ln 2$ bits if the optimal k is used.

4.3 Relative TTL

Definition 1. We define the c-bit Relative TTL, or $RTTL$, of packet P as

$$RTTL(P) \equiv TTL(P) \mod (2^c - 2)$$

where $TTL(P)$ comes from the IP header of P.

LEMMA 1. *Assume packet P is forwarded from router A to router B where both A and B are well-behaving, the RTTL* *values for P stored on A and B satisfy*

$$RTTL_A(P) = RTTL_B(P) + 1 \pmod{2^c - 2}.$$

PROOF. The proof immediately follows the fact that when a well-behaving router forwards P, the TTL field is always decremented by one. Note that we cannot guarantee this if either A or B is malicious. $\square$

When an incoming packet P is observed, a router adds the packet to TBF with $RTTL(P)$ as the associated tag. In order to determine c, the bit-storage requirement for each table entry, we need to bound the maximal length of a normal route so that *no two routers in a consecutive sequence of well-behaving routers will have the same RTTL values.* The parameter can be determined based on the underlying routing protocol and network topology settings. In the scenarios used in our simulations, we choose $c = 4$, because a maximal route length of $2^4 - 2 = 14$ is sufficient for most scenarios.

The TTL field may be modified by a compromised router in the attack path. However, if we only consider an observable AP fragment (such as the essential attack path EAP), we can guarantee that the RTTL values stored on all routers in the same OAF are continuous and monotonically decreasing (in modulo arithmetic).

4.4 Local Neighbor Lists

Even though the invalid tag rate is fairly small, the chance that at least one router returns an invalid tag in a long attack path can be much larger. This may prevent the attack graph from being fully reconstructed. As a remedy, we obtain the local connection topology from all matching routers. More specially, we obtain $NL(A)$ from router A, the set of neighbors that were within its transmission range when the packet was forwarded. It can be obtained using the standard HELLO broadcast technique: a node A broadcasts a HELLO(A) message with TTL=1 during every `HELLO_INTERVAL`. Node B which receives the HELLO(A) will include A in its $NL(B)$. The entry will be expired after one `HELLO_INTERVAL`, unless a subsequent HELLO(A) has arrived by that time. Note that we do not require HELLO messages to be authenticated. Therefore, it is possible for a malicious node to hide its own address completely or masquerade as another node. Even without a malicious neighbor, the neighbor list may not be identical to the topology when the packet was actually forwarded, due to wireless collision and node mobility. To ensure that neighbor lists can return useful results, we assume that the global parameter ρ, defined below, should be sufficiently close to 1.

Definition 2. ρ is defined as the lower bound of $Pr(A \in NL(B))$ for all distinct node pairs A and B, given that A and B were well-behaving neighbors of each other when a packet was forwarded within the latest `HELLO_INTERVAL`.

4.5 Attack Graph Construction

At the end of a traceback session for an attack packet P, an investigator collects summaries in the form of $S_i \equiv \{R_i, RTTL_{R_i}(P), NL(R_i)\}$ from all routers where P was matched. We assume that the investigator also generates a summary on behalf of the victim, of which the RTTL tag is always valid because it is obtained directly from the attack packet. We note that other summaries may come from three sources: well-behaving routers that match the digest; well-behaving routers with a false positive in its TBF; and arbitrary malicious nodes. Algorithm 1 returns an attack graph AG based on the summaries, where VIC is the address of the victim.

Algorithm 1:

```
Attack_Graph({$R_i$, $RTTL_{R_i}(P)$, $NL(R_i)$}, $VIC$, $c$) {
  $V \Leftarrow \{R_1, R_2, \ldots\}$
  $E \Leftarrow \{\}$
  $\forall\ x, y \in V$ s.t.
      ($RTTL_y(P) \neq 2^c - 2$
      $\wedge\ RTTL_x(P) = RTTL_y(P) + 1 \pmod{2^c - 2}$
      $\wedge\ x \neq VIC$)
    $E \Leftarrow E \cup \{x \to y\}$
  $\forall\ x \in V$ s.t. ($RTTL_x(P) = 2^c - 2$) {
    $\forall\ y, z \in V$ s.t.
        ($x \in NL(y)\ \wedge\ x \in NL(z)$
        $\wedge\ RTTL_z(P) \neq 2^c - 2$
        $\wedge\ RTTL_y(P) = RTTL_z(P) + 2 \pmod{2^c - 2}$
        $\wedge\ y \neq VIC$) {
      $val = RTTL_z(P) + 1 \pmod{2^c - 2}$
      Generate a pseudo node $x_{val}$
      $\forall\ y_i, z_i \in V$ s.t.
          ($RTTL_{y_i}(P) = RTTL_y(P)$
          $\wedge\ RTTL_{z_i}(P) = RTTL_z(P)$)
        $E \Leftarrow E \cup \{y_i \to x_{val},\ x_{val} \to z_i\}$
    }
  }
  return $AG \equiv \langle V, E\rangle$
}
```

The algorithm runs in $O(|V|^2)$ in the worst case. First, lines 4-8 (case 1) add a route edge $x \to y$ if x and y have consecutive and valid RTTL tags. Second, lines 9-22 (case 2) add two route edges $y \to x \to z$ if x has an invalid tag but both y and z have valid tags that are differed by two and they are both neighbors of x. If there is another satisfying sequence u, x, w where x may be assigned a different RTTL, we add route edges for them as well (since adversaries may add arbitrary neighbor relationships, allowing only one such sequence may introduce attacks that prevent the true edges from being added). By generating a pseudo x separately for each possible RTTL tag (line 16), we prevent invalid path traversals, such as $y \to x \to w$ in the previous example. Pseudo nodes with different tag values are considered different nodes, therefore may be traversed within a single route path.

