# INTRUSION DETECTION AND RESPONSE SYSTEMS FOR MOBILE AD HOC NETWORKS

A Thesis
Presented to
The Academic Faculty

by

Yi-an Huang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
December 2006

# INTRUSION DETECTION AND RESPONSE SYSTEMS FOR MOBILE AD HOC NETWORKS

Approved by:

Wenke Lee, Advisor
College of Computing
*Georgia Institute of Technology*

Jonathon Giffin
College of Computing
*Georgia Institute of Technology*

Mustaque Ahamad
College of Computing
*Georgia Institute of Technology*

Chuanyi Ji
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Umakishore Ramachandran
College of Computing
*Georgia Institute of Technology*

Date Approved: November 9, 2006

*For Xia and my family,*

*in gratitude for their love and support,*

*I owe them everything.*

# ACKNOWLEDGEMENTS

I am extremely lucky to have support, encouragement and inspiration from many people, without them this work would not have been possible.

My greatest gratitude goes to my advisor Dr. Wenke Lee for his guidance and consistent support. His knowledgeable, wise and inspiring discussions has guided me throughout my whole Ph.D. career. It was such a pleasure to work with him for all these years. Facing so many obstacles, I am lucky that he has always been there to show me the right direction and influenced me as an active thinker. Thank you, Prof. Lee!

I would also like to thank other members of my committee, Dr. Mustaque Ahamad, Dr. Umakishore Ramachandran, Dr. Jonathon Giffin and Dr. Chuanyi Ji for their interests in my work. Their insightful comments have significantly affected the substance and presentation of my work.

My fellow students made my life at Georgia Tech cheerful and memorable. I wish to thank Xinzhou Qin, David Dagon, Guofei Gu, Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Bryan Payne, Paul Royal, Takehiro Takahashi, Kapil Singh, Sanjeev Dwivedi, Oleg Kolesnikov for all of your ideas and assistance! Your friendship is my best fortune.

I would specially thank my parents. They are always supportive and encouraging, silent yet powerful. I know I owe you all my life. Finally, I cannot say more about my love and appreciation to my wife, Xia Zhou. Her exceptional kindness, caring, and love makes everything worthwhile.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ALGORITHMS

# SUMMARY

A *mobile ad hoc network* (MANET) consists of a group of autonomous mobile nodes with no infrastructure support. The MANET environment is particularly vulnerable due to its dynamic topology, less powerful mobile devices and distributed environment. Unfortunately, many existing protection and defense mechanisms designed for wired networks cannot be applied in this new environment without modifications. In this research, we develop a distributed intrusion detection and response system for MANET specific attacks, and we believe it presents a second line of defense that cannot be replaced by prevention schemes, especially in common MANET scenarios where attacks can easily be launched by insiders or compromised nodes.

In our distributed framework, *Intrusion Detection System* (IDS) agents are deployed independently on individual mobile hosts. This is desired because we do not have a single traffic concentration point where a centralized IDS server can be deployed. In addition, collaboration among IDS agents can be enabled optionally for a more effective detection model.

The foundation of our detection infrastructure is based on systematic attack analysis in the MANET environment. We use an attack taxonomy study for that purpose. Based on this study, we propose a set of misuse and anomaly detection methods that are suitable of detecting different categories of attacks, and they can handle both known and new attacks effectively. Our approaches are based on routing protocol specification with both categorical and statistical measures. They are collectively known as node-based approaches because the only input to these approaches comes from the local data collected by each node itself.

Node-based approaches is most secure but they may be too restrictive in scenarios

where attack or malicious patterns cannot be observed by any isolated node. To address this problem, we have developed cooperative detection approaches to enable collaboration among multiple IDS agents. One approach is to form IDS clusters by grouping nearby nodes, and information can be exchanged within clusters. The cluster-based scheme can result in lower false positive rate and also provide better efficiency in terms of power consumption and resource utilization compared with node-based approaches. As we have learned, security is a big issue in any distributed network without centralized authority. Our clustering protocol can be proved resilient against common security compromises without changing the decentralized assumption.

Intrusion detection will not be very useful unless proper response actions can be taken subsequently. In this research, we further address two important response techniques, traceback and filtering. Traceback schemes are useful to identify the source of a spoofing attack. Existing traceback systems are not suitable for MANET because they rely on incompatible assumptions such as trustworthy routers and static route topology. Instead, we propose a different solution, which we call hotspot-based traceback, that does not rely on these assumptions. Our solution is resilient in the face of arbitrary number of collaborative adversaries. We also develop smart filtering schemes where filters are deployed on selected routers so as to maximize the dropping rate of attack packets while minimizing the dropping rate of normal packets.

To validate our research, we present case study using both ns-2 simulation and MobiEmu emulation platform with three major ad hoc routing protocols: AODV, DSR and OLSR. We implemented various attacks that are representative based on the attack taxonomy. Our experiments show very promising results on detecting attacks in most attack categories using node-based and cluster-based approaches.

# CHAPTER I

# INTRODUCTION

A *mobile ad hoc network* (MANET) consists of a group of autonomous mobile nodes with no infrastructure support. The MANET environment is known to be vulnerable compared with traditional wired networks, due to its dynamic and distributed nature. Many recent research efforts [94, 32] attempted to apply cryptographic solutions to secure MANET, especially on routing protocols. However, existing experience in security has taught us that it is also necessary to develop intrusion detection and response techniques.

This research presents a new attack analysis technique. We first analyze routing activities by decomposing them into basic events, and then propose a new attack taxonomy based on basic events. In our work, a basic event is defined as a series of causally related network and system operations within a single node. Furthermore, normal basic events can be enumerated from protocol specifications. On the other hand, attacks targeting for routing protocols can also be considered as routing activities, except where anomalous basic events may be involved. In this study, we define a taxonomy of anomalous basic events based on the basic security goals that the adversaries attempt to compromise. This taxonomy provides a useful guideline for designing security solutions, as a general soultion should cover as many anomalous basic events as possible.

Based on the attack taxonomy study, we first develop protocol specifications in a scheme known as *Extended Finite State Automata* (EFSA). We then propose a *node-based* IDS framework that relies entirely on local information of each individual node. This framework makes use of features collected from the EFSA. In this framework, we apply two detection approaches. The first approach detects violations of the specification directly, which is referred to as a **specification-based approach**. The second approach, instead,

detects statistical anomalies by constructing statistical features from the specification and then apply machine learning methods. This **statistics-based approach** is more suitable for attacks that are temporal and statistical in nature.

The node-based framework is not perfect. There are certain classes of routing attacks that cannot be detected well with this framework. A second collaborative framework we develop attempts to address this problem. In this framework, nodes are grouped as clusters and clusterheads are elected from cluster members. Features can then be collected from multiple cluster members. Therefore, this new framework has the capability to detect more attacks that the node-based solution fails to address.

We understand the passive nature of intrusion detection systems makes it only half of the whole story. In this research, we further address two important response techniques, traceback and filtering. Without them, a defense system would not be complete. We note that a naive response scheme will not be effective if IP addresses can be easily spoofed. Hence a reliable traceback protocol is desired as it targets specifically the spoofing problem. However, a number of MANET specific vulnerabilities make existing traceback schemes designed for the wired networks unsuitable. In particular, most techniques rely on some strong assumptions such as trustworthy routers and static routes that are used by multiple packets in the same attack flow. These assumptions generally do not hold in MANET. Instead, we propose a different solution, *hotspot-based traceback*, that does not rely on these assumptions. Our solution is resilient in the face of arbitrary number of collaborative adversaries. We further develop smart filtering schemes where filters are deployed on selected routers so as to maximize the dropping rate of attack packets while minimizing the dropping rate of normal packets.

## *1.1 Motivation*

### 1.1.1 MANET Characteristics

*Mobile ad hoc networks* are particularly vulnerable due to some of their fundamental characteristics, such as open medium, dynamic topology, distributed cooperation, and constrained capability. Ironically, many of the listed features also contribute to the fact that ad hoc networks become useful and popular.

First of all, the use of wireless links renders a MANET susceptible to attacks ranging from passive eavesdropping to active interfering. Unlike wired networks where an adversary must gain physical access to the network wires or pass through several lines of defense at firewalls and gateways, attacks on a MANET can come from all directions and target at any nodes. Mobile nodes are typically autonomous units capable of roaming independently. It implies that nodes with inadequate physical protection are receptive to being captured, compromised, and hijacked. A compromised node can result in leaks of confidential information, message contamination, and node impersonation. To summarize, MANETs will unlikely have a clear line of defense, and thus every node must be prepared for encounters with an adversary directly or indirectly [95].

Decision making in ad hoc networks is typically decentralized and ad hoc network algorithms often rely on cooperative participation of all nodes. The lack of centralized authority provides golden opportunity to adversaries who can bring new types of attacks specifically designed to break these cooperative algorithms. Routing protocols are a perfect example. Most ad hoc routing protocols are inherently cooperative. Unlike in a wired network where extra protection can be (and typically is) placed on routers and gateways which are only a small subset of the entire network, any mobile node in an ad hoc network could be hijacked by an adversary who could then paralyze the entire wireless network by disseminating false routing information from that node. Such false routing information could also result in, for example, messages from all nodes being fed to the compromised node, which is hard to track down but extremely dangerous.

Finally, due to limits of current technology, most mobile hosts have limited network capacity. Disconnected operations are very common in wireless network applications [74]. For some nodes which rely on battery for power supply, energy efficiency is also an important issue, as it makes computation-intensive security measures often infeasible.

### 1.1.2 Challenges and Issues

Security threats have grown rapidly on the Internet in the past several years. For example, a *Denial-of-Service* (DoS) attack is designed to bring down a web site (or a number of sites) by flooding it (them) with huge amount of traffic. Even worse, its distributed variant DDoS, which may come from multiple sources, is much more dangerous and it has become increasingly popular.

There are two general methodologies to improve the security of any public system, either to prevent it from happening at all, or to detect it as soon as possible and take proper countermeasures. Intuitively, the former approach is more effective, and therefore, it is not surprising that most recent MANET security efforts [94, 32] adopt this approach. However, the approach has several practical issues. First, many attacks cannot be fully prevented. The highly popular and disruptive *Denial-of-Service* (DoS) attack, for example, would overload most prevention schemes. In practice, heavy use of cryptographical primitives often results in prohibitive computational and storage requirement to some application. For example, a 500 MHz Pentium can only compute digital signatures on the order of 100 1024-bit RSA signatures per second.

Therefore, we need a second wall of defense that provides the capability to detect intrusions and to alarm users. Intrusion Detection Systems (IDS) can provide such type of defense.

While IDSes have been widely used in wired networks, IDS design in MANET remains a challenging task. One of the largest issues is the lack of trust. In a typical MANET scenario, mobile nodes are self-autonomous and do not necessarily trust each other. It should

be noted that authentication and authorization schemes, even available, do not fully solve the problem, because these schemes themselves can be circumvented with node compromise, due to the lack of physical security of mobile nodes.

Second, IDS requires a well-defined attack taxonomy. In a relatively new environment such as MANET, traditional attack analysis is not effective, because most traditional techniques rely heavily on details of known vulnerabilities and attack incidents. MANET has many potential applications but none of them are widely used yet. Therefore, only limited MANET attacks have been studied in the literature.

Unlike wired networks, MANET has no single traffic concentration point, and thus no convenient place for security station or network administration either. From the architectural point of view, it implies that intrusion detection systems must be fully distributed. Unfortunately, most of today's wired IDSes rely on centralized traffic analysis, filtering and monitoring at switches, routers and gateway and thus less effective.

Furthermore, MANET nodes typically have limited resources, e.g., memory and battery power. Therefore, intrusion detection architecture and algorithms must be very efficient in such a way that complex analysis is called upon only when necessary.

Intrusion detection techniques can be categorized into *misuse detection* and *anomaly detection*. Misuse detection systems, e.g., IDIOT [48] and STAT [35], use patterns of **known** attacks or weak spots of the system to match and identify known intrusions. Anomaly detection systems, such as IDES [37], flag observed activities that deviate significantly from the established normal usage profiles as anomalies. Misuse detection has a low false positive rate on known attacks but anomaly detection is the only means to catch new attacks. Nevertheless, attacks are not well studied yet in the context of MANET, and therefore the effectiveness of misuse detection is fairly restricted, while anomaly detection could play a more important role for this platform.

Effective detection would not be useful unless some actions were to be taken, and taken promptly. One naive solution is to filter out subsequent packets from the same attack flow.

This simple approach has a serious pitfall: the source address appeared in an offending packet may be faked easily without proper countermeasures. Revealing the true identity of such an *IP spoofing* attack is non-trivial, and it has become the central topic of many recent researches [80, 75, 81, 92, 10, 4, 90]. These solutions often involve **traceback** schemes. Similar to the design of IDS, response solutions for MANET should be distributed as well.

## *1.2 System and Threat Model*

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 any node (router) may be malicious. There may be multiple malicious nodes and they can collaborate and exchange information (probably through out-of-band channels).

A malicious node may send many packets and each packet may be delivered in a different attack path.

We assume all nodes, including adversaries, have the same transmission power and all wireless links are bidirectional.

## *1.3 Problem Statement*

Since existing IDS cannot be readily applied to *Mobile Ad Hoc Networks* (MANET), in this work, we study efficient and scalable distributed approaches to build intrusion detection and response systems for this new platform. It should be noted that although our work uses MANET as the main evaluation platform, various pieces of our work may be suitable for a more general distributed network platform where centralized control may not be in place.

## *1.4 Contribution of the Thesis*

The main contribution of this work is a detailed design of a distributed intrusion detection and response system for ad hoc networks. In particular, our main contributions include

- **Systematic attack analysis for ad hoc networks**. Essential preparation step from which we can understand the strengths and limits of different security approaches, and choose the most effective solutions based on different scenarios.

- **Systematic feature collection**. We use protocol specification to categorize system behavior and collect features based on statistics of protocol states and transitions. This approach relies less on domain knowledge to find out useful features, as domain knowledge is often lacking in new areas and environment such as MANET.

- **Anomaly detection system working with large feature sets**. As we rely on systematic approach to extract features, it is inevitable to introduce a huge number of features. While we have also developed effective feature selection approach, a scalable anomaly detection system with a large set of features remains essential. In fact, the algorithm, known as *Cross Feature Analysis*, is a general approach, and we believe that it can be applied in many applications and different platforms.

- **Node-based and cluster-based detection** two complimentary detection frameworks that are especially suitable for ad hoc networks. In the first approach, each IDS agent resides on a different mobile node and performs independent detection based on its own observation. In the second approach, IDS agents collaborate with immediate neighbors to cooperate. While more effective approaches may be possible with larger scale of collaboration, we are very cautious not to achieve "better" accuracy with the cost of IDS itself being easily compromised.

- **Distributed traceback and filtering** As we have addressed, IDS will not be useful without effective response actions. It is another difficult problem as response actions

may also suffer from the lack of trust. For example, can you trust the alert sent by your neighbor that claims a third node is or may be malicious? With certain assumptions, we developed a distributed response system that can accurately identify the approximate location of attackers and filter traffic effectively with that piece of information.

## *1.5 Architecture*

Let us have a brief overview about the overall IDS architecture and and then introduce four main components of our system, namely, a systematic feature collection approach, node-based detection, cluster-based detection and distributed traceback and filtering protocols.

### 1.5.1 Overview

Intrusion detection and response in MANET must be distributed and cooperative. In our proposed architecture, as shown in Figure 1, "monitoring nodes" throughout the network each runs an IDS agent. In the node-based scheme, every node can be the monitoring node for itself. Alternatively, for better efficiency, in the cluster-based scheme, a cluster of neighboring nodes can elect a node (or a few nodes) to be the monitoring node(s) for the neighborhood.

Each IDS agent runs independently and is responsible for detecting intrusions for the local node or cluster. IDS agents on neighboring monitoring nodes can investigate collaboratively in order to not only reduce the chances of producing false alarms, but also detect intrusions that affect the whole or a part of the network. These IDS agents collectively form an integrated intrusion detection framework. The internal of an IDS agent, as shown in Figure 1, can be conceptually structured into five pieces: the feature collection module, the node-based detection engine, the cooperative detection engine, the traceback and filtering module, and a secure communication module that provides a reliable communication channel among IDS agents.

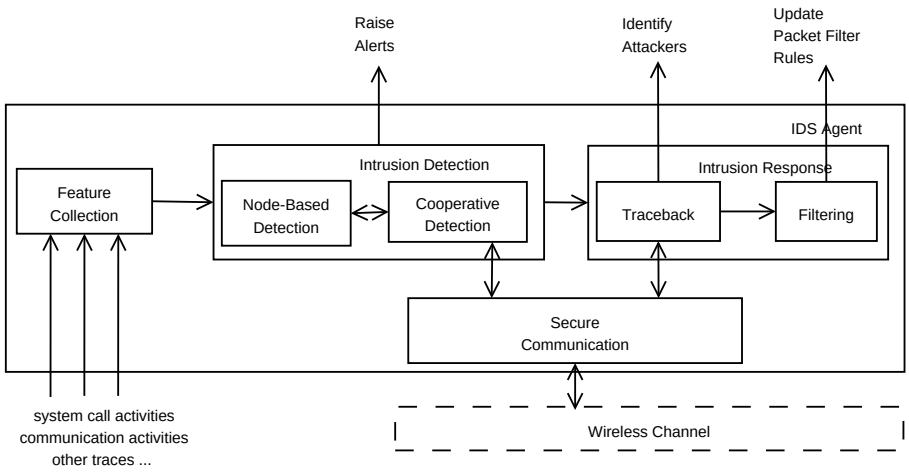



**Figure 1:** Architecture of a MANET IDS Agent

### 1.5.2 Components

**Feature Collection (Feature Construction and Selection)** Detection algorithms use observable information, known as *features* or *attributes*, that summarizes certain aspects of the underlying system or network behavior. Features can be constructed from various audit logs and system or network sensors. We develop a new systematic approach to construct features based on protocol specification. The specification-based study is presented in Chapter 3. We further perform specialized feature selection to improve detection performance.

**Node-Based Detection** The node-based detection engine analyzes the local data traces gathered by the local data collection module. It can use both misuse and anomaly detection algorithms. We present a comprehensive study of node-based detection for ad hoc routing protocols in Chapter 4.

**Cooperative Detection** An IDS agent that detects locally a known intrusion or anomaly with strong evidence (i.e., the detection rule triggered has a very high accuracy rate) can

determine independently that the network is under attack and can initiate a response. However, if a node detects an anomaly or intrusion with weak evidence, or the evidence is inconclusive but warrants broader investigation, it can initiate a cooperative global intrusion detection procedure. This procedure works by propagating the intrusion detection state information among neighboring agents. If an agent who uses alert information from other agents now finds the intrusion evidence to be sufficiently strong, they can initiate certain response action collectively. Cluster-based IDS is an inexpensive cooperative detection framework which remains very powerful. We present this work in Chapter 5.

**Traceback and Filtering** Intrusion response in MANET depends on the type of intrusion, the help (if any) from other security mechanisms, and the application-specific policy. Two typical examples of response actions include traceback and filtering. A *traceback* session reveals the true identity of a *spoofing* attack, while a *filtering* mechanism tries to filter out attack traffic with minimal impact on normal traffic. We present these response techniques in Chapter 7.

**Secure Communication** Data communication among IDS agents must be secured to ensure confidentiality, authenticity, and integrity. We do not consider the potential issue of insider attacks or compromised nodes. These issues should be taken care of by IDS itself. In other words, an IDS agent should not generally assume that another IDS agent is well-behaving unless there is some strong supportive evidence.

Secure communication primitives can be provided by either asymmetric or symmetric cryptographic operations, such as the work presented by [14] where public keys are managed in a self-organized fashion, and the one-way hash chains [31, 55] or one-way Merkle-Winternitz chains [30]. Both asymmetric and symmetric primitives have advantages and disadvantages. In our implementation, we prefer symmetric primitives because we believe public key cryptography is rather expensive and thus often prohibitive in today's technology.

## *1.6 Organization of the Thesis*

In this chapter, we introduced the basic problem in security design in mobile ad hoc networks and motivated the need to introduce distributed intrusion detection systems. In the remainder of this dissertation, we present the contributions and research activities in details as follow.

In Chapter 2, we discuss the related concepts of basic events and presents a taxonomy of anomalous basic events in MANET. It is used as a basis as to understand what kind of attacks can be covered in our IDS framework. Chapter 3 presents EFSA specification and shows how features can be collected systematically. Using these features, Chapter 4 describes the design of the node-based IDS. Chapter 5 explains why a collaborative framework should be considered and then illustrates the design of the cluster-based scheme. Chapter 6 demonstrates the effectiveness of feature selection and the performance using both node-based and cluster-based IDS frameworks with experimental results. Chapter 7 defines the general intrusion response problem in general followed by our distributed hotspot-based solution. Chapter 9 compares our work with other related work. The whole dissertation is concluded in Chapter 10.

# CHAPTER II

# ATTACK ANALYSIS

Designing an effective intrusion detection system (IDS), as well as other security mechanisms, requires a deep understanding of threat models and adversaries' attack capabilities. We note that since MANET uses a TCP/IP stack, many well-known attacks can be applied to MANET but existing security measures in wired networks can address these attacks. On the other hand, some protocols, especially routing protocols, are MANET specific. Very few attack instances of these protocols have been well studied. It follows that traditional attack analysis cannot work effectively. In this work, we describe a systematic approach to study MANET attacks based on the concept of **anomalous basic events**. We use MANET routing as the subject of our study.

## *2.1 Concepts*

A **routing process** in MANET involves causally related, cooperative operations from a number of nodes.

A **basic routing event** is defined as an indivisible local segment of a routing process. More precisely, it is the smallest set of causally related routing operations on a single node. We will use the term **basic event** for short.

For instance, *Route Discovery* is a routing process that appears in many on-demand routing protocols [39, 66]. It consists of chained actions from the source node to the destination node (or an intermediate node who knows a route to the destination) and back to the source node. This process can be decomposed into a series of basic routing events (referring to Figure 2 for an example, where $A$ is the source node and $D$ is the destination node):

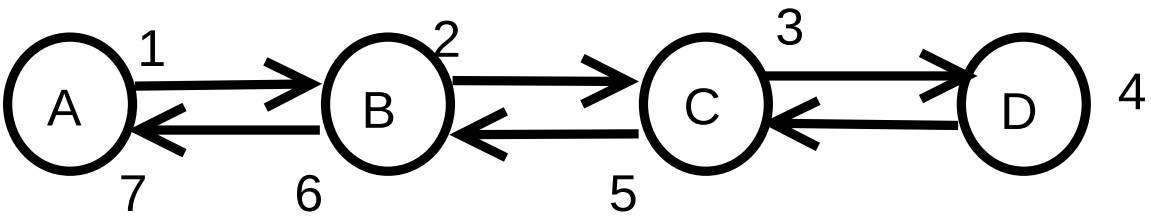



**Figure 2:** Basic Events in *Routing Discovery* Process

- The source node delivers an initial *Route Request* (Event 1);
- Each node (except for the source node and the node that has a route to the destination) in the forward path receives a *Route Request* from the previous node and forwards it (Events 2 and 3);
- The replying node receives the *Route Request* and replies with a *Route Reply* message (Event 4);
- An intermediate node in the reverse path receives a *Route Reply* message and forwards it (Events 5 and 6); and finally,
- the source node receives the *Route Reply* message and establishes a route to the destination (Event 7).

Eventually, we can find seven basic events in Figure 2.

Note that a basic (routing) event may contain one or more network or in-memory operations, such as receiving a packet, modify a routing parameter, or delivering a packet. However, the integrity of routing logic requires every basic event to be conducted in a transactional fashion, i.e., a basic event is considered successful (or normal) if and only if all these operations are performed, and performed in the specified order. We further note that in this definition, we only consider operations on a single MANET node. In principle, some event in one node may have causal relationship with another event in a second

node. However, inter-node causal relationship may not be enforceable due to the general only-trust-local-information assumption.

The requirements for normal routing behavior can thus be abstracted as a **protocol specification where all possible normal basic events on a single routing agent are enumerated**.

Therefore, any basic event that does not follow the specification can be classified as an **anomalous basic event**. Because of that, it is useful to study the anomalous basic events, because they help categorize the characteristics of basic attack components.

We should note that it is possible that some attacks may not trigger any anomalous basic events. There are a few possibilities: an attack may either involve malicious behavior on a different layer that the specification for routing protocols does not characterize, or it may involve abnormal patterns beyond a single node.

*Wormhole attacks* [31] are an example of the first case, where two wireless nodes can create a hidden tunnel through wires or wireless links with stronger transmission power. To deal with the issue, a multiple-layer framework may be desired. While this work mainly focuses on routing protocols, we believe it can be generalized to protect other network layers as well because most components in our framework are designed for general use and not routing specific.

A network scan on some known (vulnerable) ports is an example of the latter case, because each single node in this attack may not observe any illegitimate use at all. The second issue can be addressed with a collaborative IDS, such as the one we are going to present in Chapter 5.

## *2.2 Taxonomy of Anomalous Basic Events in MANET Routing Protocols*

We identify an anomalous basic event by two components, the **target** and the **operation**. A protocol agent running on a single node has different elements to operate on, with different semantics. The routing behavior of MANET typically involves three elements or targets:

*routing messages*, *data packets* and *routing table (or routing cache) entries*. Furthermore, we need to study what are the possible attack operations on these targets. Individual security requirements can be identified by examining the following well-known security goals: *Confidentiality*, *Integrity* and *Availability*. We summarize all possible combinations of routing targets and operations in Table 1. Here, we distinguish *Integrity* compromise into three distinct subcategories: add, delete and change. The exact meanings of these subcategories, however, could be interpreted slightly differently within the context of individual targets.

Conceptually, we could have characterized a normal basic event in a similar way, i.e., based on its target and its operation type. Nevertheless, different protocols define different types of normal operations and it is unlikely to combine everything into a universal taxonomy. A more logical way is probably to represent normal basic events with a different structure, such as the extended state machine approach we introduce in Chapter 3.1.1.

In MANET routing security, cryptography addresses many problems, especially those involving confidentiality and integrity issues on data packets. Intrusion detection techniques are more suitable for other security requirements. *Availability* issue, for example, is difficult for protection techniques because attack packets appear indistinguishable from normal user packets. Some *integrity* problems also require non-cryptographic solutions for efficiency reasons. For example, an attacker can compromise the routing table in a local node and change the cost of any specific route entry. It may change the sequence number or a hop count so that some specific route appears more attractive than other valid routes. Encrypting every access operation on routing entries could have been too expensive. Intrusion detection solutions can better address these issues, based on existing experience in the wired networks. We identify a number of anomalous basic events that are more suitable for intrusion detection systems in bold face in Table 1.

There are two types of anomalous basic events with asterisks in the table, *Fabrication of Routing Messages* and *Modification of Routing Messages*. There are cryptographic solutions for these types of problems, but they are not very efficient and sometimes require an

expensive key establishment phase. We want to study them in our IDS work because they are related to the routing logic and we can see later that some attacks in these categories can be detected easily.

**Table 1:** Taxonomy of Anomalous Basic Events

| Compromises to Security Goals | | Events by Targets | | |
|---|---|---|---|---|
| | | Routing Messages | Data Packets | Routing Table Entries |
| Confidentiality | | Location Disclosure | Data Disclosure | N/A |
| Integrity | Add | **Fabrication*** | Fabrication | **Add Route** |
| | Delete | **Interruption** | **Interruption** | **Delete Route** |
| | Change | **Modification*** | Modification | **Change Route Cost** |
| | | **Rushing** | | |
| Availability | | **Flooding** | **Flooding** | Routing Table Overflow |

We examine a number of basic MANET routing attacks noted in the literature [61, 85]. By comparing them (shown in Table 2) with taxonomy in Table 1, we find they match very well with the definitions of anomalous basic events. We refer to each attack with a unique name and optionally a suffix letter to denote attack variations. For example, "Route Flooding (S)" is a flooding attack of routing messages that uses a unique source address.

In addition, we use a number of more complex attack scenarios for the purpose of evaluation. These attacks may contain more than one anomalous basic events and therefore more realistic. We borrow some examples studied by Ning and Sun [61] on AODV [66] misuse. These scenarios are summarized in Table 3.

## *2.3 Summary*

We proposed a new systematic approach to categorize attacks. Our approach decomposes an attack into a number of basic events, which is very useful for the sake of attack taxonomy analysis.

The taxonomy of anomalous basic events is important because it provides an objective basis to determine how useful an IDS can be. We expect a good IDS to address as many anomalous basic events as possible. Using the taxonomy as a guideline, we propose a

**Table 2:** Basic MANET Attacks

| Attacks | Attack Description | Corresponding Anomalous Basic Events |
|---|---|---|
| Active Reply | Actively forge a *Route Reply* message while there are no corresponding incoming *Route Request* messages. | Fabrication of Routing Messages |
| False Reply | Forge *Route Reply* for a *Route Request* message for which the node is not supposed to reply. | |
| Route Drop (R) | Drop routing packets with Random source and destination addresses. | Interruption of Routing Messages |
| Route Drop (S) | Drop a fixed percentage of routing packets with the given Source address. | |
| Route Drop (D) | Drop a fixed percentage of routing packets with the given Destination address. | |
| Modify Sequence (R) | Modify the destination's sequence number Randomly. | Modification of Routing Messages |
| Modify Sequence (M) | Increase the destination's sequence number to the Maximal allowed value. | |
| Modify Hop | Reduce the hop count to a smaller value. | |
| Rushing (F) | Shorten the waiting time for *Route Replies* when a route is unavailable to increase the discovery Failure ratio. | Rushing of Routing Messages |
| Rushing (Y) | Shorten the waiting time to replY a *Route Reply* message after a *Route Request* is received. | |
| Route Flooding (R) | Flood with both source and destination addresses randomized. | Flooding of Routing Messages |
| Route Flooding (S) | Flood with the same source address but random destination addresses. | |
| Route Flooding (D) | Flood with the same destination address but random source addresses. | |
| Data Drop (R \| S \| D) | Similar to Route Drop attacks but with data packets instead. | Interruption of Data Packets |
| Data Flooding (R \| S \| D) | Similar to Route Flooding attacks but with data packets instead. | Flooding of Data Packets |
| Add Route (I) | Randomly select and validate an Invalid route entry. | Add Route of Routing Table Entries |
| Add Route (N) | Insert a New route entry with random destination address. | |
| Delete Route | Invalidate a random valid route. | Delete Route of Routing Table Entries |
| Change Sequence (R \| M) | Similar to Modify Sequence attacks but sequence numbers are altered directly on the routing table. | Change Route Cost of Routing Table Entries |
| Change Hop | Similar to Modify Hop, but hop counts are altered directly on the routing table. | |
| Overflow Table | Add excessive routes to overflow the routing table in order to evict good routes. | Routing Table Overflow of Routing Table Entries |

**Table 3:** More Complex MANET Attacks

| Attacks | Attack Description | Corresponding Anomalous Basic Events |
|---|---|---|
| Route Invasion | Inject a node in an active route. | Fabrication of Routing Messages (two RREQs) |
| Route Loop | Create a route loop. | Fabrication of Routing Messages (two RREPs) |
| Partition | Separate a network into two partitions. | Fabrication of Routing Messages (RREP) Interruption of Data Packets |

number of different IDS frameworks, illustrated in Chapter 4 and 5 respectively.

