uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Maria Alexandrovna Gorlatova
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Wormhole Attack Detection in Wireless Ad Hoc Networks

TITRE DE LA THÈSE / TITLE OF THESIS

R. Liscano
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

P. Mason
CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. Miri

F.R. Yu

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Wormhole Attack Detection in Wireless Ad Hoc Networks

Maria A. Gorlatova

A Thesis submitted to the Faculty of Graduate Studies and Postdoctoral Studies in partial fulfilment of the requirements for the degree of Master of Applied Science, Electrical Engineering

January 2007

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-32451-6
Our file Notre référence
ISBN: 978-0-494-32451-6

Abstract

This thesis deals with wormhole attack discovery in mobile wireless ad hoc networks. Two separate approaches to wormhole attack discovery are developed in this thesis. One approach – based on protocol-breaking – allows detection of wormholes that disrupt network operations by dropping network packets. Another – a novel frequency-based analysis of periodic network messages – detects wormholes that do not drop traffic. The developed wormhole attack discovery techniques are local, do not rely on specialized hardware or clock synchronization, and do not require modification to existing ad hoc network routing protocols.

In addition, tools that are necessary for ad hoc network attack research are created. Network traffic analyzer modules applicable to ad hoc network research are developed and tested. Also, a realistic implementation of a wormhole attack in the NS-2 network simulator is created.

Contents

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisors, Dr. Peter Mason and Dr. Ramiro Liscano, for providing their expertise, guidance, and assistance.

I would like to thank Dr. Peter Mason for his constant thoughtful and caring mentorship. Through his continuous involvement in my work, Peter taught me how to think like a scientist: how to critically review information, how to analyze data, how to come up with ideas, how to present results. I am thankful to Peter for being confident in me and for teaching me to be confident in myself.

Also, Peter, mostly by example, taught me that even in the most serious and formal situations it is okay to be a little childish at times and to have fun.

I would like to express my sincere gratitude to Defence Research and Development Canada (DRDC) for supporting this research, as well as for proving me with office space, laboratory access, and software I needed to complete this thesis.

I would also like to thank Maoyu Wang and Louise Lamont from the Communications Research Centre (CRC) MANET lab who developed a testbed wormhole attack implementation and were kind enough to help me with collecting traffic on their network. The experimental data set obtained with their assistance is an important part of this thesis. I would also like to thank them for sharing their expertise in the MANET area.

Also, I am very grateful to Grant Vandenberghe who developed the main functionality of the Network Traffic Analyzer (NTA) that I have extended in this thesis work. I am sincerely thankful to Grant for helping me learn NTA and get accustomed to it.

I would also like to thank NSERC for partially funding this work through a generous CGS scholarship.

Last but not least, I would like to thank my husband, Valentine Murzenok, for his endless patience and support. In particular, I want to thank Valentine for all these times when, to help me work or study at home, he did all he could to provide me a distraction-free environment. For all the times Val lacked music, TV, and attention because of my studies, I am truly thankful.

Chapter 1

Introduction

Mobile wireless ad hoc networks (MANETs) are a relatively new field of research. Such networks are fundamentally different from wired networks, as they use a wireless medium to communicate, do not rely on fixed infrastructure, and can arrange themselves into a network quickly and efficiently. In a MANET, each node can serve as a router for other nodes, which allows data to travel, utilizing multi-hop network paths, beyond the line of sight.

MANETs are particularly attractive for situations where deployment of infrastructure is costly or impractical, such as military deployments, emergency rescue operations, and short-lived conference or classroom activities. The security of such networks, however, is a great concern [1]. The open nature of the wireless medium makes it easy for outsiders to listen to network traffic or interfere with it. Lack of centralized control authority makes deployment of traditional centralized security mechanisms difficult, if not impossible. Since there are not, necessarily, any clear network entry points, it is difficult to implement perimeter-based defence mechanisms such as firewalls. Finally, in a MANET nodes might be battery-powered and have very limited resources, making the

use of heavy-weight security solutions undesirable [1, 2, 3].

A large number of routing protocols for MANETs have been proposed to enable quick and efficient network creation and restructuring. However, common ad hoc routing protocols were not designed with security in mind and assume a trusting and cooperative environment [1]. Security of MANET routing is an active research area at this time.

Wormhole attacks are among the most severe attacks on ad hoc networks and are perhaps the most difficult attacks to prevent or detect [5, 6]. A wormhole attack takes its name from physics, where a wormhole is a thin tube of space-time that connects distant regions of the universe. A network wormhole attack 'connects' distant parts of the network. A wormhole is created by two collaborating attackers. One attacker records the traffic on one end of the network, and tunnels the traffic, using an off-channel link, to the other attacker located at another end of the network. The second attacker then rebroadcasts the traffic at the other network end. These attackers' actions fool distant network nodes into believing that they are direct neighbours, forcing these nodes to communicate through the wormhole adversaries, thus giving wormhole attackers full control of the link between valid network nodes. Wormhole attacks are very difficult to detect and prevent because cryptography-based measures do not alleviate them, as wormholes do not introduce new network messages, nor do they alter existing network messages.

1.1 Contributions to the field

This thesis deals with wormhole attack discovery in wireless ad hoc networks. The contributions from this work to the field are two-fold: the tools allowing the study of wormhole attacks and the techniques for wormhole attack detection.

The tools that are necessary for the study of wormhole attacks in wireless ad

hoc networks are developed in this thesis. These are: a network traffic analyser suite capable of working with wireless ad hoc network traffic and an NS-2 wormhole attack simulator. The developed network traffic analyzer suite should allow researchers to easily study ad hoc network traffic. The NS-2 wormhole attack simulation is a realistic, detailed representation of a wormhole attack in a simulator environment; it will allow researchers to easily modify wormhole attack parameters and test novel wormhole attack detection/prevention mechanisms.

Two different traffic analysis techniques that are applicable to wormhole attack detection in MANETs are developed in this thesis. These techniques have the following properties:

- They rely on routing messages and neither introduce overhead nor require changes to existing routing protocols
- They do not require specialized hardware or clock synchronization between network nodes
- They are local techniques that are to be performed by individual nodes; no cooperation between nodes is required

One of these techniques – based on protocol-breaking – allows for easy detection of a malicious wormhole that disrupts network communications by dropping packets. Another – frequency-based analysis of routing messages – allows detection of wormholes that are 'dormant' i.e. merely present on a network but not yet malicious. This latter technique presents a novel approach to wormhole attack detection: rather than trying to detect that packets have *travelled farther than they should or faster than they should* (as other wormhole attack detection techniques do), this technique works by detecting that packets have been *processed more than they should.*

A paper based on this work was published in the proceedings of the IEEE

Military Communications conference:

- *M.A. Gorlatova*, P.C. Mason, M. Wang, L. Lamont, R. Liscano, *Detecting Wormhole Attacks in Mobile Ad Hoc Networks Through Protocol Breaking and Packet Timing Analysis*, in Proceedings of IEEE Military Communications (MILCOM), Washington, DC, October 2006

The structure of this thesis is as follows. Chapter 2 presents the background for this thesis and summarizes the prior art in the wormhole attack detection research area. This chapter also describes the testbed experiment conducted as part of this thesis work. Chapter 3 describes the tools that were created in this thesis: a network traffic analyzer MANET suite and an NS-2 wormhole attack implementation. Chapter 4 describes the developed wormhole attack detection technique that allows detection of wormholes that drop traffic. Chapter 5 describes a novel wormhole attack detection approach based on frequency-space analysis of routing message timing. Finally, chapter 6 summarizes this thesis contributions and provides further work directions.

Chapter 2

Background, Literature Review, and Experimental Settings

This chapter provides background information about certain ad hoc networking protocols, traffic analyzers, and network simulators, as well as an ad hoc networking testbed that is used in this work. It also introduces the field of wormhole attack detection and prevention, giving a brief overview of the state of the art of wormhole attack studies.

Section 2.1 provides background information pertaining to this thesis. Section 2.2 describes the wormhole attack in detail and summarizes the published work that deals with wormhole attacks in ad hoc networks. Section 2.3 describes the experimental testbed network setup that is used throughout this thesis for the demonstration of wormhole attack detection results.

2.1 Background

As this thesis deals mostly with IEEE 802.11-based ad hoc networks that use OLSR as their routing protocol, this section describes both 802.11 and OLSR. Section 2.1.1 describes IEEE 802.11, and talks about 802.11 traffic captures and traffic analyzers. Section 2.1.2 describes OLSR. Finally, section 2.1.3 introduces a network simulator NS-2 that is used in this thesis.

2.1.1 IEEE 802.11: standard, traffic captures, and analyzers

A *de facto* standard for wireless ad hoc networks is IEEE 802.11 [7], the same wireless communication standard that is used in home wireless networks. IEEE 802.11 specifies physical-layer and MAC-layer behaviour of nodes.

There are three basic packet types in 802.11: data, control, and management packets. Data packets carry data from higher-layer protocols (such as, for example, ARP or TCP/IP), control packets assist in data delivery between stations, and management packets enable stations to maintain communications. 802.11 packets can be either broadcast or unicast (sent to a specific node).[1] IEEE 802.11, like other wireless protocols, uses collision-avoidance techniques for data transfer. Before transmitting, a node listens to the media. If the media is busy, a node waits for the media to become free, then waits a random time before transmitting. Also, 802.11 uses acknowledgements and packet retransmission mechanisms to ensure data delivery over unreliable wireless channels. For example, unicast data packets need to be acknowledged with a special control packet 'ACK'. If no acknowledgement is received after a certain time, a

[1] Of course, since the media used is wireless, both broadcast and unicast packets can be heard by all nodes in the vicinity of a transmitting station. However, if a packet is broadcast, it is recorded by all nodes that overhear it. A unicasted packet is heard by everyone, but is discarded by the nodes that it is not addressed to.

packet is retransmitted. 802.11 allows for encryption at the MAC level. Originally, 802.11 used Wired Equivalent Privacy (WEP) security mechanism based on RC-4 stream cipher [8]. Later on, the 802.11i standard specified a more advanced Wi-Fi Protected Access (WPA), based on AES [8] instead of RC-4.

In order to study network traffic, the traffic must be captured and decoded. This is done with a wireless sniffer. Since IEEE 802.11 is a popular protocol, a number of sniffers and analyzers capable of working with it have been developed over the years. Some wireless sniffers, in addition to traffic capturing capabilities, include sophisticated network traffic analysis modules. For example, Ethereal [9] and AiroPeek [10] can both parse wireless packets and perform a set of analysis functions on them. They are both quite sophisticated tools, capable of parsing a wide number of different protocols on different OSI stack levels. In addition, AiroPeek can, for example, decrypt network packets that are WEP-encrypted, and identify a number of known attacks on a network (ARP spoofing, for example). However, one major drawback of prominent network traffic analyzers is that they are not easily customizable: while it is easy to use the functions they already have built in, writing custom add-ons for exploratory data analysis with these tools is complicated.

2.1.2 MANET routing and OLSR

Routing in MANETs is an active research area. In general, MANET routing protocols fall into two categories: *proactive routing protocols* that rely on periodic transmission of routing updates, and *on-demand routing protocols* that search for routes only when necessary. Among on-demand routing protocols, AODV [11] is perhaps the most popular choice. This thesis deals mostly with proactive protocols, among which the most prominent is Optimized Link-State Routing (OLSR) [12] .

OLSR is an adaptation of the Open Shortest Path First (OSPF) protocol for wired networks [13]. In OLSR, certain network nodes are selected as multi-point relay nodes (MPRs) — these nodes are responsible for data forwarding, acting as network routers. Each network node periodically sends out a HELLO message in which it lists its neighbours, their 'quality', and specifies its own 'willingness' to become an MPR. OLSR HELLO messages are 'local', they are not supposed to be forwarded by other nodes. Based on the information contained in HELLO messages coming from a node's neighbours, a node selects an MPR set such that through these MPRs all of node's two-hop neighbours are reachable. Periodically, nodes that are selected as MPRs flood the network with Topology Control (TC) messages, in which an MPR node specifies what nodes have selected it to be their MPR. The TC messages are generated only by MPRs and are relayed by other network MPRs [14].

In OLSR, HELLO messages are, by default, sent out every 2 seconds, and TC messages are sent out every 5 seconds. In order to avoid collisions, jitter – a random delay – can be added to each of the periodic messages [12]. More detail on this routing protocol can be found in OLSR RFCs [12].

2.1.3 Ad Hoc Network Simulations and NS-2

Several network simulators are capable of simulating mobile ad hoc networks. The most commonly used ad hoc network simulators are NS-2 [15], OPNET [16], and, to a lesser extend, GloMoSim [17]. For this thesis, either OPNET or NS-2 were originally considered. After careful examination of both simulators, it was determined that OPNET is poorly suited for attack detection work, as OPNET seemed to be geared towards network performance studies and does not offer a clear and easy way to look at the network traffic packet-by-packet, and, within packets, field by field. Thus, NS-2 was chosen for the simulation component of

this thesis work.

NS-2 is a prominent discrete event simulator which is focused on modeling network protocols [18]. It is free, open-source software, available for a wide variety of platforms and contains implementations of numerous networking protocols on all stack layers. The greatest advantage of NS-2 over other simulators is that NS-2 is both free (while OPNET is very expensive) and immensely popular (GloMoSim and others are not as widely adopted).

Unfortunately, NS-2 also has a number of significant issues that make working with it challenging. First of all, NS-2 is somewhat difficult to get accustomed to. Its written in a mixture of C++ and OTcl (object-oriented Tcl) [19] which is not intuitive. Also, the wireless NS-2 package is an add-on, not fully integrated with the rest of NS-2. NS-2 does not scale well and is, in general, quite slow with substantial memory requirements. Finally, documentation on NS-2 often offers contradictory and/or dated information. Originally released in 1996, NS-2 has been significantly updated throughout the years and has had a wide number of components added to it, yet it is quickly becoming outdated. In the summer of 2006, a four-year project aimed at the creation of NS-3 was announced [20].

For more detailed information on NS-2, the reader is encouraged to visit the NS-2 website [15], and to consult the NS-2 lecture notes [19] and the NS-2 manual [21]. Excellent discussion on history of NS-2 and its current limitations is provided in [20].

2.2 Literature review: wormhole attacks and proposed solutions

Routing security in ad hoc networks is often equated with strong and feasible node authentication and lightweight cryptography. A wide variety of secure

extensions to existing routing protocols have been proposed over the years. However, the majority of these protocols are focused on using cryptographical solutions to prevent unauthorized nodes from creating seemingly valid packets [1]. Unfortunately, the wormhole attack cannot be defeated by cryptographical measures, as wormhole attackers do not create separate packets — they simply replay packets already existing on the network, which pass cryptographic checks.

Virtually all generalized secure extensions proposed for currently popular routing protocols do not alleviate wormhole attacks. However, since wormhole attacks pose such a severe threat to ad hoc network security, several researchers have worked on preventing or detecting wormhole attacks specifically. Their efforts are summarized in this section.