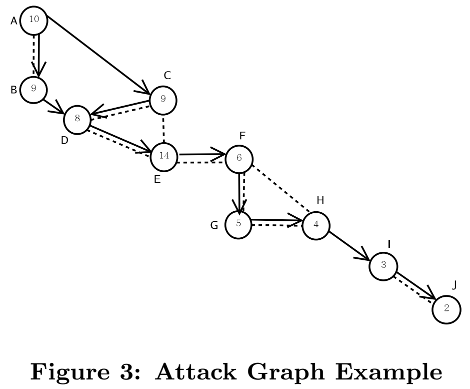


Figure 3: Attack Graph Example

Figure 3 illustrates a possible output from Algorithm 1 with $c = 4$ where letters A through J denote the output vertices and the arrows represent the output edges. The dashed lines reflect the local connection topology obtained from the neighbor lists, i.e., x and y are dash-connected if either $x \in NB(y)$ or $y \in NB(x)$. $I \to J$ and $D \to E \to F$ demonstrate cases 1 and 2, respectively.

The following theorem shows that with high probability, an edge in some OAF is embedded in the output attack graph AG. We note that EAP, the final OAF, should include VIC because the investigator, on behalf of the victim, is assumed to be well-behaving.

THEOREM 1.

$$Pr(E \in AG | E \in \text{ some } OAF) \geq (1-\mu)^2(1+\lambda\mu\rho^2)$$

where μ is the invalid tag rate defined in Section 4.2 and λ is the number of well-behaving sibling routers of E. Node z is a **sibling router** *of edge $E = x \to y$ if either $z \to x$ or $y \to z$ is in the* attack path. *An edge that connects to the victim is considered to have an additional (virtual) well-behaving sibling router on its right.*

PROOF. Assume the attack path consists of L nodes: $R_0 \to R_1 \ldots \to R_L$ where R_0 is the attack source and R_L is the victim. We also add a virtual well-behaving R_{L+1} after R_L to simplify the discussion. For each R_i, we define $V(R_i)$ to be the event that R_i returns a summary and it contains a valid RTTL tag, i.e., $V(R_i) \equiv RTTL_{R_i}(P) < 2^c - 2$.

Consider edge $E_i = R_i \to R_{i+1}$ where $i \in [1, L-1]$ and both R_i and R_{i+1} are well-behaving, we have:

$$\begin{aligned} Pr(E_i \in AG) \geq\ & Pr(V(R_i) \wedge V(R_{i+1})) \\ &+ \rho^2 Pr(V(R_i) \wedge \neg V(R_{i+1}) \wedge V(R_{i+2})) \\ &+ \rho^2 Pr(V(R_{i-1}) \wedge \neg V(R_i) \wedge V(R_2)) \end{aligned} \tag{4.1}$$

where the first term comes from case 1 (Algorithm 1), while the second and third come from case 2. Also note that Definition 2 guarantees that $Pr(R_i \in NL(R_{i+1})) \geq \rho$ and $Pr(R_{i+1} \in NL(R_i)) \geq \rho$.

We now derive $Pr(V(R_i))$. First, it may be arbitrarily small if R_i is malicious. Second, $Pr(V(R_L)) = 1$ because the investigator always returns a valid RTTL on behalf of the victim. Otherwise, $Pr(V(R_i)) = 1 - \mu$. By computing (4.1) using these $Pr(V(R_i))$ values, we prove the theorem. □

Based on Theorem 1, the probability that the essential attack path of length l is fully embedded in the output attack graph is

$$\begin{aligned} P_{EAP}(l, \mu, \rho) &= \prod_i Pr(E_i \in AG) \\ &\geq (1-\mu)^{2l}(1+2\mu\rho^2)^{l-1}(1+\mu\rho^2). \end{aligned}$$

For instance, $P_{EAP}(13, 0.00781, 0.85) \geq 0.938$. It corresponds to the longest path with the number of bits per entry $c = 4$, the number of hashes $k = 7$, and $\rho = 0.85$.

4.6 Hotspot Detection

Theorem 1 shows that with high probability, every observable AP fragment OAF is fully embedded in the output attack graph, i.e., $OAF \subseteq AG$. Algorithm 2 searches for all possible hotspot(s) in AG based on this result where IV is the address of the investigator and τ is a parameter to be determined. In this algorithm, we define a V-path to be a maximal path embedded in AG that ends at victim VIC. A V-path is maximal in the sense that any V-path is not a proper subset of another V-path. We refer to the first τ nodes of a V-path as *its head with size τ*.