# CHAPTER III

# FEATURE COLLECTION

Many detection systems use one or a few features, such as system calls [91] or call stack information [25], as model input. It is often the case that features used in these detection systems are either manually selected or derived from domain knowledge. Hence they are inherently restricted in the types of attacks to be detected. As we know, MANET is a relatively new platform, attacks are not well understood, while a few known attacks have already been found to be very different from attacks in traditional networks. We therefore believe that a more general approach is desired to collect model features. In this chapter, we develop a new systematic feature collection approach. It involves two parts, feature construction and feature selection.

## *3.1 EFSA*

In our previous study, we define a total of 141 features according to domain knowledge. These features belong to two categories, non-traffic related and traffic related. They capture the basic view of network topology and routing operations. However, since the feature set was defined manually, the effectiveness of a statistical learning model is based on the assumption that the feature set is relatively complete to cover all possible aspects of system behavior. There is no guarantee that our manually selected features could satisfy this requirement.

We address this problem by enumerating possible features derived from activities in an EFSA of the underlying routing protocol.

An *extended finite state automaton (EFSA)* is similar to a finite-state machine except that transitions and states can carry a finite set of parameters. Conventionally, we call them transition *parameters* and state *variables*. Formally, we define EFSA in Definition

1, which was originally introduced from [78]. EFSA can be derived from documentation, implementations, RFCs or other published materials.

**Definition 1.** *Extended finite state automaton* EFSA $L \equiv (\mathcal{E}, Q, s, f, V, D, \delta)$, where

$\mathcal{E}$: an alphabet of events. Each event has zero or more parameters;

$Q$: a finite set of EFSA states;

$s$: the start state. $s \in Q$;

$f$: the finish state. $f \in Q$;

$V$: state variables: $V \equiv (v_1, \ldots, v_n)$;

$D$: variable domains: $D \equiv (d_1, \ldots, d_n)$, where $d_i$ denotes the value domain for variable $v_i$;

$\delta : Q \times D \times \mathcal{E} \rightarrow (Q, D)$: transition relation.

We further distinguish two types of events: input and output events. Input events can be triggered by an incoming packet or timeouts. Output events can be sending out a packet or other actions before state transition occurs.

According to the original definition in [78], input and output events must be defined in separate transitions, as only one event is allowed in each transition. In this work, we relax the transition definition by allowing a transition to have at most one incoming event (known as the *input condition*) and one outgoing event (known as the *outgoing action*), either of which can be optional. The new definition can be described alternatively as: $\delta = \{S_o \rightarrow S_n, input \rightarrow output\}$, where the old and new states are specified in $S_o$ and $S_n$ respectively. The new definition should define the following semantics: if an output action is defined, it must be performed immediately after the input condition is met, before the new state is arrived. No other transitions are allowed unless the output action has been accomplished.

Protocol state machines are in general non-deterministic, as one incoming packet can lead to multiple states. We solve non-determinism by introducing a set of separated finite state automata. They initiate from the same state, but fork into different paths on an incoming event when the state may lead to multiple transitions. For instance, an EFSA can be defined in the context of TCP, where each EFSA corresponds to a unique connection. In on-demand routing protocols such as AODV or DSR, an extended finite state automaton may define all operations targeting a unique destination. For example, an incoming *Route Reply* message may add new routes to both the destination node and the previous hop, thus we need two EFSAs, one for each destination, to process the same message in parallel. The same *Route Reply* message may also need to be forwarded, which is conducted by a third EFSA corresponding to the originator of the *Route Recovery* process.

### 3.1.1 Case study: AODV Specification

We take *Ad hoc On-demand Distance Vector* (AODV), one of the popular MANET routing protocols [66], as a case study. In AODV, operations on a particular route entry to a single destination can be defined with a single EFSA.

We construct an AODV EFSA by following the AODV Internet draft version [66]. Our AODV EFSA is based on the AODV state machine from Bhargavan et al.'s work [5]. AODV uses hop-by-hop routing similar to distant vector based protocols such as RIP [57], but there are no periodical route advertisements. Instead, a route is created only if it is requested by data traffic where routes are not available [66].

It should be noted that the number of state machines may consistently increase, up to the number of possible nodes in the system if their lifetime is unbounded. Thus, we should remove unnecessary state machines for better memory usage. In AODV, a route entry is removed after it has been invalidated for a certain period. In other words, we can identify a final state from which no further progress could be made. Therefore, state machines reaching the final state can be deleted from the state machine repository safely.

The AODV EFSA (per destination) is shown in Figures 3 and 4. The reason that one EFSA is split into two sub-graphs are purely for the purpose of a better layout. Figure 4 is used within a certain period after a node has rebooted. After that, the normal graph (Figure 3) should be used.

The EFSA specification defines *input conditions* and *output actions* on each transition. An **input condition** (*input*) can specify timeouts or predicates and at most one packet-receiving event. It uses a C-like expression syntax where operators like &&, || etc., can be used. State variables (of the original state) and transition parameters can be accessed in input conditions. To distinguish, state variables always start with lower case letters and transition parameters start with capitalized letters. **Packet-receiving events**, predicates and **timeouts** can be used as Boolean functions within input conditions. A packet-receiving event or a predicate has its own parameters, which must be matched with provided values, unless the value is a dash (-), which specifies that the corresponding parameter can match any value. An **output action** (*output*) can specify state variable modifications, tasks and at most one packet-delivery event. Either *input* or *output* can be optional but at least one must be present.

In addition, a number of **auxiliary functions** can be used in either input conditions or output actions. They are not actually evaluated by IDS, and we use auxiliary functions mainly to improve the overall readability.

The complete AODV EFSA specification can be found at Appendix A.

### 3.1.2 Other Protocols: DSR and OLSR

We have also studied two other major routing protocols. Appendix B and Appendix C list the EFSA specification for DSR (*Dynamic Source Routing*) [39] and OLSR respectively.

## *3.2 Feature Construction*

Once we have EFSA, statistical features can be easily enumerated by looking at a few statistical properties from the EFSA. Here, we classify two different types of statistical

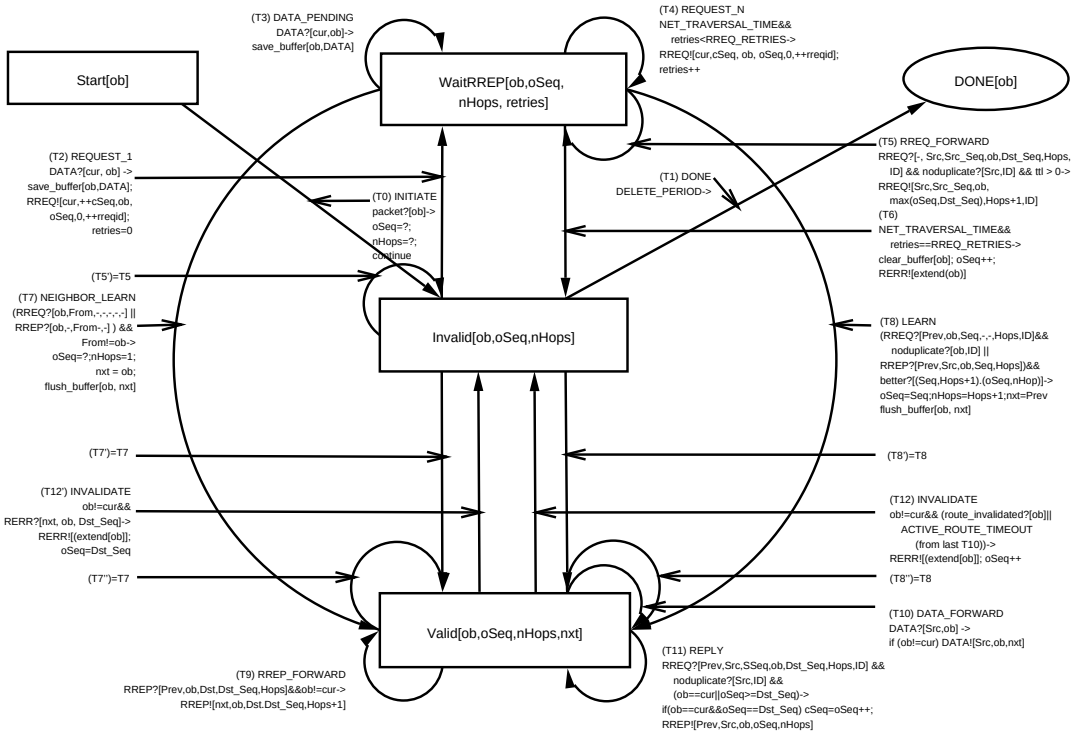



**Figure 3:** AODV Extended Finite State Automaton (for Destination *ob*): In Normal Use

features:

- *Frequencies* of states and transitions;
- *Mean* and *standard deviation* of state variables and transition parameters.

In this way, we can construct features systematically from all states and transitions.

## *3.3 Feature Selection*

When a large set of features are involved, many detection algorithms can be slow and inefficient due to irrelevant features that cause over-fitting during learning. In addition, features are not obtained for free. They are typically derived from audit traces which may consume CPU power and disk storage. As a consequence, feature selection is often necessary for both detection accuracy and performance reasons.

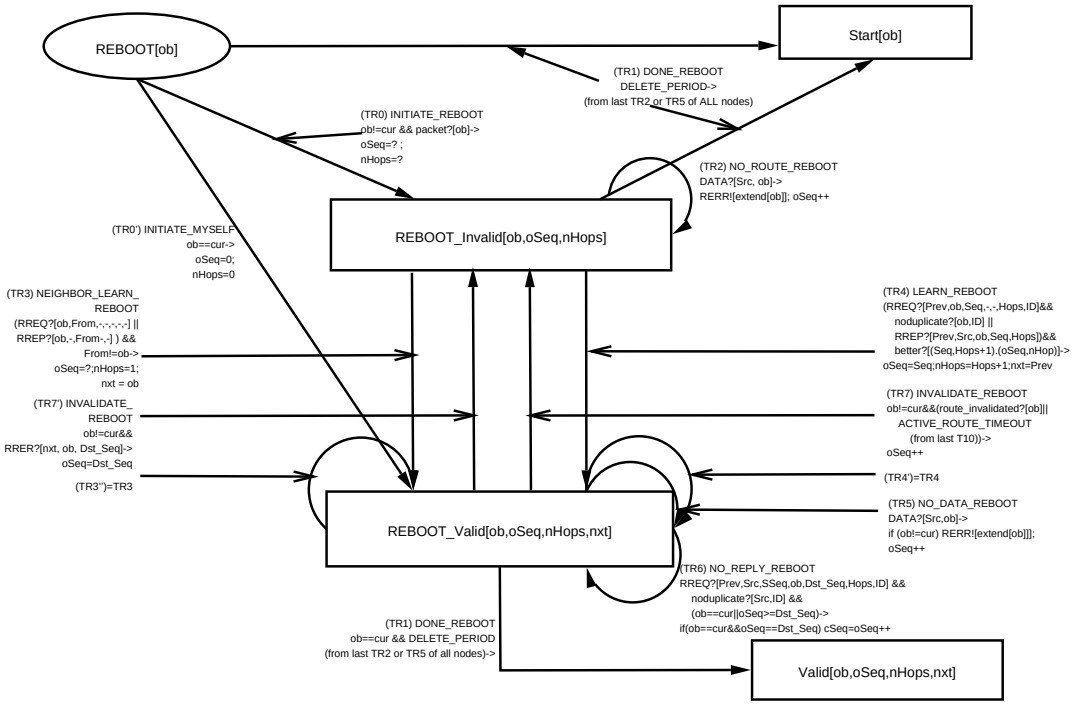



**Figure 4:** AODV Extended Finite State Automaton (for Destination *ob*): After Reboot

Finding an optimal solution to the feature selection problem is known to be NP-hard. However, many heuristics and approximation algorithms exist with fairly reasonable performance in practice. A well known feature selection algorithm is *forward selection* with Kullback-Leibler distance. At each step, a selected feature set G is expanded with a new feature F that maximizes the relative entropy (also known as Kullback-Leibler distance) between $Pr(C|G)$ and $Pr(C|G \cup \{F\})$ where $C$ is the class distribution, until some criterion is met [29]. This approach is straightforward, easy to compute, and works quite effectively in practice. However, it requires the knowledge of class distribution, which is not always known. Anomaly detection methods often have to operate on one-class data, and therefore, an alternative method may be necessary.

Standard techniques, such as *Principal Component Analysis* (PCA) [60], can be used to perform feature selection. However, Tax and Müller [84] has pointed out that the standard

technique does not work well for one-class classification (OCC) by presenting the bias-variance dilemma: removing low-variance directions is desired in standard feature selection problem, however low-variance directions may be more useful to capture outliers than high-variance directions, therefore, such removal can be counter-productive.

Based on this observation, we present a new feature selection algorithm in this work. Our idea is to estimate the tightness of the decision surface computed by classifiers, potentially with a different set of features. We expect a better feature subset will generate a "tighter" boundary. To determine how "tight" a model is, we use a "skewing" technique that generates new dataset which slightly offsets from the original training data. Therefore, a "tighter" model should have higher false positive rates than a "looser" model when testing with the "skewed" dataset. Using this notion, we present a forward selection method that incrementally adds a new feature which leads to the "tightest" model among all possible candidates. The method stops when adding a new feature no longer produces a better model.

### 3.3.1 Concepts

**Definition 2.** *Feature Set* F is the set of all possible features: $\{f_1, f_2, \ldots, f_l\}$ in an application domain, where $l = |F|$. For simplicity, let us assume every feature can take any real value in real domain $\mathcal{R}$. With some quick normalization, we can always reduce value domains to any finite real range.

**Definition 3.** *Feature Space* $\mathbf{X}_s \in \mathcal{R}^{|s|}$ with respect to feature subset $s \in 2^F$ is the set of all possible feature values when only features from the subset $s$ are used. $\mathbf{X} \equiv \mathbf{X}_F$.

**Definition 4.** *Projection function* $p(\mathbf{x}, s) : \mathbf{X} \times 2^F \rightarrow \mathbf{X}_s$ outputs the sub-vector of input feature vector $\mathbf{x}$ with regard to the specified feature subset $s$.

**Definition 5.** *One-Class Classifier* $C_s$ with respect to feature subset $s$ is an algorithm that accepts a set of training examples $\mathbf{X}_s^1 \subseteq \mathbf{X}_s$ as input and outputs a function $f : \mathbf{X}_s \rightarrow \{0, 1\}$.

When an example $\mathbf{x} \in \mathbf{X}_s$ (which may or may not in $\mathbf{X}_s^1$) is given, $f(\mathbf{x})$ outputs 0 if $\mathbf{x}$ is predicted as normal, or 1 otherwise. As a special case, $C_F$ can be written as $C$.

### 3.3.2 Skewed Data Generation

**Definition 6.** For data point $\mathbf{x} \equiv \{x_1, x_2, \ldots, x_m\}$ and $\mathbf{x} \in \mathcal{R}^m$, we define a "skewing" function $y = sk(\mathbf{x})$, where $y \equiv \{y_1, y_2, \ldots, y_m\}$ and $y_i = x_i + \alpha v_i r_i$ for all $i \in [1, m]$, where,

- $\alpha$ describes the skew degree, $\alpha > 0$;
- $v$ is the normalized standard deviation (s.t.d) vector. If $\sigma_i$ is the s.t.d on the $i^{th}$ dimension, then $v_i \equiv \frac{\sigma_i}{\sqrt{\sum_{j=1}^m \sigma_j^2}}$. It may be necessary to evaluate statistics on the whole training dataset in order to compute $v$; and
- $r$ is a normalized random unit vector. There are different ways to define what a random unit vector means. In this work, we choose to generate a random vector $\{R_i\}$ with length $m$ with $R_i$ is generated with an independent Gaussian distribution $N(0, 1)$, then $r_i \equiv \frac{R_i}{\sqrt{\sum_{j=1}^m R_j^2}}$ is a random unit vector [84].

**Definition 7.** Based on the definition of a "skewing" function, the skewed image of $\mathbf{Y}$ (where $\mathbf{Y} \subseteq \mathbf{X}$) is: $sk(\mathbf{Y}) \equiv \{sk(\mathbf{y}) | \mathbf{y} \in \mathbf{Y}\}$.

Figure 5 illustrates an example how skewed image of data point $\mathbf{x}$ is created. $v_1$ and $v_2$ shows the normalized standard deviation of the training dataset. Possible locations for $sk(\mathbf{x})$ can be seen residing on the small ellipse around point $\mathbf{x}$.

### 3.3.3 Skewed False Positive Rate

Assume the same feature set $F$ and training set $X^1$ are used by all classifiers.We use the procedure in Algorithm 1 to compare two *One-Class Classifiers* $C_F^1$ and $C_F^2$.

It should be noted that in the second step, we exclude the false positives examples before "skewing". In other words, only true negative points will be "skewed".

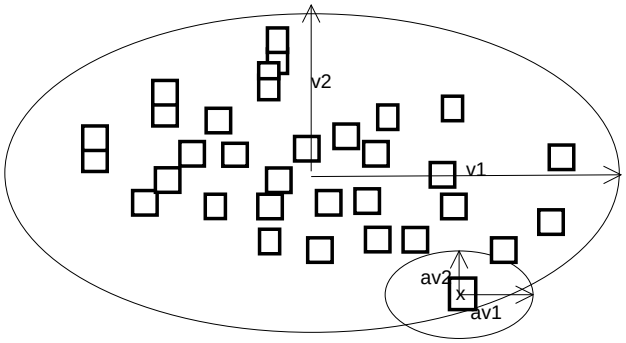



**Figure 5:** Skewed Data Generation Example

1 Generate $f^1$ from $C_F^1$ and $\mathbf{X}^1$ and $f^2$ from $C_F^2$ and $\mathbf{X}^1$;
2 Find the pre-image $\mathbf{X}_F^{1,1}$ of $f^1$ in $\mathbf{X}^1$ that maps to 0, i.e., $\mathbf{X}_F^{1,1} \equiv \{\mathbf{x} \in \mathbf{X}^1 | f^1(\mathbf{x}) = 0\}$. $\mathbf{X}_F^{2,1}$ can be similarly defined;
3 Compute the skewed images of $\mathbf{X}_F^{1,1}$ and $\mathbf{X}_F^{1,1}$ as $X_F^{1,2} \equiv sk(\mathbf{X}_F^{1,1})$ and $\mathbf{X}_F^{2,2} \equiv sk(\mathbf{X}_F^{2,1})$;
4 Evaluate the skewed false positive rate $FP^1$ using $f^1$ on $\mathbf{X}_F^{1,2}$, i.e., $FP^1 \equiv \frac{|\{\mathbf{x} \in \mathbf{X}_F^{1,2} | f^1(\mathbf{x})=1\}|}{|\mathbf{X}_F^{1,2}|}$. $FP^2$ can be similarly defined;
5 If $FP^1 > FP^2$, $C_F^1$ is better. Otherwise, $C_F^2$ is better.

**Algorithm 1**: Skewed False Positive Rate Computation

Figure 6 illustrates two classifiers. Dark boxes show data points that may become a skewed false positive point. We say "may" because of the randomization effect. When the number of these boundary points becomes very large, we can almost be certain that Classifier 1 (Figure 6(a)) shows a smaller skewed false positive rate compared with Classifier 2 (Figure 6(b)). Classifier 2 is thus "tighter", or better.

### 3.3.4 Feature Selection using Skewed False Positive Rates

Currently, we apply a simple forward selection method to choose features based on the "tightness" measure. We assume the same algorithm $C$ but different feature subsets may be used. In other words, we can define a derived classifier $C_s(\mathbf{X}^1)$ with respect to feature subset $s$ (where $\mathbf{X}^1 \subseteq \mathbf{X}_F$) as $C(p(\mathbf{X}^1, s))$ where $p$ is the projection function from Definition 4, and then compare two "classifiers" using Algorithm 1. The actual procedure is shown as follows (Here, we introduce a new parameter $\beta$ which is explained below).

**(a)** Classifier 1 **(b)** Classifier 2

**Figure 6:** Skewed False Positive Rates Example

1. Start with the empty feature set $s_0 = \{\}$.

2. Assume we have obtained $s_{i-1}$. Evaluate classifiers $L_i^j \equiv C_{s_{i-1} \cup \{f_j\}}$ for all feature $f_j \notin s_{i-1}$. We discard those classifiers whose true false positive rate is larger than $\beta$. In addition to these classifiers, we add the base classifier $L_i^0 \equiv C_{s_{i-1}}$ with no new features added. Given $l_i + 1$ classifiers, we compute the best classifier $L_i^{j_0}$ using the procedure described in Chapter 3.3.3.

3. If $s_i = F$, no more features can be added. Return $s_i$.

4. Otherwise, if $j_0 = 0$, return $s_{i-1}$.

5. Otherwise, compute $s_i \equiv s_{i-1} \cup \{f_{j_0}\}$ and proceed with Step 2.

### 3.3.5 Discussion

**Weighted by Variance** In Chapter 3.3.2, the random vector is weighted by the standard deviations of features. It is needed and we can illustrate this with a simple example. Consider an example with two dimensions where the s.t.d is $\sigma_1$ on the x-axis direction and $\sigma_2$ on the y-axis direction. Assume $\sigma_1 >> \sigma_2$ and an random vector with magnitude $\sigma_2$ is used to skew data points. If the vector is parallel to the y-axis, it is easy to see that almost all data points will become skewed false positives. If it is parallel to the x-axis however, the skewed false positive rate would become much smaller. This is unfair and weighting random vectors with standard deviations ensures that the skewed false positive rate is not affected by whichever direction the random vector points to.

**Parameter $\alpha$** $\alpha$ controls the skew degree on each data point. Currently, we choose $\alpha = 0.1 \times \sqrt{\sum_{j=1}^{m} \sigma_j^2}$ where $\sigma_j$ is the standard deviation on dimension $j$.

**Parameter $\beta$** $\beta$ controls the threshold level of false positive rates at which a classifier should be discarded. We currently use $\beta = 0.1$ which turns out to be a good trade-off.

## *3.4 Summary*

In this chapter, we present a systematic feature construction strategy and a new feature selection algorithm for the one-class classification problem. A smaller feature set is critical to improve the efficiency of large scale intrusion detection systems, in particular anomaly detection systems. We will show this with experimental results in Chapter 6.2.

# CHAPTER IV

# NODE-BASED INTRUSION DETECTION

We first present the node-based framework where each IDS agent runs independently on different mobile nodes and monitors itself and its neighborhood. It provides maximum security in a decentralized platform where strong mutual trust can be very difficult to achieve. In later chapters, we will discuss how to relax this assumption.

Before we analyze design issues of an Intrusion Detection System (IDS), let us make the following assumptions: 1) IDS should have access to internal routing elements, such as routing table entries. 2) IDS should also have the capability of intercepting incoming and outgoing packets, including data and routing messages.

In general, statistical-based detection technique, equipped with machine learning tools, can be used to detect abnormal patterns. It has the potential advantage of detecting unknown attacks. But it usually comes with a high false positive rate. Its detection performance heavily depends on selected features.

In contrast, specification-based techniques use specifications to model legitimate system behavior and do not produce false alarms. However, specification development is time consuming. Furthermore, many complex attacks do not violate the specification directly and cannot be detected using this approach.

Our detection approach combines the advantages of both techniques. Consequently, we separate anomalous basic events into two sets, events that directly violate the semantics of EFSAs, and events that require statistical measures.

## *4.1 Detection of Specification Violations*

Some anomalous basic events can be directly translated into violations of EFSAs. We identify three types of violations: **Invalid State Violation**, **Incorrect Transition Violation**

and **Unexpected Action Violation**.

*Invalid State Violation* involves a state that does not appear to be valid in the specification. In our specification, an invalid state means the combination of state variables in the current state is invalid according to the specification. For example, a state with a negative hop count is considered an invalid state. In our implementation, we keep a copy of state variables updated periodically. Thus, we can track invalid changes in state variables.

*Incorrect Transition Violations* occur if invalid transitions are detected. We verify the proper transition by comparing possible input conditions on all transitions from the current state. If a state change occurs while no input conditions can be met, this type of violation is detected. In addition, there are self-looping transitions that do not change the current state. For these transitions, we examine output actions. If some of these output actions (which include packet delivery events and state variable modifications) are detected while corresponding input conditions do not match, we also identify this type of violation. Our implementation monitors incoming and outgoing traffic to determine if input conditions and output actions are properly handled.

*Unexpected Action Violation* corresponds to the situation when the input condition during a transition matches and the new state is as expected, but the output action is not correctly or fully performed.

Let us use AODV as a case study (Figure 3 and Figure 4. We show that the specification-based approach can detect the following anomalous basic events:

Interruption of Data Packets: We monitor the transition T10, where data is forwarded when a valid route is available. An attacker interrupts data packets by receiving but not forwarding data. It is observed as a type of *Unexpected Action Violation* in the transition.

Interruption of Routing Messages: An attacker may choose to interrupt certain types of routing messages by conducting the corresponding transition but not actually sending the routing packets. For more details, *Route Request* messages are delivered in

transition T4, T5, or T5'; *Route Reply* messages are delivered in T9 or T11; *Route Error* messages are delivered in TR2, TR5, T6, T12 or T12'. They can always be identified as *Unexpected Action Violations* in the corresponding transition.

Add Route of Routing Table Entries: We monitor state change to the state when a route to *ob* becomes available (state **Valid**) from other states. If it does not go through legitimate transitions (which include T7, T8, T7' and T8'), it implies that a new route is created bypassing the normal route creation path. It is an *Incorrect Transition Violation* in these transitions.

Delete Route of Routing Table Entries: Similarly, we monitor state change in a reverse direction, i.e., from a valid state (state **Valid**) to a state when a route becomes unavailable (state **Invalid**). If it does not go through legitimate transitions (T12 and T12'), it is detected as an *Incorrect Transition Violation* of these transitions.

Change Route Cost of Routing Table Entries: We can identify changes in sequence numbers or hop counts to the routing table using the memorized state variable copy, when a valid route is available (state **Valid**). They are examples of *Invalid State Violations*.

Fabrication of Routing Messages: Currently, our approach can identify a special type of *Fabrication of Routing Messages*, namely, *Route Reply Fabrication*. We examine the transitions that deliver *Route Reply* messages (transitions T9 and T11). If the output actions are found but the input conditions do not match, we will identify an *Incorrect Transition Violation* in these transitions, which is an indication that outgoing routing messages are in fact fabricated.

To summarize, we define a violation detection matrix. It maps violation information (the violated transition(s) or state and the violation type) to an anomalous basic event. The matrix is shown in Table 4. It can be used to detect attacks that directly violate the AODV specification where we can identify the corresponding types of anomalous basic events. Detection results are summarized in Chapter 6.3.

**Table 4:** Violation Detection Matrix in AODV

| State or Transition(s) | Invalid State Violation | Incorrect Transition Violation | Unexpected Action Violation |
|---|---|---|---|
| TR2, TR5, T6 | | | Interruption of Route Errors |
| T4, T5, T5' | | | Interruption of Route Requests |
| T7, T8, T7', T8' | | Add Route | |
| T9, T11 | | Fabrication of Route Replies | Interruption of Route Replies |
| T10 | | | Interruption of Data Packets |
| T12, T12' | | Delete Route | Interruption of Route Errors |
| Valid | Change Route Cost | | |

## *4.2 Detection of Statistical Deviations*

For anomalous events that are temporal and statistical in nature, statistical features can be constructed and applied to build a machine learning model that distinguishes normal and anomalous events.

Using the taxonomy of anomalous basic events in Table 1, we identify the following anomalous basic events that cannot be addressed well using the specification-based approach.

- Flooding of Data Packets
- Flooding of Routing Messages
- Modification of Routing Messages
- Rushing of Rushing Messages

There are two different detection methods: misuse detection and anomaly detection. It is well known that misuse detection has better precision but works poorly on unknown

attacks. On the other hand, anomaly detection can detect possible anomalies without learning existing attack instances, therefore it is not biased towards known attacks. However, it is usually less accurate on the same attack whose attack signature has been known. Let us start from misuse detection first.

### 4.2.1 Misuse Detection

We first determine a set of statistical features based on activities from anomalous basic events that cannot be effectively detected using the specification-based approach. Features are computed periodically based on the specified statistics from *all* running EFSAs, and stored in audit logs for further inspection. To build a detection model, we use a number of off-line audit logs (known as training data) which contain attacks matching these anomalous basic events. Furthermore, each record is pre-labeled with the type of the corresponding anomalous basic event (or normal if the record is not associated with any attacks) because we know which attacks are used. They are processed by a classifier and a detection model is generated. The model can be a set of detection rules, or other types of detection models, depending on the actual classifier. The model is then used to detect attacks in the test data.

### 4.2.2 Anomaly Detection: Cross Feature Analysis

Anomaly detection methods are especially appealing in ad hoc networks, because they only use established normal profiles and can identify any unreasonable deviation from normal profiles as the result of some attacks. Since MANET is still under heavy development and not many MANET-specific attacks have emerged, we believe that anomaly detection is the preferred technique in the current stage.

Our anomaly detection approach is based on data mining technologies because we are interested in *automatically* constructing detection models using logs (or trails) of system and network activity data. Some intrusion detection techniques suggested in literature use probabilistic analysis where the resulting models are not straightforward to be re-evaluated by human experts [28]. Some data mining models require temporal sequence from data

stream [49], which is domain specific and highly inefficient when a large feature set is involved. The problem of anomaly detection in MANET involves a large feature set. It requires us to develop new data mining approaches.

We have developed a new approach based on *Cross-Feature Analysis* that we believe is suitable for MANET anomaly detection. We observe that strong feature correlation exists in normal behavior patterns. And such correlation can be used to detect deviations caused by abnormal (or intrusive) activities. For instance, consider a home network which is mainly composed of wireless connected home appliances and possibly a few human held wireless devices (such as PDAs). Networking controllers reside in all such nodes so that they can form an ad-hoc network. Naturally, we would expect that major portion of the established routing fabric remains stable for a long time since home appliances rarely change locations. We also require that some sensor device be installed on each node which can record useful statistics information. Let's imagine that one sensor finds out that the *packet dropping rate* increases dramatically without any noticeable change in the *change rate of routing entries*, it is highly likely something unusual has happened. The controller in the node may have been compromised to refuse forwarding incoming traffic while no route change actually takes place (which, if happens, may result in temporary packet dropping due to invalid stale routes). The relationship between the features *packet dropping rate* and *change rate of routing entries* can be captured by analyzing the normal patterns of historical data and be used later to detect (unseen) anomalies.

More formally, in the *Cross-Feature Analysis* approach, we explore correlations between each feature and all other features. Thus, the anomaly detection problem can be transformed into a set of classification sub-problems, where each sub-problem chooses a different feature as a new class label and all other features from the original problem are used as the new set of features. The outputs of each classifier are then combined to provide an anomaly detector.

The basic idea of a **cross-feature analysis** framework is to explore the correlation between one feature and all the other features, i.e., try to solve the classification problem $\{f_1, f_2, \ldots, f_{i-1}, f_{i+1}, \ldots, f_L\} \rightarrow f_i$ where $\{f_1, f_2, \ldots, f_L\}$ is the feature vector. Note that in the machine learning area, the terminology *class* in a classification system represents the task to be learned based on a set of features, and the *class label*s are all possible values a class can take. In the domain of intrusion detection, we would most likely to learn the system healthy status (the *class*) from known system information (the *features*), and *normal* or *abnormal* are both possible class labels.