Section 2.2.1 describes wormhole attacks. Section 2.2.2 discusses a technique called 'packet leashes', which disallows packets traveling farther than radio transmission range (as they are likely to do in a wormhole attack). Section 2.2.3 talks about wormhole prevention methods that rely on round trip message time (RTT) to ensure nodes claiming to be located close together really are. Section 2.2.4 discusses the work of researchers who, instead of treating wormholes, treat the network disruptions wormholes introduce. These approaches — packet leashes, message travel times, and wormhole disruption — are general. Section 2.2.5 summarized other, more specialized wormhole detection and prevention techniques suitable for only particular kinds of networks and networks with particular protocols. Finally, section 2.2.6 summarizes and discusses the current state-of-the-art in the wormhole attack research area.

2.2.1 Wormhole attack

A wormhole attack is a particularly severe attack on MANET routing where two attackers, connected by a high-speed off-channel link, are strategically placed at

different ends of a network, as shown in Figure 2.1. These attackers then record the wireless data they overhear, forward it to each other, and replay the packets at the other end of the network. By replaying valid network messages from one part of the network, wormhole attackers can make far apart nodes believe they are immediate neighbours, and force all communications between affected nodes to go through them.

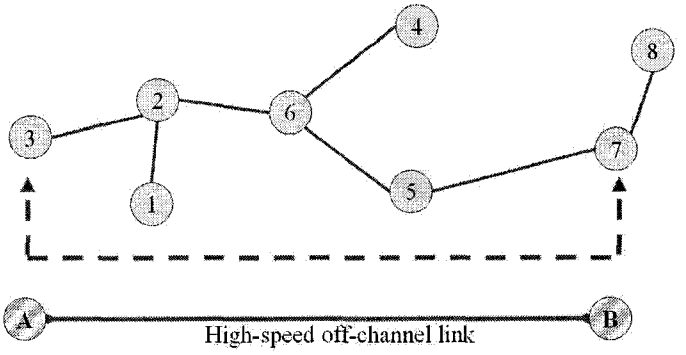


Figure 2.1: A network under a wormhole attack. Intruders A and B are connected by an off-channel link (e.g. a wired or satellite link), which they use to tunnel network data from one part of the network to the other. Without a wormhole, nodes 7 and 3 are 4 hops apart — their messages to each other should go through nodes 2, 6, and 5. When intruders A and B activate a wormhole, nodes 7 and 3 are able to directly overhear one another's messages, and are lead to believe they are immediate neighbours. Once this happens, all further communications between nodes 3 and 7 will go through the wormhole link introduced by A and B.

A wormhole attack is dangerous for both proactive and on-demand protocols [2]. When a proactive routing protocol such as OLSR [12] is used, ad hoc network nodes send periodic HELLO messages to each other indicating their participation in the network. In Figure 2.1, when node 3 sends a HELLO message, intruder A forwards it to the other end of the network and node 7 hears this HELLO message. Since 7 can hear a HELLO message from 3, it assumes itself and node 3 are direct neighbours. Thus, if 7 wants to forward

anything to 3, it will do so through the wormhole link, effectively giving the wormhole attackers full control of the communication link. If a network uses an on-demand routing protocol, such as AODV [11], the wormhole is just as effective. In on-demand protocols, when a node wants to communicate with another node, it floods its neighbours with requests, trying to determine the shortest path to the destination. In Figure 2.1, if 3 wants to communicate with 7, it sends out a request which a wormhole, once again, forwards without change to the other end of the network — directly to node 7. A request also propagates through the network via the normal channels, so 7 is lead to believe it has two possible routes to node 3: a 4-hop route through nodes 2, 6, and 5, and a single-hop direct link. Protocols will then select the shortest route, once again giving wormhole attackers full control of the link.

The majority of ad hoc routing protocols rely on the correctness of their neighbours' information for routing decisions, thus allowing wormhole-induced disruptions to have greater effects. For example, in the situation described in Figure 2.1, where nodes 3 and 7 think they are direct neighbours, nodes 5 and 8 will then think they are two hops away from node 3 (going through node 7), and will communicate with node 3 through the wormhole link as well.

Once the wormhole attackers have control of a link, they can do a number of things to actively disrupt the network. Attackers can drop the packets their link is meant to be forwarding. They can drop all packets, a random selection of packets, or specifically targeted packets.[2] Attackers can also forward packets out of order or 'switch' their link on and off.

It should, however, be noted that wormholes are dangerous by themselves, even if attackers are diligently forwarding all packets without any disruptions

[2]Two distinct situations are possible here. When no encryption is used, attackers know exactly what they are forwarding, and can target specific packets. When strong multi-layer encryption is used, attackers can either drop packets at random, or try to figure out (based on traffic patterns, packet sizes, etc.) what they are going to drop

— on some level, providing a communication service to the network. With wormholes in place, affected network nodes do not have an accurate picture of the network, which may disrupt localization-based schemes, lead to the decisions based on incorrect reasoning, etc. Wormholes can also be used to simply aggregate a large number of network packets for the purpose of traffic analysis for encryption compromise. Finally, a wormhole link is simply unreliable, as there is no way to predict what the attackers can do and when they will do it. Simply put, the wormholes are compromising network security whether they are actively disrupting routing or not.

Wormholes can be staged by insider nodes (those that are authorized to use the network), or by outsiders. An 'in-band' wormhole is a special kind of insider wormhole, where two nodes that belong to the network tunnel the packets to each other using the network's own multi-hop connections (that is, the attackers encapsulate packets and send them to each other over the network). This thesis focuses on outsider attacks, although the techniques developed in this thesis apply to the insider attacks as well. A special kind of a wormhole attack is a 'short-range' wormhole, which is formed when the wormhole attackers connect parts of the network that are distant in the network topology, but are physically located close together [5]. As will be seen later on, the methods developed in this thesis are able to detect such attacks where other methods fall short.

2.2.2 Packet leashes

Perhaps the most commonly cited wormhole prevention mechanism is 'packet leashes' by Hu *et al.* [4, 5]. Hu proposes to add a secure 'leash' containing timing and/or GPS information to each packet on a hop-by-hop basis. Based on the information contained in a packet leash, a node receiving the packet would be able to determine whether the packet has traveled a distance larger

than possible given the physical constraints.

Hu proposes two different kinds of leashes: *geographical leashes* and *temporal leashes*. Geographic leashes require each node to have access to up-to-date GPS information, and rely on loose (on the order of ms) clock synchronization. When geographical leashes are used, a node sending a packet appends to it the time the packet is sent t_s and its location p_s. A receiving node uses its own location p_r and the time it receives a packet t_r to determine the distance the packet could have traveled. Keeping in mind maximum possible node velocity v, clock synchronization error Δ, and possible GPS distance error σ, the distance between the sender and the receiver d_{sr} is upper-bounded by:

$$d_{sr} < ||p_s - p_r|| + 2v(t_r - t_s + \Delta) + \sigma \tag{2.1}$$

Geographical leashes cannot detect 'short-range' wormholes, but should function with other types of wormholes in situations when GPS coordinates are practical and available. However, modern GPS technology has significant limitations that should not be overlooked. While the price of GPS devices is going down, it remains substantial. Also, while, as Hu [4] specifies, it is possible to achieve GPS precision of about 3m with state-of-the-art GPS devices, consumer-level devices do not get (and do not require) this level of resolution. Finally, GPS systems are not versatile, as GPS devices do not function well inside buildings, under water, in the presence of strong magnetic radiation, they can be easily jammed, etc.

As opposed to geographical leashes, *temporal leashes* require much tighter clock synchronization (on the order of nanoseconds), but do not rely on GPS information. When temporal leashes are used, the sending node specifies the time it sends a packet t_s in a packet leash, and the receiving node uses its own packet reception time t_r for verification . In a slightly different variation of

temporal packet leashes, the sending node calculates an expiration time t_e after which a packet should not be accepted and puts that information in the leash. To prevent a packet from traveling farther than distance L, the expiration time is set to:

$$t_e = t_s + \frac{L}{c} - \Delta \tag{2.2}$$

where c is the speed of light and Δ is the maximum clock synchronization error.

The level of time synchronization required for temporal leashes entails the use of specialized hardware not currently practical in wireless ad hoc networks. In sensor networks, such levels of synchronization are impossible [4] at this time.

Wang [22] proposes an approach inspired by packet leashes [4], but based on end-to-end location information, rather than hop-by-hop leashes in [4]. Similar to geographic packet leashes, Wang's method requires each node to have access to up-to-date GPS information, and relies on loosely synchronized clocks. In Wang's approach, each node appends its location and time to a packet it is forwarding, and secures this information with an authentication code. The packet's destination node then verifies the nodes' coordinates (i.e. verifies that reported coordinates are within the communication range) and speeds. A minor disadvantage of this approach is that the end node is left to do all verification. Like geographical packet leashes proposed by Hu, this approach should work where GPS coordinates are appropriate.

2.2.3 Time-of-flight

Another set of wormhole prevention techniques, somewhat similar to temporal packet leashes [4], is based on the time of flight of individual packets. Wormhole attacks are possible because an attacker can make two far-apart nodes see themselves as neighbours. One possible way to prevent wormholes, as used by Capkun *et al.* [23], Hu *et al.* [24], Hong *et al.* [25], and Korkmaz [26], is to

measure round-trip travel time (RTT) of a message and its acknowledgement, estimate the distance between the nodes based on this travel time, and determine whether the calculated distance is within the maximum possible communication range.

The basis of all these approaches is the following. The RTT δ of a message in a wireless medium can, theoretically, be related to the distance d between nodes, assuming that the wireless signal travels with a speed of light c:

$$d = \frac{\delta * c}{2} \tag{2.3}$$

$$\delta = \frac{2d}{c} \tag{2.4}$$

The neighbour status of nodes is verified if d is within the radio transmission range R :

$$R > d \ (d \text{ within transmission range}) \Rightarrow$$

$$R > \frac{\delta * c}{2} \Rightarrow \tag{2.5}$$

$$\delta < \frac{2R}{c} \tag{2.6}$$

In essence, the use of RTT eliminates the need for tight clock synchronization required in temporal leashes: a node only uses its own clock to measure time. However, this approach, while accounting for message propagation, completely ignores message processing time. When a message is sent by one node and is acknowledged by another, the time it takes for a node to process a message and to reply to it is generally non-negligible, particularly in the context of bounding short distances using signals whose speed is similar to that of light in vacuum.

Outstanding clock precision and almost no capacity for error are required to bound distances on the order of hundreds of meters.

When a *de facto* standard of wireless ad hoc networks, the 802.11 MAC protocol [7] is used, such calculations are effectively impossible. 802.11 imposes a short wait time of $10\mu s$ (SIFS[3]) between the reception of a packet and sending of 802.11 acknowledgement. When 802.11 is used, transmission range R is generally about 300 meters. The speed of light c is $3 * 10^8$ m/second. Then, from equation 2.4:

$$\delta = \frac{2d}{c} = \frac{600\,m}{300,000,000\,m/s} = 0.000002s = 2 * 10^{-6} = 2\mu s \qquad (2.7)$$

Therefore, the RTT is an order of magnitude smaller than the delay required by the protocol. It is, of course, possible to account for this processing time by modifying formula 2.4 in the following manner:

$$\delta = \frac{2d}{c} + S \qquad (2.8)$$

where S is SIFS. However, note that wormhole attackers are not limited by the rules of the network, and could, with some ingenuity, send their packets without 802.11-imposed delay — thus breaking this type of defence altogether. On the other hand, if nodes were to use formula 2.4 directly, they would have to ignore 802.11-mandated delays, breaking 802.11 specifications. Hence, in order to use the approaches based on time of flight, special arrangements are required.

Capkun *et al.* [23] propose using specialized hardware that enables fast sending of one-bit challenge messages without CPU involvement so as to minimize all possible processing delays. To verify distance between the nodes, each node sends a one-bit challenge to the nodes it 'encounters', and waits for a response.

[3]This wait time is SIFS . The SIFS value depends on the version of 802.11 protocol. 802.11a specifies SIFS of $16\mu s$, 802.11b and 802.11g - $10\mu s$ [7]

A receiving node immediately sends a single-bit reply. While Capkun's use of specialized hardware is somewhat cumbersome, his method is nonetheless interesting. Hu [24] proposes a mechanism very similar to Capkun's [23], but does not use single-bit challenge approach. Instead, Hu relies on the round-trip travel time of full packets with CPU involvement, explicitly assuming medium access delays to be negligible. In addition, Hu's approach requires substantial processing of messages: upon the reception of a message, a node verifies the message correctness (i.e. performs one hash function operation) and sends an authenticated reply. Hong [25] uses a mechanism practically identical to Hu's.

Korkmaz [26] studied in detail the distance-bounding techniques described by other authors. He found that using round-trip time may lead to a high percentage of valid neighbours being rejected. He notes that although a wireless signal is akin to a light signal in vacuum, its speed is slower [26]. He also notes that even small errors in measuring time delays alter the distance measurements significantly, as was alluded to above.

Korkmaz proposes a modified statistical method based on RTT δ. He suggests using two different bounds for RTT, one based on the speed of light c ($boundC = \frac{2R}{c}$), and another based on experimentally determined speed of travel of the wireless signal s ($boundS = \frac{2R}{s}$). If the RTT δ is under $\frac{2R}{c}$, the nodes are considered neighbours. If δ is larger than $\frac{2R}{s}$, the nodes are considered non-neighbours. For the nodes with δ in between these bounds, Korkmaz suggests a probabilistic measure of 'neighbourness'. In addition, Korkmaz also proposes using received signal strength to verify the 'neighbourness' determined from the time of flight. In summary, Korkmaz's approach modifies and extends the RTT-based technique described above.

It is worth noting that while Capkun [23] proposes to use special hardware to drive the message processing time down, Hu [24] and Hong [25] simply assume

MAC delays to be negligible.

Approaches based on RTT of a packet are similar in nature to temporal packet leashes, [5], but do not require clock synchronization between nodes. The idea of these approaches is very simple: wireless nodes that claim to be neighbours should be physically close to each other, and when one node sends a packet to another, the answer should arrive very shortly, ideally within the amount of time a wireless signal would travel between the nodes. If there is a wormhole attacker involved, packets end up traveling farther, and thus cannot be returned within the time required.

2.2.4 'Effects-based' wormhole attack discovery

Several researchers have worked on the wormhole attack problem by treating a wormhole as a misbehaving link. In such approaches, a wormhole attack is not specifically identified. Rather, the wormhole's destructive behaviour is mitigated.

Baruch [27] and Chigan [28] use link rating schemes to prevent blackhole and wormhole attacks. They both rely on authenticated acknowledgements of data packets to rate links — if a link is dropping packets, the acknowledgements do not get through, the link is rated low and avoided in the future.

These approaches are geared towards discovery and prevention of only one kind of wormhole behaviour: packet loss. Wormholes can do much more than that — they can send packets out of order, confuse location-based schemes, or simply aggregate packets for traffic analysis. Even the distortion of topology information that a wormhole introduce can be a significant problem in certain networks. The real problem with wormholes is that unauthorized nodes (wormhole attackers) are able to transmit valid network messages. Techniques based on a link's performance may be suitable in certain cases, but they do not fully

address the wormhole problem.