Algorithm 2:

```
Hotspot({$RTTL_{R_i}(P)$}, $AG = \langle V, E \rangle$, $VIC$, $IV$, $c$,
  $\tau$) {
  $S \Leftarrow \{IV\}$
  $T \Leftarrow \{\}$
  Find all $V$-paths using a reverse depth-first
    search from $VIC$
  $\forall$ $V$-path $VP \equiv R_1 \to R_2 \to \ldots \to VIC$ {
    $T \Leftarrow T \cup$ all nodes in $VP$
    $S \Leftarrow S \cup \{R_1, R_2, \ldots R_\tau\}$
  }
  $\forall x \in V - T$ s.t. $(RTTL_x(P) \neq 2^c - 2$
      $\wedge$ ($\neg \exists z$ s.t. $x \to z \in E$
        $\vee$ $\neg \exists y$ s.t. $y \to x \in E$)) {
    $S \Leftarrow S \cup \{x\}$
  }
  return $S$
}
```

Lines 5-6 in Algorithm 2 find all V-paths by performing a depth-first search on the reverse graph of AG (obtained by reversing the directions of all edges in AG) and returning the (forward) paths from each leaf to the root VIC. Lines 7-10 return the head of every V-path with size τ as hotspots. Theorem 2, to be given soon, shows that with high probability, at least one true hotspot can be found through V-path searching. Lines 11-15 find additional hotspots within nodes that do not have a V-path to VIC. Here, a node x is identified as a hotspot if it does not have either an outgoing or an incoming edge. It can only happen if (a) x is malicious, (b) x is the first node of an OAF, or (c) x is the last node of an OAF. In all these cases, x is a hotspot.

We relax the earlier assumption that the investigator must be well-behaving here, by including IV directly (line 3) as a hotspot. This is useful because it thwarts a potential attack where a malicious investigator starts a traceback with a normal packet captured elsewhere.

Figure 4 demonstrates an example attack graph where Q is the victim. Algorithm 2 returns $\{A, O\}$. They correspond to V-paths $A \to \cdots \to N \to O \to P \to Q$ and $O \to B \to \cdots \to N \to A \to P \to Q$ (there are two other V-paths, but they also return either A or O). Consider a different attack scenario where H is compromised and chooses not to respond, the algorithm will return {I, G} instead, and we can see that H becomes a hotspot target.

Note that the true EAP may be one of these V-paths, but it is not necessary for us to find out the exact EAP, because the number of possible V-path heads are much smaller than the number of V-paths. The following theorem further characterizes the V-path searching procedure.

THEOREM 2. **V-path searching** *Given that every OAF is fully embedded in AG with high probability (P_0), the probability that V-path searching in Algorithm 2 returns at least one true hotspot is at least $P_0(1-\epsilon^{\tau'})$ where $\tau' = \min(\tau,$ minimum length of all V-paths).*

PROOF. If there is a V-path with length 1, i.e., it contains VIC only, VIC is obviously a hotspot and the theorem is proved. Otherwise, assume that there is at least one V-path VP with length > 1 and w is the first node of VP.

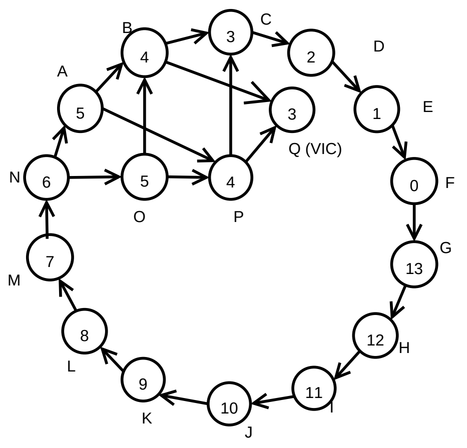


Figure 4: Hotspot Detection Example

Define an *equivalent set of* v to be the set of all nodes with $RTTL = v$ (VIC is a special case that does not belong to any equivalent set). If $RTTL_w(P) = x$, we call W the equivalent set of x (i.e., $\forall w' \in W, RTTL'_w(P) = RTTL_w(P)$), and U the equivalent set of $x-1 \pmod{2^c - 2}$. The construction of AG guarantees that $\forall u \in U \wedge w \in W$, edge $u \to w \in AG$. Since w is the head of VP, either there is no node in U or all nodes in U are connecting to other nodes in W somewhere in VP. Hence, $|U| < |W|$.

For all $w' \in W$, by simply exchanging w and w' in VP, we can obtain another valid V-path with w' as its head (of size 1). Therefore, we have $W \subseteq S$, i.e., either all nodes in the same equivalent set will be returned as hotspots or none of them will be returned.

Assume that every OAF is fully embedded in AG and there is no false positive outputs from the TBF of a well-behaving node, i.e., $\epsilon = 0$. We can assert that there is at least one node w' in W that is either malicious or the first node of an OAF, in other words, a hotspot. Otherwise, every node in W must have a preceding well-behaving node in the same OAF that must be in U. Thus, we should have $|U| \geq |W|$. This leads to a contradiction.