A basic assumption for anomaly detection is that normal and abnormal events should be able to separate from each other based on their corresponding feature vectors. In other words, given a feature vector, we can tell whether the related event is normal or not without ambiguity. This assumption is reasonable since otherwise the feature set is not sufficient and must be redefined. Under this assumption, we can name a feature vector related to a normal event a **normal vector**, for short. Similarly, we call a feature vector not related to any normal events an **abnormal vector**. Here, we assume that all feature values are discrete. A generalized extension will be discussed later in this dissertation. We re-formulate the problem as follows. For all normal vectors, we choose one feature as the target to classify (which is called the **labeled feature**), and then compute a model using all normal vectors to predict the chosen target feature value based on remaining features. In other words, we train a classification model $C_i : \{f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_L\} \rightarrow \{f_i\}$. For normal events, the prediction by $C_i$ is very likely to be the same as the true value of the feature; however, for anomalies, this prediction is likely to be different. The reason is that $C_i$ is trained from normal data, and their feature distribution and pattern are assumed to be different from those of anomalies. This implies that when normal vectors are tested against $C_i$, it has a higher probability for the true and predicted values of $f_i$ to match. Such probability is significantly lower for abnormal vectors. Therefore, by evaluating the degree of result matching, we are more likely to find difference between normal and abnormal patterns.

We name the model defined above a **sub-model with respect to** $f_i$. Obviously, relying on one sub-model with respect to one labeled feature is insufficient as we haven't considered the correlation among other features yet. Therefore the model building process is repeated for every feature and up to $L$ sub-models are trained. Once done, we have accomplished the first step of our cross-feature analysis approach, i.e. the training procedure, which is summarized in Algorithm 2.

**Data**: feature vectors of training data $f_1, \ldots, f_L$;
**Result**: classifiers $C_1, \ldots, C_L$;
**begin**
  $\forall i$, train $C_i : \{f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_L\} \rightarrow f_i$;
  return $C_1, \ldots, C_L$;
**end**

**Algorithm 2**: Cross-Feature Analysis: Training Procedure

To generalize the framework to continuous features or discrete features with an infinite value space (e.g., the integer set), we should keep in mind that they cannot be used directly as class labels since only discrete (nominal) values are accepted. We can either discretize them based on frequency or use multiple linear regression. With multiple linear regression, we use the log distance, $|log(\frac{C_i(x)}{f_i(x)})|$, to measure the difference between the prediction and the true value, where $C_i(x)$ is the predicted value from sub-model with respect to $f_i$.

Once all sub-models have been trained, we analyze trace logs as follows. When an event is analyzed, we apply the feature vector to all sub-models, and count the number of models whose predictions match the true values of the labeled features. The count is then divided by $L$, so that the output, which is called the **average match count** throughout this dissertation, is normalized. We do not need all sub-models to match. In fact, what we need is a *decision threshold*. An event is classified as anomaly if and only if the *average match count* is below the threshold. Since it is hard to develop a perfect solution to determine the decision threshold for a general anomaly detection problem directly, and in practice, a small value of false positive rate is often allowed, we can determine the threshold as follows: compute the *average match count* values on all normal events, and use a lower

bound of output values with certain confidence level (which is one minus false positive rate). As a summary, Algorithm 3 lists the strawman version of the test procedure. For convenience, $f_i(x)$ denotes the value of feature $f_i$ belonging to event $x$. $[\![\pi]\!]$ returns 1 if the predicate $\pi$ is true.

**Data**: classifiers $C_1, \ldots, C_L$, event $x = (f_1, \ldots, f_L)$, decision threshold $\theta$;
**Result**: either normal or anomaly;
**begin**
  AvgMatchCount $\leftarrow \sum_i [\![C_i(x) = f_i(x)]\!]/L$;
  **if** *AvgMatchCount* $\geq \theta$ **then** return "normal";
  **else** return "anomaly";
**end**

**Algorithm 3**: Cross-Feature Analysis: Testing Procedure Using *Average Match Count*

One straightforward improvement to the strawman algorithm is to use probability instead of the 0-1 count, the probability values for every possible class are available from most inductive learners (e.g., decision trees, induction rules, naive Bayes, etc.) This approach can improve detection accuracy since a sub-model should be preferred where the labeled feature has stronger confidence to appear in normal data. Algorithm 3 can actually be regarded as a special case under the assumption that the predicted class is the only valid class and hence has a probability of 1.0, so the probability for the true class is either 1 (when the rule matches) or 0 (otherwise). More strictly, assume that $p(f_i(x)|x)$ is the estimated probability for the true class of the labeled feature, we define **average probability** as the average output value of probabilities associated with true classes over all classifiers. The optimized version is shown in Algorithm 4.

We now discuss in detail how the probability function can be calculated in some popular classification algorithms. Decision tree learners (such as C4.5 [72]) uses a divide-and-conquer strategy to group examples with the same feature values until it reaches the leaves of the tree where it cannot distinguish the examples any further. Suppose that $n$ is the total number of examples in a leaf node and $n_i$ is the number of examples with class label

**Data**: classifiers $C_1, \ldots, C_L$, event $x = (f_1, \ldots, f_L)$, decision threshold $\theta$;
**Result**: either normal or anomaly;
**begin**
    AvgProbability $\leftarrow \sum_i p(f_i(x)|x)/L$;
    **if** *AvgProbability* $\geq \theta$ **then** return "normal";
    **else** return "anomaly";
**end**

**Algorithm 4**: Cross-Feature Analysis: Testing Procedure Using *Average Probability*

$\ell_i$ in the same leaf. $p(\ell_i|x) = \frac{n_i}{n}$ is the probability that $x$ is an instance of class $\ell_i$. We calculate probability in a similar way for decision rule classifiers, e.g. RIPPER [19]. For naive Bayes classifiers[1], we assume that $a_j$'s are the feature values of $x$, $p(\ell_i)$ is the prior probability or frequency of class $\ell_i$ in the training data, and $p(a_j|\ell_i)$ is the prior probability to observe feature attribute value $a_j$ given class label $\ell_i$, then the score $n(\ell_i|x)$ for class label $\ell_i$ is: $n(\ell_i|x) = p(\ell_i) \prod_j p(a_j|\ell_i)$ and the probability is calculated on the basis of $n(\ell_i|x)$ as $p(\ell_i|x) = \frac{n(\ell_i|x)}{\sum_k n(\ell_k|x)}$.

**An Illustrative Example** We use a simplified example to demonstrate our framework. Consider an ad-hoc network with two nodes. Packets can only be delivered from one end to the other if they are within each other's transmission range. We define the following three features. 1) Is the *other node reachable*? 2) Is there any *packet delivered during last 5 seconds*, and 3) is there any *packet cached for delivery during last 5 seconds*? For simplicity, we assume all features are binary valued, i.e., either **True** or **False**. All normal events are enumerated in Table 5. We then construct three sub-models with respect to each feature, shown in Table 6. The "Probability" columns here denote the probability associated with predicted classes. We use an illustrative classifier in this example that works as follows. If only one class is seen in all normal events where other non-labeled features have been assigned with a particular set of values, the single class is selected as the

[1]One such implementation (NBC) is publicly available at http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html.

predicted class with the associated probability of 1.0. If both classes are seen, label *True* is always selected with the associated probability of 0.5. If none are seen (which means the combination of the other two feature values never appears in normal data), we select the label which appears more in other rules, with the associated probability of 0.5. To compute the probability for the true class, we use the probability associated with the predicted class if it matches, or one minus the associated probability if it does not. For example, the situation when a route is viable, no data is cached and no data is therefore delivered is a normal case. We apply the corresponding feature vector, $\{True, False, False\}$, into all three sub-models and all match the predicted classes. But the first sub-model with respect to the feature "Reachable?" has a probability of 0.5 only, which is obvious since when no data is delivered, it does not matter whether the route is up or not. The average match count is then calculated as $(1 + 1 + 1)/3 = 1$, and the average probability is $(1 + 1 + 0.5)/3 = 0.83$. Suppose we use a threshold of 0.5, then both values tell that the event is normal, which is right. A complete list of the *average match counts* and *average probabilities* for all possible events (both normal and abnormal) is shown in Table 7. Note that we use abbreviations here where AMC stands for the *Average match count*, and AP is the *Average probability*. The results clearly show that given a threshold of 0.5, both Algorithm 3 and 4 work well to separate normal and abnormal events, while Algorithm 4 works better as it achieves perfect accuracy (Algorithm 3 has one false alarm with the input $\{False, False, False\}$).

**Table 5:** Normal Events in the 2-node Network Example

| Reachable? | Delivered? | Cached? |
|---|---|---|
| True | True | True |
| True | False | False |
| False | False | True |
| False | False | False |

**Table 6:** Cross-Feature Analysis Models in the 2-node Example

**(a)** Sub-model with respect to 'Reachable?'

| Delivered? | Cached? | Reachable? | Probability |
|---|---|---|---|
| True | True | True | 1.0 |
| False | False | True | 0.5 |
| False | True | False | 1.0 |
| True | False | True | 0.5 |

**(b)** Sub-model with respect to 'Delivered?'

| Reachable? | Cached? | Delivered? | Probability |
|---|---|---|---|
| True | True | True | 1.0 |
| True | False | False | 1.0 |
| False | True | False | 1.0 |
| False | False | False | 1.0 |

**(c)** Sub-model with respect to 'Cached?'

| Reachable? | Delivered? | Cached? | Probability |
|---|---|---|---|
| True | True | True | 1.0 |
| True | False | False | 1.0 |
| False | False | True | 0.5 |
| False | True | True | 0.5 |

## *4.3 Topology Aware Normalization*

A typical procedure to perform anomaly detection is to collect example data from a training profile and then to build the anomaly detection model (or simply *model*) based on the training data. However, ad hoc networks are well known for their dynamic topology and ever-changing routing structure. Therefore, one training profile must choose a set of topology and scenario parameters. One parameter example is the maximum moving speed (*speed*) enforced by all nodes. It varies greatly for different application scenarios. For example, PDAs held by pedestrians will have a different speed range from wireless devices installed on moving vehicles.

To address the problem caused by profiles using different topology parameters, one simple approach is to build a series of models, one for each profile. The limitation of this approach is obvious. There are simply infinite number of possible profiles. It is very

**Table 7:** Event Outcome in the 2-node Example

| Reachable? | Delivered? | Cached? | Class | AMC | AP |
|---|---|---|---|---|---|
| True | True | True | Normal | 1 | 1 |
| True | False | False | Normal | 1 | 0.83 |
| False | False | True | Normal | 1 | 0.83 |
| False | False | False | Normal | 0.33 | 0.67 |
| True | True | False | Abnormal | 0.33 | 0.17 |
| True | False | True | Abnormal | 0 | 0 |
| False | True | True | Abnormal | 0.33 | 0.17 |
| False | True | False | Abnormal | 0 | 0.33 |

uncommon that a specific application can fix all topology parameters. Generally, there are always some parameters remain variable. Therefore, an efficient topology aware detection approach is highly desired.

In this work, we present a topology-aware method to normalize features, where all profiles are merged into a cluster, and a single (anomaly detection) model can be applied to all profiles in the cluster. We present comparison results in terms of detection accuracy to validate the approach.

We first present a brief overview how existing anomaly detection algorithms work and why it is not practical to be used by scenarios with a variety of possible topology parameters. We then present a topology aware algorithm that can address this problem.

### 4.3.1 Concepts

Here, we consider *anomaly detection* as a machine learning based One-Class Classification problem. It consists of two steps. During the training step, examples are collected from a training dataset that contains only normal data. Each example is composed of a number of features (or attributes). We use the *Cross Feature Analysis* algorithm in Chapter 4.2.2. It should be noted that, however, the proposed procedures can be easily migrated to other algorithms in general.

A topology parameter, or simply *parameter*, is a variable that defines the basic characteristics of a topology. We list the parameters in Table 8. Each parameter also defines a

default value. These default values define the “standard” scenario which is used in other experiments in our experiments. We note that these parameters are used in many other research work as well.

**Table 8:** A Typical Ad Hoc Scenario

| parameter | meaning | default value |
|---|---|---|
| nn | number of nodes | 20 |
| speed | maximum speed of nodes | 20 meters/second |
| pause | pause time between any two moves | 50 seconds |
| radius | transmission range | 250 meters |
| mc | maximum number of connections | 20 |

A *scenario* is defined as a set of value assignments for *all* parameters. Similarly, an *application* restricts the value ranges of certain parameters, but not necessarily all. A “pedestrian” application may specify the speed range as [0, 5] and the pause range as [0, 600], for example.

### 4.3.2 Analysis on Feature Patterns

It is important to realize that the features that are collected either through domain knowledge or EFSA may be implicitly dependent on topology parameters. To understand that, we perform a simple test. We first choose a "standard" scenario. We then vary only one parameter from the "standard" scenario, and obtain the means of all features on the varied scenarios.

For a case study, we use the AODV EFSA feature set (for details, please see Chapter 6.6). We then demonstrate the means on *some* relevant feature values in Table 9.

From this table, we can see that many features are affected by topology parameters. We can characterize the trends how parameters affect the means of each feature. For example, when *radius* increases, all features (including those not listed here) will decrease except for T10, which will also increase. The phenomenon can be explained as follows. When

**Table 9:** Feature Statistics with Different Topology Parameters

| Features | Valid | T2 | T5 | T10 | seq | rreq_seq |
|---|---|---|---|---|---|---|
| nn = 5 | 1.6 | 0.16 | 1.3 | 63 | 1.8 | 0.29 |
| nn = 20 | 2 | 0.015 | 1.7 | 23 | 0.54 | 0.2 |
| nn = 50 | 2.2 | 0.0094 | 2.1 | 9.7 | 0.24 | 0.062 |
| radius = 100 | 3.8 | 0.41 | 3.5 | 12 | 3.8 | 0.28 |
| radius = 250 | 1.5 | 0.035 | 1.1 | 50 | 0.77 | 0.31 |
| radius = 800 | 0.34 | 0.036 | 0.26 | 58 | 0.24 | 0.042 |
| speed = 5 | 1.1 | 0.025 | 0.81 | 52 | 0.56 | 0.19 |
| speed = 20 | 1.4 | 0.021 | 1 | 44 | 0.78 | 0.31 |
| speed = 80 | 2.7 | 0.051 | 2 | 45 | 1.4 | 0.49 |
| pause = 40 | 1.3 | 0.031 | 0.97 | 45 | 0.71 | 0.28 |
| pause = 160 | 1.3 | 0.043 | 1 | 50 | 0.66 | 0.28 |
| pause = 640 | 1.1 | 0.068 | 0.97 | 54 | 0.73 | 0.14 |
| mc = 3 | 1.1 | 0.039 | 0.82 | 19 | 0.68 | 0.2 |
| mc = 20 | 4.6 | 0.053 | 3.1 | 110 | 2.4 | 0.76 |
| mc = 80 | 99 | 0.92 | 65 | 290 | 70 | 1.5 |

radius increases, more nodes are connected directly (in 1-hop), therefore most routing activity are reduced, so as the number of transitions and the number of route messages (route requests and route replies). The *data forward transition* (T10) increases since successful data forwarding rate now becomes more likely. Similar observations can also be seen on other parameters.

### 4.3.3 Challenge

Based on the above observation, it is unwise to feed examples from different scenarios blindly into one "super" model (experimental results in Chapter 6.6 also verify that). Another alternative is to build a separate model for each possible scenario. This is not always feasible, because the possible combinations of parameter values are infinite in general. Even in applications where the possible parameter combinations are limited, the maintenance of multiple models remains extremely expensive.

### 4.3.4 Topology Aware Normalization Algorithm

We propose the following changes to the anomaly detection algorithm as part of the topology aware effort:

1 Add topology parameters as features;
2 Normalize feature values in such a way that each feature has a zero-mean and a standard deviation of one.

**Algorithm 5**: Topology Aware Normalization Algorithm

Either step can be optional. In our current experiment however, both are used.

The first step can be implemented trivially (we assume the topology parameters are always known in advance). The second step involves a pre-learning step for each different scenario, where the statistics of every feature can be obtained (per scenario). The pre-learning step in unbiased, because we always use the normal data (only) as the training dataset, and therefore they are not influenced by attacks.

In practice, it is not always practical to do pre-learning for each individual scenario. Instead, we can perform pre-learning for a group of scenarios where parameter values are close to each other. One variation is to divide the value space of each parameter into $n$ buckets (where $n$ is a pre-determined parameter). Scenarios with all parameters belong to the same bucket will share the same statistics. We will show more results using this variation scheme in Chapter 6.6.

## *4.4 Summary*

Based on the attack taxonomy study, we use protocol specifications to model normal protocol behavior, and develop features that can be used by intrusion detection systems. By applying both specification-based and statistical-based detection approaches using these features, we have the advantages of both. Specification-based approach has no false alarm, statistical-based approach can detect attacks that are statistical or temporal in nature.

In particular, we developed an innovative anomaly detection approach that explores the

correlation among different features and use them to build a detection model purely based on normal data.

We also present a novel topology aware anomaly detection mechanism. Using this mechanism, samples collected from multiple profiles can be used to build a single anomaly detection model. So far, we are not aware of any published literature that addresses how to perform topology aware intrusion detection so far, partially because this problem is more or less specific to MANETs and remains less explored.

# CHAPTER V

# CLUSTER-BASED INTRUSION DETECTION

## *5.1 Motivation*

It may not be efficient and often not necessary to run an IDS agent on every MANET node. Furthermore, some sophisticated attacks can evade a node-based IDS or make it rather inefficient. Let us consider the following scenarios:

System level adversaries: An IDS agent should not be trusted when its hosting system encounters a powerful adversary, which we call a *system level adversary*. A system level adversary can control the whole system and thus alter the IDS behavior arbitrarily. Neighbor-monitoring sensors [59] can mitigate the issue, but only when the number of system level adversaries can be bounded. Furthermore, the use of neighbor-monitoring sensors can be prohibitive because of energy consumption, as we have discussed in Chapter 1.5.

Grouped attacks: In situations where attacks affect nodes in a whole network segment in a similar way, IDS agents on these nodes may eventually come up with similar detection outputs. We call these attacks *grouped attacks*. One common example is DoS attacks such as flooding or packet dropping. Redundancy with IDS agents can be helpful in terms of fault-tolerance, where network sustainability is the highest priority, but a more efficient solution may be preferred in other scenarios.

Distributed attacks: Local observations do not always contain sufficient evidence to detect an attack. In other words, an attack may behave benignly from the view of every affected node individually. They are known as *distributed attacks*. Consider the *Sinkhole* attack where all traffic is redirected to go through a particular node (here

the node is known as the sinkhole). When an individual IDS agent detects that all outbound traffic is forwarded by the same next hop, it cannot ascertain whether there is an attack or not simply because there is one neighbor connecting to the outside world.

These problems can be mitigated by a new cluster-based framework. We define a cluster as a group of mobile nodes that includes one or more special nodes, known as *clusterheads*. To ensure that the clusterhead has the ability to watch the activities on every node of the cluster, we require that all wireless interfaces support promiscuous mode operations. In our design, nodes belonging to a cluster must reside within one hop from the clusterhead, while IDS agents are only enabled on clusterheads.

Using the cluster-based framework, the above issues can be handled effectively:

System level adversaries: In a cluster-based IDS, clusterheads can optimize energy use by scheduling only a subset of cluster members who will activate their neighbor-monitoring sensors at one time. Other members can minimize their energy consumption at the same time.

Grouped attacks: In a cluster-based IDS, only clusterheads run IDS agents, and only a subset of cluster members collects information through local IDS sensors at one time. Therefore, the overall computational overhead can be significantly reduced.

Distributed attacks: A cluster-based structure can help detect these attacks. We take Sinkhole as an example. If a source routing based protocol, such as DSR [39], is used, we can easily verify that a single host appears in every source route from every cluster member to every possible destination. With high probability, the evidence is convincing enough to raise an alarm. In a general routing protocol, we can also perform traffic analysis based on the distribution of the last hops before a packet is delivered outside the whole cluster. Given destinations of all packets are distributed uniformly

throughout the whole network, alarms can be raised with certain confidence if some member appears with extremely high usage as the last hop.

While the cluster-based framework has many advantages, it cannot completely replace the node-based solution. For example, it can be more vulnerable in the case of *Denial-of-Service* attacks because IDS agents are now run on fewer nodes. It remains our future work to improve its security and reliability. We will study a few factors that impact the effectiveness of a cluster-based solution in Chapter 6.4 and Chapter 6.5.

## *5.2 Cluster Formation Protocols*

### 5.2.1 Overview

A MANET can be organized into a number of clusters in such a way that every node is a member of at least one cluster. A cluster is defined as a group of nodes that are *close* to each other. The criteria of 'close' is that a node in the cluster, the *clusterhead*, has all other members, known as *citizens*, in its 1-hop vicinity. As a special case, a node that cannot be reached by anyone else (or under other special circumstances as described below) forms a *single node cluster*, or SNC. The *size* of a cluster is defined as the number of nodes in the cluster (including both clusterhead and citizens) and is denoted as $S_C$.

It is imperative that the clusterhead assignment be fair and secure. By fairness, we mean that every node should have a fair chance to serve as a clusterhead. Note that fairness has two components, *fair election*, and *equal service time*. We currently do not consider differentiated capability and preference (such as criteria based on network or CPU load, unless they can be verifiable) and assume that every node is equally eligible. Thus, *fair election* implies randomness in election decision, while *equal service time* can be implemented by periodical fair re-election. By security, we mean that none of the nodes can manipulate the selection process to increase (or decrease) the chance for it (or another node) to be selected. Obviously, if randomness of the election process can be guaranteed, then security can be guaranteed too.

Although there are other cluster formation protocols available, they do not satisfy our requirements discussed above. For example, the *Leader Election* (or cluster organization) algorithms in [86, 3] choose either a common evaluation function or node-specific utility functions to compute a score for every node, and the node with maximal score is elected as the clusterhead. It does not guarantee the random selection of clusterheads, because a node can easily advertise a high score for itself, unless care is taken to make the evaluation process verifiable and with non-repudiation.

Before we describe our clustering formation protocols, let us state a few assumptions.

- Each node contains a unique and ordered identifier.
- Every node can overhear traffic within its transmission range (this is a common requirement by MANET monitoring schemes, e.g. [59]).
- Neighbor information is always available. Usually this is implemented by periodically broadcasting HELLO messages and listening to the neighbors' response. Given the assumption, we can obtain the number of neighbors of node $i$. Let us denote the value to be $N_i$.
- A secure, fast and reliable node to node communication infrastructure is available. The infrastructure has to be light-weighted because MANET nodes are often resource-constrained. Recently, efficient protocols for MANET are proposed, such as TESLA [70], which carries only symmetric cryptographic functions. These protocols make certain assumption that loose synchronized clocks are available.

The basic idea of our cluster formation protocols is as follows.

Clique Formation: We first form special head-less clusters, known as cliques. This stage provides necessary preparation for a later clusterhead election. Clique formation algorithms have been extensively studied. In particular, we adopt the algorithm from [45].

Clusterhead Election: Each clique member chooses a random token and broadcasts it to the whole clique. After a full round of token exchange, each member independently runs a modulo-XOR function that combines all random tokens. The modulo-XOR function satisfies the *full randomness* property in that as long as at least one non-malicious member produces a true random input, the output from that function will be randomized. The function output is then used as the initial seed of a cryptographically secure pseudo-random sequence generator. Assume we need to generate $m$ clusterheads per cluster, we can generate the random sequence until $m$ distinct values have appeared. These values identify the $m$ elected clusterheads.

Cluster Reconstruction: We deal with node mobility by dynamically adjusting cluster membership, in an optional *Cluster Recovery Stage*. If a citizen loses the connection with its previous clusterhead or a clusterhead loses all its citizens, it broadcasts an ADD_REQUEST message. A clusterhead who receives the message replies with an ADD_REPLY. The lost node acknowledges the first received ADD_REPLY and establishes cluster membership accordingly. If there is no ADD_REPLY after a predefined timeout, the orphan creates a single member cluster.

To ensure long-term fairness, even if there is no membership change involved, each cluster will have a limited life cycle. After a *re-election period* from its formation, every cluster will invalidate itself and start a new election. Note that the new election chooses a set of new clusterheads that completely independent from the previous elected clusterheads.

Note that it may be necessary to have multiple clusterheads elected for a cluster so that clusterheads are also being watched through mutual monitoring. This will reduce the risk of compromise. It is obvious that if there is only one clusterhead and it is compromised, the whole IDS cluster would not be functional. It also discourages a clusterhead from behaving selfishly, e.g., not doing the required monitoring work. We will investigate how to decide

the number of clusterheads needed and the election schedules based on run-time conditions in Chapter 6.4.

The details of cluster formation algorithms are discussed in the next subsection.

### 5.2.2 Details of Cluster Formation Protocols

Figure 7 shows a finite state machine demonstrating the states of the MANET nodes and the state transitions that are enabled by the cluster formation protocols.

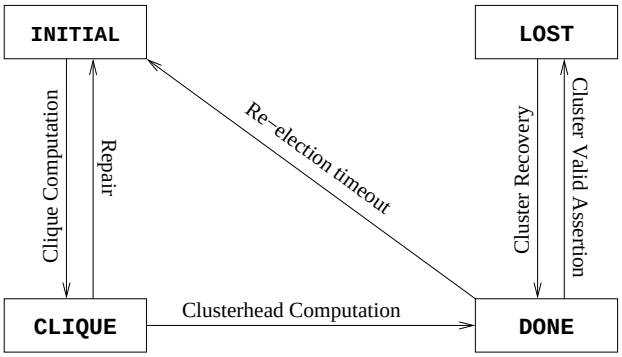



**Figure 7:** Finite State Machine of the Cluster Formation Protocols

Initially, all nodes are in an INITIAL state. They temporarily assume themselves SNCs, so that they can do intrusion detection for themselves, just as in the per-node based approach. We perform an initial clusterhead setup round, which is composed with two protocols: *Clique Computation* and *Clusterhead Computation*.

**Clique Computation Protocol** A clique is defined as a group of nodes where every pair of members can communicate via a direct wireless link. Note that the definition of a *clique* is stricter than the definition of a *cluster*. The clique requirement can be relaxed right after the clusterhead has been computed. That is, only the clusterhead needs to have direct links with all members. We use the cluster formation algorithm from [45] to compute cliques. Once the protocol is finished, every node is aware of its fellow clique members. We denote the clique containing $i$ as $CL_i$, i.e., $\forall j \in CL_i, CL_j = CL_i$. We define $CL'_i = CL_i - \{i\}$.

Once the Clique Computation Protocol has finished, all nodes enter CLIQUE state.

**Clusterhead Computation Protocol** The purpose of this protocol is to randomly select $m$ nodes in the computed clique as clusterheads. Without loss of generality, let us describe the procedure on the $i$-th node in Algorithm 6.

1 Generate a random integer $R_i$;
2 Broadcast a message ELECTION_START=($ID_i$,HASH($ID_i, R_i$)) to $CL'_i$. HASH is a common hash function. A corresponding timer $T_1$ is set up. On Receiving *all* ELECTION_START from $CL'_i$, broadcast the message ELECTION=($ID_i, R_i$) to clique $CL'_i$;
3 If $T_1$ is up, every node for whom ELECTION_START has not be received is excluded from
4 $CL_i$Jump to Step 4; On Receiving ELECTION from node $j$, verify its hash value matches the value in the ELECTION_START message from $j$. $R_j$ is then stored locally;
5 If $R_j$ from all members $CL'_i$ have arrived, compute H=SEL($R_0, R_1, R_2, ..., R_{S_C-1}$) where SEL is the selection function;
6 The function output is then used as the initial seed of a cryptographically secure pseudo-random sequence generator. Assume we need to generate $m$ clusterheads, we can generate the random sequence until $m$ distinct values have appeared. These values identify the $m$ elected clusterheads: $H_1$ to $H_m$. Assume they are ordered so that all nodes have consistent view of indices of each clusterhead;
7 For clusterhead $H_k$, if $H_k \neq i$ (i.e., node $i$ is a citizen), do the following;
   a) Send ELECTION_DONE to $H_k$;
   b) Wait for ELECTION_REPLY from $H_k$, then enter DONE state;
8 Otherwise, as a clusterhead, $H_k$ (or $i$) performs following ;
   a) Set up a timer $T_2$;
   b) On Receiving ELECTION_DONE, verify it is from $CL'_i$;
   c) If $T_2$ is up, citizens from whom ELECTION_DONE has not be received are excluded from $CL_i$. Broadcast ELECTION_REPLY to $CL'_i$ and enter DONE state.

**Algorithm 6**: Clusterhead Computation Protocol

We use several techniques to guarantee the fairness and security of the election process. Most importantly, each node $i$ contributes a random value $R_i$ to the input, and then a common selection function is used by all nodes to compute a integer from 0 to $S_C - 1$ from a total of $S_C$ inputs. The output of the election function must have a uniform distribution in $[0, S_C - 1]$. The selection function we use is simply the modular Exclusive OR (or XOR) function, i.e., $f(R_0, R_1, R_2, ..., R_{S_C-1}) = (\bigoplus_{i=0}^{S_C-1} R_i)$ MOD $S_C$. A nice property of XOR is that as long as one input is random (i.e., from a "well-behaving" node), the output is random. The random values are fully exchanged within the cluster (clique) and the selection function is computed in a distributed manner, i.e., on each node, to decide the clusterhead.

This guarantees that the same clusterhead be computed by all cluster members.

Once a clusterhead is determined, it copies the clique member list to a citizen list $CT_C$. The suffix C denotes the current cluster controlled by the clusterhead. All cluster are independent, even if memberships of two clusters may overlap.

**Cluster Valid Assertion Protocol** All nodes in the DONE state should perform the assertion protocol listed in Algorithm 7.

1 Since the network topology tends to change in an ad hoc network, connections between the elected clusterhead and some citizens nodes may be broken from time to time. If a link between a citizen $Z$ and a clusterhead $H$ has been broken, $Z$ will check if it is in another cluster. If not, it enters LOST state and activates the *Cluster Recovery Protocol*. Also, $Z$ is removed from $H$'s citizen list $CT_C$. If there is no more citizens in cluster $C$, $H$ becomes a citizen if it belongs to another cluster. Otherwise, $H$ enters LOST state and activates the *Cluster Recovery Protocol*;

2 Even if no membership change has occurred, the clusterhead cannot function forever because it is neither fair in terms of service and unsafe in terms of the long time single-point control and monitoring. We enforce a mandatory re-election timeout, $T_r$. Once the $T_r$ expires, all nodes in the cluster enters the INITIAL state and start a new clusterhead setup round. If the clique property still holds, the *Clique Computation* step can be skipped.

**Algorithm 7**: Cluster Valid Assertion Protocol

**Cluster Recovery Protocol** In the case that a citizen loses its connection with previous clusterhead or a clusterhead loses all its citizens, the node enters LOST state and initiate the *Cluster Recovery Protocol* to re-discover a new clusterhead. Again, without loss of generality, we discuss the protocol on the $i$-th node in Algorithm 8.

#### *5.2.2.1 Discussion*

Since clusters can overlap, a node can belong to multiple clusters. Therefore, the notation $CL_i$ of node $i$ can actually take multiple values. For simplicity of the protocol description, we use the singular form but keep in mind that a node in multiple clusters should perform the *Clusterhead Computation Protocol* for each of its clusters independently.