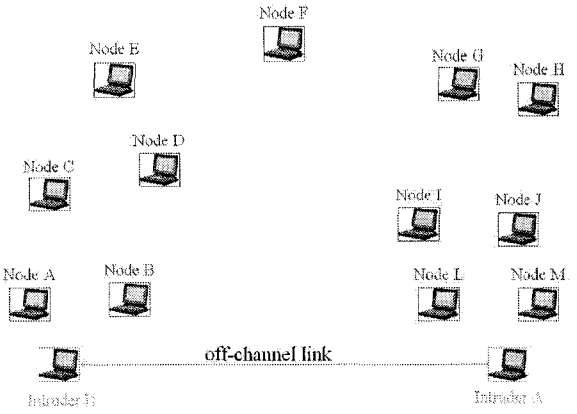


Figure 2.2: When a wormhole is treated as a misbehaving link, attackers are not detected and can create wormhole attacks targeting other nodes on the network. Detection only occurs after performance metrics are triggered, meaning there is latency in attack identification.

Consider the scenario shown in Figure 2.2. Say that, originally, intruders are creating a wormhole between nodes A and M. To the network, it seems that nodes A and M are direct neighbours, and the link between them is evaluated using a link rating system. When the rating system determines the link A-M to be lossy, it avoids it — which can be detected by the attackers. They can then simply move on: create a fake link between, say, nodes B and L, or even B and M. Since the methods proposed in [27] and [28] do not differentiate between poorly performing links and wormhole intruders, discovery of a bad link between A and M does not trigger a security investigation, and the attackers can thus indefinitely continue to disrupt the network.

2.2.5 Specialized techniques

A wide variety of wormhole attack mitigation techniques have been proposed for specific kinds of networks: sensor networks, static networks, networks running particular protocols, or networks where nodes use directional antennas. In this section, these techniques are described and their applicability to general MANETs is discussed.

Nodes with directional antennas

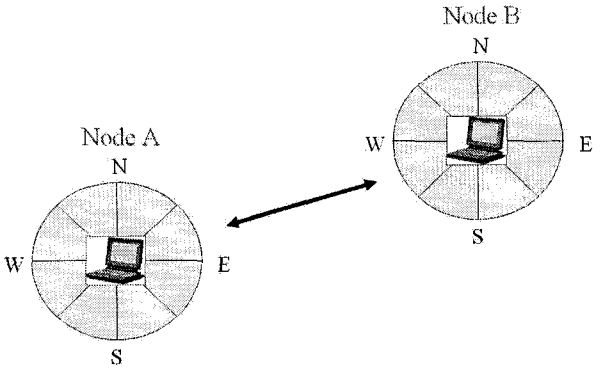


Figure 2.3: Nodes using directional antennas. When nodes A and B communicate, they send their messages on specific 'sectors': node A uses its North-East sector, node B - South-West. Therefore both nodes know how they are located with respect to each other. From knowing the sector on which it receives B's messages, A knows that B is located to the North-East. Had nodes A and B used omni-directional antennas, A would not be able to say anything at all about B's location.

Directional antennas have been extensively studied in the literature [29]. When directional antennas are used, nodes use specific 'sectors' of their antennas to communicate with each other, as shown in Figure 2.3. Therefore, a node receiving a message from its neighbour has some information about the location of that neighbour — it knows the relative orientation of the neighbour with respect to itself, as demonstrated in Figure 2.3. This extra bit of information

makes wormhole discovery much easier than in networks with exclusively omni-directional antennas.

In [29], Hu and Evans propose a solution to wormhole attacks for ad hoc networks in which all nodes are equipped with directional antennas. Wormholes introduce substantial inconsistencies in the network, and can easily be detected. In SERLOC [30], Lazos *et al.* use a slightly different approach: only a few nodes need to be equipped with directional antennas, but these nodes also have to be location-aware. These nodes then send out localization beacons, based on which regular network nodes determine their own relative location.

The methods proposed by Hu [29] and Lazos [30] are both promising, and could be easily applied to networks that use directional antennas. However, such methods do require directional antennas and would not be suitable for most MANETs.

Sensor networks: network visualization

Wang and Bhargava [31] introduce an approach in which network visualization is used for discovery of wormhole attacks in stationary sensor networks. In their approach, each sensor estimates the distance to its neighbours using the received signal strength. During the initial sensor deployment, all sensors send this distance information to the central controller, which calculates the network's physical topology based on individual sensor distance measurements. With no wormholes present, the network topology should be more or less flat, while a wormhole would be seen as a 'string' pulling different ends of the network together.

Wang's approach [31] has several aspects that may limit its applicability to general ad hoc networks. Wang assumes a dense sensor network of a polygon shape deployed on a flat surface — an assumption perhaps justified for sensor networks, but not practical for ad hoc networks. For sparsely located ad

hoc network nodes, the estimated physical topology may not be precise. Also, this method requires a central controller and is thus not readily suitable for decentralized networks. Finally, while this method should, theoretically, be extendable to mobile networks, Wang does not study how node mobility would affect the results.

Overall, Wang's method requires more research to be applicable to sparse, decentralized, or mobile ad hoc networks, but seems promising.

Sensor networks: use of location-aware guards

Lazos *et al.* [32] develop a 'graph-theoretical' approach to wormhole attack prevention based on the use of Location-Aware 'Guard' Nodes[4] (LAGNs).

In [32], Lazos uses 'local broadcast keys' — keys valid only between one-hop neighbours — to defy wormhole attackers: a message encrypted with a local key at one end of the network cannot be decrypted at another end. However, the establishment of such keys is non-trivial in the possible presence of wormholes. Lazos proposes to use hashed messages from LAGNs to detect wormholes during the key establishment. LAGNs are assumed to be trusted, and, since their location is known, a node can detect certain inconsistencies in messages from different LAGNs if a wormhole is present. Without a wormhole, a node should not be able to hear two LAGNs that are far from each other, and should not be able to hear the same message from one guard twice. Use of LAGNs, in essence, allows the nodes not equipped with GPS devices to perceive network irregularities a wormhole introduces, and to get some idea about their relative position in space.

Lazos's method [32] is elegant. However, it seems more suitable for dense stationary sensor networks than for mobile ad hoc networks.For example, LAGNs

[4]As explained above, Lazos also worked on a wormhole-resistant localization scheme for sensor networks [30], from which this wormhole attack prevention technique seems to directly follow

in this scheme are assumed to have longer communication range than regular network nodes — a good assumption for sensor networks (i.e. where sensor motes are regular nodes, laptops are LAGNs), but not usually available with mobile ad hoc networks. Also, the assumption of trusted LAGNs is better justified for sensor networks (where controllers can act as LAGNs) than for standard ad hoc networks. Nonetheless, Lazos' method is relatively lightweight and may hold promise for sensor networks and particular types of non-sensor ad hoc networks.[5]

Stationary networks: LiteWorp

Khalil *et al.* [33] propose a protocol for wormhole attack discovery in static networks they call LiteWorp. In LiteWorp, once deployed, nodes obtain full two-hop routing information from their neighbours. While in a standard ad hoc routing protocol nodes usually keep track of who their neighbours are, in LiteWorp they also know who the neighbours' neighbours are — they can take advantage of two-hop, rather than one-hop, neighbour information. This information can be exploited to detect wormhole attacks.[6]

After authentication, nodes do not accept messages from those they did not originally register as neighbours. Also, nodes observe their neighbours' behaviour to determine whether data packets are being properly forwarded by the neighbour — a so-called 'watchdog' approach. LiteWorp adds a novel wormhole-specific twist to the standard watchdog behaviour: nodes not only verify that all packets are forwarded properly, but also make sure that no node is sending packets it did not receive (as would be the case with a wormhole)

[5] While the need for specialized high-range location-aware 'guards' is probably limiting for emergency and tactical operations, it may be suitable for a mixed wired/wireless networks (i.e. office networks, rooftop, etc.) where stationary high-power wireless access points may serve as LAGNs

[6] To exploit this data fully, LiteWorp packets not only include 'sender' information, but also 'previous hop' information, rarely found in other routing protocols

LiteWorp, however, would not work in a scenario where node mobility is a factor. Since node's neighbours are determined and detected only once in LiteWorp and the packets from non-neighbouring nodes are rejected, no node movement is allowable. Therefore, LiteWorp is applicable to static networks only.[7]

Networks with on-demand multipath routing: a statistical analysis approach

Song *et al.* [34] approach the wormhole attack from a different angle. They propose a wormhole discovery mechanism based on statistical analysis of multipath routing, observing that a link created by a wormhole is very attractive in routing sense, and will be selected and requested (for routing) with unusually high frequency. This unusual route selection frequency can be statistically detected and used to identify wormhole links. Such a statistical analysis approach is fundamentally different from the majority of others where, in general, wormhole detection is related to locating a node in absolute or relative terms (based on network topology, time of packet transmission, GPS coordinates, with respect to GPS-aware nodes, etc).

Song's method requires neither special hardware nor any changes to existing routing protocols. In fact, it does not even require aggregation of any special information, as it only uses routing data already available to a node. These factors allow for easy integration of this method into intrusion detection systems.

However, Song's method is somewhat limited in scope as it applies only to routing protocols that are both on-demand and multipath. Non-multipath on-demand protocols do not provide enough information for the determination of link frequencies. While on-demand routing protocols keep complete information

[7]Note, that for purely static networks there is also a trivial solution to wormhole attacks: static routing

about routes they discover, proactive ones rely on next-hop information only, which does not allow the calculation of link frequencies. Nonetheless, within its scope Song's method is promising, and could be integrated in a real-world system with little effort.

2.2.6 Summary

Wormhole attacks, in which adversaries tunnel network data from one end of the network to another using an off-channel link, are a severe routing security concern in mobile wireless ad hoc networks. Wormhole attacks cannot be prevented by cryptographic measures as in a wormhole attack the attackers do not create any packets themselves, but simply forward the packets they hear coming from valid network nodes.

Several researchers use distance-bounding techniques to detect network packets that travel distances beyond radio range, thus preventing packets that have gone through the wormhole from being accepted. However, the majority of these techniques rely on specialized hardware and may not be practical. Of distance-bounding techniques, GPS-based ones are particularly interesting, as, of the specialized hardware proposed to combat wormhole attacks, GPS is perhaps the most general in purpose and most widely available. The effectiveness of GPS-based wormhole attack solutions is intuitively solid: a packet cannot travel to another end of the network undetected if all nodes know precisely where they are located and where their neighbours are (a 'short-range' wormhole is an exception to this; it cannot be detected by GPS-based measures). Unfortunately, GPS-based wormhole-combatting techniques inherit the limitations of GPS technology. They cannot be used where GPS does not work (underwater, inside buildings, etc.), or in small sensor networks(due to the resolution of GPS devices).

Some of the techniques created for particular network types appear promising. Network visualization techniques presented in [31] for dense sensor networks do not require special hardware and work well in specific instances. The technique based on anomalous frequency of route selection with the presence of wormholes [34] is intriguing. This technique, although it applies to multipath on-demand protocols only, is appealing because it is lightweight and, unlike many others, can be easily and immediately integrated into a MANET intrusion detection system (IDS). In essence, this technique is akin to those employed by IDS systems in wired networks (for example, on a wired network a port scan can be detected by observing high and abnormal rate of port requests) and could potentially be useful.

Overall, a significant amount of work has been done on solving wormhole attack problem. In itself, this body of work demonstrates the complexity of, and concern about, this type of attack. A standard solution is still lacking, although several promising solutions applicable to specific types of networks and situations have been described.

2.3 A MANET testbed and a wormhole experiment setup

A number of previous works in the area of wormhole attack discovery focus on analytical work and simulations. There is, however, much to learn from a physical implementation of a wormhole attack. The Communications Research Center Canada (CRC) [35] MANET lab has set up a MANET testbed, and has programmed a wormhole attack in it; the network traffic generated by their testbed wormhole attack implementation was collected in order to study wormhole behaviour. In this section, the experimental testbed created by the CRC

MANET lab is described. Then, the experimental setup that was used for the purpose of collecting the network traffic of a network under the wormhole attack is detailed.

The MANET testbed created by CRC MANET lab has several laptops physically located in the same room. In order to enable multihop communications, the nodes are programmed to believe they are far from each other. Node separation is simulated by generating artifical GPS coordinates for each node using the Mobile Network Emulator (MNE). The MNE includes a GPS generator and an MNE information exchange channel, as well as a location computing system. The MNE allows each node to propagate location information to other nodes through the MNE channel. At any given time, each node knows who is 'located' within its transmission range, and will only accept the packets coming from these nodes. This testbed setup allows testbed users to study multi-hop MANET networks without having to move the network nodes.

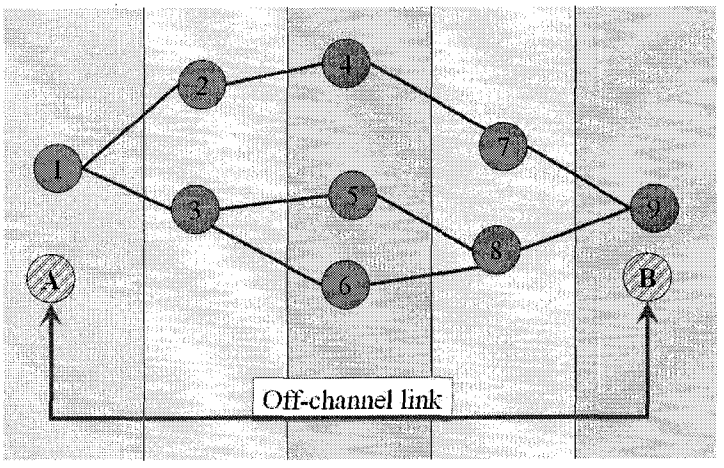


Figure 2.4: A testbed wormhole attack: experimental setup. Altogether, there were 9 valid nodes on the network and 2 intruders (A and B) connecting distant network parts. One of the intruders was programmed to randomly drop a portion of the traffic it was retransmitting.

The CRC MANET lab has also developed wormhole attackers. These wormhole attackers were introduced to the network, one at one end of the network and another at the opposite end, as shown in Figure 2.4. The attackers were fully functional: their actions and effectiveness in no way relied on them being in a testbed and they would have been just as efficient in a real-world deployment. Each one of them encapsulates the packets it receives wirelessly, transmits them over a wired link, and sends, wirelessly, the packets it receives on the wired side. One of the intruders (intruder A in Figure 2.4) simply forwards every packet it receives. The other attacker (intruder B in Figure 2.4), instead of forwarding everything it hears, drops packets at random. This wormhole behaviour is stealthy: by losing some — but not all — packets, a wormhole link acts like a link with poor connectivity, leading users to suspect connectivity issues rather than intrusions. The CRC MANET lab has set up a live video demo to demonstrate this effect. In their demo, a video from a node on one end of the network was streamed to a node on the other end of the network. When a wormhole gained control of a link that was used to transmit the video, the video started to break up as frames were lost. The effect produced by the wormhole was perceptibly the same as would have been produced by poor connectivity or server overload. If this had happened when a user was watching a live video off the internet, the user would not assume intrusion – the user would think there is an issue with their internet provider, or with the server sending the video.