We now consider the general case $\epsilon \geq 0$. A false positive node has the same effect as a malicious node, but the algorithm will fail if we only return false positives as hotspots. The probability that a false positive is chosen as the head (with size 1) of a V-path is at most ϵ. Therefore, the probability that all τ' nodes in the head of every V-path are false positives is at most $\epsilon^{\tau'}$. $\square$

Furthermore, we can bound the total number of hotspots returned by Theorem 2. First, note that a normal route does not exceed $2^c - 2$ hops. A compromised router B, however, can break the maximum-length rule by (1) introducing an artificial TTL gap so that two well-behaving routers separated by B may have the same RTTL tag, (2) forwarding the packet through a non-optimal route. Given that a single B can only increase the maximum route length by at most $2^c - 2$, we can derive that an attack path containing at most q compromised routers will have at most $1+q$ well-behaving nodes in any equivalent set, therefore at least $\frac{1}{2+q}$ outputs from V-path searching are true hotspots.

5. HOTSPOT-BASED TRACEBACK PROTOCOLS

Bloom Filter Capacity: The digest table, implemented as a *Tagged Bloom Filter* (TBF), is used to record the packets captured at each router. It is sufficient to only store the traffic within the most recent `TRACEBACK_INTERVAL`, which is defined as the largest time gap between the time a router recorded a packet and the time the packet may be queried for the purpose of traceback. `TRACEBACK_INTERVAL` is associated with the capability of IDS agents: The faster an IDS agent can detect attacks, the shorter the `TRACEBACK_INTERVAL` has to be. Note that `TRCEBACK_INTERVAL` may be larger than `HELLO_INTERVAL`. It introduces a problem because the neighbor list when a traceback is triggered may not be consistent with the one when the packet was forwarded. The problem can be solved by maintaining a set of buffers where each buffer $B_i \equiv \langle TBF_i, NL_i \rangle$ stores the data for each `HELLO_INTERVAL`. To guarantee that the needed buffer can be available within `TRACEBACK_INTERVAL`, we only need to maintain up to $max = \frac{\texttt{TRACEBACK\_INTERVAL}}{\texttt{HELLO\_INTERVAL}} + 1$ buffers and reuse them cyclically. When a packet is queried, we start with the most recent buffer and check backwards if there is no match in the current buffer, until all max buffers have been checked.

Protocol 1 - Investigator Directed: We first introduce the basic protocol, where the investigator sends out traceback request, collects the response from all matching routers, and computes the hotspot list. The protocol involves three rounds:

- Request The investigator broadcasts a traceback request to the entire network. The request is defined as `INV(vic, iv, seq, hash)`[3]. , where `vic` and `iv` are the addresses of the victim and the investigator respectively, `hash` is the digest of the attack packet `P`, and `seq` is a sequence number that is automatically incremented by the investigator for every new `INV` in order to prevent replay attacks.

- Reply Every node responds to the `INV` message by querying its digest table, after it determines that the message contains a valid signature and is not a duplicate. If the digest table contains no match, the request is silently ignored. Otherwise, the router sends a summary of its own results, `ACK(vic, seq, router, rttl, nl)`, back to the investigator, where the router's address, the $RTTL$ tag associated with the packet, and the neighbor list associated with the matching TBF are included.

- Collection The investigator waits for the worst-case RTT (round-trip time) plus the largest router processing time to ensure that all matching routers have responded. It then runs the *Attack Graph Construction Algorithm* (Algorithm 1) and the *Hotspot Detection Algorithm* (Algorithm 2) sequentially. A list of hotspots is reported to an offline authority where hotspot analysis will be conducted.

Protocol 2 - Volunteer Directed: The *Investigator Directed* protocol suffers from the *bandwidth consumption* at-

[3]Although we do not state explicitly, all protocol messages use the broadcast authentication protocol to protect its authenticity and integrity.

tack where the investigator may not have sufficient resources to receive and handle all ACK messages. One solution is to give priority to ACK messages and other traffic may be discarded if necessary. However, the adversaries may still be able to launch Denial-of-Service attacks with bogus ACK messages because verifying the authenticity of an ACK message is a non-trivial operation. Instead, we propose a new protocol, namely *Volunteer Directed.* It chooses a number of third party nodes that perform ACK collection and hotspot computation independently. One possible way to choose these nodes is to deploy behavior-based trust management [16] so that only the most trusted servers are used. We propose here a more general solution that does not require such an infrastructure. It is similar to *Investigator Directed* but with the following differences:

1. Two additional fields, `rttl` and `nl`, containing the RTTL tag and the neighbor list observed by the victim, are attached to the INV message.

2. Every router that receives INV determines whether it wants to be a volunteer with probability α, prior to digest table lookup. If it volunteers, a `VLT(vic, seq, volunteer)` message is broadcast.

3. A matching router caches the ACK message. Instead of being sent to the investigator, the cached ACK message is transmitted to a volunteer from whom a valid VLT message is received. To ensure the cache size does not increase infinitely, an expiration time is associated with every cached ACK.

4. A volunteer performs the same *Collection* round as performed by the investigator in Protocol 1.

By having multiple volunteers, the communication overhead slightly increases. Hence, one may prefer to suppress duplicated volunteers when the first volunteer broadcasts its intent. However, this can also help adversaries because a malicious node can volunteer quickly to suppress others, and then never deliver the hotspot list to the offline authority. There is another more serious problem. We can ensure that a malicious node does not forge summaries from well-behaving routers because they are signed (the signature can be examined by the offline authority if necessary). But a malicious node can deny that some summaries have ever been received, thus changing the attack graph and eventually altering the resulting hotspot list. Unless a complicated protocol that implements non-repudiation is used, this situation cannot be prevented. Our approach, instead, is to keep multiple volunteers with the hope that at least one of them is well-behaving.