1 A request message ADD_REQUEST=($ID_i$) is broadcast with a timer $T_3$;
2 A clusterhead $H$ receives the request and replies ADD_REPLY=($ID_H$) only after a short delay $T_d$ (0.5s in our implementation). The delay is introduced in hope that a connection has been stable for $T_d$ can remain to be stable for a fairly long time;
3 Node $i$ replies the first ADD_REPLY it received, i.e., ADD_ACK=($ID_i$). And enters DONE state. Additional ADD_REPLYs are ignored;
4 On Receiving ADD_ACK, $H$ adds $i$ into its $CT_C$;
5 If $T_3$ is up and no ADD_REPLY is received, there is no active clusterhead nearby. Node $i$ enters INITIAL state to wait for other lost citizens to form new cliques and elect their new clusterheads.

**Algorithm 8**: Cluster Recovery Protocol

In the *Clusterhead Computation Protocol*, we assume the topology remains static during computation. In a mobile environment, this assumption does not always hold. A remedy is for each cluster member to monitor the neighborhood actively. Once a link is broken, a REPAIR message is broadcast by both ends of the link, and all other nodes in the cluster will be aware of that. All nodes in the cluster then re-enter INITIAL state and restart the *Clique Computation Protocol*.

We require that all nodes have direct links to each other (i.e., they are in a clique) in the cluster formation process. This is intentional so that spoofed messages can be detected and contested because the nodes can overhear each other. Whenever such dispute arises, the nodes can switch to a more secure way (e.g., authenticated channels) to exchange messages.

Finally, our protocols are meant to be a framework that can be customized according to operational conditions and security needs. For example, a malicious node has a $\frac{1}{S_C}$ chance to be elected as the clusterhead. It can then launch certain attacks without being detected because it is the only node in the neighborhood that is supposed to run the IDS and its IDS may have been disabled already. If this chance is not acceptable, we can elect multiple clusterheads each of which runs a separate IDS to monitor the whole cluster. The worst case is to run an IDS agent on each cluster member. There is obviously a trade-off between efficiency and security. We will investigate how to dynamically adjust the number of clusterheads (or monitoring nodes) according to resource constraints and potential threats.

### 5.2.3 Security Concerns

As an approach dedicated to detect malicious behavior, our protocol itself has to be secure in the first place. In addition to conventional attacks such as man-in-the-middle and replay attacks (which are addressed by enforcing a secure communication channel and sequence numbers verifiable by neighbors), we also address the following specialized attacks with particular consideration.

**Defending against delayed random value distribution** In order to prevent a malicious node from manipulating the election outcome, e.g., by sending its random number only after it receives the random numbers from all other nodes, the exchange of random numbers among the nodes proceeds in two rounds. First, each node computes a random number and its hash using a common hash function, then sends out only the hash value. Second, only after receiving all hash values from all other nodes, a node sends out the actual random number. A multi-round process for exchanging the random numbers, which corresponds to Steps 1 through 4 in the *Clusterhead Computation Protocol*, is used to prevent cheating.

**Defending against intentional timeout for certain advantages** In the *Cluster Recovery Protocol*, the new member will not have a chance to be elected as a clusterhead in the beginning unless a new re-election period occurs (or if the clusterhead leaves the area). This is intentional so as to reduce the chance that change of clusterhead occurs too often. However, the property involves a fairness issue. A node can refuse to acknowledge being elected as a clusterhead in the cluster computation stages but later on dispatches an ADD_REQUEST to join the cluster. In this way, it will be exempted from serving as a clusterhead (a special type of *Denial-of-Service*). A similar attack works in the opposite way. An attacker can refuse to take the responsibility as a citizen (or non-clusterhead member) by repeated timeout until the compromised node is elected as a clusterhead. This gives the attacker the advantage of a clusterhead but not willing to conform to a citizen's liability

when other nodes are clusterheads. To defeat both of these attacks, we add a retreating suspicion counter in the cluster computation protocol which counts how many times an elected node refuses to respond. If it happens more than certain times (three in our experiments), the node is excluded from further clusterhead computation and an exception is reported about the misbehavior of that node.

## *5.3 IDS Agents in a Cluster-Based IDS*

In each cluster, clusterheads independently run IDS agents, while citizens do not. A clusterhead sends a FEATURE_REQUEST message that specifies what features the clusterhead needs to a randomly chosen cluster member. The requested node begins to collect the required information and replies with a FEATURE_REPLY after a feature-sampling period. The clusterhead can then make another request. In practice, the sampling period is often chosen based on the threat model in a real environment.

### 5.3.1 General Detection Methods

In general, similar detection methods can be applied in a cluster-based IDS. For example, we can apply a similar statistical method as what Chapter 4.2 describes, except that features may be collected from multiple nodes. However, the nodes in a cluster need to have the incentive to participate in this cluster-based scheme. We envision that the following scenarios can achieve better performance when cluster-based models are applied.

**Anomaly Detection** In the case of *grouped attacks*, one approach is to instruct the clusterhead to use the detection model similar to the one used in the node-based scheme. Instead of measuring features and analyzing events only for the local node, the "new" detection model, at the same sampling rate, randomly selects a node belonging to the cluster, and computes the same features and analyzes the corresponding event. It is easy to see that over time, when compared with the node-based scheme, each node spends only a fraction of the energy.

**Misuse Detection** A cluster-based framework also has a greater potential in detecting attacks where attack patterns cannot be observed by any single node. Because a clusterhead can collect features from all members in the cluster, it can evaluate a complicated misuse rule using features spanning on multiple nodes. One example can be found in Chapter 5.1.

### 5.3.2 Cluster-Based Specific Methods

We can also use misuse detection techniques with customized detection rules such as the Sinkhole detection discussed in Chapter 5.1. We show more techniques to detect other attacks below.

Dropping: An attacker drops packets that it should forward according to the routing topology. Depending on the purpose of the adversary, packets can be dropped either unconditionally, probabilistically or selectively, based on the destination address or other criteria. Detection of the attack can be done by either *Route Analysis* or *Traffic Analysis*.

Route Analysis: For source routing based protocols such as DSR [39], the detection of this attack can be achieved by neighbor-monitoring. If a node receives a packet with a source route that specifies itself is not the destination, other nodes can detect the anomaly if A does not forward it after a certain timeout. For distance vector based protocols, it cannot be directly applied without protocol modification [93].

Traffic Analysis: A node ($A$) can identify that flow consistency does not hold, i.e., the number of incoming packets minus the number of packets destined to $A$ does not match the number of outgoing packets minus the number of packets originated from $A$. Different from Route analysis, a neighbor $B$ should only analyze flows that were forwarded from $B$ to $A$. As packets received by $A$ from other directions may not be overheard by $B$. This approach applies to both source routing based and distance vector based protocols.

Flooding: An attacker requests services that a certain node has to offer repeatedly in order to prevent the node from going into an idle or power preserving state [82]. *Sleep deprivation* is a typical example of *Denial-of-Service* attacks that can significantly reduce the capability of IDS response. For that purpose, we do not recommend the victim to perform further detections. Instead, if a feature-sampling request arrives, a node suffering this attack can choose to ignore it. As a special passive response, a clusterhead should treat the lack of response as a signal of DoS type of attacks. Even not directly on the victim node, it is hardly a problem for many nodes to detect huge volume of traffic in the victim's neighborhood. Once an early detection of the attack signature is found, it is recommended that other nodes reduce their overhearing that would otherwise significantly suffer from energy loss as well.

Sinkhole: An attacker attempts to absorb data to all destinations by claiming itself has the best route to every destination. The attack is realized by altering route topology. Once successful, collected data packets can be silently discarded, altered or forwarded as usual, but the risk of traffic analysis or information disclosure will significantly increase [40]. Sinkhole detection has been addressed in Chapter 5.1.

Partition: A network (or one of its sub-regions) is separated into two or more partitions so that every route path originating from a node in one partition and destined to a node in a different partition will always include some attackers. The attack may also be realized by altering route topology. Attackers can further drop all traffic that comes across them, which is actually the narrow definition in some literature. We use the current definition for two reasons. First, the narrow definition can be easily implemented by adding an additional *Dropping* attack. Second, partition can still be harmful without dropping if it instead performs some other attacks similar to what *Sinkhole* does. Detection of the attack can be done using either *Route Analysis* or *Traffic Analysis* methods.

Route Analysis: Detection of this attacks benefits from DSR where source routes from multiple nodes can be used for further analysis. For each feature-sampling period, a random member can return a random source route to the clusterhead. The clusterhead in turn compares the source route with previous ones from other members. A partition can be recognized if a small number of nodes appear in each of source route. Note that we currently adopt this approach only to detect a partition implemented with a single node. Otherwise, the response time appears to be very long before an attack is detected. We are investigating on methods that are more efficient.

Traffic Analysis: Traffic can be analyzed similarly as Sinkhole detection does. We should monitor the outbound traffic volume on last hops. The only difference is Sinkhole corrupts traffic to all possible destinations, while this attack only manipulates the traffic to another network partition. Thus, the last hop distribution looks much flatter than *Sinkhole* does.

## *5.4 Summary*

Compared with node-based IDS framework, a cluster-based IDS framework has many advantages. First, it can detect anomalous basic events that the node-based agent can detect because we can always use the same features. However, since the cluster-based solution can collect information from multiple nodes, it has a better chance to detect inter-node modification patterns. Sinkhole detection is such an example. Furthermore, since the same sets of anomalous basic events can often be found in many nodes in the context of group attacks, a cluster-based IDS agent can also avoid running the same detection model unnecessarily on every individual node.

However, the chance of clusterhead compromise introduces more risks with the cluster-based framework. Although we have introduced a fair and secure cluster formation protocol, there is still a non-zero possibility that clusterheads may be compromised. In Chapter

6, we evaluate the performance of both frameworks and compare trade-offs under different system parameters.

# CHAPTER VI

# PERFORMANCE EVALUATION OF INTRUSION DETECTION SYSTEMS

In this chapter, we present experimental results for our proposed detection frameworks. We first discuss the effectiveness of feature selection, followed by the study of detection accuracy using node-based and cluster-based frameworks. We then evaluate a number of factors that affect the stability of the cluster-based framework and compare the performance of node-based and cluster-based frameworks. Finally, we show how topology aware normalization can be used to reduce the space requirement of detection models while remaining fairly accurate.

## *6.1 General Setup*

**Testing Environment** We use MobiEmu [96] as the basic evaluation platform. MobiEmu is an experimental testbed that emulates MANET environment with a local wired network. Mobile topology is emulated through Linux's packet filtering mechanism. Different from many simulation tools, MobiEmu provides a scalable application-level emulation platform, which is critical for us to evaluate the intrusion detection framework efficiently on a reasonably large network. We choose the AODV-UIUC implementation [41], which is designed specifically to work with the MobiEmu platform. We further extend MobiEmu with a toolset of security library and utilities. The toolset, known as S-MobiEmu, is further explained in Chapter 8. We build our IDS system on top of S-MobiEmu, but the toolset is not limited for IDS use and therefore greatly simplifies general development of security protocols for ad hoc networks.

Since MobiEmu is still in development and AODV is currently the only protocol supported by MobiEmu, we use AODV as the main case study, for which we also performed experiments on the ns-2 [22] simulator with very similar results. For comparison, we also studied two other popular ad hoc protocols, DSR and OLSR, but on ns-2 only. We also use EFSA-based study for these protocols, with their EFSA specifications are shown in Appendix B and C. We will show some interesting results that are unique to these protocols in later sections.

**Experimental Parameters** The following parameters are used throughout our experiments. Mobility scenarios are generated using a random way-point model with 50 nodes moving in an area of 1000m by 1000m. The pause time between movements is 10s and the maximum movement speed is 20.0m/s. Randomized TCP and UDP/CBR (Constant Bit Rate) traffic are used but the maximum number of connections is set to 20; and the average traffic rate is 4 packets per second. These parameters define a typical MANET scenario with modest traffic load and mobility, and they are similar to the parameters used in other MANET experiments, such as [69, 58, 59].

One problem is each experiment can only explore one possible scenario. Therefore the detection model may not work for all possible scenarios (for instance, with high mobility or under high traffic load). The topology aware normalization experiments 6.6 partially address this problem.

We test our framework with multiple independent runs. A normal run contains only normal background traffic. An attack run, in addition, contains multiple attack instances which are randomly generated from attacks specified in Tables 2 and 3 or a subset according to certain criteria.

**Misuse Detection** we generate five normal profiles (NORMAL), with 10000 examples each, is used. We also generate set of five basic intrusion profiles (ALL), with 20000 examples each are generated for evaluation purposes. In these profiles, a number of basic

attacks are randomly injected at random locations and with a duration length of 200 examples. Within the total number of examples, 80% are normal. Finally, a set of complicated intrusion profiles (COMP) of 1000 records, which implement some complicated intrusion scenarios are used for evaluation purposes.

**Anomaly Detection** We use the NORMAL profiles for training, and ALL and COMP for testing. We control the false positive rate on the training data to be no more than 1% during the training process, whenever possible, because in practice, a low false positive rate is essential to be useful for human adminstrators to perform useful analysis.

## *6.2 Feature Selection*

In our case study using the AODV protocol [66], the feature set contains 118 statistical features.

Figure 8 shows the skewed false positive rates when more features are added. For illustration purposes, we do not stop immediately when the stop threshold would have held us from continuing the algorithm. Instead, we show the possible consequence when more features are added even after the threshold.

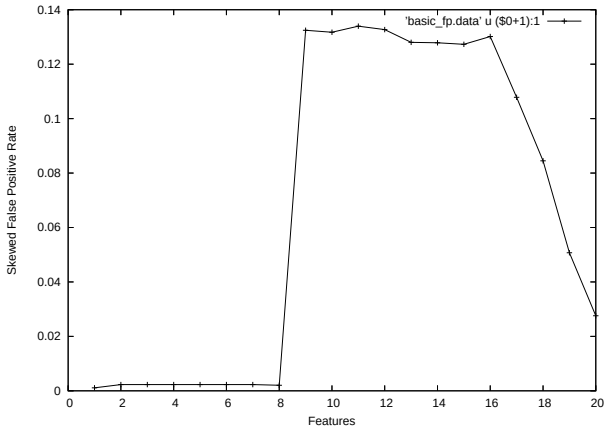
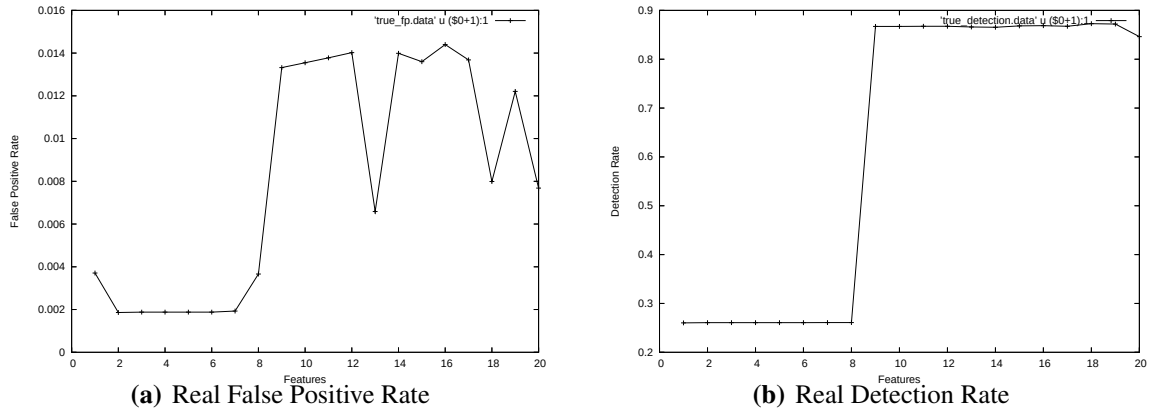



**Figure 8:** Skewed False Positive Rates

The stop threshold would have stopped the algorithm when 9 features are used, since

the skewed false positive rates would start decreasing if more features were added.

Figure 9 shows the true false positive rates and detection rates on the test data with various number of features. Since the test data contains both normal and intrusion data, we show both the false positive rate and the detection rate. For reference, the base rates (i.e., when all features are used) are about false positive rate = 0.013 and true positive rate = 0.78.




**(a)** Real False Positive Rate **(b)** Real Detection Rate

**Figure 9:** Feature Selection Evaluation Results

In fact, we find out that the performance with a subset of features can be better than with the full feature set. In particular, if more than 18 features are added, the performance (both the skewed false positive rates and the accuracy on the test data) becomes to drop again. This is a well known issue in many classifiers because of the potential noise and over-fitting on some "poisoning" features. Therefore, our experiment validates our claim that feature selection can actually improve the overall accuracy as well as the performance gain with reduced number of features.

While we are testing more features than required, we see that the performance no longer increases in general. This shows that our stop threshold is effective enough in practice. However, we do observe that in a few circumstances, the detection rate remains the same, but the (real) false positive rate can still be improved (for example, with 13 and 18 features). Since this is not a constant factor, we believe that it is possibly due to a sampling issue. To

reduce the potential variances in the sampling data, we need to perform more experiments to smooth them out in future.

## *6.3 Node-Based Detection*

### 6.3.1 Detection of AODV Specification Violations

The following attacks are detected in the test data as direct violations of the EFSA, verifying our previous analysis that these attacks match anomalous basic events that can be directly detected by verifying the specification. For complex attacks, a different network size may be used if appropriate. Note that detection rates are 100% and false positive rates are 0% for attacks when the specification-based approach is used, based on the assumption that specfications are 100% accurate. While it is not actually perfect, our specification characterizes most major functionalities and therefore the assumption is mostly reasonable.

Data Drop (R | S | D): detected as *Interruption of Data Packets*.

Route Drop (R | S | D): detected as *Interruption of Routing Messages*.

Add Route (I | N): detected as *Add Route of Routing Table Entries*.

Delete Route: detected as *Delete Route of Routing Table Entries*.

Change Sequence (R | M); Change Hop: detected as *Change Route Cost of Routing Table Entries*.

Active Reply; False Reply: detected as *Route Reply Fabrication*.

Route Invasion; Route Loop: They are detected since they use fabricated routing messages similar to what the **Active Reply** attack does. In particular, *Route Invasion* uses *Route Request* messages, and *Route Loop* uses *Route Reply* messages. With the same set of transitions in **Route Drop**, we can detect them as *Incorrect Transition Violations* in *Route Request* or *Route Reply* delivery transitions.

Partition: This attack can be detected since it uses a fabricated routing message (*Route Reply*) and interrupts data packets. Therefore, monitoring the transitions related to *Route Reply* (as in **Route Drop**), and the transition related to data packet forwarding (T10, as described in **Data Drop**), we can detect this attack with the following violations identified: *Incorrect Transition Violation* in *Route Reply* delivery transitions and *Unexpected Action Violation* in the data forwarding transition.

### 6.3.2 Detection of AODV Statistical Deviations

Some attacks are temporal and statistical in nature and should be detected using the statistical approach. The following are four representative examples of such attacks: **Data Flooding (S | D | R)**; **Route Flooding (S | D | R)**; **Modify sequence (R | M)**; **Rushing (F | Y)**.

**Misuse detection** We first train separately with each training dataset with known intrusion data. The same test dataset is used to evaluate the learned model.

For each type of anomalous basic event described in Chapter 4.2.1, we discuss what features are needed to capture its behavior. All features are defined within a sampling window. We use a sampling window of five seconds in all cases. In addition, features are normalized in a scale of 0 to 50.

Flooding of Data Packets: In order to capture this anomalous event, we need to capture the volume of incoming data packets. In AODV, data packets can be accepted under three different situations: when a valid route is available (which is transition T10), when a route is unavailable and no route request has been sent yet (transition T2) or when a route is unavailable and a route request has been sent to solicit a route for the destination (transition T3). Accordingly, we should monitor frequencies of all these data packet receiving transitions. We define three statistical features, *Data1*, *Data2*, and *Data3*, for each transition (T10, T2 and T3) respectively.

Flooding of Routing Messages: Similarly, we need to monitor the frequencies of transitions where routing messages are received. However, a larger set of transitions need to be observed because we need to take into account of every type of routing messages (which include 15 transitions, T5, T5', T7, T8, T7', T8', T7", T8", T9, T11, TR3, TR4, TR3', TR4', and TR6). In order not to introduce too many features, we use an aggregated feature *Routing* which denotes the frequency of all these transitions. Note that it is not the same as monitoring the rate of incoming routing messages. An incoming routing message may not be processed by any EFSA in a node. We need only to consider messages that are being processed.

Modification of Routing Messages: Currently, we consider only modifications to the sequence number field. We define *Seq* as the highest destination sequence number in routing messages during transitions where they are received (see above for the transitions involved in routing messages).

Rushing of Routing Messages: We monitor two features where some typical routing process may be rushed. *Rushing1* is the frequency of the transition where a route discovery process fails because the number of *Route Requests* sent has exceeded a threshold (RREQ_RETRIES) or certain timeout has elapsed (NET_TRAVERSAL_TIME in transition T6). *Rushing2* is the frequency of the transition where a *Route Request* message was received and it is replied by delivering a *Route Reply* message (transition T11).

To demonstrate, we use RIPPER [19] as the classifier. The output rules are combined into a single rule set. One example of the rule set is shown below.

```
Data_Flooding :- Data1>=29 (1068/43).
Data_Flooding :- Data2>=26 (309/10).
Data_Flooding :- Data3>=47 (1032/17).
Routing_Flooding :- Routing>=20 (1988/103).
```

```
Routing_Flooding :- Routing>=16, Rushing2>=4 (291/17).
Routing_Modification :- Rushing2>=4, Seq>=27 (49/3).
Routing_Modification :- Data1>=13, Seq>=32 (842/380).
Routing_Modification :- Seq>=50 (1231/0).
Routing_Rushing :- Data3<=10, Rushing1>=19 (1173/120).
Routing_Rushing :- Rushing2>=6 (1038/191).
```

For example, the first rule says the anomalous basic event *Flooding of Data Packets* is detected if *Data1*, the normalized frequency of transition T10, is larger than 29. Using this rule, 1068 attack instances are correctly detected, while 43 normal instances are incorrectly identified as attacks.

**Table 10:** AODV: Detection and False Positive Rates with Statistical-based Approach

**(a)** Attack Detection Rates

| Attack | Detection rate |
|---|---|
| Data Flooding (S) | 93±3% |
| Data Flooding (D) | 91±4% |
| Data Flooding (R) | 92±4% |
| Route Flooding (S) | 89±3% |
| Route Flooding (D) | 91±2% |
| Route Flooding (R) | 89±3% |
| Modify sequence (R) | 59±19% |
| Modify sequence (M) | 100±0% |
| Rushing (F) | 91±3% |
| Rushing (Y) | 85±4% |

**(b)** Detection and False Positive Rates of Anomalous Basic Events

| Anomalous Basic Event | Detection Rate | False Positive Rate |
|---|---|---|
| Flooding of Data Packets | 92±3% | 5±1% |
| Flooding of Routing Messages | 91±3% | 9±4% |
| Modification of Routing Messages | 79±10% | 32±8% |
| Rushing of Routing Messages | 88±4% | 14±2% |

The detailed detection results are shown in Table 10. We show the detection rates of tested attacks (in Table 10(a)). We consider a successful detection of an attack record if and only if the corresponding anomalous basic event is correctly identified. We also show the

detection and false positive rates (in Table 10(b)) directly against anomalous basic events. We analyze these results for each type of anomalous basic event below.

Flooding of Data Packets and Routing Messages: We implement flooding as traffic over 20 packets per second. For flooding of data packets, 92% can be detected. They are detected by observing abnormally high volume on at least one of related statistics, *Data1*, *Data2*, or *Data3*. Similar results are also observed for flooding of routing messages.

Modification of Routing Messages: The corresponding detection result is not very satisfactory. It shows high variations in both the detection and false positive rates. In fact, the corresponding detection rule assumes that this anomalous basic event can be predicted when at least some incoming packet has a sequence number larger than certain threshold. It is not a rule that can be generally applied. Randomly generated sequence numbers may only be partially detected as attacks. We further discuss problem in the end of this section. In contrast, for a special type of sequence modification (*Modify Sequence (M)*), the detection rate is perfect. Because we know that it is very rare for the largest sequence number to appear in the sequence number field of routing messages.

Rushing of Routing Messages: Detection performance varies significantly on different rushing attacks, namely, *Rushing (F)* and *Rushing (Y)*. In *Rushing (F)*, the attacker tries to shorten the waiting time for a *Route Reply* message even if a route is not available yet. Because more requests to the same destination may follow if route discovery was prematurely interrupted, the attack results in abnormally high frequency where the route discovery process is terminated (*Rushing1*). In *Rushing (Y)*, the attacker expedites *Route Reply* delivery when a *Route Request* message has been received. It can be captured because the corresponding transition (T11) now occurs

more frequently than a computed threshold (*Rushing2*). Nevertheless, we also observe significant false alarms in detecting these attacks. It results from irregularity of route topology change due to MANET's dynamic nature. Some normal nodes may temporarily suffer a high route request volume that exceeds these thresholds.

**Discussion** Comparing with the taxonomy of anomalous basic events in Table 1, we realize that a few of them cannot be detected effectively yet. First, we cannot detect **Route Message Modification with incoming packets** in which the modification patterns are not known in advance. We identify the problem as it requires knowledge beyond a local node. However, these attacks can usually be detected using other security mechanisms or by other nodes. If the message comes from external sources, it may be successfully prevented by a cryptographic authentication scheme. Otherwise (i.e., it was delivered by the routing agent from another legitimate node), the IDS agent running on that node may have detected the attack. In addition, **Rushing** attacks cannot be detected very effectively, especially when features beyond the routing protocol, such as delays in the MAC layer, are involved. Our system can be improved if we were able to extend our detection architecture across multiple network layers. It is part of our future work.

**Anomaly Detection** The evaluation procedure is similar to misuse detection learning models, except that we only label data as normal or abnormal (instead of the anomalous basic events). However, we should point out that for certain attacks, especially the ones related to routing, it is not necessary to identify every anomalous event (or data point) in order to detect the attack because there may be many anomalous events caused by the attack. Therefore, we can use a post-processing procedure to count the number of detected anomalous events within each sliding time window, and conclude that an attack is present if the count is the majority or above a threshold. Using such a post-processing scheme, we can improve the detection rate and lower the false positive rate. Another observation is that our detection models run at a frequency of the feature sample rate rather than continuously.

They can potentially be the more efficient alternative than cryptography-based prevention scheme.

Our simulation shows that our anomaly detection method, using *Cross-Feature Analysis*, can detect flooding attacks with a detection rate of 95% and a false positive rate less than 1%. Its performance on packet modification attacks is also close to what the misuse detection rules can achieve in general. Considering that the anomaly detection model is trained with pure normal data, the results are very promising.

### 6.3.3 Detection of OLSR Specification Violations

OLSR is a proactive routing protocol, and therefore it has several unique properties. We show the detection results for OLSR specific attacks in the following sections.

Detection of a typical message fabrication attack, *Hello message insertion*, has been illustrated by Orset et al. [62]. In this attack, an adversary attempts to advertise a non-existent symmetric link to other nodes and by doing that, can perturb the routing calculation of these nodes. According to the EFSA in Appendix C, if the receiver of the Hello message is not its neighbor, it will stay in State **I**, but a transition that can accept a Hello message with link code SYM (denoting a symmetric link) can only occur from State **A**, **S** or **M**. Therefore, it can be detected as *Fabrication of Routing Messages*.

Another OLSR specific attack, known as *MPR attack* [73], prevents a MPR from forwarding messages. Because the OLSR specifications [18] state that a node will never retransmit a message if the sender is not in its MPR selector list, this optimization is used for performance reasons but it can be easily abused. For example in Figure 10, B is an MPR of A and C is an MPR of B. A misbehaving node X can overhear messages sent by node A. Since X is not the MPR of any node, it should do nothing. Instead, X maliciously retransmits the message to C. C discards this message because X is not one of its MPR selectors. However, this also implies that C will not forward the same message from B as well. Therefore, the message is lost.

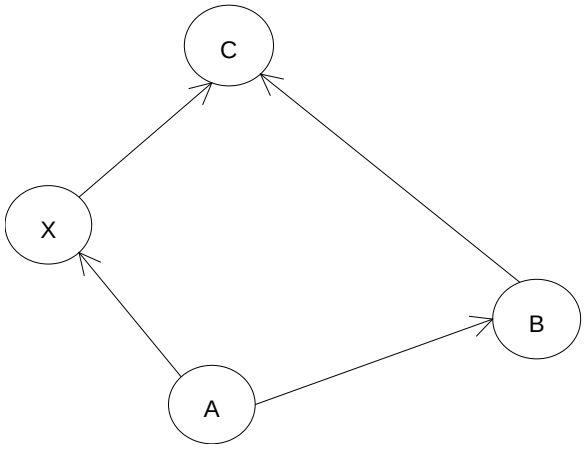



**Figure 10:** MPR Attack Example

*MPR attack* can also be detected in a similar way to *Hello message insertion*. Note that the only transition that perform message forwarding is transition T20 which can only be satisfied when the forwarding node is an MPR of the receiver. Because the input conditions are not fully satisfied while the output action is taken anyway, this is one typical example of *Incorrect Transition Violation*.

In addition, the following attacks can also be detected by detecting OLSR specification violations:

Data Drop (R | S | D): detected as *Interruption of Data Packets*.

Route Drop (R | S | D): detected as *Interruption of Routing Messages*.

Add Link (I | N): detected as *Add Route of Routing Table Entries*.

Delete Link: detected as *Delete Route of Routing Table Entries*.

### 6.3.4 Detection of OLSR Statistical Deviations

One typical OLSR attack that is statistical in nature is “message bombing" [73]. It is essentially equivalent to data or route message flooding in the previous AODV analysis. Nevertheless, it should be noted that for better performance, we need to perform some

optimization. Since MPRs are the only nodes that forward data packets, we add one additional boolean feature **MPR** to the standard feature list used by node-based IDS agents. **MPR** is 1 if and only if the current node has been chosen as one MPR by some node. Since MPR and non-MPR nodes may observe different types and volumes of traffic, we expect the additional feature can help improving detection performance. Table 11 shows that this is actually the case. For simplicity, we use labeled training datasets in this table.

**Table 11:** OLSR: Detection and False Positive Rates of Anomalous Basic Events

| Anomalous Basic Event | Without MPR | | With MPR | |
|---|---|---|---|---|
| | Detection Rate | False Positive Rate | Detection Rate | False Positive Rate |
| Flooding of Data Packets | 88±4% | 2±1% | 93±4% | 4±2% |
| Flooding of Routing Messages | 85±4% | 3±2% | 91±3% | 5±2% |

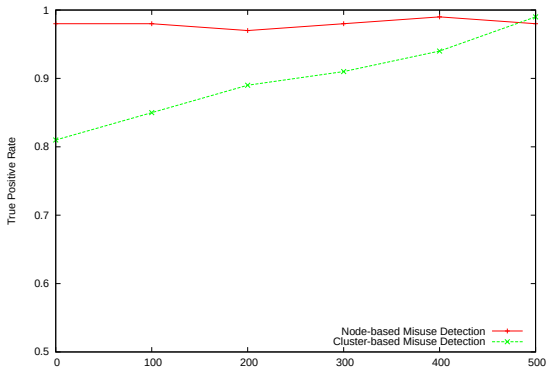
## *6.4 Cluster-Based Detection: Detection Accuracy Study*

Chapter 5.1 argued that distributed attacks could evade a node-based IDS, but not so easily with a cluster-based IDS. We used the Sinkhole detection as an example, because it is very difficult for a node-based IDS to detect Sinkholes, while the cluster-based approach can be applied in neighboring clusterheads to detect these attacks.