Above, an experimental testbed and the implemented wormhole attackers are described. Here, the actual network setup that was enabled when the network traffic was collected is described. For the purpose of traffic collection, two experiments under the same conditions were conducted, one with a wormhole and one without. When the wormhole was introduced on the network, the routing tables of network nodes were affected in the predicted way: nodes,

located at the opposite ends of the wormhole, believed themselves to be direct single-hop neighbours. The network setup was as follows:

- *Network Topology*: the network consisted of 9 laptop nodes, arranged (with artificial GPS coordinates) into the formation shown in Figure 2.4. As shown, the maximum number of hops on the network was 4. Two wormhole attackers, connected by a wired link, were introduced on the network, at the positions shown in Figure 2.4

- *Routing*: As a routing protocol, OLSR [12] was used. Default OLSR parameters were not used. HELLO message interval was ~0.315 seconds. A simplified version of HELLO messages jitter was used: a HELLO message was sent out either after 0.3 seconds, or after 0.33 seconds. The Topology Control (TC) message interval was set to 1 second.

- *Network load*: network traffic consisted of OLSR management traffic and several nodes pinging each other. However, since all nodes were in the same room (used the same wireless channel) and the intervals between OLSR messages were low, the wireless channel was relatively busy.

- *MAC address spoofing* Not enabled. Intruders did not try to hide by spoofing their MAC addresses. [8]

The traffic, generated by this experiment, was captured with a wireless sniffer AiroPeek [10]. Since all nodes were physically located close to each other in this experiment, it was possible to sniff and record all network traffic at the same time.

[8] Subsequent analysis of this data, however, does not rely on having 'visible' intruders MAC addresses, and is viable even if the wormhole attackers do spoof theirs.

Chapter 3

Tool development: a traffic analyzer and an NS-2 wormhole attack simulation

In order to do research quickly and efficiently, it is important to have the right tools. This chapter describes the research 'tools', a traffic analyzer suite and a simulation, that were developed in this thesis.

Section 3.1 describes the network traffic analyzer suite that will allow researchers to work with ad hoc network traffic without having to implement low-level analyzer capabilities. This section also demonstrates the capabilities of the developed suite by presenting a small case study on trivial wormhole attack detection techniques completed using the Network Traffic Analyzer (NTA).

This chapter also describes another tool developed in this thesis, an NS-2 wormhole attack simulation. Although a number of researchers work with wormhole attacks, no literature on realistic wormhole attack simulations exists.

Section 3.2 describes the schematics of the created wormhole attack implementation and comments on whether this wormhole attack model is realistic.

3.1 Wireless traffic analysis: NTA

This section presents an NTA-MANET traffic analyzer suite created for working with wireless ad hoc network traffic. First, this section introduces the MATLAB-based NTA tool developed at Defence Research and Development Canada (DRDC) [36]. The NTA-MANET suite that was contributed to the NTA as part of this thesis is then described. Finally, a small case study on a trivial wormhole attack detection technique is discussed to demonstrate what the NTA-MANET suite is capable of.

Network Traffic Analyser (NTA) is a MATLAB-based tool developed in-house by the Network Information Operations (NIO) section of DRDC for statistical analysis of network traffic. Unfortunately, commercially available traffic analyzers are not easily customizeable. While it is easy to use the functions these analyzers have built into them, writing custom add-ons for exploratory data analysis with these tools is complicated. NTA, on the other hand, was created specifically to be flexible. NTA was originally created for wired networks. Within this work, additional functionality was implemented in the NTA, allowing NTA to work with wireless (IEEE 802.11) ad hoc network traffic.

3.1.1 NTA: overview

NTA works with the data captures obtained by a sniffer. The captured data then needs to be stored in libpcap format [37] — a special binary file format commonly used for packet manipulations and analysis.[1] The capturing, storing,

[1]The same binary format is used by the well-known packet-capturing tools tcpdump and Ethereal, just to name a few.

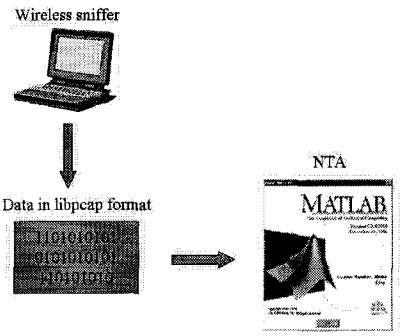


Figure 3.1: Pre-processing of data for NTA. First, wireless data is captured with a wireless sniffer. Next, the captured data is converted to a binary libpcap format. The data in libpcap format can be loaded directly to NTA, where it is parsed and dissected. Some wireless sniffers are able to store the captured data directly in libpcap format; for other sniffers conversion tools are available.

and uploading of data to NTA is demonstrated schematically in Figure 3.1. Although not all wireless sniffers store the data in libpcap format directly (some of them use their own proprietary formats), a number of conversion tools are freely available. When the data is uploaded to NTA, each packet in the capture file is identified, its different headers are 'peeled off', and each of the headers (MAC, IP, etc) is dissected into their components (addresses, flags, etc). NTA fully decodes Medium Access Control (MAC), Logical Link Control (LLC), and Internet Protocol (IP) headers. It also recognizes and dissects UDP, TCP, and ICMP headers. Within the NTA, network packets are represented as simple MATLAB arrays, where each packet occupies a separate row, and each identified packet field has a dedicated column. As a result, once the data is in NTA, working with it becomes very easy.

Once protocol parsing is implemented in NTA, working with that protocol's data is simple. However, implementing parsing of a new networking protocol in NTA requires a great deal of development time. Figure 3.2 shows schematically

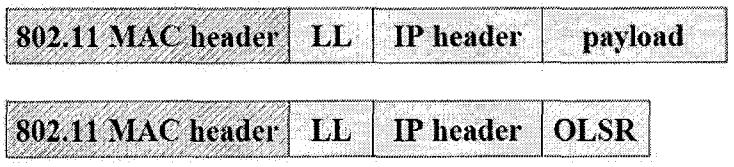


Figure 3.2: A schematic view of 802.11 packets' headers, a non-OLSR packet (top) and an OLSR packet (bottom). Within the NTA-MANET suite, parsing of 802.11 headers and parsing of OLSR data were implemented.

the structure of wireless packets, a non-OLSR packet (top) and an OLSR packet (bottom). Prior to this work, NTA could parse IP headers, but could not deal with 802.11 headers and treated OLSR data as payload. Within the scope of this work, IEEE 802.11 parsing and processing within NTA were implemented. Also, much like other traffic analyzers, the basic version of NTA did not have any MANET functionality, so several MANET-related functions were implemented within NTA as well. Finally, OLSR parsing within NTA was also implemented. Together, this added NTA functionality is named the 'NTA-MANET' suite. The capabilities of NTA-MANET are described in the next section.

3.1.2 NTA MANET suite

NTA-MANET's basic functionality that was developed includes the following:

- *Uploading wireless traffic to NTA.* Previously, NTA depended on the capture being done by a wired sniffer. This dependency was eliminated, allowing users to work with wireless data the same way they work with wired data.

- *Complete parsing of 802.11-related parts of packets, including 802.11 headers of different packets, and full parsing of 802.11 management and control packets*

- *Parsing of OLSR packets*: this MANET-specific functionality gives NTA users easy access to the data contained in OLSR routing packets.

- *Working with MAC-addresses of stations: filtering, selection, following, etc* In the analysis of wired traces, MAC addresses of individual nodes sometimes tend to be ignored. In the wired world, the IP address of a packet is often all that is necessary. In the wireless world, where network nodes act as routers, this is frequently not the case.

Given the above functionality, the tool can now be used to easily write custom functions. With it, a researcher wanting to, for example, study the behaviour of a particular network access point, does not have to spend time on writing a parser for 802.11 beacon packets. This researcher would simply upload their experimental data to NTA and with that will have an easy access to all data that 802.11 beacons contain, they would be able to obtain statistics on any field, filter on any field parameter or a combination of parameters, etc.

The main value of NTA is the ease with which NTA can be extended for a particular need of a particular user. However, within the scope of this work several more specific functions that could be of interest to other NTA users were developed as well. Within NTA, novel functions were developed which allow users to:

- *List stations and networks present in the trace; list stations belonging to a particular network.* When turning on a wireless card in a city, one often sees a number of different networks. The nodes belonging to another, non-related network, may be of no interest to a researcher. Thus a simple way to identify which nodes belong to which network is a good addition. Note that NTA is designed in such a way that creating functions like this is trivial.

- *Trace particular packets on a network as they travel from node to node* With this function, it is possible to see the paths that data travels on a network. There are many applications of this functionality. For example, it can be determined which nodes are the most important for the network, or whether a blackhole attack is happening, etc. It is interesting to note that the same information could have been obtained if each node was to keep track of the packets it was processing. With this function, however, the information from the nodes is not needed and node cooperation is not required – the information can be obtained directly from traffic. One can imagine a passive intruder listening in on the network – this intruder would be working with precisely the same information, with just the wireless network traffic. This function could help, for example, to study what an attacker can find out about a network by simply listening in on network's traffic.

- *Print the summary of each wireless station's behaviour, i.e. show if it is an access point, what nodes it communicated with, etc.; print the total summary of the wireless trace (how many packets were captured, what networks were observed, etc).* These summary functions present an easy way for a person to get started when working with a particular trace, and provide a good brief description of a trace.

- *Identifying packets that are repeated.* This function could be used for detection of replay attacks. It had been developed for detection of wormhole attacks through detection packet repetitions, within the case study that is discussed in the next section.

3.1.3 'Trivial' detection of wormhole attack with NTA: packet repetitions

Having explained the NTA functionality that was developed, it is now possible to demonstrate how NTA can be used for discovery of wormhole attacks in network traffic. Here, a trivial way of attack discovery is shown — using packet repetitions — which would not be easily implemented in a real distributed network. In this section, the traffic obtained in the experiment described in section 2.3 is processed, and wormhole attackers in this experiment are detected.

A wormhole is hard to discover because wireless nodes cannot directly listen to the off-channel communications between the two intruders. Each network node only has access to wireless traffic, and only to a small local subset of it. With a single global wireless sniffer that was present in the experiment used in this work (see section 2.3), access to all wireless traffic on the network is provided. With a global sniffer (or a carefully selected set of distributed sniffers), a wormhole can be detected by looking at packet repetitions in the wireless trace.

To see how packet repetitions would be useful for wormhole discovery, consider the network under a wormhole attack shown in Figure 2.4. Imagine that station 1 is trying to communicate with station 9. Since a wormhole is present in this network, station 1 believes that it has a direct link to station 9. Let us say that station 1 is sending traffic to station 9. Intruder A listens to the traffic, and forwards the packet sent by station 1 to intruder B. Intruder B retransmits the packet that station 1 has sent. Therefore, before reaching station 9, the packet from station 1 is actually sent three times: twice wirelessly, and once on the wired link between intruders. While it is not possible to see what the intruders send to each other, it is easy to see, having access to all network traffic, that the exact same packet is resent twice on the wireless medium. By

detecting packet repetitions, a wormhole attack can thus be detected.

It should be mentioned that there are cases when packets are validly repeated on a network. For example, packets that are legitimately forwarded by neighbour nodes are 'repetitions' as well, only the packets are not *identical* as their hop counts are different. A legitimate repetition also happens when a packet is retransmitted on the 802.11 MAC layer. In this case, a packet is marked as a 'retry' in accordance with the 802.11 rules. Therefore, by assessing packet repetitions in a network trace, a wormhole attack can be detected. If the intruder's MAC address is not spoofed, a large number of packets being re-sent with a different source MAC address would appear in the trace. If the intruder's MAC address is spoofed, it will appear as if a station repeats its own packets (even those that were acknowledged or those that are broadcasted).

To detect these packet repetitions in the experimental data capture, an NTA function that compares captured packets with each other was created. Since the MAC address of the intruder could be spoofed, this function not only looks for different stations repeating each other's packets, but also for stations that seem to be repeating their own packets.

In general, packet P_2 is considered to be the repetition of packet P_1 if all of the following criteria are met:

- *Packet P_2 was recorded by a sniffer within a certain relatively short time after packet P_1.* Wormhole attackers should be capable of a fast transmission of data, so the time between repetitions should be relatively small.
- *P_1 and P_2 are both 802.11 data packets.* Only data packets are transmitted through the wormhole. 802.11 management and control packets are not repeated by a wormhole.
- *The 802.11 payloads of P_1 and P_2 are equal* — that is, the actual message of the packet is the same. Within 802.11 payloads, there are IP headers,

hop counts, etc. Note that this equality does not depend on the presence of encryption: it is not necessary to identify what is within a packet to be able to compare it to another packet. If no encryption is in place, the comparison can be simplified and accelerated. For example, one can start with comparing IP attributes (for example, addresses and hop counts), and compare the complete payload only if IP attributes of packets match.

If an intruder is spoofing a MAC address of a valid station, it could be troublesome to distinguish the packets that are validly repeated because of poor connectivity and those maliciously repeated. Let us say that a packet P_1 is of the type that requires 802.11 acknowledgement (it is not a broadcast packet). When the connectivity is poor, the packet P_1 may end up being resent, and that could be confused with an actual malicious packet repetitions. There are two basic rules that intruders *may* break in this case. First, when a valid packet is repeated because of poor connectivity, the 'retry' bit on the first packet should be '0', and on the second and all consecutive packets — '1'. When P_2, following a P_1 with a retry bit set to 0, is sent the retry flag set to '0', P_2 is improper and suspicious. However, packets on the MAC-level are often repeated more than once (up to the limit specified by the wireless card), and in the 802.11 standard the packets sent for the second, third, and so on times are identical, with their retry bits set to one. From this follows the second rule: if P_1's retry bit is 1, and P_2's is zero, P_2 is improper. But when if, for example, P_1 and P_2 both have the retry bit of 1, it cannot be determined whether its a valid repetition due to poor network connectivity, or a malicious intruder spoofing another node's MAC address. This property has the potential to be the source of false positives in attack detection. In this case, secondary checks are needed to find out whether an intrusion is in place. A valid node would, for example, have a smaller number of packet repetitions. Note that this point is a concern

only for the packets that need to be acknowledged. If OLSR is used as a routing protocol, in order for wormhole attackers to create a link, they would have to repeat broadcasted packets, which should not be repeated.

The rules for packet replays, specified above, were implemented within the NTA-MANET suite. Since the NTA-MANET suite described above allows to have direct access to all fields of 802.11 packets, implementing these rules was straightforward.

Sending node (S) MAC address	Replaying node (R) MAC address	# of replays by R
06:25:A7:03:40	06:25:A7:03:40	**785**
06:25:17:D6:1F	06:25:17:D5:BF	**391**
06:25:17:D6:1F	06:25:17:D6:1F	**4**

Table 3.1: Packet repetitions detected by NTA. The first column shows the MAC addresses of nodes that had their packets repeated. The next column shows the MAC addresses of the nodes that repeated packets. The last column shows how many packets were repeated. From this table, both wormhole attackers in this network and the nodes affected by the wormhole are identified.

The experimentally obtained data was then processed to determine how many packets are detected as 'repetitions' by this method. The replay statistics, obtained with NTA from the MANET experiment with a wormhole are shown in Table 3.1. This table was generated by detecting packet replays (i.e. packets P_1 and P_2 in the discussion above), and obtaining statistics on the characteristics of these packets: what stations were doing the repetitions, whose data they were repeating, etc. The first column of Table 3.1 shows the stations S whose packets were replayed. The second column shows the stations R that were doing the replaying. The third column shows how many packets were identified as replays for each S, R pair. From this result, both intruders in the experiment can be identified: the first intruder is a station with a MAC address 06:25:29:C6:CD, and the second intruder is a station with a MAC address 06:25:17:D5:BF. Here, the nodes had their packets replayed by these intruders are also identified.