Protocol 3 - Fast Filtering: In the third protocol, we consider the most critical scenario: the investigator requires immediate response to filter out the attack flows. We define an attack flow as a number of attack packets delivered from the same attack source but possibly with different (spoofed) source addresses. The previous protocols require offline analysis and are thus not suitable to guide packet filtering. Instead, we can deploy a *Fast Filtering* approach where filtering is conducted on the routers that actually forward the attack packet in *real-time.* This protocol works by assuming the topology does not change too frequently, thus most of these routers will still be used to forward subsequent packets in the same attack flow.

A straightforward approach requires all routers that contain a matching digest of an attack packet to drop subsequent traffic destined for the victim. The major problem with this approach is false positives. Not only does the attack flow get dropped, normal flows destined for the victim may use these routers as well, and thus suffer from the unconditional dropping. Instead, our approach assigns a different dropping probability for each router based on its distance to the victim. Intuitively, a smaller dropping probability should be used on a router closer to the victim, because the chance that this router is used by normal flows is likely to be larger.

Assume we choose a dropping probability p_i for a router whose distance to the victim is i, the *Fast Filtering* protocol can then be briefly described as follows. A matching router R estimates its distance to the victim as $i = (RTTL_R - RTTL_{VIC}) \pmod{2^c - 2}$. It then adds a filter rule that drops all packets destined for the victim with probability p_i. The filtering rule can later be removed if so requested by the victim.

How to assign these probabilities optimally is a hard problem. We illustrate one solution based on a simplified theoretical analysis with a random compromise model. Consider a uniform distribution in an infinitely large map where each node has d neighbors (Figure 5). The sources of the traffic destined for V are randomly distributed, while the optimal route with the shortest path can always be found. We assume every node may be compromised with probability β. Assume the attack path is X to V, where X is the attack source and V is the victim. We assume that the attack path contains k intermediate routers, R_1 to R_k, where the index stands for the distance to V.

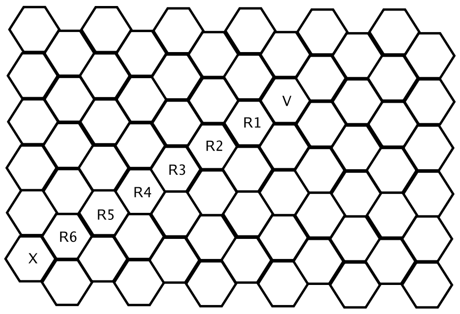


Figure 5: An Ad Hoc Network with Uniform Distribution ($d = 6$)

First, we observe that there are $i \times d$ nodes that are i hops away from V. Therefore, the probability that a node which is at least i hops away from V chooses R_i in its route to V is $\frac{1}{id}$. Hence, the probability that a packet from a random source to V uses router R_i (but not R_{i+1}) is $\frac{1}{i(i+1)d}$. In this case, we denote R_i as the *first filtering router* of this packet.

Assume that router R_i drops any packet destined for V with probability p_i. A compromised router would never drop attack packets, but drop packets in normal flows with probability p_i[4]. We define PA as the probability that an attack packet from X is dropped and PB as the probability that a normal packet is dropped. The objective of the optimization

[4]A slightly different attack model can use probability 1 instead. We do not address this variation here but the analysis will be similar.

problem can be defined by maximizing PA while requiring PB to be no more than γ, the highest loss rate that can be tolerated on normal traffic.

An attack packet is not dropped when all intermediate routers are either compromised or not dropping the packet (with probability $1 - p_i$). A normal packet is not dropped when all routers from R_i to R_1 choose not to drop it where R_i is the first filtering router of the packet. The optimization problem can thus be formalized as:

$$\text{maximize } PA = 1 - \prod_{i=1}^{k}(1 - p_i(1-\beta)), \text{ given}$$

$$PB = \sum_{i=1}^{k} \frac{1}{i(i+1)d}(1 - \prod_{j=1}^{i}(1-p_j)) \le \gamma$$

The optimal solution to this problem can be given by $p_1 = \cdots = p_{i-1} = 0$, $p_i = (d\gamma + \frac{1}{k+1})i(i+1) - i$, $p_{i+1} = \cdots = p_k = 1$, where $i = \lfloor \frac{1}{d\gamma + \frac{1}{k+1}} \rfloor$. The optimal PA is $1 - \beta^{k-i}(1 - p_i(1-\beta))$.

In practice, we do not know k precisely. An estimation with the average route length can be considered as an alternative.

6. SIMULATION RESULTS

6.1 Platform Setup

To evaluate the hotspot-based traceback protocols, we use ns-2 [8], a well known simulator that is widely used for evaluating ad hoc protocols. We use the two-ray ground reflection radio propagation model and the AODV [17] routing protocol. Our scenarios contain 25 nodes randomly deployed in a square space of 2,500 m by 300 m to guarantee that the average route length is not too short. The radio transmission rage is 250 m. We deploy 20 source-destination pairs with UDP/CBR (Constant Bit Rate) traffic and the average traffic rate is 4 packets per second (we also tested TCP traffic and the results are similar). Each experiment lasts for 1,000 seconds and the average results of 10 experiments are reported.