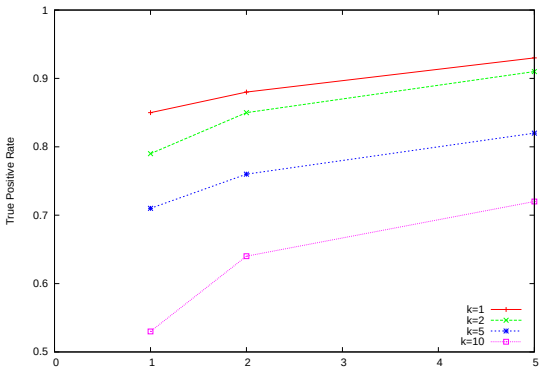
We show the detection accuracy of both solutions in Figure 11. For the cluster-based solution, we use $m$=5 clusterheads per cluster and we assume the maximum number of system level compromised nodes $k$ is 2. In all cases, we control the false positive rates to be no more than 2%. We can see that the cluster-based solution does have fairly good detection accuracy where the average detection rate is 91% at a modest mobility level (pause time = 300s).

Furthermore, the cluster-based solution can mitigate the problem of system level adversaries where the node-based solution can be easily defeated. We choose a modest mobility level (pause time = 300s) and experiment with various numbers of collaborative attackers.

We assume that if a clusterhead is compromised, its IDS functionality is completely disabled. By varying $k$ from 1 to 10, we see the detection rate decreases sharply when $m$ is fixed. If $k = 10$, Figure 11(b) shows the true positive rate can be as low as 53% with a single clusterhead. With the help of multiple clusters, the detection accuracy quickly recovers to a relatively high level. We can see that if $m = 5$, 70% of the attacks can be detected even under very high compromise level.




**(a)** Sinkhole Detection, m=5, k=2




**(b)** Sinkhole Detection with Multiple Attackers, Pause Time=300s

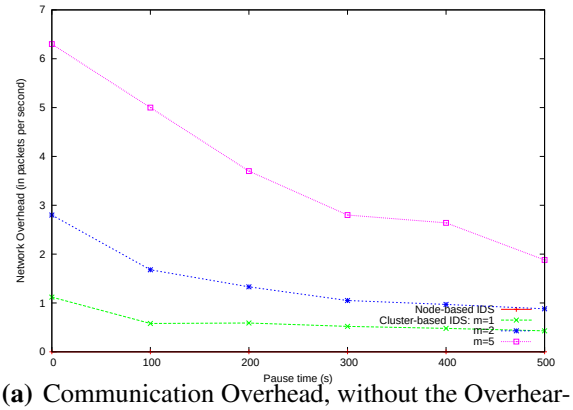
**Figure 11:** Cluster-Based Detection Accuracy Study

## *6.5 Cluster-Based Detection: Effectiveness Study*

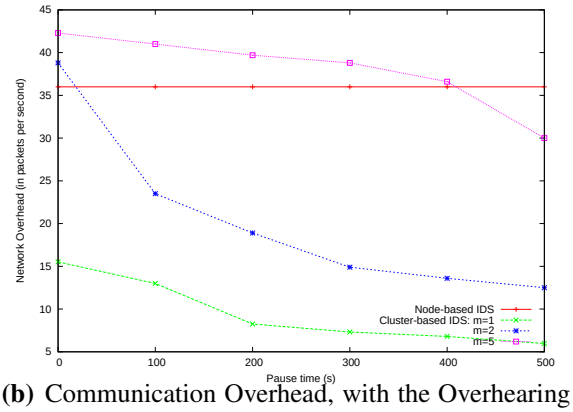
As we have shown, many attacks can be detected effectively with a node-based IDS. Chapter 5.1 discussed a number of limitations of this approach. We verify two statements made there.

First, we argued that a cluster-based IDS introduces less energy consumption while achieving the same detection accuracy. The impact of communication overhead is widely agreed to be the largest source of energy consumption in a MANET environment. At first glance, it seems that the node-based IDS does not deliver or receive any messages at all and it should not cause any communication overhead. We note that it is not the case if we consider that overhearing neighboring traffic also consumes energy. As we have explained, the use of overhearing capability is essential to defeat system level adversaries. We compare

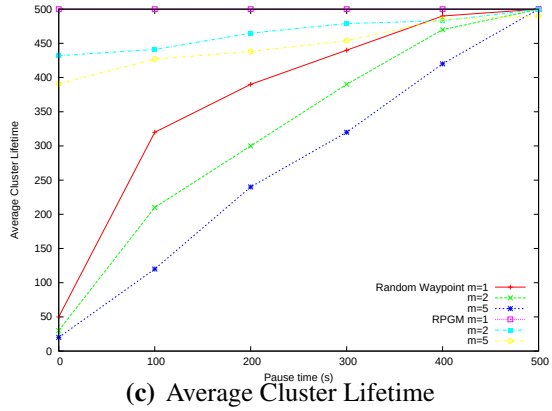
the communication overhead in Figure 12(a) and 12(b), with the only difference between the two figures lies in whether we consider the overhearing effect or not. We measure the overhead in packets per second. We note that Figure 12(a) shows that a node-based solution has zero energy consumption without considering the overhearing effect! It appears inappropriate to ignore this effect. Figure 12(b) shows that the cluster-based scheme, provided with appropriate parameters, is a more energy efficient solution.



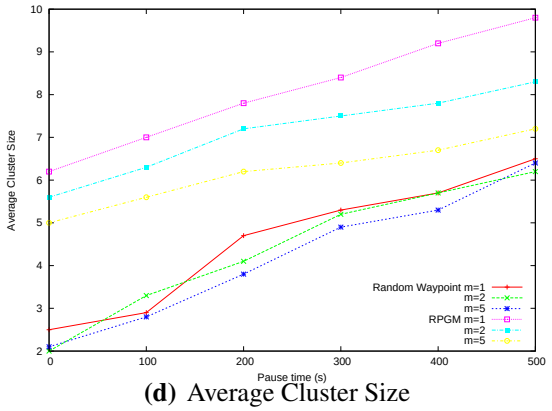

**(a)** Communication Overhead, without the Overhearing Effect (note that the node-based IDS overhead is always zero)




**(b)** Communication Overhead, with the Overhearing Effect




**(c)** Average Cluster Lifetime




**(d)** Average Cluster Size

**Figure 12:** Cluster-Based Detection Effectiveness Study

Another issue relates to grouped attacks such as data flooding. They can be detected effectively by both the node-based and cluster-based solutions. Therefore, an intuitive way of determining which solution is better is to study the cluster stability. If the formed clusters are stable, the cluster-based solution is more efficient because few IDS agents and sensors

are used. On the other hand, if clusters change too frequently or the average cluster size is too small, the communication overhead during the cluster formation stage dominates. We use two measures, the *average cluster size* and the *average cluster lifetime*, to measure cluster stability.

We evaluate several factors that may affect cluster stability. We first consider the impact due to node mobility. By varying the pause time, we test against different mobility levels in Figure 12(c) and 12(d) when different numbers of clusterheads ($m$) are used. We observe that increasing $m$ reduces a cluster's lifetime. Yet the reduction is only observable when the mobility level is very high (i.e., when the pause time is near zero). In that case, a low $m$ is preferred. However, in the worst case we have experimented, where the pause time is zero and $m$ is 5, we still have an average cluster size of 2 and average cluster lifetime of 20 seconds, in which four feature-sampling periods can be finished. Note that $m$ only specifies the maximum number of clusterheads and a smaller value may be used if there are less than $m$ members. We observe that the cluster-based solution is more stable and thus probably more feasible in a relatively static ad hoc network.

Another factor that matters is the mobility model that we use. By default, we assume a *Random Waypoint* mobility model. In many application scenarios, it may not be the best one. In particular, group models can be more suitable when mobile nodes move together in groups. Numerous studies have shown that the choice of a mobility model can have significant influence on the performance of a MANET network protocol [13]. In our study, we experiment with a generic group model, the *Reference Point Group Mobility* model. In such a model, group movements are based on a logical center for the group. Random motion vectors are generated independently for each group center and each individual node from its group center. The overall movement vector for each node is then the summation of both. In our experiments, we define ten groups with five nodes per group. We compare its performance with the *Random Waypoint* model in Figure 12(c) and 12(d). We learn that the effective cluster lifetime by applying the group model is in fact only bounded by the

cluster re-election period (500 seconds). It is not hard to understand, since nodes within a group are much closer to each other, they can always form a cluster with an adequate size among themselves. Nodes outside a group help build a larger cluster only when the low mobility allows.

## *6.6 Topology Aware Normalization*

We again use the AODV protocol [66] as a case study for topology aware normalization experiments, with the same 118 feature set constructed from EFSA.

### 6.6.1 Unconditional Grouping

Recall that we define a scenario as a group of topology parameters. We show the detection rate and false positive rates using the following four methods: separate models, one for each scenario (model 1); a global "super" model with no normalization (model 2); a global "super model" where each example is normalized based on the Gaussian parameters (mean and standard deviation) computed from each individual scenario through "pre-learning" (model 3); and finally, a global "super" model with feature normalization and parameter grouping (with bucket parameter $n = 5$) (model 4). The results for AODV is shown in Table 12. Note that to simulate the whole feature space, we need to generate *many* scenarios. We currently use 500.

**Table 12:** AODV Detection and False Positive Fates: Comparison Study

| Model | Detection Rate | False Positive Rate |
|---|---|---|
| 1 | 81.2 | 1.0% |
| 2 | 68.3 | 34.2% |
| 3 | 73.2 | 9.83% |
| 4 | 73.8 | 12.2% |

We can see that a all-in-one model without normalization behaves the worst. The false positive rate is extremely high, partially reflecting the analysis we presented before. A normalization step greatly improves the false positive rates. The detection rates, however,

are still worse compared with separate models. This is due to the fact abnormal records are normalized using the base statistics collected from normal dataset, and therefore attack features tend to become more "normal". We also observe the bucketing scheme has similar performance compared with the model where pre-learning is conducted on per scenario basis, with only slight degradation in the performance.

It should be noted although the separate model seems to work best, it is extremely costly, becausewe essentially recompute the whole detection model on the fly whenever a new scenario is added. It is therefore infeasible to obtain real-time performance on a rapid changing environment. The second slowest model is model 3, because "pre-learning" should still be conducted on a per scenario basis. However, since the process involves only statistical estimation, a sampling with a small subset can be relatively safe and therefore it is still much faster than the separate model. Model 4 is faster, because of parameter bucketing, and the performance degradation is still acceptable compared with model 3.

### 6.6.2 Subgrouping

We can view the separate model as one end of a scale spectrum where examples from different scenarios are always separated into different training sets. On the other hand, the "super" models go to the other end where examples from all scenarios (after normalization) are clustered and used to build a single model. Hence, a further improvement could be made if we allowed the existence of multiple models, where each model clusters a subset of scenarios. These subsets are mutually exclusive but add up to the whole feature space.

Finding the optimal splitting strategy is NP-hard. But it is a good heuristic to split along topology parameters. We again use a bucket size $n$. For each topology parameter, we divide the value space of that parameter evenly into $n$ buckets and thus split the whole feature space into $n$ buckets. We then train all scenarios in each bucket using either model 3 or 4 above, and then compare the average performance over $n$ buckets. The first round is evaluated by testing all possible topology parameters and choose the best one. This process

can be continued as needed, and it is in fact, a standard feature selection problem.

In reality, the computation is so expensive that we cannot go very deep. In fact, we are only able to evaluate the first round. By using model 3 (pre-learning only), we obtain the best result by splitting along parameter *nn* (the number of nodes) in Table 13.

**Table 13:** AODV Detection and False Positive Fates: Comparison Study by Subgrouping

| Feature | Detection Rate | False Positive Rate |
|---|---|---|
| nn | 76.7% | 5.31% |
| radius | 77.4% | 7.29% |
| speed | 73.5% | 11.7% |
| pause | 74.8% | 8.66% |
| mc | 74.9% | 9.01% |

## *6.7 Summary*

We showed that effective use of feature selection and topology aware normalization can improve the overall detection accuracy or space requirement. We also illustrated the detection results using different frameworks: node-based and cluster-based.

It is very difficult to provide a guideline on how to choose between node-based IDS and cluster-based IDS. With experimental results, we show that cluster-based IDS is more energy efficient and it can mitigate the problem of system level adversaries. However, the additional benefit is highly related with the cluster size. It is probably more suitable to apply cluster-based IDS in a relatively static network or with a generic group model where cluster sizes are more stable. Overall, the framework choice depends on the actual application scenarios, potential security threats, and parameter settings.

# CHAPTER VII

# HOTSPOT-BASED TRACEBACK

## *7.1 Introduction*

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 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. [80], 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 chapter, 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.

## *7.2 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 work, 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.

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 in Chapter 5) can detect intrusions on behalf of the victim. Therefore, the traceback can be triggered even if the victim itself is compromised. In this work, we refer to the IDS agent that triggers a traceback session as the **investigator**.

### 7.2.1 Secure Communication

In the protocols discussed in this work, 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 [14]. Alternatively, we can use light-weighted symmetric cryptography based on one-way hash chains and time synchronization, such as TESLA [31] or $\mu$TESLA [55]. 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 [33], 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 [32], 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 [59] 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, our statistics-based IDS, may 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.

### 7.2.2 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 work, 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 [59], physical security [82, 15] 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.

### 7.2.3 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

[1] We will define the digest function in Chapter 7.3.1.

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 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.

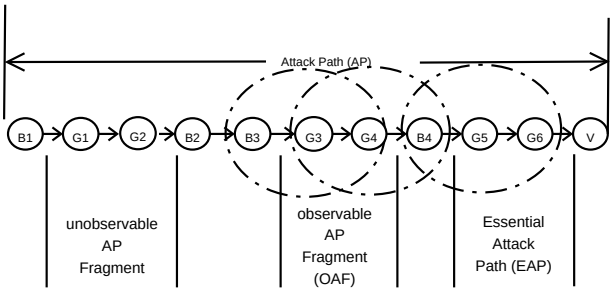



**Figure 13:** Attack Path

Figure 13 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 13. 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 work, we focus on the problem of how to detect hotspots.

## *7.3 Existing 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 [75, 81, 92], 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.

### 7.3.1 The Source Path Isolation Engine (SPIE)

Let us first review some of the basic elements in the SPIE [80] approach.

**Infrastructure** In this framework, every SPIE-enabled router runs a *Data Generation Agent* (DGA) where a Bloom filter [7] 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 Chapter 7.4.2. 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 [80]. 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*.

## *7.4 Basic Mechanisms*

### 7.4.1 Overview

First, let us assume that the investigator for a traceback session itself is well-behaving. This requirement will be relaxed later in Chapter 7.4.6.

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 (Chapter 7.4.3) and the neighbor list, i.e., the nodes within its own transmission range (Chapter 7.4.4). 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 Chapter 7.4.5 and Chapter 7.4.6, respectively.

### 7.4.2 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* [7] 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 pre-image 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 [6], 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 Algorithm 9.

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

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

[2]The *Counting Bloom Filter* presented in [23] also uses multiple-bit entries, but it serves a different purpose and its operations are different.

```
insert(element, t) begin
    foreach i in [1, k] do
        if TBF[h_i(element)] = 2^c − 1 then
            TBF[h_i(element)] ⇐ t;
        TBF[h_i(element)] ⇐ 2^c − 2;
end
test(element) begin
    if ∃ i, s.t. TBF[h_i(element)] = 2^c − 1 then
        return 2^c − 1;
    return min_{i∈[1,k]}(TBF[h_i(element)]);
end
```

**Algorithm 9**: Tagged Bloom Filter Operations

Similar analysis can show that the $\epsilon$, the false positive rate, is equal to $\mu$. We summarize $\epsilon$ and $\mu$ in Table 14.

**Table 14:** Tagged Bloom Filter: Probability Matrix

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

We observe that $\epsilon$ (or $\mu$) 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.

### 7.4.3 Relative TTL

**Definition 8.** 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. □

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).

### 7.4.4 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 $\rho$, defined below, should be sufficiently close to 1.

**Definition 9.** $\rho$ 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`.

### 7.4.5 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 10 returns an attack graph $AG$ based on the summaries, where $VIC$ is the address of the victim.

The algorithm runs in $O(|V|^2)$ in the worst case. First, lines 4-5 (case 1) add a route edge $x \rightarrow y$ if $x$ and $y$ have consecutive and valid RTTL tags. Second, lines 6-11 (case 2) add two route edges $y \rightarrow x \rightarrow z$ 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 9), we prevent invalid path traversals, such as $y \rightarrow x \rightarrow w$ in the previous example. Pseudo nodes with different tag values are considered different nodes, therefore may be traversed within a single route path.

```
1  Attack_Graph({R_i, RTTL_{R_i}(P), NL(R_i)}, VIC, c) begin
2      V ⇐ {R_1, R_2, ...};
3      E ⇐ {};
4      foreach x, y in V s.t. (RTTL_y(P) ≠ 2^c − 2 ∧RTTL_x(P) = RTTL_y(P) + 1 (mod 2^c − 2)
       ∧x ≠ VIC) do
5          E ⇐ E ∪ {x→y};
6      foreach x in V s.t. (RTTL_x(P) = 2^c − 2) do
7          foreach y, z in V s.t. (x ∈ NL(y) ∧ x ∈ NL(z) ∧RTTL_z(P) ≠ 2^c − 2
           ∧RTTL_y(P) = RTTL_z(P) + 2 (mod 2^c − 2) ∧y ≠ VIC) do
8              val = RTTL_z(P) + 1 (mod 2^c − 2);
9              Generate a pseudo node x_val;
10             foreach y_i, z_i in V s.t. (RTTL_{y_i}(P) = RTTL_y(P) ∧RTTL_{z_i}(P) = RTTL_z(P)) do
11                 E ⇐ E ∪ {y_i→x_val, x_val→z_i};
12     return AG ≡ ⟨V, E⟩;
13 end
```

**Algorithm 10**: Hotspot Attack Graph Construction

Figure 14 illustrates a possible output from Algorithm 10 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 $y \in NB(x)$. $I \rightarrow J$ and $D \rightarrow E \rightarrow F$ 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.**

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

*where $\mu$ is the invalid tag rate defined in Chapter 7.4.2 and $\lambda$ is the number of well-behaving sibling routers of E. Node z is a* **sibling router** *of edge $E = x \rightarrow y$ if either $z \rightarrow x$ or $y \rightarrow z$ 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: $R_0 \rightarrow R_1 \ldots \rightarrow R_L$ where $R_0$ is the

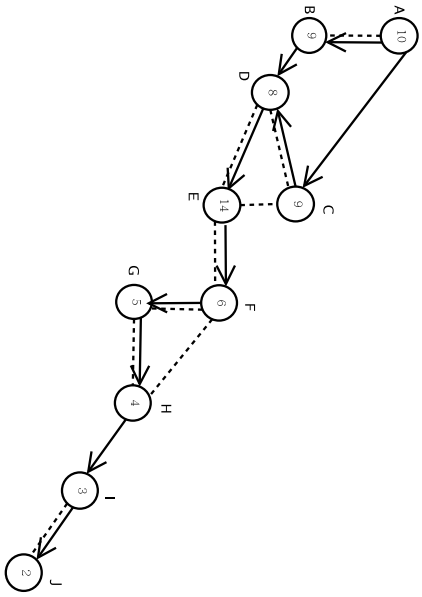



**Figure 14:** Attack Graph Example

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

$$\begin{aligned} Pr(E_i \in AG) \geq\ & Pr(V(R_i) \wedge V(R_{i+1})) \\ & + \rho^2 Pr(V(R_i) \wedge \neg V(R_{i+1}) \wedge V(R_{i+2})) \\ & + \rho^2 Pr(V(R_{i-1}) \wedge \neg V(R_i) \wedge V(R_2)) \end{aligned} \tag{7.4.1}$$

where the first term comes from case 1 (Algorithm 10), while the second and third come from case 2. Also note that Definition 9 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 (7.4.1) 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

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

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 $\rho = 0.85$.

### 7.4.6 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 11 searches for all possible hotspot(s) in $AG$ based on this result where $IV$ is the address of the investigator and $\tau$ 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 $\tau$ nodes of a $V$-path as *its head with size* $\tau$.

```
1  Hotspot({RTTL_{R_i}(P)}, AG = ⟨V, E⟩, VIC, IV, c, τ) begin
2      S ⇐ {IV};
3      T ⇐ {};
4      Find all V-paths using a reverse depth-first search from VIC;
5      foreach V-path VP ≡ R_1 → R_2 → ··· → VIC do
6          T ⇐ T∪{all nodes in VP};
7          S ⇐ S ∪ {R_1, R_2, ... R_τ};
8      foreach x in V − T s.t. (RTTL_x(P) ≠ 2^c − 2 ∧(¬∃z s.t. x→z ∈ E ∨¬∃y s.t. y→x ∈ E)) do
9          S ⇐ S ∪ {x};
10     return S;
11 end
```

**Algorithm 11**: Hotspot Detection Algorithm

Lines 4 in Algorithm 11 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 5-7 return the head of every $V$-path with size $\tau$ 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 8-10 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 2) 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 15 demonstrates an example attack graph where $Q$ is the victim. Algorithm 11 returns $\{A, O\}$. They correspond to $V$-paths $A \rightarrow \cdots \rightarrow N \rightarrow O \rightarrow P \rightarrow Q$ and $O \rightarrow B \rightarrow \cdots \rightarrow N \rightarrow A \rightarrow P \rightarrow Q$ (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.

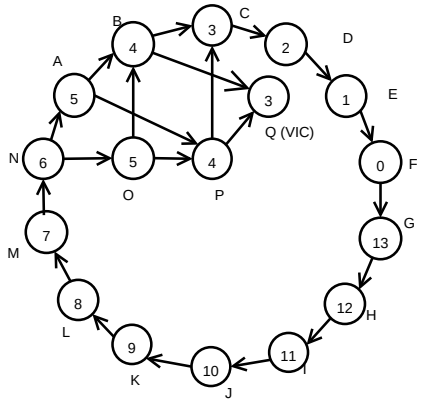



**Figure 15:** Hotspot Detection Example

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 11 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 $VP$.

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 $u \rightarrow w \in AG$. 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 $\epsilon$. Therefore, the probability that all $\tau'$ nodes in the head of every $V$-path are false positives is at most $\epsilon^{\tau'}$. □

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.

## *7.5 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 these buffers are always available within one cycle of `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 10) and the *Hotspot Detection Algorithm* (Algorithm 11) 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* attack 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,

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

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 [59] 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 $\alpha$, prior to digest table lookup. If it volunteers, broadcast message `VLT(vic, seq, volunteer)`.

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 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 16).

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 $\beta$. 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 $V$.

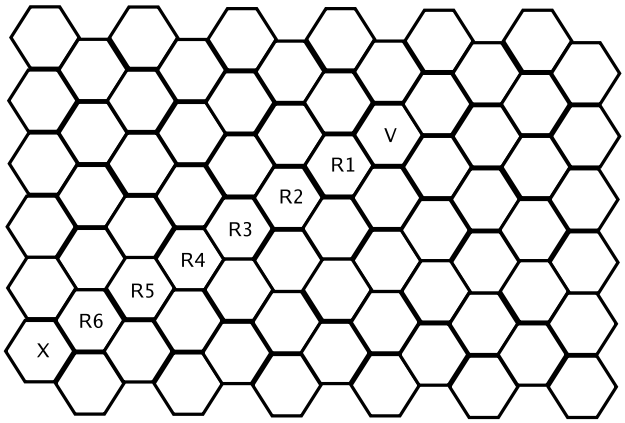



**Figure 16:** An Ad Hoc Network with Uniform Distribution ($d = 6$)

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

An attack packet is not dropped when all intermediate routers are either compromised

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

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:

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

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

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.

## *7.6 Simulation Results*

### 7.6.1 Platform Setup

For most parameters, we use the same setup as Chapter 6.1. A few other parameters are highlighted below.

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 $\rho \geq 0.85$ under different levels of mobility.

### 7.6.2 TBF Storage Requirement

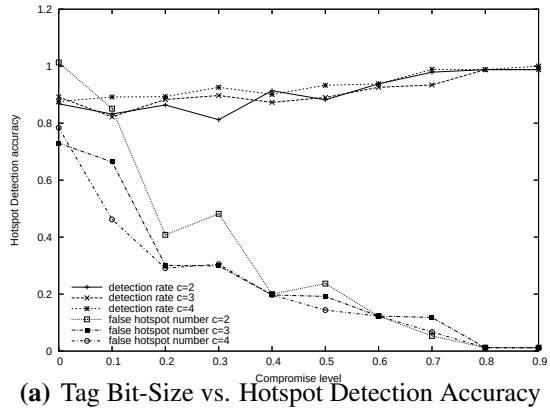
It is important to estimate the storage requirement because a TBF must be fully stored in memory to support real-time membership queries. For simplicity at this moment, assume a

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 $\epsilon$ 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.

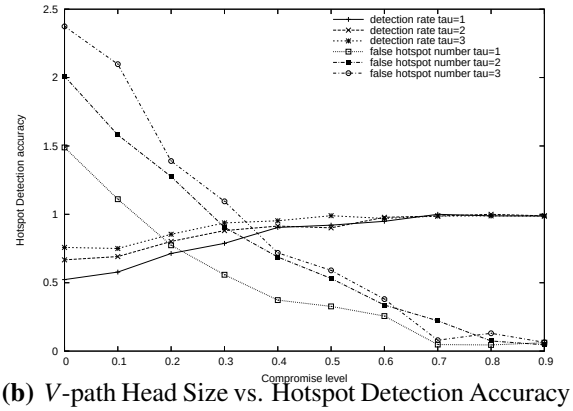
In our simulations, we choose HELLO_INTERVAL to be 5 seconds so that a total of $60/5 + 1 = 13$ buffers are used. A careful reader would find out that the worst-case false positive rate with the same $k$ will be larger when multiple (and smaller) buffers are used. By using $k = 11$, we obtain the same 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.

### 7.6.3 Hotspot Detection

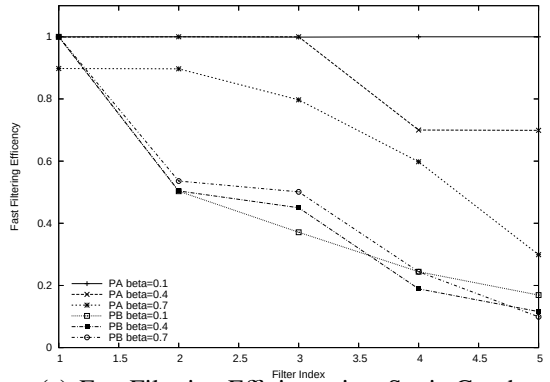
In order to evaluate the effectiveness of the *Hotspot Detection Algorithm*, we use a random compromise model, where every node may be compromised with an equal probability $\beta$. We consider the following two attacks:



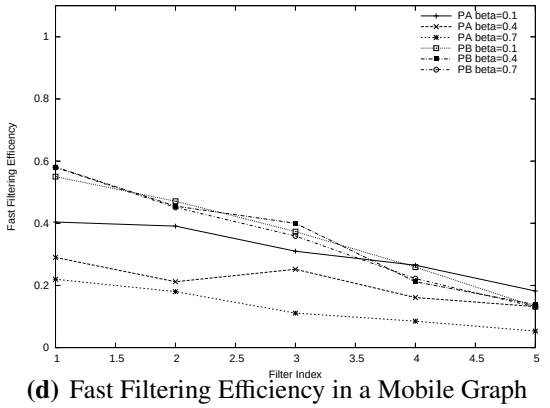

**(a)** Tag Bit-Size vs. Hotspot Detection Accuracy




**(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 17:** Hotspot-Based Protocol Performance

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 $\beta$ 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 17(b) shows both measures when the compromise level $\beta$ 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.

To show similar curves for different $\tau$ 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 17(a) shows both measures where $\tau$ varies from 1 to 3 ($c$ is fixed to 4). We can see that the detection rate benefits from a larger $\tau$, but the false hotspot number suffers. In general, $\tau$ 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.

### 7.6.4 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 $\beta$, and observe the attack dropping rate $PA$ and normal dropping rate $PB$ when various $FI$ values are used in Figure 17(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 $\gamma$. 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 17(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.

## *7.7 Summary*

In this work, 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.

# CHAPTER VIII

# S-MOBIEMU: SECURITY PROTOCOL EVALUATION PLATFORM

We implemented our intrusion detection architectures and algorithms, and evaluated our research prototypes using both simulation and emulation platforms.

Our first platform is ns-2 simulator [22]. Through our simulation study, we have already gained extensive and valuable experience in using simulation software to implement and evaluate our algorithms. We have also migrated our algorithms from ns-2 based simulation to a more realistic platform. We use the MobiEmu software developed in HRL by Zhang and Li [96], which provides a mobile ad hoc emulation on top of Local Area Networks (LAN).

The reason why we apply this migration is multi-folded. The software uses packet filtering based on source address to emulate dynamic topology. We believe MobiEmu can be used as a base platform to build security solutions for MANET. It allows us to run a wide range of MANET scenarios without the need to physically relocate participating nodes, therefore we can reliably repeat the experiments for a large set of test cases and on a practically large scale. In addition, we can easily evaluate real applications and obtain real performance measurement data. Our experience of using MobiEmu in secure MANET research has further validated these benefits.

Developing and evaluating secure mobile ad hoc networks in real systems is a complex process that involves careful design of attack test cases and security countermeasures, as well as meaningful performance measurements to evaluate both the impact of attacks and the performance of security solutions. It is desirable to have a development and testing environment that can automate this process. Although there are several such tools available

for wired networks (such as LARIAT [53, 54]), little work has been done in the wireless domains, and to the best of our knowledge, no such secure MANET testing systems exist in the open literature.

For this purpose, we are developing a software framework, called *S-MobiEmu*, in the secure MANET routing domain based on MobiEmu. In addition to the basic emulation functionalities provided by MobiEmu, S-MobiEmu adds an attack emulation layer with necessary API for easy development and execution of attack test cases. The test case repository is implemented as an attack library, which extends from a core foundation library consisting of a full set of basic attacks, defined in Table 2. The repository is extensible as compound attacks can be constructed using existing attacks as building blocks. Initially, we have included several such well-known compound attack scenarios. Finally, a set of measurement tools are also provided in a performance measurement toolkit.

Our successful experience confirms that the platform can greatly facilitate the development of security solutions on MANET.

The history of security research and practice has taught us that security is an on-going process and any secure system should undergo rigid test and re-test with carefully designed attack test cases. Securing Mobile Ad-hoc Networks (MANET) should also follow this process and it is extremely important to evaluate secure MANET software in real systems and under real attacks.

Like any security system, a thorough evaluation of secure MANET software requires a cycle of four steps (Figure 18). First, we must understand application objectives because the ultimate test of success for a secure MANET is how well the MANET application achieves its designed mission goal in spite of threats and attacks. This *application understanding* will help us design experiments, including the choice of test cases and evaluation models. In *test case development*, we need to come up with a set of carefully designed attack scenarios. Much like other testing in software engineering disciplines, this should come after an extensive analysis of the potential threats to MANET objectives and the set

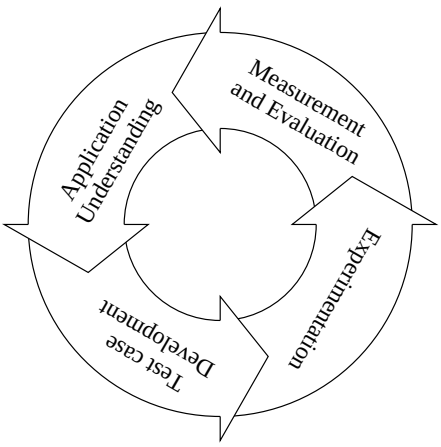



**Figure 18:** The Circle of Securing MANET

of test cases should cover these threats extensively. Systematic approaches like our *attack taxonomy* study in Chapter 2 can be used here in secure MANET test case development.