The third line in Table 3.1 demonstrates a false-positive result that appears for the reasons discussed previously. This line shows that station with a MAC address 06:25:17:D6:1F 'replayed' four of its own packets. This is clearly a false-positive result since it is questionable whether wormhole attackers would get any benefit by replaying only four packets. Note that the intruder, shown on line 2 of Table 3.1, replayed almost 400 of this station's packets, two orders of magnitude more than the self-repetition case shown on the third line. No other false-positive results were obtained in this experiment.

3.1.4 Discussion

In this section, a Network Traffic Analyzer (NTA) MANET suite that was created in this thesis work has been described. This traffic analyzer suite allows researchers to easily work with MANET data, without having to 're-invent the wheel' by writing low-level 802.11-related and ad-hoc-network-related traffic analyzer functionality. This section also presented a case study on the detection of a wormhole attack using packet repetitions. It was shown that if a global wireless sniffer is present on a network, a wormhole can be detected easily by accessing packet repetitions.

Applicability of 'packet-repetitions' method to real-world networks

While this case study was performed to demonstrate NTA's capabilities, it is possible to comment on the applicability of packet repetitions to wormhole attack detection in general.

When a global sniffer is present on a network (for example, a powerful network controller is overseeing a sensor network), detection of a wormhole attack with this technique is very simple, if not to say trivial. For a case where a global sniffer or a network controller is present, this technique can be implemented on

the fly. Unfortunately, an implementation of such technique on a traditional fully distributed network without a global network monitor would be challenging. In order to do so, it will be necessary, for example, to use specialized agents that would carry hashes of network messages and compare them node-to-node. This approach would also require tight time synchronization among network nodes and would lead to an increase in network overhead. So, while the packet-repetition technique does not appear practical for networks without a global wireless sniffer, it is a solid technique that can be easily implemented if a global sniffer is available. Note that in the majority of ad hoc networks it is not feasible to have a global sniffer, so this technique is applicable only in a limited number of deployment scenarios.

3.2 Wormhole implementation with NS-2

In the previous section, the tools that were created in order to analyze the traffic generated by physical experiments are described. In this section, another 'tool' — an implementation of a wormhole attack in a network simulator NS-2 — is described.

While the importance of experimental work cannot be stressed strongly enough, a reliable wormhole attack simulation is also needed in order to have an extendable, easily customizable model that would be able to generate traffic of different networks in different conditions under wormhole attacks. With a simulator, it is easy to alter various wormhole parameters and network settings — something that may not be easily done in an experimental testbed. While a number of researchers have studied wormhole attacks, many of them did not develop wormhole attack implementations [5, 22, 26], and the researchers that did develop wormhole attack implementations do not provide any details (neither schematics nor the code) on how they have simulated wormhole attacks.

Khalil [33] mentions that their wormhole attack implementation works by making packets 'heard' on one end of the network immediately available on the other end of the network. This model grossly oversimplifies real wormhole attackers' activities, and, in fact, could not be used at all for working with intrusion detection techniques such as those developed later on in this thesis (chapter 5).

In this thesis, a realistic wormhole attack in the NS-2 was developed and tested. Section 3.2.1 introduces NS-2 and discusses the challenges that one is faced with when trying to create a wormhole attack in this simulator. Section 3.2.2 describes the model of a wormhole attack that was created in NS-2. Appendix A shows, through the demonstration of routing changes and by tracing packets going through the wormhole that this wormhole attack schematics is effective.

3.2.1 An overview of NS-2

The overall architecture of NS-2 is similar to the OSI protocol stack network model [18]. NS-2 has a wide variety of different protocols built into it on all levels of the OSI protocol stack, starting from the physical layer (where it has built-in models of wired, wireless, and satellite media), and going up to the application layer (where it can simulate, for example, FTP, telnet, or constant bit-rate traffic) [38]. NS-2 has two distinct levels — an OTcl level and a C++ level. On the C++ level, NS-2 components — links, agents, address classifiers — are defined and on the scripting level (OTcl level) these components are put together (as to avoid recompiling every time a structural change is made [20]).

In NS-2, a simulated topology consists of nodes, connected by links (if a wired network is being simulated). In order to simulate traffic, NS-2 uses 'agents' and 'applications'. For example, to get UDP traffic going between two nodes, the node generating messages must have a UDP agent attached to it, while the node

receiving traffic should have a NULL agent. On top of these agents, an application — for example, a generator of constant bit rate traffic — is attached. The nodes have to be connected by links to each other, either directly as immediate neighbours, or through intermediate nodes that route traffic accordingly. If wireless networking is used, instead of relying on wired links, nodes use shared media to send packets to each other. In order for a wireless node to route packets for its neighbours, it has to have a routing agent running on it. The difference between wired and wireless nodes in NS-2 is highlighted below.

Node structure in NS-2

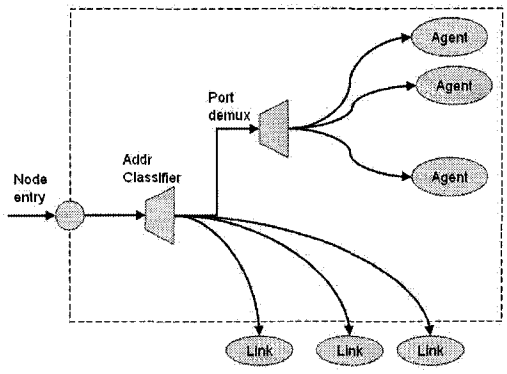


Figure 3.3: An NS-2 wired node. Packets, received by a node, are handled by the node's entry point — the address classifier. The address classifier determines whether a packet is destined to the node itself (in which case it is routed to the port demultiplexer), or to other nodes (it is then sent on to the corresponding wired link). The port demultiplexer, in turn, determines which agent should be receiving this packet. When node's agents themselves generate a packet, the packet, too, is sent to the node's entry point, where it is routed accordingly.

Wireless and wired nodes have different structures in NS-2. The structure of a wired node is shown in Figure 3.3, and the structure of a wireless node is shown in Figure 3.4.[2]

[2]Figures 3.3 and 3.4 are taken directly from the NS-2 Manual [21].

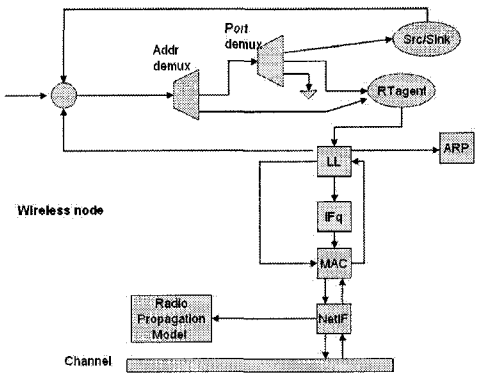


Figure 3.4: An NS-2 wireless node. In the NS-2 wired domain, the connectivity is determined by wired links. In wireless NS-2 world, there is a concept of a shared wireless channel, which the nodes use to communicate. When a packet is delivered to a node on the channel, it is handled by the MAC and LL layers, and is then sent up to the node entry point — the IP address demultiplexer. If a packet is destined to the node itself, it is sent to the port demultiplexer. Otherwise, it is handed down to the routing agent, which determines where to send it next. All packets that are placed on the wireless channel go through routing agent, link layer, and MAC components.

A wired NS-2 node, shown in Figure 3.3, is relatively simple. A packet enters a node through the 'entry point', to which an IP-level packet address classifier is attached. Based on the packet's IP address, the packet is either sent to one of the wired links or, if a packet is destined to the node itself, to the port demultiplexer which will determine what application agent a packet is supposed to go to. It is important to note that packets a node itself generates also go to the node's entry point and are sorted from there.

An NS-2 wireless node, shown in Figure 3.4, has a more complex structure. While wired nodes rely on links to communicate, wireless nodes use media sensing, and thus have a number of additional components providing for media access control. For a wireless node, when a packet 'arrives' to the node's location (functionality handled by the NS-2 'channel' component), it is treated by the node's MAC layer (whose functionality is, for 802.11 networks, almost identical

to the actual functionality of 802.11 MAC — looking at packet's integrity, determining whether the packet should travel up the stack, checking its flags, etc.) and Link Layer (LL), and then passed on to the node's entry point. Similar to the wired node, the wireless node's entry point is the IP address demultiplexer. Depending on the packet's IP address, the address classifier sends the packets to the port demultiplexer (if the packets are addressed to the node itself) or to the routing agent, which determines what to do with other packets.

Note that for a wired node the address classifier is used to determine where to send the packet next — a functionality performed by the NS-2 routing agent in the wireless domain. However, while the NS-2 routing agents do the packet classification (among other things), they are much more than simple packet classifiers. In essence, the routing agents perform the full functionality of real-world ad hoc routing daemons. For example, an OLSR routing agent creates and sends out routing messages, processes the routing messages received from other nodes, and maintains an IP table based on this information — i.e, it performs the full functionality described in the OLSR RFCs [12]. In NS-2, this routing agent is also involved in routing data packets while in an actual network the routing table generated by the OLSR routing daemon would be used for this, but the daemon would not be involved in the actual routing.

Issues: need for multiple interfaces and wired/wireless domain combination

A particular weakness of NS-2 pertaining to wormhole attack modeling is its inability to work with multiple interfaces [21]. Note, for one thing, that there is no 'interface' classifier in NS-2 — an IP-level address classifier handles everything a node receives. The very design of the wired and wireless nodes makes it impossible to have several interfaces: an NS-2 node is designed to have one single entry point; the presence of such a singular entry point is fundamental

and would be a challenge to change. Unfortunately, a proper wormhole attacker needs to have 3 interfaces (two wireless and one wired) — not something NS-2 allows.

Another issue that makes modeling of wormhole attacks non-trivial is the need to mix wired and wireless domains in a single simulation. In NS-2, the wireless block is an add-on, and it is not fully compatible with the wired components. When combining wired and wireless nodes in a single simulation, all kinds of issues are encountered, including problems with tracing (trace formats are different) and visualization (on the wired side, the nodes are placed in a visualization according to certain connectivity rules, while in the wireless world the location is predetermined — the mixture of both leads to node misplacement in visualization). Note that there is a significant difference in routing between the wired and wireless domains. In the wired world, the routing is done with links (indeed, an address classifier actually directs packets to the right links), while in the wireless domain the packets are routed with the help of routing agents and no links are present. This makes a straightforward combination of wired links and wireless media access impossible. Overall, using wired links on a wireless network, as a wormhole is supposed to do, is challenging in NS-2.

3.2.2 An NS-2 model of a wormhole attack

In this work, a wormhole attack is simulated with NS-2. A wormhole should capture packets on one end of a network and forward them on the other. Ideally, each of the wormhole attackers should have two wireless interfaces (one for sending packets and one for recording) and a wired one (for transferring packets to each other). Unfortunately, due to NS's inability to work with multiple interfaces (as described in the previous section), creation of such attackers in NS-2 is not feasible. In this section, a work-around to this problem is described, and

the schematics of a wormhole attack implementation with NS-2 are presented.

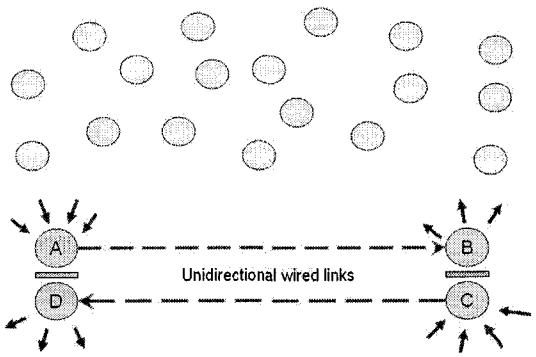


Figure 3.5: Schematics of the wormhole attack implementation in NS-2. Each end of a wormhole has two components: a sink that records all it hears (A and C on the diagram) and a source that replays everything it receives on the wired side (B and D) wirelessly. Sinks and sources are connected by unidirectional links. For example, sink A records everything it hears wirelessly and transfers it over a unidirectional wired link to source B. Source B, in turn, replays wirelessly all packets it receives from sink A. Along with a sink and a source, each end of the wormhole includes a firewall to prevent infinite loops arising from sinks replaying packets coming from sources.

The overall scheme of the wormhole attack implementation in NS-2 is shown in Figure 3.5. Since it was not possible to create a single intruder node, two specialized types of nodes were instead created, such that, when combined together, they form a single wormhole intruder. These two node types are a sink node which captures wireless traffic and sends it on the wired link (nodes A and C in Figure 3.5) and a source node (nodes B and D in Figure 3.5) that puts everything it receives on a wired channel on the wireless media.

By separating an intruder like this — into a sink and a source — two intruder components, both unidirectional, are created. This makes it possible to avoid the most significant issue arising from the mixture of wired and wireless domains — the inability to classify packets by interfaces. The structure of the created sink and source nodes is simple enough to avoid the need for complex

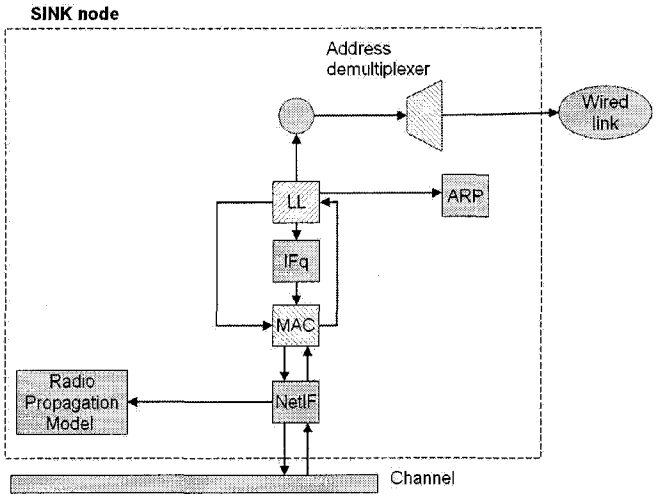


Figure 3.6: A wormhole sink node. A sink node is a simplified version of a general NS-2 wireless node. Striped parts are those that were changed to create a sink: the medium access component, the link layer, and the address demultiplexer. On the MAC layer, the SINK is programmed to accept and reject particular packets. On the LL layer, the changes are introduced to make sure all packets, even those that are supposed to be handled by the ARP layer, go up the protocol stack. The address demultiplexer is programmed to forward everything it receives up to the single wired link, regardless of IP addresses.

packet classification: since sinks and sources are unidirectional, the classification is simply 'what comes in one way goes out the other way'. Corresponding source and sink nodes should not be able to hear each other (as to not infinitely replay the information back and forth) – the capability shown as a firewall in Figure 3.5. Finally, unidirectional wired links are used to connect corresponding sink and source pairs together.[3] Both sink and source nodes, whose roles in a wormhole are explained above, are modified NS-2 wireless nodes. A schematic representation of a sink node is shown in Figure 3.6 while the scheme of a source node is given in Figure 3.7. Essentially, both sink and source nodes are gen-

[3]The implications of using this attack model and its limitations in modeling an actual attack are discussed in section 3.2.3.

eral wireless nodes with a number of components not activated and a number of components substantially modified. Those that were modified are shown in stripes in Figures 3.6 and 3.7.