We use a random waypoint model with maximum velocity = 20 m/s and pause time = 50 seconds to show the effectiveness of our scheme in a *high mobility* scenario. The number of bits per RTTL tag $c = 4$, and the size of a V-path head $\tau = 1$. These parameters are used throughout our experiments unless stated otherwise.

We assume that it is sufficient to store traffic within the last minute for traceback purposes, i.e., `TRACEBACK_INTERVAL` = 60 seconds. We choose `HELLO_INTERVAL` to be 5 seconds, by which our simulation shows the neighbor list consistency probability $\rho \ge 0.85$ under different levels of mobility.

6.2 TBF Storage Requirement

It is important to estimate the storage requirement because a TBF must be fully stored in memory to support real-time membership queries. For simplicity at this moment, assume a large TBF is used where all incoming packets within the most recent `TRACEBACK_INTERVAL` are recorded. Assume the average packet size is 1,000 bits, the number of bits per RTTL tag $c = 4$, the number of hashes $k = 7$ (which corresponds to a false positive rate ϵ or invalid tag rate $\mu \approx 0.0781\%$). We first consider a normal operating environment that delivers the user experience similar to a DSL connection where the average bandwidth does not exceed 1Mbps. A TBF-enabled router needs $1M/1000{*}60{*}4{*}7/\ln 2 \approx 2.4M$ to store a TBF table for one minute's data, which should not be a constraint for most modern devices. Secondly, we consider the worst case where a high-end scenario with the full capacity provided by the 802.11a or 802.11g wireless standard is used and the maximum bandwidth over a wireless link is 54Mbps. The footprint rises to about $131M$. While this is affordable even based on commodity hardware, the requirement can be heavily reduced in practice because the achievable bandwidth is much lower due to wireless collision and other physical constraints. Instead, a straightforward linear directory where each entry contains a 160-bit SHA-1 packet digest and a 4-bit tag will take approximately $(160 + 4) * 54M/1000 * 60 = 531M$ for one-minute storage of traffic with full 802.11a(g) capacity.

In our simulations, we choose `HELLO_INTERVAL` to be 5 seconds so that a total of $60/5 + 1 = 13$ buffers are used. A careful reader would find out that the worst-case false positive rate with the same k will be larger when multiple (and smaller) buffers are used. By using $k = 11$, we obtain the same false positive rate compared with a full TBF when $k = 7$. This corresponds to the memory footprints of $3.8M$ and $206M$ in the low-end and high-end scenarios respectively.

6.3 Hotspot Detection

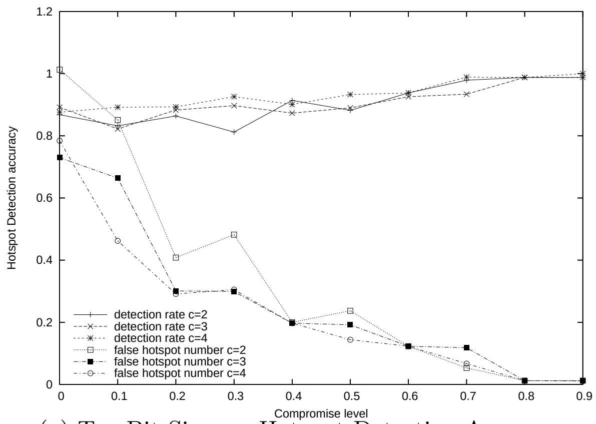
In order to evaluate the effectiveness of the *Hotspot Detection Algorithm*, we use a random compromise model, where every node may be compromised with an equal probability β. We consider the following two attacks:

1. A router in an attack path is compromised and modifies the TTL field to a random value.

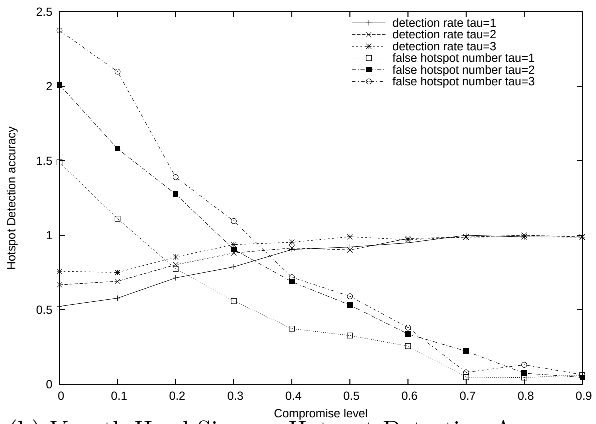
2. A compromised node sends a summary for a traceback session with a random RTTL and a random neighboring list.

Note that a compromised router that performs Attack 1 will also perform Attack 2 with probability β in response to a traceback request or simply not respond.