Next, secure MANET should be evaluated through *experimentation*. Unlike simulation, actual experiments can allow both the actual application and security codes to run in the same condition as in actual deployment. When the attack test cases are injected into the experiment to create real intrusions, the behavior of the secure MANET system can be observed and studied in face of these attacks. Furthermore, meaningful *measurement and evaluation* can be conducted to gain qualitative and quantitative assessments.

Our past experience has also taught us that this process is non-trivial and time-consuming, and it is desirable to have software tools and environments to automate and facilitate some of the tasks. Although there are some such tools available for wired networks (such as LARIAT [53, 54]), little work has been done in the wireless domains and to the best of our knowledge no such secure MANET testing systems exist in the open literature.

The goal of this research is to develop such environment as an experimentation platform for evaluating secure MANET. Given the potentially large amount of test cases and MANET scenarios, this platform should support reproducible experiments and posses the ability to inject attacks (test cases) automatically during an experiment. Further, this platform should provide easy programming support for test case development, and a way to

organize attack test cases in an extensible repository.

- The ability to run reproducible experiments repeatedly.
- The ability to inject artificial attacks (i.e., test cases) during experiments.
- Programming support for test case development.
- Extensible repository for organizing test cases.
- Performance measurement and evaluation tools.

We have developed a software system that meets the above goal. In the rest of this chapter, we will describe the software architecture and explain each major components in details. Especially, we will focus on the methodology and practice of attack emulation in Chapters 8.3 and 8.4.

## *8.1 Architecture*

We have developed such a software platform to facilitate the security development and evaluation that meets the above goal. Architecture-wise it includes the following components:

- *A network emulator to provide a high-fidelity communication environment for repeatable and scalable mobile ad-hoc network experiments.* This will allow us to test real security code in real applications and real systems. The emulation of underlying communication environment will allow us to test security in a wide range of different scenarios and mobility patterns.
- *An attack emulation system that is capable of injecting attack test cases during the experiments.* It installs hooks in certain MANET components and provides a programming abstraction (an API) so that researchers can write attack logics in a way independent from the actual MANET implementations. This is particularly useful

because there can be a large number of test cases and it is impractical to modify a huge number of MANET components to implement each test case.

- *An extensible repository for test cases.* The attack library should have the structure to organize all the test cases and make it easily extensible to accommodate future attacks. Based on the results of our attack taxonomy study, all attacks in MANET can be composed from a set of *basic attacks* or other compound attacks. We should therefore provide an object-oriented hierarchy in the repository to organize the test cases, to assist attack composition into more complex attacks, and to make it easy to add new atomic or compound attacks.

- *A collection of instrumental, measurement, and data analysis tools* to measure the effectiveness and performance of the security solution in the experiments. This includes tools to log traffic and security-related events, tools to observe the state of the network, and finally, tools to assess the effectiveness of the attacks and the state of the applications.

### 8.1.1 Rationale for the Emulation Approach

The evaluation of secure MANET must be under a realistic MANET environment. Although simulation tools like ns-2 [22] and QualNet is widely used for other MANET related experiments, they are not very suitable for this purpose. First, they do not have real applications and thus attacks on application level cannot be easily ported and evaluated. Second, it is impractical to obtain real and meaningful measurement data in a simulation platform. And most importantly, the security of a real system should only be evaluated with the real system and not on a simulated one.

The experimental environment should also be reproducible because this is important in exploring design space and evaluating alternatives. A full-blown test with real wireless hardware may not be repeatable because it is difficult to reproduce the extra wireless

communication environment and it can be too costly to try a wide range of mobility scenarios [96]. Comparatively, the emulation approach has the advantage of both. We therefore believe that emulation is the right approach for experiment with real applications and real systems and yet be faithful to the actual communication environment (MANET).

### 8.1.2 S-MobiEmu

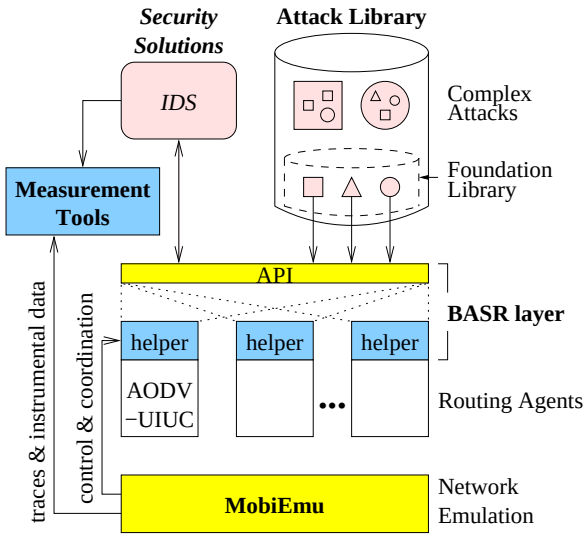



**Figure 19:** S-MobiEmu Software Architecture

We have implemented this platform in a software system called S-MobiEmu, using ad-hoc routing as an example application domain and intrusion detection as a security solution case study. Figure 19 illustrates the software components and their relationship with respect to the security solution being studied.

To support reproducible experiments, we build upon a publicly available wireless network emulator called MobiEmu [96]. We add an attack emulation layer, called *Basic Ad-hoc Security Routines* (BASR), as a common abstraction layer for attack injection and for test case development. The test case repository is implemented as an attack library, which extends from a core foundation library consisting of a full set of basic attacks. The repository is extensible as complex attacks can be constructed using existing attacks as

building blocks. Initially, we have included several such well-known complex attack scenarios. Finally, a set of measurement tools are also provided in a performance measurement toolkit. Since functionwise this can be considered as an extension to MobiEmu, we call it S-MobiEmu, with "S" meaning security.

## *8.2 Emulating MANET*

The network emulation system in S-MobiEmu is based on MobiEmu [96] – a software tool for testing "live" MANET systems in a laboratory setting. MobiEmu uses a cluster of $n$ Linux machines to emulate a MANET of $n$ nodes (see Figure 20). Although these testbed hosts are physically well-connected, the packet delivery behavior has been modified at the network device layer to generate the effect of real-world wireless communications and network dynamics. With MobiEmu, MANET software can be tested under the same wireless communication characteristics and networking environment as if it were running in a real MANET deployment.

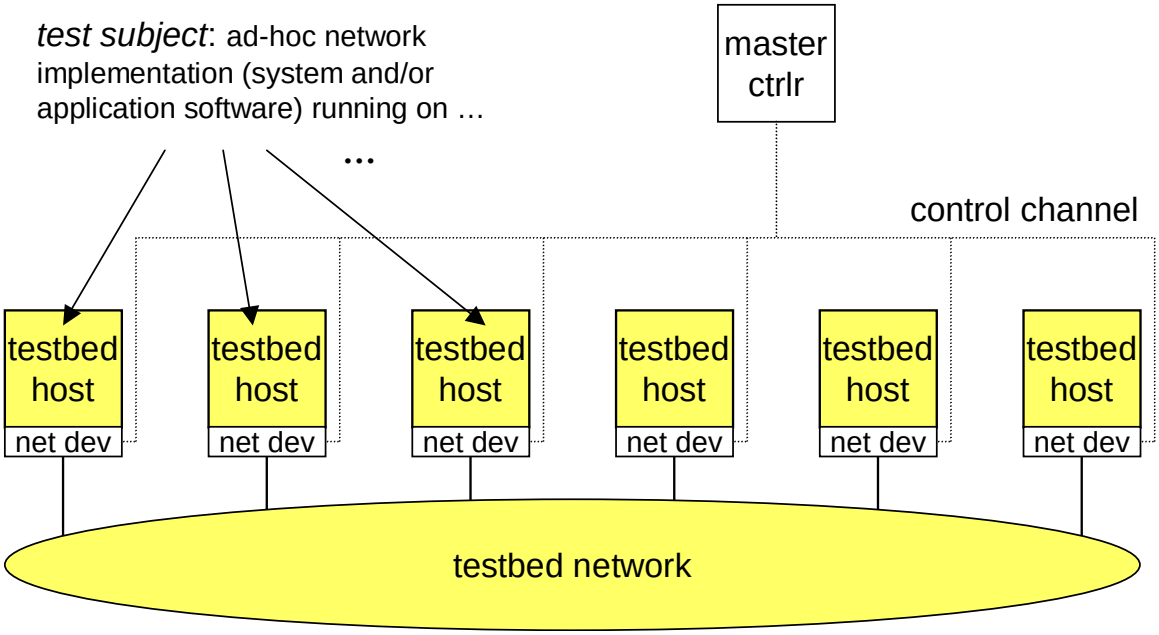



**Figure 20:** Emulating MANET with MobiEmu

MobiEmu experiments are driven by predefined *network scenario*, which is expressed

in a history of node motions and link characteristics changes. The node motions can determine current connectivity topology, and the link characteristics include bandwidth, delay, bit-error-rate, and loss rate. MobiEmu software enforces the topology and link characteristics by setting proper packet filtering and queuing rules at the device driver layer.

The MobiEmu system operates in a master/slave architecture. The master controller runs at a dedicated host outside the testbed network; a slave controller runs at each testbed host. The master controller controls all slaves and synchronizes their actions: the master dictates when changes (to topology and link characteristics) are needed according to the scenario and instructs the slaves to enforce such changes. The master/slave communication is on a separate control channel, which may be overlay on the testbed network if the overall load is low.

There are many benefits of using MobiEmu to evaluate secure MANET. First, all networking and above is real in MobiEmu, meeting our requirements to run experiments with actual secure MANET code. Second, since the communication effects are emulated, these experiments are reproducible. And third, since MobiEmu allows we run a wide range of MANET scenario without the need to physically move the nodes, we can easily repeat the experiments for a large set of test cases. Our experience of using MobiEmu in secure MANET research has further validated these points.

We have therefore use MobiEmu as the basis of our S-MobiEmu platform. To run experiments, secure MANET software (i.e., the test subject) will be loaded in each testbed host and run as if it were in a real deployment. S-MobiEmu accepts all MobiEmu scenarios, although not all attack test cases would make sense in all network scenarios. MobiEmu master controller has been extended to control and coordinate with attack emulation so that both network emulation and attack emulation are in sync.

## *8.3 Attack Emulation*

The attack emulation layer in S-MobiEmu provides the means to test the security aspect of the MANET system running in MobiEmu. It interacts with each node's software stack to inject the effects of a network under attacks. For example, if a MANET attack aims at compromising a node's routing agent and falsifying its route table, the attack emulation layer will instruct the routing agent to make such alteration in its route table as if it were indeed compromised. Then, the whole system can be put under test to see how it responds to such route alteration event.

### 8.3.1 The BASR Layer

The software layer that implements the attack emulation layer is called BASR (Basic Ad-hoc Security Routines). It consists of a set of "helper" modules that implement a library of convenient security routines and a common API for attack test cases (see Figure 19). Currently, BASR is designed for ad-hoc routing although it is extensible to support other application domains.

The purpose of BASR is to isolate the implementation of attack test cases and security systems from the routing protocol code as much as possible. In real network security scenarios, routing agents can be compromised and driven into running malicious codes. Implementing attacks or countermeasures to such attacks often requires modification of the routing agents. It is obviously inconvenient and error-prone to modify the routing agents every time a new possible attack is studied. Instead, the BASR layer abstracts the most common security routines into a common API to expedite the design of attack cases and security systems, thus minimizing direct code-injection into the routing agents.

The implementation of these "helper" modules is obviously routing protocol and implementation dependent. It is indeed necessary to modify route agent source code to implement the security functions provided by the API. That is, each instance of routing protocol implementation should be paired with an instance of the helper module as illustrated in

Figure 19. So far, we have implemented a BASR instance for AODV-UIUC [41], a public implementation of the AODV routing protocol. Further implementation on other protocols, such as DSR, is currently under development.

### 8.3.2 API Details

BASR supports the following three types of common routines:

1. Capturing and intercepting incoming and outgoing packets – the pcap [36] library is used to capture network packets, including both data packets and routing messages.

2. Overhearing traffic in neighboring nodes – wireless interface is put in the promiscuous mode to monitor traffic in the proximity of this node.

3. Access to routing table entries – routing table entries that usually reside internally to routing agents are made available in a shared memory block.

Here are the function prototypes of these common APIs:

- `register_callback(bool incoming, int type,`
  `addr_t src, addr_t dst, func callback);`

  This function creates a packet-matching rule and associates it with the given callback function. The callback function will be called, when a packet is received (when `incoming` value is true) or sent (when `incoming` value is false) at the wireless interface matches the given source, destination and protocol type. Protocol type can be specified as `TCP`, `UDP`, `RREQ`, `RREP`, `RERR`, etc., or bitwise-ORs of them, such as `TCP|UDP`. Source or destination address can be any IP address or wildcards. The callback function has the following form:

  ```
  int callback(addr_t src, addr_t dst,
  void * data, int len);
  ```

Sequential calls of this API (possibly from different processes) will register a chain of callback functions that will be invoked in the reversed order of registrations.

- `register_overhear_callback(int type,`
  `addr_t src, addr_t dst, func callback);`

  This function registers a callback function similar to the previous one, but it matches only those packets that are overheard in the neighborhood. The callback function takes an additional parameter that specifies the particular neighbor from which the packet is overheard.

- `rentry * read_route_entry(int dst);`
  `write_route_entry(int dst, rentry * new_rentry);`

  They provide read and write access to the routing table entry (rentry) corresponding to the given destination. The rentry structure includes fields essential to the routing protocol, such as destination, next hop (or source route), hops and sequence number, etc.

- `rentry * read_local_entry();`
  `write_local_entry(rentry * new_rentry);`

  They provide the interface to read and modify information of the host node itself. The interfaces are similar to the `read_route_entry` and `write_route_entry`.

### 8.3.3 An Example Attack Written in the API

We now use a simple example to demonstrate how we can use the API to program attack test cases. Let us assume a possible attack scenario: an attacker Malice tries to eavesdrop in communication from Alice to Bob. Let us assume they reside on nodes M, A and B respectively. Malice can achieve the goal by several means. The simplest approach (Approach I) is to intercept all traffic from A to B on the local interface of M. It only works when M is in the route path from A to B. The second approach (Approach II) improves by overhearing

nearby traffic as well. It works when there is at least some of M's neighbors resides in the interested route path. The most aggressive approach (Approach III) tries to proactively advertise a new route from A to B that contains M. Thus, no matter where Malice resides, it may always intercept the expected communication. Chapter 8.4.1 will describe the detailed technique to advertise a false route, briefly, a *Route Request* message is fabricated and it contains falsified originator and target fields, namely, B and A. By manipulating sequence number fields in the message, all nodes who receive the request will forward the message to other nodes. As a side effect, they will also update their route to B (the originator) via M (the previous hop). Eventually, A will also receive the message and update the route path to B accordingly, which contains M.

The following pseudo code segment illustrates how we can implement these three approaches with the BASR library. We assume that `disclose_data()` is a callback function that attempts to extract useful information from an intercepted data packet, and the function `broadcast()` broadcasts a packet. Here we provide a simplified RREQ structure only for demonstration purposes.

**Listing 1:** Eavesdrop Attack Example using BASR library

```
Eavesdrop_Approach_I(addr_t A, addr_t B) {
        BASR::register_callback(true, TCP|UDP, A, B,
                disclose_data);
}

Eavesdrop_Approach_II(addr_t A, addr_t B) {
        Eavesdrop_Approach_I(A,B);
        BASR::register_overhear_callback(TCP|UDP, A, B,
                disclose_data);
}

struct RREQ {
        addr_t src;      // the originator
        addr_t dst;      // the target
        int src_seq;
        int dst_seq;
        addr_t ip_src;   // the forwarder
};

Eavesdrop_Approach_III(addr_t A, addr_t B) {
```

```
        Eavesdrop_Approach_I(A,B);
        addr_t M=BASR::read_local_entry()->dst;
        int aseq=BASR::read_route_entry(A)->seq;
        int bseq=BASR::read_route_entry(B)->seq;
        RREQ rreq(B, A, bseq+1, aseq+1, M);
        broadcast(rreq);
}
```

## *8.4 Attack Library*

The Attack Library in S-MobiEmu is a well-organized and extensible collection of carefully designed attacks and test cases. It also provides the structure to assist researchers in developing new test cases in a new study.

The attack library organizes attacks and test cases in a hierarchy structure based on their composition. The core of the attack library is a collection called *Attack Foundation Library*, which contains all the *atomic* attacks that define the basic attack behavior on a single node. These attacks can be used as building blocks to construct *compound* attacks or complex test cases. These more sophisticated attacks can also span over multiple nodes.

### 8.4.1 Attack Foundation Library for Ad-hoc Routing

We used the attack taxonomy to build an attack foundation library that includes the basic attacks. All basic attacks listed in Table 2 are implemented in the attack foundation library.

### 8.4.2 Extending the Attack Library

The attack library is extensible because compound attacks can be built from the basic attacks in the foundation library or other attacks. Here we will present several realistic attacks that we developed and included in the extended attack library. They can serve as the common test cases to test, evaluate and compare different security solutions in their response to MANET threats. Furthermore, they can also be used as building blocks of more complicated attacks.

We now show a few attack examples in pseudo codes.

**Route Invasion** Inject a node in an active route.

**Listing 2:** Route Invasion

```
Route_Invasion(double duration /*unused*/, addr_t src, addr_t dst) {
        if  !read_route_entry(src) ||
                !read_route_entry(dst) {
                return NO_ATTACK;
        }
        cur=read_local_entry()->dst;
        cseq=read_local_entry()->seq;
        sseq=read_route_entry(src)->seq;
        dseq=read_route_entry(dst)->seq;
        False_Request(dst, src, dseq+1, sseq+1, cur);
        False_Request(cur, dst, cseq, dseq+1, cur);
}
```

If the route from `src` to `dst` exists, the attacker first generates a `False_Request` basic attack with a larger sequence number for `dst`. It will make all nodes, including `src`, update their routes to `dst` using `cur` as the next hop. Then, the attacker generates a second `False_Request` attack, which will launch a route discovery process to establish the route from `cur` to `dst`. Eventually, `cur` will be injected in the route from `src` to `dst`.

Note that this script does not prevent the route to be changed back later. We can implement a persistent version of this attack by calling the basic script repeatedly. The pseudo code looks like this:

**Listing 3:** Route Invasion Persistent Version

```
Route_Invasion_P(double duration, addr_t src, addr_t dst) {
        while(duration>0) {
                Route_Invasion(0, src, dst);
                sleep(period);
                duration=duration-period;
        }
}
```

Similar techniques may be applied to many other attacks as well.

**Route Loop** Create a route loop.

**Listing 4:** Route Loop

```
Route_Loop(double duration /*unused*/, addr_t src, addr_t dst) {
        if  !read_route_entry(src) ||
                !read_route_entry(dst) {
                return NO_ATTACK;
        }
        cur=read_local_entry()->dst;
        prev=read_route_entry(src).next_hop;
        next=read_route_entry(dst).next_hop;
        dseq=read_route_entry(dst).seq;
        Add_Route(dst, prev, dseq+1);
        Active_Reply(src, dst, dseq+1, cur, next);
}
```

If the attacker is close to a route from `src` to `dst` such that two subsequent nodes in this route, `prev` and `next`, are in the attacker's 1-hop neighborhood, the attacker can first add a route to `dst` using `prev` as the next hop. It then generates an `Active_Reply` basic attack to `next`, using a larger sequence number for `dst` in the RREP message. It will make `next` update its route to `dst` via `cur`. When `prev` receives a packet from `src`, the packet is forwarded according to the normal path and it will eventually reach `next`. However, `next` now thinks the best route to `dst` is through `cur` and `cur` forwards it back to `prev`. This effectively creates a loop from `src` to `dst` and all packets will be dropped in the route when their TTLs drop to zero.

A similar attack can be implemented when the attacker is not close to the targeted route. The attacker can first find a victim node V that is close to the route. Instead of calling `Add_Route` locally on V (which will require an additional compromise on V), the attacker can use either `False_Request` or `Active_Reply` to force V to update its route to `dst` via V's corresponding `prev`. The rest is similar.

**Sinkhole** Create a sinkhole that redirects all neighboring traffic to a particular node.

**Listing 5:** Sinkhole

```
Sinkhole(double duration /*unused*/, addr_t victim) {
        cur=read_local_entry()->dst;
        sseq=read_route_entry(victim)->seq;
        dst=random address that does not exist;
        dseq=random sequence number;
```

```
        False_Request(victim, dst, sseq+1, dseq, cur);
        Data_Drop_D(1.0, victim, TCP|UDP);
}
```

The attacker generates a `False_Request` that appears to come from the `victim` and to a non-existent destination. Since nobody has a route to that destination, the RREQ will eventually flood throughout the whole network. As a side effect, all nodes that receive the RREQ will update its route to the victim via `cur`. Eventually, `cur` becomes a Sinkhole for `victim`.

Note that the above attack examples only act on a single host. However, it is not difficult to develop a powerful distributed attack with two or more compromised hosts based on similar techniques.

## *8.5 Measurement and Evaluation Tools*

Performance measurement tools are designed to evaluate the effectiveness of the security solution in maintaining application mission objective when under attacks. They are very useful to compare alternative security solutions. Since different security solutions have different requirements and may target different ranges of attacks or threats, there is no single measurement that can be used alone to determine the best solution. To provide an objective basis for decision-making, we should support multiple measurement tools based on different performance models. How to prioritize and assess these metrics wisely and choose the best security solution(s) is a research problem that goes beyond our scope.

In S-MobiEmu, we build a set of measurement tools based on the cost-benefit analysis model [50]. They can be extended to build other measurement and evaluation tools.

Every security solution comes with a cost. We can identify the major cost factors as *response cost* and *operational cost* [50]. Response cost is the cost to perform responsive actions based on the intrusion evidence indicated by the security solution. Operational cost is the cost of applying security functions (e.g., encryption or intrusion analysis).

On the other hand, there is also a benefit by deploying a security solution. One benefit measurement is *damage cost*, which describes the degree of damage to the system that is caused by an attack when the security solution is not available. Another measurement is *effectiveness*, which describes how effective the security solution can reduce the damage cost of a particular attack.

1. Operational Cost
   - (a) System Resource Consumption
     - i. CPU Usage
     - ii. Memory Usage
   - (b) Network Resource Consumption
     - i. Communication Overhead
     - ii. Overhearing Overhead
2. Effectiveness
   - (a) Detection Accuracy
     - i. Detection Rate
     - ii. False Positive Rate

**Figure 21:** Performance Measurement and Evaluation Library for Ad-hoc Routing

In our framework, we consider only the *objective* measures that are relevant to routing security. In particular, we do not include the *response cost* and *damage cost* because they are application and environment specific, and can thus be *subjective*.

Most metrics in the tree are self-evident. We describe the operational cost in the amount of resource consumption, which can roughly be classified as system resource (such as CPU, memory, disk, etc.) consumption, and network resource (such as incoming and outgoing network traffic) consumption. In particular, we consider the the amount of overhearing traffic as an overhead as well. The usefulness of this metric can be shown by the energy efficiency problem. In wireless networks, energy efficiency is a very important issue. It is widely agreed that both communication overhead and overhearing overhead contribute

to the majority of energy consumption in a MANET environment. Therefore, energy consumption can be measured (approximately) in terms of both communication overhead and overhearing overhead.

## *8.6 Discussion*

### 8.6.1 Experience of Using S-MobiEmu

The objective of our IDS research is to investigate IDS techniques and to develop IDS-based security systems for MANET. Using S-MobiEmu, we have implemented two IDS frameworks, *node-based* framework from Chapter 4, and *cluster-based* framework from Chapter 5.

We expect our IDS can detect routing anomalies by utilizing information on both the internal states of the underlying routing protocol and the patterns of network events. In our implementation, we found that the BASR approach serves us very well for this purpose. We can fully reconstruct the protocol specification indirectly through BASR hooks and use the specification to detect anomalies. The implementation is non-trivial but it can be done fairly efficiently.

We also experimented with a similar intrusion detection system on the simulation platform ns-2. Compared with that experience, development using S-MobiEmu is easier, because of fewer resource constraints. By using the *user-mode Linux extension* to MobiEmu [96], we were able to experiment on an emulation platform of as many as 100 virtual nodes. Since each test experiment can be conducted in real-time, it turns out to be much faster than a simulated run. Thus, we were able to conduct a larger number of experiments with a wider parameter selection.

We further state that our implementation with BASR has additional security advantages than a straightforward implementation without BASR. We note that a traditional IDS solution requires a trace log from the routing protocol process as input. Let us assume an attacker may not have the source code to the routing protocol and therefore cannot tamper

with the normal protocol behavior directly. However, the attacker may still be able to obtain the needed privileges to modify the trace log file right before IDS can access it. This attack will not succeed in our implementation because we do not use the trace log as an intermediate audit log file. Instead, the IDS uses (read-only) helper hooks directly from the routing protocol.

### 8.6.2 Code Complexity

The BASR module for AODV is about 400 lines in C. The attack library, which includes the implementation of 28 basic attacks, which form the *Attack Foundation Library* and about 10 compound attacks, is implemented in about 3,500 lines in C++. The performance measurement toolkit contains about 800 lines of code. For our case study, the node-based IDS has about 15,000 lines of code. The cluster-based IDS has about 8,000 lines of code, excluding the shared code base from the node-based IDS.

### 8.6.3 Limitation

We would like to point out that S-MobiEmu is not suitable for studying attacks in physical layer (such as jamming), because the wireless communication is emulated. However, if we replace the network emulator (MobiEmu) with a real deployed MANET network, it is possible to use the rest of S-MobiEmu platform to run experiments, but such experiments may not be reproducible for the reasons we have explained earlier. Similarly, we may have to run real experiments for MAC-layer security study, because today's wireless MAC is almost always implemented in firmware and is inaccessible. However, if the MAC protocols are implemented in host OS, like in some new architecture such as "Native WiFi", we may be able to use S-MobiEmu in emulation mode. We also envision that S-MobiEmu can be extended to support MAC-layer security study in future software-defined radio platforms where MAC protocols are programmable in DSP or FPGA.

## *8.7 Summary*

In this work, we have explained the needs to have an experimental environment to assist the development and evaluation of secure MANET. We have developed one such platform called S-MobiEmu. It allows us to test actual secure MANET code in repeatable experiments. It provides the necessary programming abstraction for us to design and implement attack test cases and the flexibility for us to extend the attack library in the future. We have tested S-MobiEmu in our own secure MANET research. We used it to evaluate an Intrusion Detection System and gained very positive results. We believe that S-MobiEmu will be a very useful tool for secure MANET research community.

# CHAPTER IX

# BACKGROUND AND RELATED WORK

In this chapter, we discussed background and related work to our research.

## *9.1 Intrusion Detection*

As to intrusion detection in wired environments, since its early introduction [2, 21], it has received increasing interests from researchers and even vendors. The representative misuse detection systems are IDIOT [47] and STAT [35], which use Colored Petri Networks and State Transition Diagrams, respectively, to represent and pattern-match known intrusions.

These two are also host-based systems that monitor operating system events. NIDES [1] has a statistical component for anomaly detection, e.g., it can use system resource usages (e.g., process size) to detect anomalies. Bro [65] is a representative network-based IDS that performs packet capturing, filtering, and re-assembly, and invokes user-specified event handlers with intrusion detection and response logic. Both misuse and anomaly detection schemes are used to detect attacks to the Open Shortest Path First (OSPF) routing protocol [87]. We use the knowledge and experience from these efforts in our research.

### 9.1.1 Misuse Detection Approaches

There are many approaches in misuse detection. They differ in the representation as well as the matching algorithm employed to detect intrusion patterns.

**Expert system** The expert system contains a set of rules that describe the facts about attacks and inferences can be made from these rules. A rule is triggered when specified conditions are satisfied. Expert systems are typically very slow, because all of the audit data need to be imported as facts. Therefore, expert systems are rarely seen in commercial

products.

*Event Monitoring Enabling Responses to Anomalous Live Disturbances* (EMERALD) [71] is an extension of the Intrusion Detection Expert System (IDES) [37] and *Next Generation Intrusion Detection System* (NIDES) [1] by SRI International. EMERALD uses a rule-based expert system component for misuse detection. A forward-chaining rule-based expert system development toolset called the Production-Based Expert System Toolset (P-BEST) [52] is utilized to develop a modern generic signature-analysis engine. A chain of rules is established utilizing P-BEST to form the signature database.

**Pattern Recognition** Pattern recognition methods encodes knowledge about existing intrusions with signatures as patterns, and intrusion patterns are matched against audit logs. This method is efficient with concise representation of pattern rules, and therefore it is widely used in commercial tools.

**Colored Petri Networks** Intrusion signatures can also represented with *Colored Petri Networks* (CPNs). It is more general and provides the capability to write complex intrusion scenarios. However, it is relatively computationally expensive and thus less used in practice. *Intrusion Detection In Our Time* (IDIOT) is the one example that uses CPNs [48].

**State Transition Analysis** As yet another reprensentation, state transition diagrams can be used to represent intrusions as well. It is proposed by Porras and Kemmerer [35], which is implemented in USTAT - a real-time intrusion detection system for UNIX [34].

Vigna and Kemmerer [88] proposed NetSTAT that extends the original state transition analysis technique (STAT) [35]. It models an attack as a sequence of states and transitions in a finite state machine.

In our work, finite state machines are modeled for normal events, not abnormal events. Specification-based intrusion detection was proposed by Ko et al. [43] and Sekar et al. [78].

Specification-based approaches reduce false alarms by using manually developed specifications. Nevertheless, many attacks do not directly violate specifications and thus, specification-based approaches cannot detect them effectively. In our work, we apply both specification-based and statistical-based approaches to provide better detection accuracy and performance.

### 9.1.2 Anomaly Detection Approaches

Anomaly based IDSes assume that an intrusion can be detected by observing a deviation from normal or expected behavior of the systems or users. Normalcy is defined by the previously observed subject behavior, which is usually created during a training phase. The normal profile is later compared with the current activity. If a deviation is observed, IDS flag the unusual activity and generate an alarm. The advantages of anomaly detection based IDSes include that they might be complete to detect attacks, i.e., they can detect attempts that try to exploit new and unforeseen vulnerabilities. They are also less system-dependent. Disadvantages are that they may have very high false positive rates and are more difficult to configure because the comprehensive knowledge of the expected behavior of the system is required. They usually require a periodic online learning process in order to build the up-to-date normal behavior profile. Anomaly detection approach is harder to implement, which make them inappropriate for commercial use.

Anomaly detection is a very challenging problem. Early approaches [1] use statistical measures of system features, e.g., CPU usage, to build normal profiles. Lane et al. recently study machine learning approaches for modeling user behavior [49]. There have been studies on modeling program behavior, using system call traces and learning-based approaches [27, 77] and specification-based approaches [42]. Ghosh and Schwartzbard [28] propose using a neural network to learn a profile of normality. Hyperview [20] is another example of IDS that uses neural networks. Fan et al. [24] transform a one-class

problem with only normal data into a new problem with two classes by introducing artificial anomalies. Lee and Stolfo [51] proposed to use data-mining approach to construct intrusion detection models. There are many other anomaly detection techniques, such as PAYL [89].