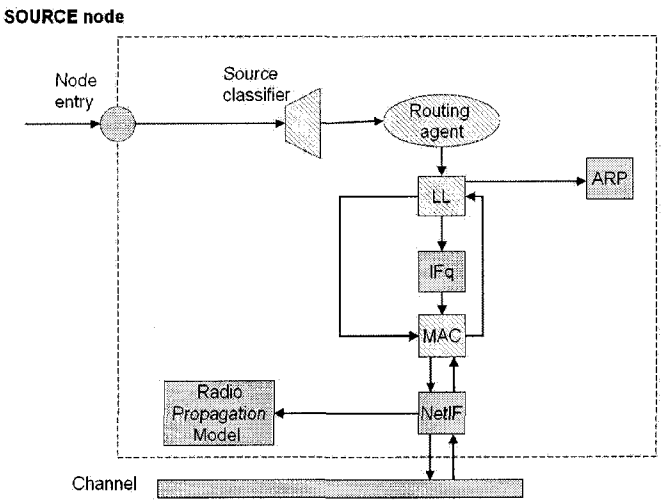


Figure 3.7: A wormhole source node. A source node takes everything it receives on the wire and sends it on the wired medium. To create a source node, the 'wireless' node is simplified, and the following components are altered: the address classifier(so that it forwards everything to the routing agent), the routing agent, and the lower wireless media-related stack layers. At the MAC layer, a source node should ignore everything it hears.

As shown in Figure 3.6, the following wireless node modules were reprogrammed to create a sink node: the MAC module, the LL module, and the address demultiplexer.

At the MAC layer, a wormhole sink should be similar to a real-world wireless card operating in promiscuous mode — listening and recording all packets it hears, but not transmitting anything. However, unlike a wireless card in promiscuous mode, the sink node should perform some filtering on the packets it captures as well since some packets should not be resent on the other end of the network. For instance, packets coming from the corresponding source node

have to be rejected to avoid infinite packet resending loops — the functionality shown as a firewall in Figure 3.5 and implemented as a MAC-address filter in the MAC layer of NS-2 sink node.

On the link layer (LL) level, a standard non-wormhole wireless node determines whether the ARP module is to be accessed so that MAC-to-IP mapping can be done. However, the intruders should not use ARP. Instead, ARP packets should be transmitted on the wormhole link. Thus, the LL module for the sink node is modified to make sure that ARP-related packets are not transferred to the ARP layer. For the sink node, the LL layer is programmed to send everything, even ARP packets, up the protocol stack.

Finally, to create a sink node, the address demultiplexer of a wireless node is modified. On the level of address demultiplexing, the sink node is very simple: the sink address classifier takes everything that a node receives and forwards it on the node's only link.

A source node, which is supposed to forward wirelessly all packets it receives on the wired link without any changes, is shown in Figure 3.7. Just like the sink node, the source node is a simplified version of the general wireless node. For the source node, the following components are modified: the address classifier, the routing agent, the LL and the MAC layers (shown in stripes in Figure 3.7).

In the source node, an IP address classifier is modified to send everything it receives to the routing agent instead of performing address demultiplexing. The operation of the routing agent is significantly altered: for a source node, the routing agent does nothing more than simply push the packets down the protocol stack without any changes. The modifications for the source node LL module are similar to the modifications done for the sink node: the LL module is modified to make sure the ARP module is never addressed. Finally, the MAC layer is significantly altered: at the MAC layer a source node is programmed to

reject all 802.11 data packets, even those directed to the broadcast or to itself.

To create one complete wormhole attacker, a sink and a source must be placed together as shown in Figure 3.5. Then, a sink from one end of the network is connected to a source on another end of the network (shown by arrows in Figure 3.5).

Overall, each packet relayed by the wormhole is handled by several wormhole components. First, it is captured by the intruder's MAC layer where an intruder determines whether this packet should be forwarded or not. Then, the packet goes up to to the LL (where its simply pushed up) and to the sink classifier which places the packet on the wired link between the intruders. After traveling through the source classifier, which forwards that packet to the routing agent, the routing agent forwards the packet down to the LL and finally to the MAC layer where the packet is resent wirelessly.

Finally, it should be noted that the link structure in NS-2 is very simple: a user specifies the delay on the link that they would like to simulate and each packet is held on the link by that delay. In order to be able to better control links between wormhole attackers (for example, induce some randomness on the delay between links), specialized 'wormhole links' – slightly modified versions of NS-2 wired links – were also created.[4]

3.2.3 Discussion

In this work, a fully functional wormhole attack simulation was developed. This is, of course, a base model which can now be programmed for specific wormhole behaviour or for specific network types. With this attack framework ready, the attack behaviour can now be easily modified in a number of ways. Appendix A demonstrates how this wormhole affects the routing tables of a particular

[4]The importance of controlling wormhole links will become apparent in chapter 5.

network; with very little coding, it is possible to study a wormhole's effect on different network topologies and to study the effect of node mobility. With ease, traffic on a network can be added to see how traffic affects wormhole behaviour and its effect. It is also very easy to program this wormhole to transfer only particular kinds of packets or drop a certain percentage of packets going through it. Overall, an extendable basic model of a wormhole in NS-2 was created 'from scratch' (as no literature on the implementation of wormhole attacks in NS-2 was available).

The goal of this work was to obtain a realistic simulator representation of a wormhole in an ad hoc network. In this simulation, the steps of wormhole processing of packets – capturing of a packet by one intruder, packets going through the off-channel link to the other end of the network, packet retransmission by another intruder – are implemented without simplifications. However, there are limitations to how close to reality this model comes. These limitations are discussed below.

Limitations

It should be noted that the intruders can create a wormhole in a multitude of ways. The intruders could be connected with a dedicated wired link or perhaps a congested satellite link. They could be using standard TCP/IP to send the packets they capture to each other, or could come up with their own transport protocol. The intruders themselves could have highly specialized, dedicated machines that spend all their CPU time on the creation of a wormhole, or they could be engaged in a myriad of other tasks. Since it is not known exactly what the intruders would do, there are intrinsic limitations to the modelling of wormhole attacks. In this work, a specific wormhole attack setup was modelled: the intruders, connected by a wired link, use IP tunneling to transfer data to

each other. Here, the implications of the differences between a real-life wormhole implementation and the NS-2 simulation that was created here are discussed.

In this model, the corresponding co-located sink and source intruders are independent of each other, while in reality each would most likely be just a single node with two different wireless cards and an interface to an off-channel link. This implies that in a real wormhole the sink and the source would share information (as well as resources), while in this NS-2 simulation they don't. Also, because sink and source are independent, two separate uni-directional wired links had to be used, rather than a shared bi-directional link that real attackers are more likely to use.

As was explained in the previous sections, in NS-2 all packets a node forwards go through the routing agent. For a real network, that is often not the case, particularly where proactive or table-driven routing agents are used. This may throw off timing in a network. The delays of packets on wired links can also affect the experiments that have to do with timing, especially with statistical uncertainties in the timing. In NS-2, the delay of a particular wired link is actually selected by the user and is specified in the Tcl script. In reality, of course, this is not the case. As a result, strict attention must be paid to applying certain types of timing analysis when using the simulator in order to ensure that artifacts of this particular implementation do not lead to spurious conclusions about the generality of the techniques.

Chapter 4

Protocol breaking for wormhole attack detection

In this chapter, protocol-breaking techniques for wormhole attack detection are introduced. In general, approaches based on discovering out-of-protocol behaviour are often used in intrusion detection [8]; this chapter demonstrates wormhole-specific out-of-protocol behaviour that may be exhibited by attackers. The techniques described in this chapter are based on the periodic nature of proactive routing protocols and can be used to detect a wormhole attack that maliciously drops packets when trying to inflict damage on a network.

Previously, it was mentioned that a wormhole attack cannot be prevented with encryption. A wormhole can still create a spoofed link even if all network traffic is encrypted and can still disrupt that link as it pleases. However, encryption does complicate the life of wormhole intruders. A cunning strategy for a wormhole that is trying to disrupt network traffic would be to forward all routing traffic, while maliciously interfering with transmitted data (working like a blackhole attack). Without encryption, a wormhole knows what packet it

receives and can determine whether this particular packet is to be sent forward. With encryption, a wormhole would not know whether a packet is a control packet or a data packet without complicated derivations.[1] If a wormhole cannot determine which packets are related to routing, when trying to disrupt network traffic it will end up dropping some of the routing messages — something that can easily be detected with protocol-breaking techniques described in this section.

Section 4.1 introduces the protocol-breaking techniques that were developed. Section 4.2 shows how these techniques detect a wormhole attacker in experimental data. Finally, section 4.3 addresses possible shortcomings of protocol-breaking approach to wormhole attack discovery.

4.1 Protocol-breaking techniques: an introduction

Proactive MANET routing protocols, such as OLSR, rely on periodic transmission of particular messages. In OLSR, these messages are HELLO messages and Topology Control (TC) messages. HELLO messages are generated by each node and are not to be retransmitted by other nodes. By default, OLSR HELLO messages are sent out every 2 seconds. TC messages are generated by nodes chosen as MultiPoint Relays (MPRs). TC messages are replayed by all neighbours, and are, by default, sent out every 5 seconds [12]. As described above, when a wormhole does not know what the routing messages are while dropping traffic, it will inevitably end up dropping some of the routing messages. Due to the expected periodicity of HELLO messages, a node that drops HELLO messages can be detected easily with local techniques. In this section two protocol-breaking

[1] It should be noted that a motivated resourceful attacker could still figure out what routing messages look like. This is addressed in section 4.3

techniques are presented, HELLO Message Time Interval (HMTI) profiling and its extension, maximum packet intervals technique, which are both based on the idea that HELLO message loss can be detected locally.

To characterize the loss of HELLO messages, the times between successive HELLOs can be accessed. Let us say that the HELLO message interval in a network is K. The time between successive HELLO messages should be approximately K (to be precise, it should be in the range specified by both HELLO message interval and the jitter on the HELLO messages). If a single HELLO message is lost, the time between successive HELLO messages becomes $\sim 2K$. If two successive messages in a row are lost, the time between successive ones is then $\sim 3K$, etc. Traffic loss can be detected by looking at the intervals between successive HELLO messages. If a node does not drop any packets, the majority of intervals between its HELLO messages are expected to lie within $\sim K$ 'valid' range. If traffic is being dropped, the number of HELLO message intervals in the 'valid range' will decrease — something that can be easily detected. This developed approach was named *Hello Message Time Interval (HMTI) profiling.* A valid node — one that does not drop any messages — should have almost all of its HMTIs in the valid range. For the intruder, the percentage of HMTIs in the $1K$ range should be much lower, and should depend on the percentage of traffic the intruder drops.

HMTI profiling — calculating the percentage of HMTIs in the valid range — is a local technique that network nodes can very easily implement. Knowing the network's HELLO message interval K and the OLSR HELLO message jitter value, the node can identify a 'valid' range, 'second' range (around $2K$) range, and any higher ones. When this setup is completed, a node may simply have a counter running, calculating the number of HMTIs for its particular neighbours that fall within particular ranges. When the percentage of HMTIs in the valid

range falls below a certain threshold, the node can trigger an intrusion alarm. Note that since it is the difference between the HELLO message times that is being accessed and not the actual times themselves, no timing synchronization is necessary for the implementation of HMTI profiling.

In the beginning of this discussion, it was mentioned that both HELLO and TC messages are periodic in OLSR. Since TC messages are also sent out regularly, the same approach could be applied to them as well (*TC Message Time Interval (TMTI) profiling*). However, TC messages are only sent out by the MPR nodes. In addition, TC messages can be validly retransmitted on the network. These properties make TC message profiling more complex with no added benefit.

An extension to the previous idea about looking at the spaces between HELLO messages is the idea to look at the spaces between all messages coming from a node. Let us say, once again, that the HELLO message interval on a network is K, and that the TC message interval is higher than the HELLO message interval (which is usually the case in OLSR). With this, the intervals between successive packets coming from a node should be upper-bounded by the HELLO message interval K. If an interval between successive messages is higher than K, it indicates that traffic is lost, and may signal an intrusion. This protocol-breaking technique is named *maximum packet interval profiling.*

4.2 Experimental results

In this section, the experimental results that show how the developed techniques apply to experimental data are discussed. The experimental data, presented in this section, were obtained in the experiment described in section 2.3.

Figure 4.1 shows intervals between HELLO messages of an intruder that

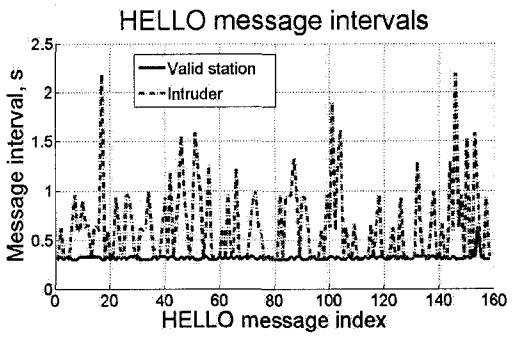


Figure 4.1: HMTIs obtained from the experimental work with the wormhole attack, where solid line shows an HMTI profile of a valid station, and a dashed line gives the HMTIs of an intruder. It can be observed that the HMTI values for the intruder are often higher than the HMTI values of a valid node.

drops packets (dashed line) and a valid node (solid line).[2] In the experiment, the HELLO interval K was set around 0.3 seconds. In Figure 4.1, it can be seen that for a valid node the vast majority of HMTIs lies around 0.3 second — in the $1K$ 'valid' range. For the intruder (dashes line), this is not the case: lots of HMTIs at the $2K$ ($\sim$ 0.6 seconds) and $3K$ ($\sim$ 0.9 seconds) ranges, and even higher, can be observed.

With NTA (see chapter 3), statistics on the percentage of HMTIs in the valid range can be easily obtained. It was determined, with NTA, that for valid nodes 98% of their HMTIs lie in the $1K$ range (the range of 0.28 to 0.34 second was used). For the intruder that drops packets, only 48% of its HMTIs lie in the $1K$ range. An additional 24% of the intruder's HMTIs lie in the $2K$ range while the other 28% fall to the $3K$ range and above.

Figure 4.2 presents the experimental results showing the maximum packet interval technique. The top part of Figure 4.2 shows a valid node. Almost all of its traffic (over 99% of it) falls below the maximum packet interval range.

[2]While a particular node is shown here, the HMTI profiles of all valid nodes were very consistent, similar to the profile of this individual node.

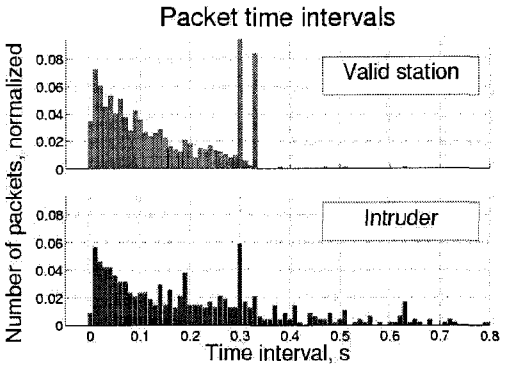


Figure 4.2: Normalized histogram of the packet intervals. A valid node is shown on top, while the intruder is presented on the bottom. It can be observed that for a valid node the packet interval is upper-bounded by the HELLO message interval, as it should be. For the intruder, that is not the case: a great deal of packet intervals are much higher than the HELLO message interval.