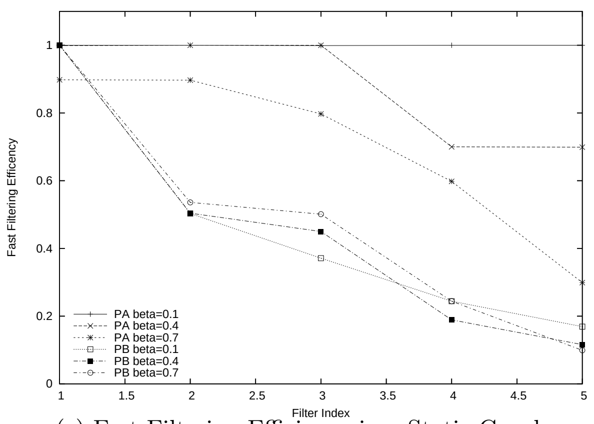
We use the *detection rate*, the probability when at least one true hotspot is detected from V-path searching, as the effectiveness measure. We also use the *average number of false hotspots* per traceback session as the efficiency measure, because it reflects the extra overhead when hotspot analysis is performed. Figure 6(b) shows both measures when the compromise level β ranges from 0 to 1 with different values of c, the number of bits for each RTTL tag. We observe that the false hotspot number decreases when the compromise level increases, because the V-path searching will more likely to stop at a malicious router under a high compromise level. We also observe that the false hotspot number decreases significantly when c changes from 2 to 3 but the difference is hardly distinguishable when c changes from 3 to 4. It shows that most routes does not exceed $2^3 - 2 = 6$ hops. This matches the topology setting used in our experiments. In the mean time, the detection rate increases slightly when the compromise level increases because it becomes less and less likely for a false-positive well-behaving node to be the head of a V-path. Overall, we can obtain 85% accuracy in detecting at least one true hotspot while the average false hotspot number is less than one in the worst case.



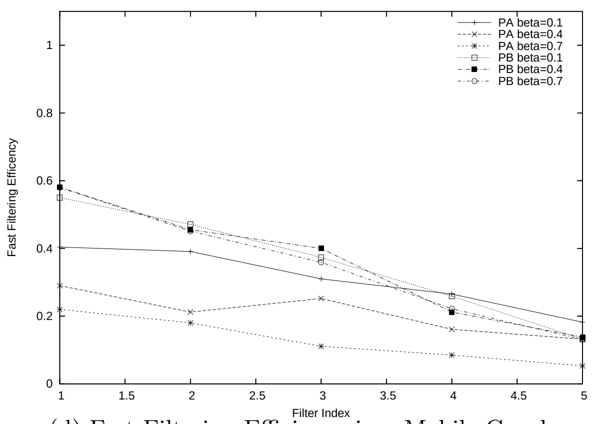
(a) Tag Bit-Size vs. Hotspot Detection Accuracy



(b) V-path Head Size vs. Hotspot Detection Accuracy



(c) Fast Filtering Efficiency in a Static Graph



(d) Fast Filtering Efficiency in a Mobile Graph

Figure 6: Hotspot-Based Protocol Performance

To show similar curves for different τ values, we lower the maximum capacity of each TBF (to the tenth of its optimal size) so that the false positive rate becomes much higher. Figure 6(a) shows both measures where τ varies from 1 to 3 (c is fixed to 4). We can see that the detection rate benefits from a larger τ, but the false hotspot number suffers. In general, τ must be chosen properly to provide a good trade-off between the detection rate and the false hotspot number if TBFs with a smaller capacity have to be used due to memory constraints. In normal scenarios with the optimal TBF parameters, we observe that $\tau = 1$ is typically good enough.

6.4 Fast Filtering

To evaluate the *Fast Filtering* protocol, we choose a single attack path from each scenario and apply *fast filtering* on all routers in this attack path. The filter index FI is defined as: $1 \leq FI \leq L$, where L is the total number of routers in the attack path. We set the dropping probability for router R_i (where i is the distance to the victim) as follows: $\forall i < FI, p_i = 0$; otherwise $p_i = 1$. For the purpose of illustration, we first use a static topology (pause time = 1,000 s). We vary the compromise level β, and observe the attack dropping rate PA and normal dropping rate PB when various FI values are used in Figure 6(c). We see the similar behavior as the theoretical analysis shows: in a low compromise level, PA is sufficiently close to 1, where the highest filter index can be used. In a medium or high compromise level, PB decreases with a larger filter index, but PA decreases as well. Therefore, the optimal point should be defined at the smallest FI where PB does not exceed the pre-determined upper bound γ. For instance, if $\gamma = 0.5$, we can choose $FI = 3$, where $PA \geq 0.8$, even in a high compromise level.

We then experiment with a mobile topology (pause time = 50 s). In Figure 6(d), we can see that the attack dropping rate becomes lower but a similar pattern as the static scenarios can be observed in both rates.

Summary: The simulations show that our traceback approaches are effective and efficient in detecting hotspots. In addition, we can use fast filtering techniques to effectively filter out attack traffic with high accuracy.

7. RELATED WORK

IP traceback schemes include hop-by-hop tracing [4], out-of-band ICMP traceback [1], in-band probabilistic packet marking [18, 20, 24] and watermark-based [23] techniques. Since they are designed for the traditional wired networks (more specifically, the Internet) where the core infrastructure is well protected, the effectiveness of these solutions rely heavily on the assumption that the intermediate routers would not be compromised. Some solutions require a centralized management server, others assume the global routing topology to be static and thus may be obtained or cached locally as guidance. As we have discussed in Section 2, none of these assumptions can be guaranteed to hold in a typical MANET environment. It should be noted that our traceback scheme does not attempt to detect invalid source addresses, therefore our work is different from the detection of address duplication [22].