Several approaches can be used to perform general one-class classification, such as K-means [56], *Principal Direction Partitioning* [8], *Self Organizing Maps* [44].

However, most of them are applied to a single series of features and cannot be generalized well to heterogeneous feature sets, while our cross-feature analysis can capture inter-feature correlation and automatically construct anomaly detection models.

### 9.1.3 Specification-Based Detection

Bhargavan et al. [5] analyzed simulations of the AODV protocol. Their work included a prototype AODV state machine. Our AODV EFSA is based on their work but has been heavily extended. Ning and Sun [61] also studied the AODV protocol and used the definition of atomic misuses, which is similar to our definition of basic events. However, our definition is more general because we have a systematic study of taxonomy of anomalous basic events in MANET routing protocols.

Recently, Tseng et al. [85] proposed a different specification-based detection approach. They assume the availability of a cooperative network monitor architecture, which can verify routing request-reply flows and identify many attacks. Nevertheless, there are security issues as well in the network monitor architecture which were not clearly addressed.

Orset et al. [62] extended our work and analyzed attacks targeting at the OLSR (*Optimized Link State Routing Protocol*) protocol, and it shows that a similar EFSA-based detection approach can detect some typical OLSR attacks. Their work uses only deterministic rules and therefore can only detect direct violations against the specification. We have improved over their work and provided comprehensive analysis with both deterministic and statistical methods to detect different attacks according to our attack taxonomy.

### 9.1.4 Other Approaches

Schnackenberg et al. [76] proposed a low cost *Intruder Detection and Isolation Protocol* (IDIP) architecture which achieves intrusion detection and tracking, automatic response by isolating problematic nodes. In their protocol, each IDIP enabled node makes a local decision about the proper response while it also forwards a central coordinator to collect related attack reports to form a better global picture. Wu et al. [16] discussed a real-time attack source identification system, by instructing intermediate routers to authenticate packets (using IPSEC) destined to the victim and a binary search on routers in possible paths can nail know exactly which router or link has been compromised. Although the approach is interesting, it assumes the topology is fixed and known before hand, which is hard to achieve in real time in MANETs. A more distributed approach is preferred.

## *9.2 Other Security Efforts in Ad Hoc Networks*

### 9.2.1 Misbehavior Monitoring and Incentive Based Routing

Watchdog and pathrater approach, discussed by Marti et al. [59], introduces two related techniques to detect and isolate misbehaving nodes, which are nodes that do not forward packets. In the "watchdog" approach, a node forwarding a packet verifies the next hop also forwards it. If not, a failure tally is incremented and misbehavior will be recognized if the tally exceeds certain threshold. The "pathrater" module then utilizes this knowledge of misbehaving nodes to avoid them in path selection. The approach is limited in several aspects. First of all, overhearing does not always work in case of collisions or weak signals. Secondly, pathrater actually awards the misbehaving node, if its motivation comes from selfishness, i.e., not "serving" others can reduce its battery power consumption. It does not prevent the misbehaving node from sending or receiving its own packets.

CONFIDANT [9] extends Marti's approach in numerous ways. Misbehaving nodes are not only excluded from forwarding routes, but also from requesting their own routes. Also, it includes a trust manager to evaluate the level of trust of alert reports and a reputation

system to rate each node. Only reports from trusted sources are processed. However, trust management in MANETs has not been well studied yet. For example, it is not clear how fast the trust level can be adjusted for a compromised node, especially if it has a high trust level initially. Detection decision only comes from the residing node's own observation, which is not always sufficient. Trust relationship, which is pre-determined in this work, does not always model the right relationship in run-time. Especially in case of nodes that are compromised after trust relationship has been established.

Buttyan et al. [12] suggests the use of tamper-resistant hardware on each node to encourage cooperation. Nodes are assumed to be unwilling to forward packets, unless it is stimulated. In this approach, a protected credit counter runs on the tamper-resistant device. It increases by one when a packet is forwarded. It refuses to send its own packets if the counter is smaller than a threshold $n$. Public key technology is used to exchange credit counter information among neighbors and verify if forwarding is really successful. The scheme has a few strong assumptions, including tamper-resistant hardware and public key technology, which may not be widely available in MANET.

SPARTA, suggested by Krugel et al. [46], builds IDS based on mobile agents. It also features an event definition language (EDL), which describes multiple-step correlated attacks from an intrusion specification database. However, we have not seen details on how these specifications are generated and used for well-known routing attacks.

### 9.2.2 Key Management

Key generation, distribution and management in MANET is challenging because of the absence of central management. Zhou and Haas [97] introduced a routing protocol independent distributed key management service. It exploits redundancies in the network topology and uses secret sharing to provide reliable key management. Stajano and Anderson [82] proposed a scheme that establishes secure transient association between mobile devices by "imprinting" according to the analogy to duckling acknowledging the first moving subject

they see as their mother. The devices can be imprinted several times.

### 9.2.3 Secure Routing

Although there are secure routing approaches in wired networks, such as [17, 79], they usually come with large communication overhead and do not work well in MANET because of its dynamically changing network topology.

Several researchers have recently proposed new secure routing protocols or security enhancement for existing protocols for MANET. For example, Papadimitratos et al. [63] propose a Secure Message Transmission (SMT) protocol which disperses a message into N pieces and transmits them through different paths, given a topology map of the network. A successful reception of any M-out-of-N pieces allows the reconstruction of the original message. The method protects data packet transmission, but the protection of routing topology need to be further strengthened.

Their Secure Routing Protocol (SRP) attempts to address this problem by establishing a prior secret association between every pair of the network nodes and protecting routing request and reply messages using the secret association. Using secret key cryptography leads to an efficient solution but key management is hard to scale. The prohibitive use of cached routes from intermediate nodes is also restricted.

Zapata [94] proposes the use of asymmetric cryptography to secure the AODV protocol [66].

Hu et al. [32] consider the problem of avoiding expensive public key computation in authentication in Ariadne, a secure version of the DSR protocol [39]. It primarily uses TESLA [70] an efficient broadcast authentication protocol that requires loose time synchronization, to secure route discovery and maintenance. To use TELSA for authentication, a sender generates a hash chain and determines a schedule to publish the keys of the hash chain. The key required to authenticate a packet will not be published before the packet has arrived at the receiver so that an adversary cannot have captured the key and forged the

packet. This is the same underlying broadcast authentication protocol that we currently use

Partridge et al. [64] reported that signal processing techniques can be used to perform traffic analysis on packet streams, even in encrypted form. In its current stage, it provides limited information other than timing information, which makes it not suitable for intrusion detection in higher network layers.

## *9.3 Intrusion Response*

Intrusion response in the ad hoc network is a new problem. As far as we know, there is no other research yet that study how to perform traceback and filtering in a dynamic network without infrastructure support. In the context of wired networks, IP traceback schemes include hash-based traceback [80], hop-by-hop tracing [10], out-of-band ICMP traceback [4], in-band probabilistic packet marking [75, 81, 92] and watermark-based [90] 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. Unfortunately, none of these solutions can be applied directly to MANET because none of these assumptions can be guaranteed to hold in this new environment.

Egress (ingress) filtering [26] can be deployed at routers to enforce that all traffic with illegal source addresses be blocked from establishing outbound (inbound) connections. For example, a packet with a forged and invalid source address may not even be able to leave its originating network because the gateway router can identify that the source address does not belong to the valid address range that it owns. In a pure MANET scenario, node mobility inevitably leads to dynamic route paths, and thus there is no equivalent concept of a gateway. Consequently, it is usually difficult to find a good place to enforce these filters. There are certain exceptions, especially in a hybrid network, where access points or

wireless gateways are good candidates to apply egress (ingress) filter techniques.

## *9.4 Feature Selection*

Many algorithms, such as Evolutionary Algorithms [29], SVM [11], PCA [60], can be used to perform feature selection. In this study, we use the simple forward selection algorithm which is very efficient and works pretty well in practice.

The problem of feature selection in one-class classification problem has been investigated by [84]. They studied, in particular, PCA, that can be applied to reduce the dimensionality for one-class classification problem. They find that retaining the high variance directions is not always the best option for one-class classification. However, it remains to be seen whether the same problem holds in other feature selection methods, such as Kernel PCA, ICA, etc.

## *9.5 Cluster-Based Routing*

Hierarchical network is an effective way to group (or cluster) a large number of nodes in a network. Distributed algorithms to form clusters have been studied extensively, for example, [3, 45, 86]. Most of these approaches have the drawback that the clusterhead computation can be easily manipulated (cheated) to elect an arbitrary compromised node. Nevertheless, the head-less cluster formation and maintenance protocol [45] does not have such problem. We use this cluster formation scheme as the basis of clique computation protocol because the clique structure allows us to effectively compute a selection function on random inputs from each member.

## *9.6 S-MobiEmu*

To the best of our knowledge, there is no similar secure MANET testing system reported in the open literature. In wired network security, the best known test environment is perhaps LARIAT, an IDS testbed used in the 1998 and 1999 DARPA Intrusion Detection evaluation [53, 54]. LARIAT provides a configurable test environment where intrusion detection

modules can be “plugged” in the testbed to capture audit data and invoke response. It provides many ways to configure background traffic and attack generation. However, it does not provide APIs to extend its attack library to accommodate more/new attacks.

# CHAPTER X

# CONCLUSION AND FUTURE WORK

In this chapter, we summarize our research, review contributions, and discuss the needed future work.

## *10.1 Conclusion*

*Mobile ad hoc networks* (MANET) are particularly vulnerable due to some of the fundamental characteristics, such as open medium, dynamic topology, distributed cooperation, and constrained capability.

*Intrusion Detection Systems* (IDS) provide the second wall of defense that is essential to the overall security of MANET. However, IDS design in MANET is a challenging task, because it requires a distributed design of IDS that is lacking from most wired IDSes available today. In addition, we believe that proper response systems are also critical, as effective detection will not be useful unless some actions can be taken properlyy.

Our central research problem is therefore the design of a scalable and distributed approach of intrusion detection and response systems for MANET. It should be noted that although our work uses MANET as the main evaluation platform, various pieces of our work, such as feature selection, *Cross-Feature Analysis*, and cluster formation protocols, may be suitable for a more generalized distributed network platform without centralized control.

In particular, our research work answers the following questions:

1. What kind of threats should be considered? Our taxonomy study of anomalous basic events identifies different categories of intrusions and we can identify categories that are suitable for detection approaches.

2. What statistical features should be used to construct the model? In history, short sequences of system calls from privileged processes can be used to effectively distinguish normal and intrusive behavior and utilized to construct host based IDSes. TCPdump data could be used to construct network based IDS. However, these features are not suitable to construct detection models to guard against MANET specific attacks. In our approach, we used a specification-based approach, where features can be enumerated automatically.

3. How do we build a anomaly detection model purely on normal data and with heterogeneous features? We build the *Cross-Feature Analysis* approach which is shown to be effective in ad hoc intrusion detection. In fact, it is a general approach that can be applied in many applications and different areas.

4. Why do we need two different frameworks: node-based and cluster-based detection? They complement each other, and both are very useful with different assumptions. Because of the importance of routing protocols in MANETs, we use many types of basic attacks and complex attacks as examples to illustrate the effectiveness of the node-based IDS. Simulation results illustrate that our IDS can achieve very good false positive rate and detection rate on most attacks.

5. What can be done after an attack is detected? We develop a distributed response framework: hotspot-based traceback and filtering. It is automatic and distributed, and thus suitable for the ad hoc environment.

We have conducted extensive simulations to evaluate the performance of our node-based and cluster-based IDS. Furthermore, we developed a security software toolkit that facilities new security protocol design. Our work is based on the MobiEmu emulation platform. It is superior than simulations in many aspects because real applications can be launched and evaluated, and performance measures can be also taken from real environments.

## *10.2 Future Work*

Up to now, not many research efforts have been devoted to MANET IDSes. This thesis provides our initial work in this respect. As a very new and promising research area, there are several interesting and important future directions:

- Instead of focusing on specific routing protocols, we can apply similar detection framework on all layers such as *Medium Access Control* (MAC) layers or application layers.
- Encourage further collaboration among nodes within two or more hops. In fact, the hotspot framework is an initial effort of distributed protocols that do not rely on intermediate routers to be trustworthy. However, communication between IDS agents may need to exchange information frequently, thus the overhead is fairly high and better algorithms are yet to be found.
- Further study on MANET specific attacks and comprehensive defense strategies. As we can envision from the security effort in wired networks, the security issue is going to be more and more challenging.
- Improve topology aware normalization and anomaly detection model generalization in general.
- Add more support from reliable infrastructure whenever it is available.

# APPENDIX A

# AODV EFSA SPECIFICATION

We construct AODV EFSAs by following the AODV Internet draft [66]. AODV uses hop-by-hop routing similar to distant vector based protocols. AODV does not use periodical route advertisements. Instead, a route is created only if it is demanded by data traffic [66].

We construct an AODV EFSA by following the AODV Internet draft version [66]. Our AODV EFSA is based on the AODV state machine from Bhargavan et al.'s work [5]. AODV uses hop-by-hop routing similar to distant vector based protocols such as RIP [57], but there are no periodical route advertisements. Instead, a route is created only if it is requested by data traffic where routes are not available [66].

The AODV EFSA (per destination) is shown in Figures 3 and 4. The reason that one EFSA is split into two sub-graphs are purely for the purpose of a better layout. Figure 4 is used within a certain period after a node has rebooted. After that, the normal Figure 3 should be used.

We define a unique EFSA for each destination host. We use the abbreviation *ob*, which stands for the **ob**served node, to specify the destination. Thus we can use EFSA(*ob*) to denote the EFSA for *ob*. In addition, there is a global variable **cur** that defines the node's own address. There is also a global variable **cSeq** denoting the node's own sequence number.

States: There are eight states per EFSA where the initial and final states correspond to the special situations when there is no actual route entry for the destination and when it has just been removed. A few other states include: *Valid*, which indicates that a route to the destination is available and valid; *Invalid* and *WaitRREP*, which indicate that a route is either unavailable yet or has been invalidated due to broken links. AODV keeps invalidated entries for efficiency reasons. *WaitRREP* is used only when

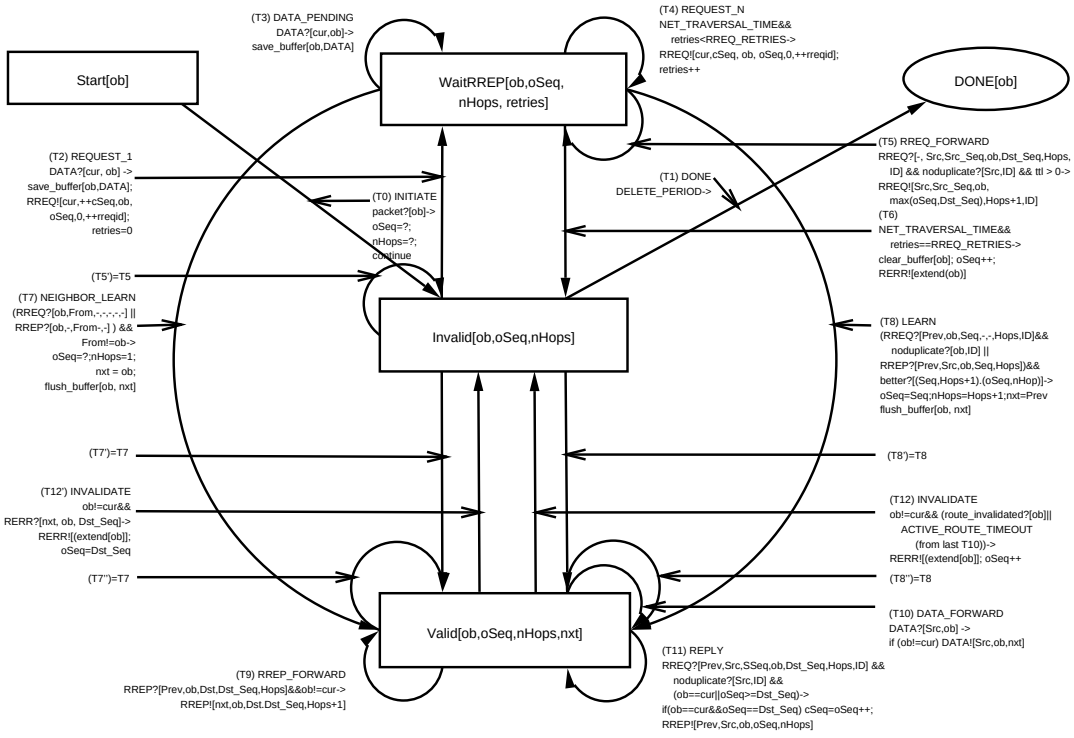



**Figure 22:** AODV Extended Finite State Automaton (for Destination *ob*): In Normal Use

a *Route Request* message has been delivered and a *Route Reply* is expected. The reboot subgraph has two other states that we will omit here.

Transitions: There are 19 transitions per EFSA.

Note that we only capture major AODV functionalities in the EFSA. Some specified protocol behavior relies on information from other layers, which we choose not to model for now. For instance, AODV can optionally use link-layer notifications to maintain local connectivity. Furthermore, since AODV, as well as many other MANET protocols, is evolving quickly, some parts of the specification are likely to change in future. We decide that it is not necessary to reproduce every detail as long as the major need of IDS can be satisfied. Instead, we abstract these functionalities with **predicates** (in input conditions) and **tasks** (in output actions). They are silently ignored by our current IDS implementation. Consequently, IDS currently cannot capture anomalies in these categories. Some missing

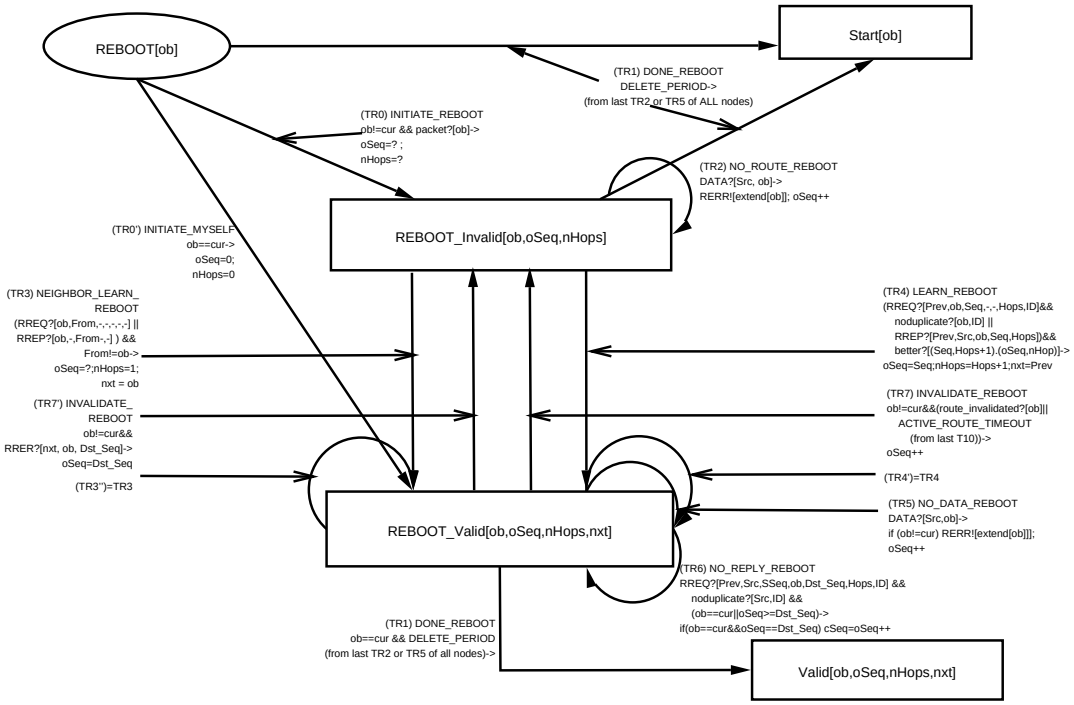



**Figure 23:** AODV Extended Finite State Automaton (for Destination *ob*): After Reboot

functionalities are as follows:

1. Hello Messages Handling;
2. 'Gratuitous RREP' flag and 'Destination only' flag in RREQ messages;
3. RREP Acknowledgment Handling;
4. RREP Subnet Prefix Usage;
5. Layer-2 Notification;
6. ICMP Messages;
7. Precursors List;
8. Computation of RERR Targets;

9. Rate Limiting of RREQ and RERR Messages;
10. RREP Blacklist;
11. Expanded Ring Search;
12. Buffering;
13. TTL handling.

We list all packet-receiving events, packet-delivery events, timeouts and auxiliary functions used in Figure 22 and 23 below. As a convention, packet-receiving events and boolean axillary functions end with '?', while packet-delivery events end with '!'. We use **Src**, **Dst**, **Src_Seq**, **Dst_Seq**, **Hops**, **Next**, **Prev**, **ID** to denote the following packet fields: *Source address*, *Destination address*, *Source Sequence Number*, *Destination Sequence Number*, *Number of Hops*, *Next Hop*, *Previous Hop* and *Route Request Identification Number* respectively.

- Packet-receiving events:

  DATA?[Src, Dst]: data packet.

  RREQ?[Prev, Src, Src_Seq, Dst, Dst_Seq, Hops, ID]: *Route Request*.

  RREP?[Prev, Src, Dst, Dst_Seq, Hops]: *Route Reply*. It should be noted that *Src* in RREP is consistent with the definition of *Src* in RREQ, that is, it specifies the originator of the route discovery process. In other words, it is actually the destination field in the IP header of a RREP packet.

  RERR?[Src, Dst, Dst_Seq]: *Route Error*.

- Packet-delivery events:

  DATA![Src, Dst, Next]: data packet.

  RREQ![Src, Src_Seq, Dst, Dst_Seq, Hops, ID]: *Route Request*.

RREP![Next, Src, Dst, Dst_Seq, Hops]: *Route Reply*. We explicitly specify *Next* since RREP uses unicast instead of broadcast.

RERR![Dsts]: *Route Error*. *Dsts* are the list of affected destinations.

- Timeouts:

  DELETE_PERIOD: timeout before an invalidated route is removed.

  ACTIVE_ROUTE_TIMEOUT: timeout before a valid route is invalidated due to inactivity.

  NET_TRAVERSAL_TIME: timeout before a *Route Reply* is received in response to *Route Request*.

- Auxiliary functions:

  noduplicate?(Src, ID): whether a *Route Request* message from *Src* with *ID* is not seen before.

  route_invalidated?(Dst): whether a route to *Dst* has been invalidated.

  packet?[Dst]: whether there is an incoming packet.

  better?([seq1, hop1],[seq2, hop2]]: whether a packet with seq1 and hop1 is more recent than a packet with seq2 and hop2.

  extend[Dst]: return a list of destinations whose routes include *Dst* as the next hop. Obviously, $Dst \in extend(Dst)$.

  save_buffer[Dst, DATA]: buffer *DATA* in buffer.

  flush_buffer[Dst, Next]: deliver all packets in buffer that are destined to *Dst* through *Next* and remove them from buffer.

  clear_buffer[Dst]: remove all data to *Dst* from buffer.

### A.0.1 Basic AODV Transition Flow

A normal state transition starts from the **Start** state, which represents the state where no routing entry (valid or invalid) exists for the observed node. The current state will change to **Invalid** if there is a packet targeted for the observed node (transition T0 in Figure 22). Note that we assume it enters **Invalid** first and if the packet helps establish a route, it will perform another transition (which can be transition T2, T5', T7' or T8'. See related descriptions below). When there is a data packet prepared to send to *ob* while the current state is **Invalid**, it buffers the data, broadcasts a RREQ (Route Request) message and enters **WaitRREP** (T2). During **WaitRREP**, it will not send another RREQ to the same destination even if another data packet is requesting the route to *ob* (T3). However, after timeout NET_TRAVERSAL_TIME, if there is no RREP (Route Reply) received for *ob*, it broadcasts another RREQ message (T4). This process repeats until a maximum of RREQ_RETRIES times of RREQ have been broadcast. If there is still no RREP received, the current state switches to **Invalid** and sends (or broadcasts) RERR to notify neighbors about the failure of data transmission to *ob*. The RERR message contains notification of undeliverable destinations, which includes not only *ob*, but also other destinations that use *ob* as the next hop (T6). In either **WaitRREP** or **Invalid**, if there is another incoming RREQ which also requests a route to *ob*, it is forwarded immediately (T5 or T5').

In either **WaitRREP** or **Invalid**, if there is a RREQ or RREP message with *ob* as the previous hop, the protocol assumes there is a direct link to *ob* and enters **Valid** (T7 or T7'). Similarly, if a RREQ originated from *ob* is overheard, we can take the route directly (T8 or T8'). Finally, if the corresponding RREP is received, it is used to create a new route and enters **Valid** (T8 or T8').

During **Valid**, if there is a RREQ or RREP message with *ob* as the previous hop, the protocol assumes there is a direct link to *ob* and replaces the current route with it (T7"). Similarly, if a RREQ originated from *ob* or another RREP received whose destination field is *ob*, the route will be updated if and only if the receiving message contains a better route

(T8").

If there is an incoming data packet, *ob*!=**cur**, and a valid route to *ob* exists, the packet is forwarded (T10). If there is a RREP packet originated from *ob*, while we are in the active route, the RREP packet is forwarded (T9). If there is a RREQ packet requesting a route to *ob* while we are in the **Valid** state, a RREP message is replied to the previous hop (T11). Under all three situations described, the current state remains **Valid**.

A valid route can be broken later if 1) a received RERR message includes *ob* in the unreachable destination list and the RERR is sent by the next hop to *ob*; or 2) the route is invalidated by other means (link layer notification, timeout of hello messages) or ACTIVE_ROUTE_TIMEOUT has elapsed with no active route activities. In either case, a RERR message is sent (or broadcast) and the current state becomes **Invalid** (T12 or T12').

Finally, if an invalid route is not validated after DELETE_PERIOD, it is removed from the route table and the EFSA enters the final **Done** state (T1), which will have the EFSA removed.

When a node reboots, all previous routes or states are lost. However, there may be neighboring nodes that still consider the node as the next hop in an active route. To avoid this situation, the system starts from **REBOOT**. It behaves like the normal graph but it will not forward any routing or data messages. If a data packet needs to be forwarded, a stale route before the reboot must have been used. A RERR message is then sent (or broadcast) instead, in order to notify neighbors that the route should be invalidated (TR2 or TR4). After DELETE_PERIOD, the state changes to **Start** (TR1), when normal routing behavior starts.

# APPENDIX B

# DSR EFSA SPECIFICATION

We construct DSR EFSAs by following the DSR Internet draft [39]. DSR uses source routing and it is also a on-demand routing protocol [39]. We use the MobiCache implementation of DSR in ns-2, and therefore many details also follow that implementation.

Our DSR EFSA is shown in Figure 24. Unlike the AODV EFSA, DSR does not have the special logic to handle packets during reboot time differently.

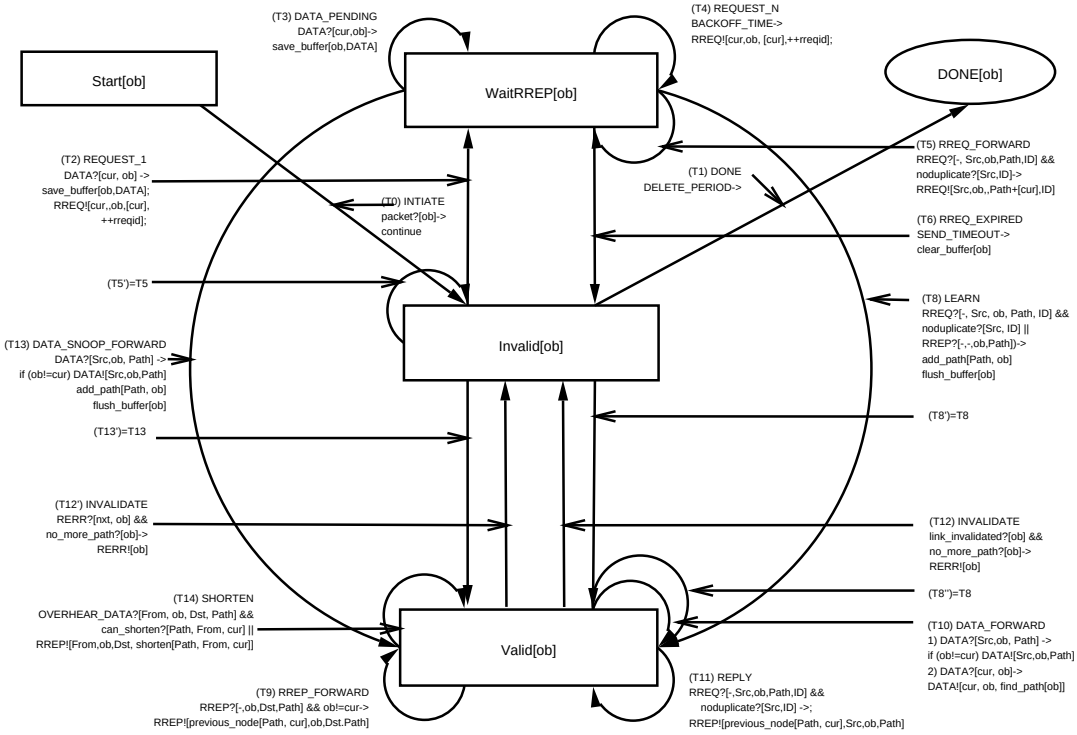



**Figure 24:** DSR Extended Finite State Automaton (for Destination *ob*):

Similar to the case of AODV, we define a unique EFSA for each destination host. We use the abbreviation *ob*, which stands for the **ob**served node, to specify the destination. Thus we can use EFSA(*ob*) to denote the EFSA for *ob*. And there is also a global variable

**cur** that defines the node's own address.

In our EFSA, DSR has five states and 14 transitions. Note that for compatibility with the AODV notation, we reserve transition T7 which does not actually exist in DSR. DSR also introduces two new transitions T13 (snooping) and T14 (automatic route shortening).

Again, we do not include all DSR functionalities because they cannot be modeled for now. Some missing functionalities are listed below:

1. Packet Salvaging;
2. Flow state Extension;
3. Layer-2 Notification;
4. Increased Spreading of Routing Error Messages;
5. Preventing Route Reply Storms;
6. Expanded Ring Search.

We list all packet-receiving events, packet-delivery events, timeouts and auxiliary functions used in Figure 24 below. As a convention, packet-receiving events and boolean axillary functions end with '?', while packet-delivery events end with '!'. We use **Src**, **Dst**, **Path**, **ID** to denote the following packet fields: *Source address*, *Destination address*, *Source Route* and *Route Request Identification Number* respectively.

- Packet-receiving events:

  DATA?[Src, Dst, Path]: data packet with source route.

  DATA?[Src, Dst]: data packet without source route.

  OVERHEARD_DATA?[From, Src, Dst, Path]: data packet overheard from another node *From*.

  RREQ?[Prev, Src, Dst, Path, ID]: *Route Request*.

RREP?[Prev, Src, Dst, Path]: *Route Reply*.

RERR?[Src, Dst]: *Route Error*.

- Packet-delivery events:

  DATA![Src, Dst, Next]: data packet.