The bottom part of Figure 4.2 shows an intruder that drops traffic. It can be observed from Figure 4.2 that the maximum-packet-interval rule is repeatedly broken in the intruder's case.

4.3 Discussion

In this chapter, protocol-breaking techniques for wormhole attack discovery were introduced. These techniques allow to easily detect the presence of wormhole attackers that drop messages. In this section, the advantages of the proposed approach are discussed. Then, the issue of poor connectivity in relation to HMTI profiling is addressed and a question of detection evasion is examined. Although the discussion below focuses on the HMTI profiling, the same comments apply to maximum-packet-interval technique, as both techniques are related to the loss of HELLO messages in a wormhole attack.

The protocol-breaking approach introduced here is simple, and can be implemented locally. All that a node needs to get an HMTI profile of its neighbours

is to keep track of when the HELLO messages from the neighbours were received. A node does not need to cooperate with other nodes, and does not need any additional information. There is no need to modify the routing protocol in any way in order to implement HMTI profiling. Also, HMTI profiling is not computationally expensive.

Protocol breaking with a wormhole attack occurs when the attackers cannot figure out which packets are to be dropped and end up dropping routing messages along with data traffic. If the network traffic is not encrypted, determining whether a packet is related to routing traffic or not is trivial for intruders. Thus, if the network traffic is not encrypted, protocol-breaking techniques on their own will not work. However, this should not be seen as a severe limitation of protocol-breaking techniques as encryption is essential for network security. If no encryption is in place, the network administrator is exposing the network to vulnerabilities that are much easier to implement than wormhole attacks.

If network traffic is encrypted, motivated intruders can, in theory, still determine what messages are related to routing. Routing messages have several defining characteristics that may make them trackable even if encryption is in place. Such messages are broadcast, and broadcast messages can be easily identified based on the 802.11 MAC layer destination addresses. Also, routing messages are sent out with a fixed periodicity. In addition, routing messages are supposed to be sent out even if there is no other traffic. They are supposed to be among the first things that a node sends out when joining a network. In essence, while these wormhole detection techniques use predictable nature of routing messages in order to identify attackers, attackers can also take advantage of the exact same thing to defy being detected. Although identification of encrypted routing messages in network traffic is an interesting subject, further discussion of it is beyond the scope of this work. Overall, it is fair to

assume that a motivated and resourceful attacker could find a way to identify encrypted routing messages, and can thus defeat protocol-breaking techniques that are discussed in this chapter.

Another concern with this type of protocol breaking is that a number of HELLO messages can be lost due to network problems: poor connectivity and congestion, for example, can all results in HELLO messages being lost or corrupted. A certain amount of packet loss is, indeed, expected on a wireless network. In the experiment that was analyzed in the previous section, network load was relatively low, and yet about 2 percent of HMTIs still fell outside the valid range. It could be a challenge to determine whether there are indeed attackers in a network that maliciously drop traffic or if the high number of node's HMTIs is due to connectivity issues.

It is important to note that connectivity problems have other detectable 'symptoms' in addition to the number of lost packets. A node can see that the signal strength of its neighbour is low and expect a higher number of packets coming from that neighbour to be lost. Also, a node with connectivity problems may have a high rate of 'retry' packets coming to and from it. A node that is overloaded with traffic will be, for example, sending or receiving a large number of packets. For the purpose of intrusion detection, these 'symptoms' can be used as secondary checks if a node identifies that the percentage of its neighbour's HMTIs in the valid range is low. Also, note that wormhole attackers cannot create their own packets, and thus a wormhole that drops packets will always have fewer packets than the node whose packets a wormhole is resending. So, while connectivity issues are an important consideration for HMTI profiling, connectivity problems do not invalidate the protocol-breaking approach.

Finally, it should be mentioned that protocol-breaking techniques only work when a wormhole attacker is trying to disrupt network operations. While a

number of researchers work on identifying a wormhole when a wormhole drops packets [27, 28], a wormhole is actually dangerous even if it doesn't disrupt network operations, particularly in sensitive or tactical networks. In the next chapter, a novel technique is presented that allows to detect wormhole attackers that do not drop any traffic.

Chapter 5

Wormhole attack detection with frequency analysis

In the previous chapter, a wormhole attack detection mechanism was described that allows detection of wormhole attacks when wormhole attackers disrupt network operations by dropping network traffic. In this chapter, a more sophisticated wormhole attack detection technique is presented. This technique is based on frequency analysis of packet arrival times and is capable of detecting wormhole attacks even when the wormhole attackers do not drop any network traffic (*dormant* wormholes).

When a HELLO message is transmitted directly from one neighbour to the other, it is only transmitted once: one neighbour sends it, another receives it and does not retransmit it any further. When a message is transmitted through a wormhole, this is not the case. Several message retransmissions and substantial message processing is involved when packets travel through a wormhole: first, a message is captured by one intruder and is encapsulated, then it is sent over an off-channel link, decapsulated by the other intruder, and is finally re-sent

on the wireless media. All this processing adds a delay to the HELLO message times. It will be shown in this chapter that this delay can be modelled by a random variable drawn from a statistical distribution. For ease of representation, this delay and other statistical delays will be called *random* delays. It will be shown in this chapter that the presence of this random delay can be detected with frequency analysis of a time series generated from HELLO message reception times. This wormhole attack detection technique relies on having a number of HELLO messages, not just one message, but does not require time synchronization or any specialized hardware.

Section 5.1 shows, mathematically, how the addition of a random wormhole delay is visible in the frequency domain. For the frequency domain analysis, power spectral densities are proposed and studied. Section 5.2 comments on non-wormhole delays occurring in a network and describes how power spectral density-based detection performs with the presence of these delays. Section 5.3 shows the experimental results that were obtained in a testbed wormhole attack experiment and in an NS-2 wormhole attack simulation. In section 5.4, techniques that wormhole attackers may try to use to counteract this detection method are described and their futility is demonstrated. Finally, the summary of this chapter is provided in section 5.5

5.1 A mathematical description

This section demonstrates how power spectral densities can be used to detect wormhole attackers. In section 5.1.1, a brief mathematical review of the power spectral densities (PSDs) and of certain Fourier Transform (F.T.) properties is provided. Then, in section 5.1.2 analytical expressions for signals with and without random delays are derived and it is explained why adding random delays to samples of a perfectly periodic signal leads to a decline in the signal's PSD.

Finally, section 5.1.3 discusses the effects of finite number of packets and finite sampling frequency on the PSDs.

5.1.1 Mathematical background and introduction to Power Spectral Densities (PSDs)

The power spectral density (PSD), as its name suggests, describes how a signal's power is distributed with frequency. PSD is used for frequency-domain analysis of random signals and noise [40]. There are two equivalent ways to calculate a PSD of a signal: a direct way where the PSD is calculated from the Fourier Transform of the signal, and an indirect way where signal's autocorrelation function is first obtained and this autocorrelation function is then Fourier-Transformed. The second, indirect, way is important since in analytical calculations the direct way is often difficult to use ([40] p. 63). In this section, both ways to calculate PSDs are described. Also, this sections summarizes several important properties of Fourier Transforms.

The *direct way* of calculating a PSD of a signal involves calculation of the Fourier Transform of the signal itself. To get PSD $P_w(f)$ of signal $w(t)$, the signal's Fourier Transform $W_T(f)$ is first calculated: [1]

$$W_T(f) = \int_{-\frac{T}{2}}^{\frac{T}{2}} w(t)\, e^{-j2\pi ft} dt \tag{5.1}$$

Then the PSD $P_w(f)$ of signal $w(t)$ can be calculated as :

$$P_W(f) = \lim_{T\to\infty} \left(\frac{\overline{[|W_T(f)|^2]}}{T}\right) \tag{5.2}$$

where $\overline{W_T(f)}$ is ensemble average of $W_T(f)$. [2] Since this formula involves en-

[1] [40], p. 43
[2] [40] p. 406

semble average, it is often impractical. In practice, the PSD of a random signal is often approximated with the following approximation of formula 5.2: [3]

$$P_W(f) = \frac{|W_T(f)|^2}{T} \tag{5.3}$$

The *indirect way* of PSD calculations involves obtaining a signal's autocorrelation function (AF) and then taking the Fourier Transform of this autocorrelation function. For a deterministic signal $W(t)$, the formula for AF $R_{ww}(\tau)$ is:[4]

$$R_{ww}(\tau) = \int_{\tau=-\infty}^{\tau=+\infty} W(t)W(t-\tau) \tag{5.4}$$

For a random signal $W(t)$, its AF $R_{ww}(\tau)$ can be expressed in terms of expectations:

$$R_{ww}(\tau) = E[W(t)W(t+\tau)] \tag{5.5}$$

PSD $P_W(f)$ is the Fourier transform of the autocorrelation function $R_{ww}(\tau)$: [5]

$$P_W(f) = F[R_{ww}(\tau)] = \int_{-\infty}^{\infty} R_{ww}(\tau)\, e^{-j2\pi f\tau} d\tau \tag{5.6}$$

Also, there are two important Fourier-Transform properties that are applicable to this work:

Property A. It is an essential property of the Fourier Transform: the more 'concentrated' a signal is in time-domain, the more 'distributed' it is in frequency-domain, and vice versa [40]. For example, F.T. of a dirac delta function δ (a signal, 'concentrated' to the maximum in time-domain) is 1 for any frequency – that is, the F.T. of an extremely 'concentrated' time-domain δ is

[3] [40], p. 417. Note that there also exist other, more advanced techniques for approximating PSDs

[4] In this notation (R_{ww}), ww indicates that a signal W is correlated with itself. If signals Y and X were to be crosscorrelated, their crosscorrelation function would be written as R_{yx}

[5] The PSD can be obtained this way by Wiener-Khintchine theorem([40] p. 63, 406), which applies to wide-sense stationary processes.

maximally distributed (over all spectrum) in frequency. On the other hand, a constant in time domain (i.e. $x(t) = 1$) has a F.T. of δ — that is, it is maximally 'distributed' in time-domain and maximally 'concentrated' in frequency-domain.

Property B applies to F.T.s of periodic signals. Let us say that a signal $x(t)$ is periodic with period T (that is, the fundamental frequency of the signal is $\frac{1}{T}$), and $\widetilde{x_T(t)}$ is $x(t)$ on one period. Then $F[x(t)]$ consists of spectral lines that are located at the multiples of signal's fundamental frequency $\frac{1}{T}$. The magnitude of the spectral lines – the envelope of $F[x(t)]$– is defined by the magnitude of the Fourier transform of $\widetilde{x_T(t)}$: [6]

$$envelope\ of\ F[x(t)] \sim |F[\widetilde{x_T(t)}]| \tag{5.7}$$

Finally, in order to calculate the PSDs of a signal formed by the packet arrival times, a mathematical representation of packet arrival times is necessary. In this work, a signal $x(t)$ is created by putting δ functions at the packet arrival times, and zeroes elsewhere. Mathematically, it can be constructed as

$$x(t) = \sum \delta(t - T_p) \tag{5.8}$$

where T_p are packet arrival times. [7]

5.1.2 PSD declines when a delay is introduced

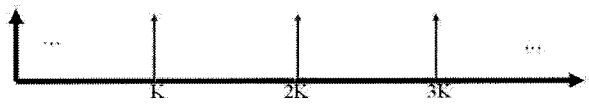
This section shows how PSDs can be used to detect small offsets in packet periodicity — that is, how they can be used to detect the presence of random packet delays introduced by wormhole attackers.

A valid station sends out its HELLO packets with perfect periodicity, as

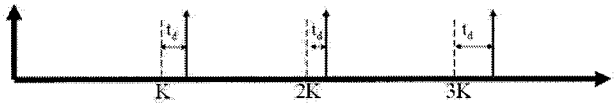
[6] [40], p. 73

[7] This signal model is ideal: in reality, there are no infinite summations and no precise δ functions. These issues are addressed in section 5.1.3.

shown in Figure 5.1(a).[8] Let us call this signal $U(t)$. When a wormhole processes packets sent out by a valid station, a small random delay is added to each of HELLO packets. This addition of a random delay to regular packets is shown schematically in Figure 5.1(b). Let us call a signal formed this way $V(t)$. Below, the PSD of a perfectly periodic signal $U(t)$ is derived and it is shown that the derived PSD has constant envelope over all frequencies. Then, the PSD of a signal $V(t)$ that has an embedded delay is derived and it is shown that the PSD of $V(t)$ declines with frequency.



(a) A valid station sends its HELLO packets every K seconds



(b) A packet is sent, by a valid station, every K second, and wormhole processing adds a small random delay t_d to each of the packet times

Figure 5.1: Signals that are formed by periodic HELLO messages for the valid node and for a wormhole attacker. Figure 5.1(a) shows a signal $U(t)$ that is formed when a valid station sends its HELLO messages every K seconds. Figure 5.1(b) shows a signal $V(t)$ that is formed when a random delay t_d is added to each regularly transmitted packet. Since each t_d is random, delays added to different packets also differ (but their ensemble obeys a certain statistical distribution).

First, an analytical PSD for the perfectly periodic signal $U(t)$ is derived. The *indirect way* is used to calculate the PSD of $U(t)$: the autocorrelation function, R, of $U(t)$ is calculated, then the obtained R is Fourier-Transformed. Since this signal is deterministic, R can be obtained by straightforward application of Formula 5.4. The process of obtaining R can be easily visualized: when

[8] In this section, ideal, theoretical conditions are assumed, ignoring, for now, jitters, effect of finite sampling frequency, travel time of the messages, etc. These practical issues are addressed in sections 5.1.3 and 5.2.1.

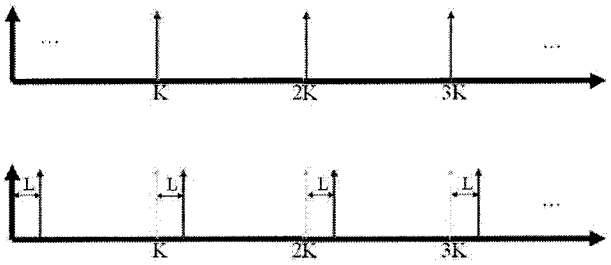


Figure 5.2: Autocorrelation of valid station's packet timings. Bottom shows time-shifted signal that is multiplied with the non-time-shifted version. It is obvious that the autocorrelation function will be periodic with period K, and non-zero only when the lag L is a multiple of K.

obtaining AF, one copy of a signal 'slides' with respect to another (fixed) copy, and these two copies are multiplied. This process is shown schematically in Figure 5.2. From the visualization shown in Figure 5.2, it is clear that R of $U(t)$ is 0 everywhere except in the points where two 'copies' match – i.e. where the shift of one copy with respect to the other is an integer multiple of K, the period of $U(t)$. At these points, R is represented by δ functions. Thus R of $U(t)$ is equal to $U(t)$ – the autocorrelation is the same as the original signal itself.