Without an effective Intrusion Detection System (IDS) that produces accurate and timely intrusion alerts, traceback and other response techniques would simply be futile. Unfortunately, most traditional IDS solutions [7] are centralized and thus not suitable for MANET without a lot of modification. One of the earliest work that tries to address the problem is the "watchdog" system [16], where malicious behavior is detected through neighborhood monitoring. Node-based and cluster-based IDS approaches [12, 13] provide a general-purpose detection framework based on protocol specification analysis and statistical learning tools. Successful experience in detecting many known attacks has been reported based on these approaches. For example, detection of several forms of DoS attacks such as Blackhole (also known as Sinkhole) and Sleep Deprivation Torture, can be done [12].

8. CONCLUSION

In this paper, we presented a distributed traceback approach where no trustworthy infrastructure is needed. Different from other traceback systems, we showed that in our scheme, a single packet can be effectively used in traceback even when the routing topology has been changed. Thus, our scheme is very suitable for MANET that consists of mobile nodes. Our algorithms are able to detect the approximate locations where adversaries reside (but not necessarily the original attack source), even in the face of arbitrary number of adversaries. Furthermore, we presented several traceback protocols. In particular, we showed that a network-wide filtering scheme can be implemented effectively on top of the traceback framework so that its impact on normal traffic is minimal.

Our system requires a traceback to be triggered promptly by an investigator. Otherwise, the digests will be lost after some fixed interval. Although it may be desired to traceback a historical packet when later evidence suggests that it is intrusive, we cannot lengthen the time window infinitely due to memory constraints. One possible solution is to trade off larger memory footprint with possibly slower access time. That is, we can utilize disk storage to store old Bloom filters. Since real-time analysis is seldom needed in this case, the trade-off is typically acceptable.

9. REFERENCES

[1] S. M. Bellovin. ICMP traceback messages. Internet draft draft-bellovin-itrace-00.txt, Network Working Group, Mar. 2000. expired 2000.

[2] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'99)*, pages 216–233, London, UK, 1999. Springer-Verlag.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, July 1970.

[4] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *Proceedings of the USENIX LISA Conference*, Dec. 2000.

[5] S. Capkun, L. Buttyán, and J.-P. Hubaux. Self-organized public-key management for mobile ad hoc networks. In *Proceedings of the ACM International Workshop on Wireless Security (WiSe'02)*, 2002.

[6] S. Capkun, J.-P. Hubaux, and L. Buttyán. Mobility helps security in ad hoc networks. In *Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*, 2003.

[7] S. Cheung and K. N. Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *Proceedings of the New Security Paradigms Workshop*, Cumbria, UK, Sept. 1997.

[8] K. Fall, K. Varadhan, and the VINT project. *The ns Manual (formerly ns Notes and Documentation)*, 2000.

[9] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[10] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom'02)*, Sept. 2002.

[11] Y.-C. Hu, A. Perrig, and D. B. Johnson. Packet leashes: A defense against wormhole attacks in wireless networks. In *Proceedings of IEEE INFOCOM*, pages 1976–1986, San Francisco, CA, Apr. 2003.

[12] Y. Huang and W. Lee. A cooperative intrusion detection system for ad hoc networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03)*, Oct. 2003.

[13] Y. Huang and W. Lee. Attack analysis and detection for ad hoc routing protocols. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, pages 125–145, French Riviera, France, Sept. 2004.

[14] J.-P. Hubaux, L. Buttyán, and S. Capkun. The quest for security in mobile ad hoc networks. In *Proceeding of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01)*, Long Beach, CA, 2001.

[15] D. Liu and P. Ning. Multilevel μTESLA: Broadcast authentication for distributed sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):800–836, Nov. 2004.

[16] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (Mobicom'00)*, pages 255–265, 2000.

[17] C. E. Perkins, E. M. Belding-Royer, and I. Chakeres. Ad hoc on demand distance vector (AODV) routing. Internet draft draft-perkins-manet-aodvbis-00.txt, Internet Engineering Task Force, Oct. 2003. (Work in Progress).

[18] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for IP traceback. *ACM/IEEE Transactions on Networking*, 9(3):226–239, June 2001.

[19] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP traceback. In *Proceedings of the ACM Conference on Communications Architectures, Protocols and Applications(SIGCOMM'01)*, 2001.

[20] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings of the IEEE INFOCOM*, volume 2, 2001.

[21] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. *Security Protocols. 7th International Workshop Proceedings, Lecture Notes in Computer Science*, pages 172–194, 1999.

[22] N. H. Vaidya. Weak duplicate address detection in mobile ad hoc networks. In *Proceeding of the Third ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, pages 206–216, Lausanne, Switzerland, June 2002.

[23] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill. Sleepy watermark tracing: An active intrusion response framework. In *Proceedings of the 16th International Information Security Conference (IFIP/Sec'01)*, June 2001.

[24] A. Yaar, A. Perrig, and D. X. Song. FIT: Fast internet traceback. In *Proceedings of IEEE INFOCOM*, Miami, FL, Mar. 2005.