  RREQ![Src, Dst, Path, ID]: *Route Request*.

  RREP![Next, Src, Dst, Path]: *Route Reply*. We explicitly specify *Next* since RREP uses unicast instead of broadcast.

  RERR![Dst]: *Route Error* for destination *Dst*.

- Timeouts:

  DELETE_PERIOD: timeout before an invalidated route is removed.

  BACKOFF_TIME: timeout before another *Route Request* is sent.

  SEND_TIMEOUT: timeout before a *Route Reply* is received in response to *Route Request*.

- Auxiliary functions:

  noduplicate?[Src, ID]: whether a *Route Request* message from *Src* with its rreq_id equal to *ID* **or less** is not seen before.

  link_invalidated?[Dst]: whether some link in any route to *Dst* has been invalidated.

  packet?[Dst]: whether there is an incoming packet.

  no_more_path?[Dst]: whether there is no more source route destined to *Dst*.

  can_shorten?[path, From, Node]: whether *Node* is in a later position than *From* in source route *Path* with more than one hop.

  save_buffer[Dst, DATA]: buffer *DATA* in buffer.

flush_buffer[Dst]: deliver all packets in buffer that are destined to *Dst* through *Next* and remove them from buffer.

clear_buffer[Dst]: remove all data to *Dst* from buffer.

add_path[Path, Dst]: add a new source route *Path* destined to *Dst*.

find_path[Dst]: find and return a source route destined to *Dst*.

previous_node[Path, Node]: find the node in source route *Path* prior to node *Node*.

shorten_path[Path, From, Node]: shorten *Path* by removing nodes from *From* to *Node* (both exclusive).

### B.0.2 Basic DSR Transition Flow

Most transition flow in DSR is similar to the AODV EFSA, therefore we do not repeat the detailed explanation from Appendix A.0.1. However, it should be noted that our EFSA for DSR reflects a relatively high abstract model because DSR maintains route caches differently from AODV. In particular, there may be multiple paths to each destination, and each path may be implicitly discarded when any link in the path is invalid. Therefore there remain a few important differences.

- In transition T12, a state switch from **Valid** to **Invalid** occurs when there are no more paths to destination *ob*.
- DSR does not learn new routes from RREQ or RREP messages with *ob* as the previous hop, therefore T7 (or T7') does not exist in the DSR EFSA.
- Instead, DSR snoops data packets with source route. If the source route is destined to *ob*, it is learned and it enables state switch for *ob* from **Invalid** (or **WaitRREP**) to **Valid** (T13 or T13').
- DSR also allows automatic route shortening when a node overhears a data packet with source route and it observes itself is included in the source route but in a later

position than the node it overhears from with more than one hop. This means a shorter route is possible. The node then send a "gratuitous" Route Reply to the originator of the data packet with the shortened path.

Figure 25 provides an example for route shortening. Node *E* overhears a data packet sent from *C* to D which originates from *A* and is destined to *G* with source route following the solid arrows. At this point, node *E* finds out a shorter route from *A* to *G* is possible by simply omitting node *D*. Therefore, it sends a RREP to node *A* with the shortened path.

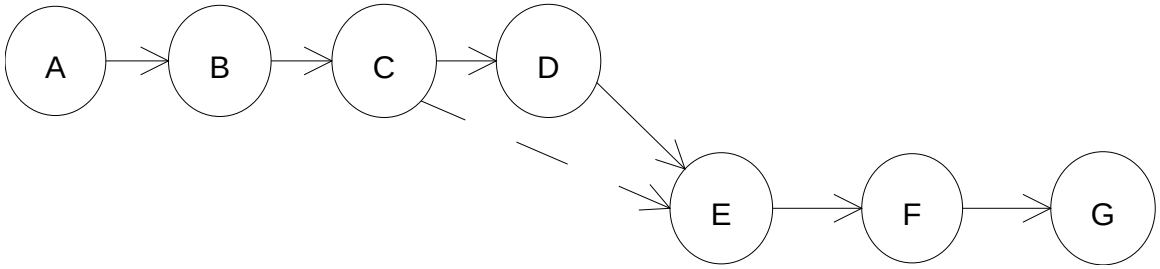



**Figure 25:** DSR Automatic Route Shortening Example:

# APPENDIX C

# OLSR EFSA SPECIFICATION

We construct OLSR EFSAs by following the OLSR Internet draft [18]. OLSR (*Optimized Link State Routing Protocol*) is a proactive link state routing protocol for ad hoc networks. It is optimized to reduce the flooding of link state information by selecting a subset of nodes called *Multipoint Relays (MPR)*. It also advertises only partial link state information in order to compute shortest path routes. The minimum set of links are the links from MPR nodes to nodes that select them. OLSR is well suited to large and dense mobile networks [18] where the traffic is random and sporadic rather than being almost exclusively between a small specific set of nodes.

Our OLSR EFSA is shown in Figure 26. It is based on Orset et al.'s work [62]. Because OLSR is a link state routing protocol, links play a central role in OLSR's core functions. Therefore, each OLSR EFSA represents a link between two nodes, instead of a route entry to a particular destination as the other two protocols.

In our EFSA, OLSR has four states and 23 transitions. For simplicity, we assume each node has only one interface and it can only claim one link to the same node [62].

Section 6.1.1 in RFC 3626 [18] defines all link codes. In our work, we only use the following link codes:

ASYM: Asymmetric link;

SYM: Normal symmetric link;

MPR: Symmetric link where the destination node is selected as the MPR of the source node.

We list all packet-receiving events, packet-delivery events, timeouts and auxiliary functions used in Figure 26 below. As a convention, packet-receiving events and boolean axillary functions end with '?', while packet-delivery events end with '!'. We use **Src**, **Dst**, **LinkCode** to denote the following packet fields: *Source address*, *Destination address* and the *LinkCode* field.

- Packet-receiving events:

  DATA?[Prev]: data packet received from the previous node *Prev*.

  Hello?[Src]: *Hello* message without link information.

  Hello?[Src, Dst, LinkCode]: *Hello* message that claims link *Src* to *Dst* with given the link code.

  TC?[Src, Dst]: *Topology Control* message received from *Src*, in which *Dst* is listed because *Dst* is one of its MPR selectors.

- Packet-delivery events:

  DATA![]: data packet transmitted according to the current routing table.

  Hello![Src]: *Hello* message without link information.

  Hello![Src, Dst, LinkCode]: *Hello* message that claims link *Src* to *Dst* with given the link code.

  TC![Src, Dst]: *Topology Control* message sent from *Src*, in which *Dst* is listed because *Dst* is one of its MPR selectors.

- Timeouts:

  HelloTimeOut: timeout before a *Hello* message is received.

  UpdateTimeOut: timeout before a link state updating message is received.

  TcTimerOut: timeout before a *Topology Control* message is received.

- Auxiliary functions:

  has_data?: whether there is a data packet to be sent by the current node.

  recv_tc: accept and handle an incoming *Topology Control* message.

- Boolean variables:

  SentHello: whether a hello message has been sent.

  InAsym: whether the link to *ob* is asymmetric.

  InMprSel: whether *ob* selects the current node as one of its MPRs.

  InMpr: whether the current node selects *ob* as one of its MPRs.

### C.0.3 Basic OLSR Transition Flow

We refer more details of OLSR transition flow to [62].

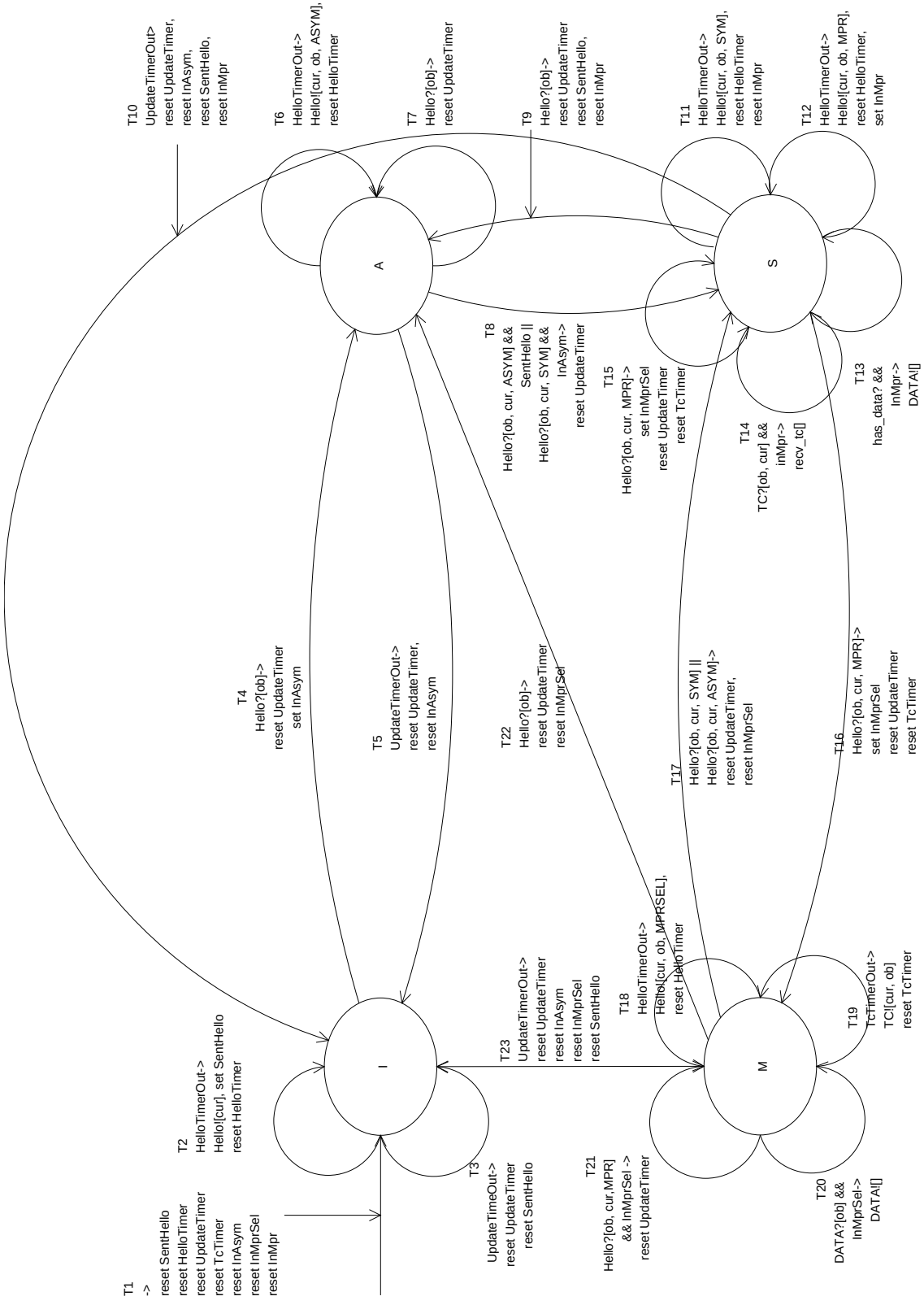



**Figure 26:** OLSR Extended Finite State Automaton (for Link between *cur* and *ob*):

# REFERENCES


[1] Anderson, D., Frivold, T., and Valdes, A., "Next-generation intrusion detection expert system (NIDES): A summary," Tech. Rep. SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, CA, May 1995. 9.1, 9.1.1, 9.1.2

[2] Anderson, J. P., "Computer security threat monitoring and surveillance," tech. rep., James P. Anderson Co., Fort Washington, PA, Apr. 1980. 9.1

[3] Basagni, S., "Distributed clustering for ad hoc networks," in *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'99)*, (Perth, Western Australia), pp. 310–315, June 1999. 5.2.1, 9.5

[4] Bellovin, S. M., "ICMP traceback messages," Internet draft draft-bellovin-itrace-00.txt, Network Working Group, Mar. 2000. expired 2000. 1.1.2, 9.3

[5] Bhargavan, K., Gunter, C. A., Kim, M., Lee, I., Obradovic, D., Sokolsky, O., and Viswanathan, M., "Verisim: Formal analysis of network simulations," *IEEE Transactions on Software Engineering*, 2002. 3.1.1, 9.1.3, A

[6] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., and Rogaway, P., "UMAC: Fast and secure message authentication," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'99)*, (London, UK), pp. 216–233, Springer-Verlag, 1999. 7.4.2

[7] Bloom, B. H., "Space/time trade-offs in hash coding with allowable errors," *Communications of ACM*, vol. 13, pp. 422–426, July 1970. 7.3.1, 7.4.2

[8] Boley, D., Borst, V., and Gini, M., "An unsupervised clustering tool for unstructured data," in *Proceedings of the Interntional Joint Conference on Aritical Intelligence (IJCAI'99)*, (Stockholm), Aug. 1999. 9.1.2

[9] Buchegger, S. and Boudec, J.-Y. L., "Performance analysis of the CONFIDANT protocol: Cooperation of nodes — fairness in dynamic ad-hoc networks," in *Proceedings of the Third IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, (Lausanne, Switzerland), June 2002. 9.2.1

[10] Burch, H. and Cheswick, B., "Tracing anonymous packets to their approximate source," in *Proceedings of the USENIX LISA Conference*, Dec. 2000. 1.1.2, 9.3

[11] Burges, C. J. C., "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998. 9.4

[12] Buttyán, L. and Hubaux, J.-P., "Stimulating cooperation in self-organizing mobile ad hoc networks," *ACM Journal for Mobile Networks (MONET), special issue on Mobile Ad Hoc Networks*, 2002. 9.2.1

[13] Camp, T., Boleng, J., and Davies, V., "A survey of mobility models for ad hoc network research," *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, pp. 483–502, 2002. 6.5

[14] Capkun, S., Buttyán, L., and Hubaux, J.-P., "Self-organized public-key management for mobile ad hoc networks," in *Proceedings of the ACM International Workshop on Wireless Security (WiSe'02)*, 2002. 1.5.2, 7.2.1

[15] Capkun, S., Hubaux, J.-P., and Buttyán, L., "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. 7.2.2

[16] Chang, H.-Y., Chen, P., Hayatnagarkar, A. N., Narayan, R., Sheth, P., Vo, N., Wu, C.-L., Wu, S. F., Zhang, L., Zhang, X., Gong, F., Jou, Y. F., Sargor, C., and Wu, X.-Y., "Design and implementation of a real-time decentralized source identification system for untrusted IP packets," in *Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX I)*, Jan. 2000. 9.1.4

[17] Cheung, S. and Levitt, K. N., "Protecting routing infrastructures from denial of service using cooperative intrusion detection," in *Proceedings of the New Security Paradigms Workshop*, (Cumbria, UK), Sept. 1997. 9.2.3

[18] Clausen, T., (editors), P. J., Adjih, C., Laouiti, A., Minet, P., Muhlethaler, P., Qayyum, A., and L.Viennot, "Optimized link state routing protocol (OLSR)," RFC 3626, Internet Engineering Task Force, Oct. 2003. 6.3.3, C, C

[19] Cohen, W. W., "Fast effective rule induction," in *Proceedings of the International Conference on Machine Learning (ICML'95)*, pp. 115–123, 1995. 4.2.2, 6.3.2

[20] Debar, H., Becker, M., and Siboni, D., "A neural network component for an intrusion detection system," in *Proceedings of 1992 IEEE Symposium on Research in Security and Privacy*, (Oakland, CA), pp. 240–250, May 1992. 9.1.2

[21] Denning, D. E., "An intrusion detection model," *IEEE Transactions on Software Engineering*, vol. 13, Feb. 1987. 9.1

[22] Fall, K., Varadhan, K., and the VINT project, *The ns Manual (formerly ns Notes and Documentation)*, 2000. 6.1, 8, 8.1.1

[23] Fan, L., Cao, P., Almeida, J., and Broder, A. Z., "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000. 2

[24] Fan, W. and Stolfo, S. J., "Ensemble-based adaptive intrusion detection," in *Proceedings of the Second SIAM International Conference on Data Mining (SDM'02)*, 2002. 9.1.2

[25] Feng, H. H., Kolesnikov, O. M., Fogla, P., Lee, W., and Gong, W., "Anomaly detection using call stack information," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 62–77, 2003. 3

[26] Ferguson, P. and Senie, D., "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," RFC 2267, Internet Engineering Task Force, Jan. 1998. 9.3

[27] Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A., "A sense of self for Unix processes," in *Proceedings of the IEEE Symposium on Security and Privacy*, (Los Alamitos, CA), pp. 120–128, 1996. 9.1.2

[28] Ghosh, A. K. and Schwartzbard, A., "A study in using neural networks for anomaly and misuse detection," in *Proceedings of the 8th USENIX Security Symposium*, 1999. 4.2.2, 9.1.2

[29] Guyon, I. and Elisseeff, A., "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003. 3.3, 9.4

[30] Hu, Y.-C. and Johnson, D. B., "Securing quality-of-service route discovery in on-demand routing for ad hoc networks," in *Proceedings of the 2nd ACM Workshop on Security of Ad hoc and Sensor Networks (SASN'04)*, (Washington, DC), pp. 106–117, 2004. 1.5.2

[31] Hu, Y.-C., Perrig, A., and Johnson, D. B., "Packet leashes: A defense against wormhole attacks in wireless networks," in *Proceedings of IEEE INFOCOM*, (San Francisco, CA), pp. 1976–1986, Apr. 2003. 1.5.2, 2.1, 7.2.1

[32] Hu, Y.-C., Perrig, A., and Johnson, D. B., "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. 1, 1.1.2, 7.2.1, 9.2.3

[33] Hubaux, J.-P., Buttyán, L., and Capkun, S., "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. 7.2.1

[34] Ilgun, K., "USTAT: A real-time intrusion detection system for Unix," Master's thesis, University of California at Santa Barbara, Nov. 1992. 9.1.1

[35] Ilgun, K., Kemmerer, R. A., and Porras, P. A., "State transition analysis: A rule-based intrusion detection approach," *Software Engineering*, vol. 21, no. 3, pp. 181–199, 1995. 1.1.2, 9.1, 9.1.1

[36] Jacobson, V., Leres, C., and McCanne, S., "`tcpdump`." available via anonumous ftp to ftp.ee.lbl.gov, June 1989. 1

[37] Javitz, H. S. and Valdes, A., "The SRI IDES statistical anomaly detector," in *Proceedings of the IEEE Research in Security and Privacy*, (Oakland, CA), pp. 316–376, May 1991. 1.1.2, 9.1.1

[38] Johnson, D. B. and Maltz, D. A., "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing* (Imielinski, T. and Korth, H., eds.), pp. 153–181, Kluwer Academic Publishers, 1996.

[39] Johnson, D. B., Maltz, D. A., and Broch, J., "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks," in *Ad Hoc Networking* (Perkins, C. E., ed.), ch. 5, pp. 139–172, Addison-Wesley, 2001. 2.1, 3.1.2, 5.1, 5.3.2, 9.2.3, B

[40] Karlof, C. and Wagner, D., "Secure routing in wireless sensor networks: Attacks and countermeasures," in *Proceedings of the WMCA*, 2003. 5.3.2

[41] Kawadia, V., Zhang, Y., and Gupta, B., "System services for ad-hoc routing: Architecture, implementation and experiences," in *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys'03)*, (San Francisco, CA), May 2003. 6.1, 8.3.1

[42] Ko, C., "Logic induction of valid behavior specifications for intrusion detection," in *Proceedings of the IEEE Symposium on Security and Privacy*, (Oakland, CA), 2000. 9.1.2

[43] Ko, C., Ruschitzka, M., and Levitt, K. N., "Execution monitoring of security-critical programs in distributed systems: A specification-based approach," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 134–144, 1997. 9.1.1

[44] Kohonen, T., "Self-organizing maps," 1995. 9.1.2

[45] Krishna, P., Vaidya, N. H., Chatterjee, M., and Pradhan, D. K., "A cluster-based approach for routing in dynamic networks," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 2, pp. 49–64, 1997. 5.2.1, 5.2.2, 9.5

[46] Krügel, C. and Toth, T., "Flexible, mobile agent based intruion detection for dynamic networks," in *Proceedings of the European Wireless Conference (EW'02)*, 2002. 9.2.1

[47] Kumar, S., *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, Aug. 1995. 9.1

[48] Kumar, S. and Spafford, E. H., "An application of pattern matching in intrusion detection," Tech. Rep. 94-013, Department of Computer Sciences, Purdue University, 1994. 1.1.2, 9.1.1

[49] Lane, T. and Brodley, C. E., "Temporal sequence learning and data reduction for anomaly detection," *ACM Transactions on Information and System Security*, vol. 2, no. 3, pp. 295–331, 1999. 4.2.2, 9.1.2

[50] Lee, W., Fan, W., Miller, M., Stolfo, S. J., and Zadok, E., "Toward cost-sensitive modeling for intrusion detection and response," *Journal of Computer Security*, vol. 10, no. 1,2, 2002. 8.5

[51] Lee, W. and Stolfo, S. J., "A framework for constructing features and models for intrusion detection systems," *Information and System Security*, vol. 3, no. 4, pp. 227–261, 2000. 9.1.2

[52] Lindqvist, U. and Porras, P. A., "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 146–161, 1999. 9.1.1

[53] Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., and Zissman, M. A., "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation," in *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX I)*, vol. 2, Jan. 2000. 8, 8, 9.6

[54] Lippmann, R. P., Haines, J. W., Fried, D. J., Korba, J., and Das, K. J., "Analysis and results of the 1999 DARPA off-line intrusion detection evaluation," in *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID'00)*, Oct. 2000. 8, 8, 9.6

[55] Liu, D. and Ning, P., "Multilevel $\mu$TESLA: Broadcast authentication for distributed sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, pp. 800–836, Nov. 2004. 1.5.2, 7.2.1

[56] MacQueen, J. B., "Some methods for classification and analysis of multivariate observations," in *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press*, vol. 1, pp. 281–297, 1967. 9.1.2

[57] Malkin, G., "RIP version 2 - carrying additional information," RFC 1723, Internet Engineering Task Force, Nov. 1994. 3.1.1, A

[58] Maltz, D. A., Broch, J., Jetcheva, J. G., and Johnson, D. B., "The effects of on-demand behavior in routing protocols for multi-hop wireless ad hoc networks," *IEEE Journal on Selected Areas in Communications*, Aug. 1999. 6.1

[59] Marti, S., Giuli, T. J., Lai, K., and Baker, M., "Mitigating routing misbehavior in mobile ad hoc networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (Mobicom'00)*, pp. 255–265, 2000. 5.1, 5.2.1, 6.1, 7.2.1, 7.2.2, 7.5, 9.2.1

[60] Mika, S., Schölkopf, B., Smola, A. J., Müller, K.-R., Scholz, M., and Rätsch, G., "Kerenel PCA and de-noising in feature spaces," *Advances in Neural Information Processing*, pp. 536–542, 1999. 3.3, 9.4

[61] Ning, P. and Sun, K., "How to misuse AODV: A case study of insider attacks against mobile ad-hoc routing protocols," in *Proceedings of the 4th Annual IEEE Information Assurance Workshop*, pp. 60–67, June 2003. 2.2, 2.2, 9.1.3

[62] Orset, J.-M., Alcalde, B., and Cavalli, A., "An EFSM-based intrusion detection system for ad hoc networks," in *Proceedings of the 3rd International Symposium of Automated Technology for Verification and Analysis (ATVA'05)*, (Taipei, Taiwan), Oct. 2005. 6.3.3, 9.1.3, C, C, C.0.3

[63] Papadimitratos, P. and Haas, Z. J., "Securing mobile ad hoc networks," *Handbook of Wireless Networks and Mobile Computing*, 2002. 9.2.3

[64] Partridge, C., Cousins, D., Jackson, A. W., Krishman, R., Saxena, T., and Strayer, W. T., "Using signal processing to analyze wireless data traffic," in *Proceedings of the ACM Workshop on Wireless Security (WiSe'02)*, (Atlanta, GA), Sept. 2002. 9.2.3

[65] Paxson, V., "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999. 9.1

[66] Perkins, C. E., Belding-Royer, E. M., and Chakeres, I., "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). 2.1, 2.2, 3.1.1, 6.2, 6.6, 9.2.3, A

[67] Perkins, C. E., Belding-Royer, E. M., and Das, S. R., "Ad hoc on-demand distance vector (AODV) routing," Internet draft draft-ietf-manet-aodv-13.txt, Internet Engineering Task Force, Feb. 2003. expired 2003.

[68] Perkins, C. E. and Royer, E. M., "Ad hoc on-demand distance vector routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, (New Orleans, LA), pp. 90–100, Feb. 1999.

[69] Perkins, C. E., Royer, E. M., Das, S. R., and Marina, M. K., "Performance comparison of two on-demand routing protocols for ad hoc networks," *IEEE Personal Communications Magazine special issue on Ad hoc Networking*, pp. 16–28, Feb. 2001. 6.1

[70] Perrig, A., Canetti, R., Tygar, D., and Song, D. X., "The TESLA broadcast authentication protocol," *Cryptobytes (RSA Laboratories, Summer/Fall)*, vol. 5, no. 2, pp. 2–13, 2002. 5.2.1, 9.2.3

[71] Porras, P. A. and Neumann, P. G., "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proceedings of the 20th National Information Systems Security Conference*, Oct. 1997. 9.1.1

[72] Quinlan, J. R., *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993. 4.2.2

[73] Raffo, D., *Security Schemes for the OLSR Protocol for Ad Hoc Networks*. PhD thesis, Université Paris 6, Sept. 2005. Computer Science. 6.3.3, 6.3.4

[74] Satyanarayanan, M., Kistler, J. J., Mummert, L. B., Ebling, M. R., Kumar, P., and Lu, Q., "Experience with disconnected operation in a mobile computing environment," in *Proceedings of the USENIX Symposium on Mobile & Location-Independent Computing*, pp. 11–28, 1993. 1.1.1

[75] Savage, S., Wetherall, D., Karlin, A., and Anderson, T., "Network support for IP traceback," *ACM/IEEE Transactions on Networking*, vol. 9, pp. 226–239, June 2001. 1.1.2, 7.3, 9.3

[76] Schnackenberg, D., Djahandari, K., and Sterne, D., "Infrastructure for intrusion detection and response," in *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX I)*, Jan. 2000. 9.1.4

[77] Sekar, R. C., Bendre, M., Dhurjati, D., and Bollineni, P., "A fast automaton-based method for detecting anomalous program behaviors," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 144–155, 2001. 9.1.2

[78] Sekar, R. C., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., and Zhou, S., "Specification-based anomaly detection: A new approach for detecting network intrusions," in *Proceedings of the 9th ACM Conference on Computer and Communication Security (CCS'02)*, 2002. 3.1, 3.1, 9.1.1

[79] Smith, B. R., Murthy, S., and Garcia-Luna-Aceves, J. J., "Securing distance-vector routing protocols," in *Proceedings of the Symposium on Network and Distributed System Security (NDSS'97)*, pp. 85–92, 1997. 9.2.3

[80] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Kent, S. T., and Strayer, W. T., "Hash-based IP traceback," in *Proceedings of the ACM Conference on Communications Architectures, Protocols and Applications(SIGCOMM'01)*, 2001. 1.1.2, 7.1, 7.3.1, 7.3.1, 9.3

[81] Song, D. X. and Perrig, A., "Advanced and authenticated marking schemes for IP traceback," in *Proceedings of the IEEE INFOCOM*, vol. 2, 2001. 1.1.2, 7.3, 9.3

[82] Stajano, F. and Anderson, R., "The resurrecting duckling: Security issues for adhoc wireless networks," *Security Protocols. 7th International Workshop Proceedings, Lecture Notes in Computer Science*, pp. 172–194, 1999. 5.3.2, 7.2.2, 9.2.2

[83] Systems, I. S., *RealSecure Network Protection*, Nov. 2003.

[84] Tax, D. M. and Müller, K.-R., "Feature extraction for one-class classification," in *Proceedings of Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP'03)*, pp. 342–349, 2003. 3.3, 6, 9.4

[85] Tseng, C.-Y., Balasubramanyam, P., Ko, C., Limprasittiporn, R., Rowe, J., and Levitt, K. N., "A specification-based intrusion detection system for AODV," in *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03)*, (Fairfax, VA), Oct. 2003. 2.2, 9.1.3

[86] Vasudevan, S., DeCleene, B., Immerman, N., Kurose, J., and Towsley, D., "Leader election algorithms for wireless ad hoc networks," in *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, Apr. 2003. 5.2.1, 9.5

[87] Vetter, B. M., Wang, F., and Wu, S. F., "An experimental study of insider attacks for the OSPF routing protocol," in *Proceedings of the 5th IEEE International Conference on Network Protocols (ICNP'97)*, (Atlanta, GA), IEEE press, Oct. 1997. 9.1

[88] Vigna, G. and Kemmerer, R. A., "NetSTAT: A network-based intrusion detection approach," in *Proceedings of the 14th Annual Computer Security Applications Conference*, 1998. 9.1.1

[89] Wang, K. and Stolfo, S. J., "Anomalous payload-based network intrusion detection," in *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, (French Riviera, France), pp. 102–124, Sept. 2004. 9.1.2

[90] Wang, X., Reeves, D. S., Wu, S. F., and Yuill, J., "Sleepy watermark tracing: An active intrusion response framework," in *Proceedings of the 16th International Information Security Conference (IFIP/Sec'01)*, June 2001. 1.1.2, 9.3

[91] Warrender, C., Forrest, S., and Pearlmutter, B. A., "Detecting intrusions using system calls: Alternative data models," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 133–145, 1999. 3

[92] Yaar, A., Perrig, A., and Song, D. X., "FIT: Fast internet traceback," in *Proceedings of IEEE INFOCOM*, (Miami, FL), Mar. 2005. 1.1.2, 7.3, 9.3

[93] Yang, H., Meng, X., and Lu, S., "Self-organized network layer security in mobile ad hoc networks," in *Proceedings of the ACM Workshop on Wireless Security (WiSe'02)*, (Atlanta, GA), Sept. 2002. 5.3.2

[94] Zapata, M. G., "Secure ad hoc on-demand distance vector (SAODV) routing," Internet draft draft-guerrero-manet-saodv-00.txt, Internet Engineering Task Force, Aug. 2001. expired 2002. 1, 1.1.2, 9.2.3

[95] Zhang, Y. and Lee, W., "Intrusion detection in wireless ad-hoc networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (Mobicom'00)*, pp. 275–283, 2000. 1.1.1

[96] Zhang, Y. and Li, W., "An integrated environment for testing mobile ad-hoc networks," in *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, (Lausanne, Switzerland), June 2002. 6.1, 8, 8.1.1, 8.1.2, 8.2, 8.6.1

[97] Zhou, L. and Haas, Z. J., "Securing ad hoc networks," *IEEE Network*, vol. 13, no. 6, pp. 24–30, 1999. 9.2.2

# VITA

Yi-an Huang was born in Shanghai, China. In 1999, he received his Bachelor Degree in Computer Science from Fudan University, Shanghai, China. During this period, he took internship in Microsoft Great Asian Support Center in 1998. After that, he was awarded the degree of Masters of Science in Computer Science from North Carolina State University in 2001. Subsequently, he joined the Computer Science Ph.D program in the College of Computing at the Georgia Institute of Technology in Fall 2001. During his study, he conducted Ph.D research under the advisement of Dr. Wenke Lee in the area of intrusion detection, in particular in mobile ad hoc networks. He has also been in research collaborations with IBM T.J. Watson Research Center during summer internships.