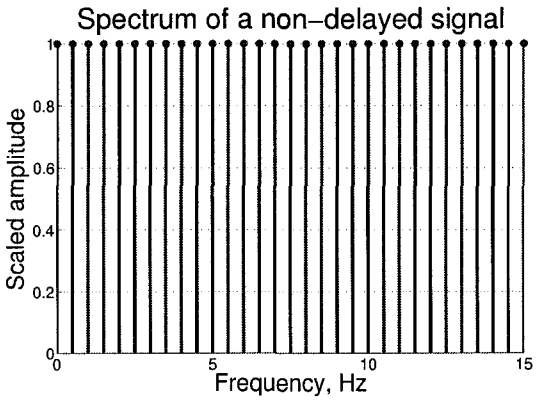


Figure 5.3: PSD of a perfectly regular signal $U(t)$ with $K = 2$. The δ functions are located at integer multiples of $\frac{1}{K}$, and the envelope of the PSD is constant.

The PSD of $U(t)$ can be obtained by Fourier-Transforming R. The F.T. pair for a periodic pulse train (with period K) is: [9]

$$\sum_{i=-\infty}^{\infty} \delta(t - iK) \Longleftrightarrow f_0 \sum_{n=-\infty}^{\infty} \delta(f - nf_0),\ f_0 = 1/K \tag{5.9}$$

By directly applying formula 5.9 to the obtained R, the PSD of $U(t)$ is determined to be a train of δ functions located at integer multiples of $\frac{1}{K}$. A sample PSD of $U(t)$ (for the case of $K = 2$) is shown graphically in Figure 5.3. Note that once R is obtained, it is also possible to use F.T. property B (see section 5.1.1) to derive the PSD of $U(t)$. The autocorrelation function R derived above is periodic with period K, and on one period R is just a single δ function. The F.T. pair for a δ function at an offset t_0 is:[10]

$$\delta(t - t_0) \Longleftrightarrow e^{-j2\pi f t_0} \tag{5.10}$$

and

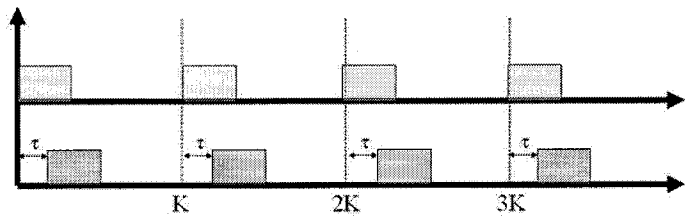
$$|e^{-j2\pi f t_0}| = 1 \tag{5.11}$$

By property B (see section 5.1.1), the F.T. of R (i.e. the PSD of $U(t)$) is a sampled version of the expression given by Formula 5.11 – a sampled version of a straight line – with samples located at integer multiples of $\frac{1}{K}$. Note that this result agrees with property A (see section 5.1.1): on each period, the time-domain signal $U(t)$ is 'concentrated' to the maximum (it is just a δ function), and the frequency-domain view of this signal is 'distributed' to the maximum.

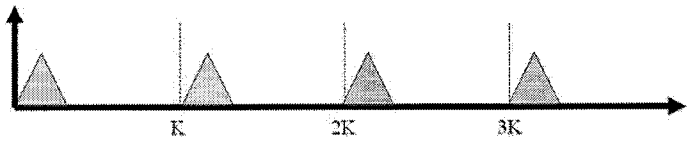
Next, a signal $V(t)$ that is generated when a wormhole adds random delays to each HELLO message sent out by a valid station is examined. Using F.T. Property A (section 5.1.1), it can be intuitively seen that the PSD of $V(t)$ will

[9] [40], p. 62
[10] [40], p. 62

be more 'concentrated' than the PSD of $U(t)$, as $V(t)$ is more 'distributed' in the time-domain.



(a) To obtain an autocorrelation, a signal is multiplied with a time-shifted version of itself. The bottom signal 'slides' with respect to the top signal. In this case – the case of two uniform distributions – two squares are shifted and multiplied.



(b) As a result of convolving two square signals, a triangular signal is obtained on one period, with the same periodicity as the original signal.

Figure 5.4: Autocorrelation of a signal with a random uniform delay. Figure shows how a 'signal' is shifted with respect to itself to obtain the autocorrelation. Figure shows the resulting autocorrelation function. It can be seen from these figures that for a signal with a random uniform delay the autocorrelation function is periodic with the same period as the original signal, and has a triangular shape

First let us examine a signal $V(t)$ that is formed when a wormhole adds a random delay from *uniform* distribution to packets transmitted with interval K. Again, it is possible to use the indirect way to calculate the PSD of $V(t)$. Since this signal is not deterministic, Formula 5.5 can be used to calculate its autocorrelation R. Although this formula involves expectations, the process of obtaining R is still easy to visualize: to obtain R, two 'signals' are shifted with respect to one another, and then multiplied (similar to a convolution), where the 'signals' that are shifted with respect to each other incorporate statistical delay distributions, as shown schematically in Figure 5.4. The derived autocorrelation

R is periodic with period K, and on one period it is represented by a triangular 'signal' (a triangle-shaped distribution), as shown in Figure 5.4.

To get the PSD of $V(t)$, R needs to be Fourier-transformed. Again, property B can be used to calculate the F.T. of R. In this case on one interval R is triangular. Thus, by Property B, the Fourier Transform of R is expected to have an envelope consistent with the Fourier Transform of a triangular signal. The Fourier Transform pair for a triangular pulse $\triangle$ (with width T) is: [11]

$$\triangle(t/T) \Longleftrightarrow T[Sa(\pi fT)]^2 \tag{5.12}$$

where

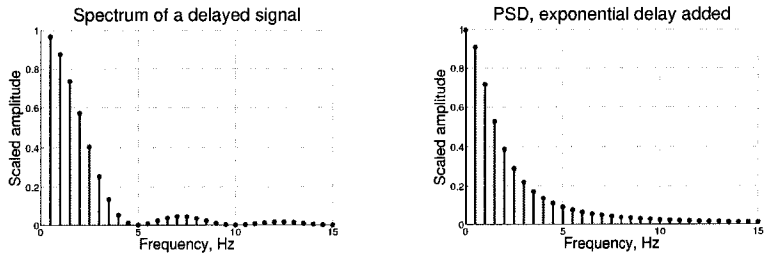
$$Sa(x) = \frac{sin(x)}{x} = sinc(\frac{x}{\pi}) \tag{5.13}$$

Therefore the PSD of a $V(t)$ consists of spectral lines located at integer multiples of $\frac{1}{K}$, and has an envelope proportional to $sinc^2$.

The PSD of a particular $V(t)$ (where $V(t)$ is such that $K = 2$, and incorporated delay is uniform with mean $\frac{K}{20}$) is shown in Figure 5.5(a). As is to be expected from Property A, the PSD of this signal is more 'concentrated' in frequency domain than the PSD of a perfectly regular signal $U(t)$. The PSD of $V(t)$ has strong components in low frequencies, but quickly reduces to zero as frequency increases.

For completeness, the PSD of a signal $V(t)$ that is formed when small exponential delays are added to samples of a regular signal $U(t)$ is also derived. The PSD of $V(t)$ can be approximated using an approximation to the *direct way* of calculating PSDs (Formula 5.3). Once again, because of the periodic nature of the signal (Property B), the PSD has a line spectrum with spectral lines located at $\frac{1}{K}$. The envelope of the PSD is proportional to the square of the signal's Fourier transform. An exponential distribution can be approximated by a

[11] [40], p. 54, 62



(a) PSD of a signal with an embedded uniform delay ($K = 2$, delay mean $\frac{K}{20}$)

(b) PSD of a signal with an embeded exponential delay ($K = 2$, delay mean $\frac{K}{20}$)

Figure 5.5: PSDs of signals with uniform and exponential delays

one-sided exponential signal. The Fourier Transform of a one-sided exponential is:[12]

$$e^{-\frac{t}{T}} \Rightarrow \frac{T}{1 + j2\pi fT} \tag{5.14}$$

Using Formulas 5.14 and 5.3, it can be determined that the envelope of the PSD is proportional to:

$$|\frac{T}{1 + j2\pi fT}|^2 = \frac{T^2}{1 + (2\pi fT)^2} \tag{5.15}$$

A sample PSD of a signal $V(t)$ with an embedded exponential delay (for $K = 2$, delay mean $= \frac{K}{20}$) is shown in Figure 5.5(b).

So far it has been demonstrated that addition of uniform and exponential random delays to a signal generated by regular HELLO packets leads to the decline in the PSD of a signal. Property A of Fourier Transform allows the claim that introduction of any random delay will lead to the decline in the PSD. This claim follows directly from Property A: when a delay is introduced, a signal becomes more 'spread out' in time domain, thus, by property A, it becomes more 'concentrated' in the frequency domain. From a wormhole analysis point of view, it has been argued that wormhole processing adds a random delay to each of the packet samples, and thus wormhole processing of packets should

[12][40], p. 62

result in a decline of PSD calculated from HELLO message times.

5.1.3 Issues with finiteness

In the previous section, the following model for a signal $x(t)$ created by HELLO message arrival times T_p was used:

$$x(t) = \sum \delta(t - T_p) \tag{5.16}$$

This signal model describes a continuous infinite-length signal. In real life, of course, the signals are sampled: discrete signals and finite in length. Thus the issues of having a finite sampling frequency F_s, and a finite number of sample points (packet arrival times) have to be addressed.

The sampling frequency F_s indicates how many samples per second are obtained. The higher the sampling frequency, the more computational steps are needed to analyze signals. However, the maximum sampling frequency is limited not only by the computational power that can be devoted to this technique, but also by the timestamp resolution, as the packet arrival times themselves are finite, with resolution Δ. Thus the maximum possible F_s is $\frac{1}{\Delta}$.

The sampling frequency determines the highest component that can be seen in a PSD. For a sampling frequency F_s, the maximum frequency that can be identified in a PSD F_{max} is, by Nyquist theorem: [13]

$$F_{max} = \frac{F_s}{2} \tag{5.17}$$

Theoretical derivations in sections 5.1 and 5.1.2 rely on having an infinite number of HELLO packets; in reality, the number of HELLO packets is always finite as the experiments are inherently finite. In addition, if the HELLO mes-

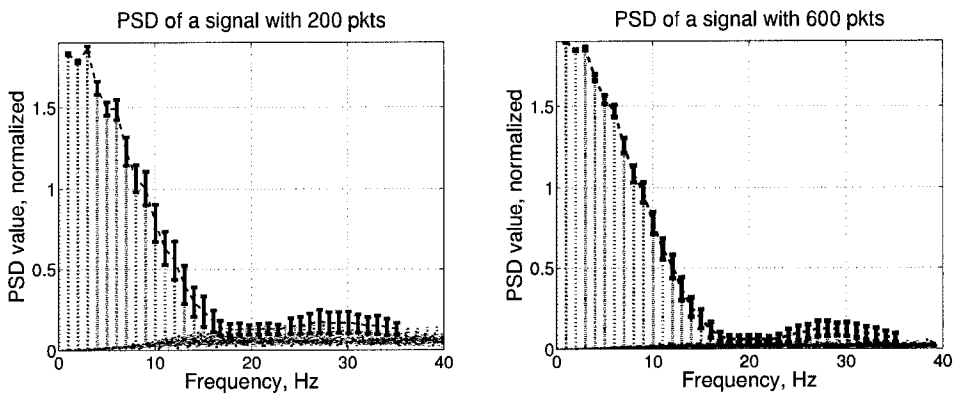
[13] [40], p. 87

sage interval K is large, then accumulating a large number of packets could take a long time and could be problematic.

How does having a finite number of packets affect PSDs? Intuitively, the fewer packets there are, the more individual signals that have same statistical parameters vary, and hence the more PSDs vary. To access the effect of the finite number of packets, a MATLAB simulation is created. In this MATLAB simulation, the following parameters are used:

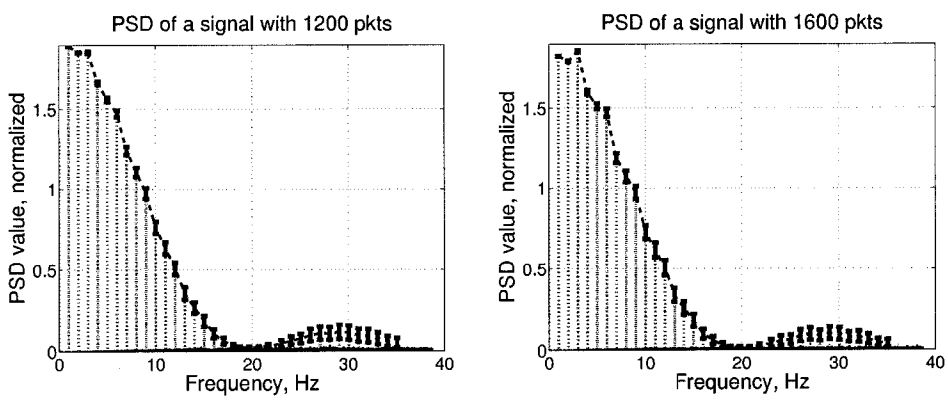
- HELLO message interval $K = 1$s
- Sampling frequency $F_s = 500$ Hz
- The introduced delay t_d is uniformly distributed between 0 and $\frac{K}{20} = 0.05$s (i.e. delay mean $= \frac{K}{40}$)
- The PSDs are evaluated for four different values for the number of packets P: $P = 200$, $P = 600$, $P = 1200$, and $P = 1600$.
- For each of the four values of P, 30 independent experiments are completed. In each of these experiments a signal is generated that has the specified number of packets P and the specified delay t_d.

Figure 5.6 shows the PSDs that are obtain in this simulation. In Figure 5.6, the error bars demonstrate the range of the PSD values that are obtained when PSDs are calculated independently in 30 separate trials. As can be seen in Figure 5.6, if the number of packets used to calculate a PSD is relatively small, the PSD values obtained in individual experiments are varied, and can differ significantly, particular in the higher frequency ranges. As the number of packets increases, the range of PSD values obtained in independent experiments decreases. For example, when only 200 packets are used (Figure 5.6(a)), the



(a) PSDs of signals with 200 packets

(b) PSDs of signals with 600 packets



(c) PSDs of signals with 1200 packets

(d) PSDs of signals with 1600 packets

Figure 5.6: PSDs of signals obtained by keeping the experimental parameters the same and varying the number of packets used to obtain a PSD. In the figures above, the error bars show the range of measurements obtained by running 30 independent experiments with the same settings. Note that the error bars become smaller (which implies that the PSDs converge) as the number of packets increases.

error bars are large, particularly at high frequencies. When the number of packets used is increased, the error bars become smaller as the PSDs obtained in separate experiments converge. From a theoretical point of view, this is to be expected: the more samples from a distribution there are, the better these samples characterize a distribution, and the closer an empirical PSD is to the theoretical calculations.

5.2 Jitter and other causes of HELLO message delays

In the previous section, a mathematical model for a signal where HELLO messages are transmitted periodically was described. It was shown that an addition of small random delays leads to a decline in the PSD of a signal. In a network, however, there could be other delays added to HELLO messages (other than wormhole delays) — jitters, contention-based delays, etc. In section 5.2.1, HELLO message jitter is discussed. Section 5.2.2 demonstrates a jitter waveform that eases the detection of wormhole attacks. In section 5.2.3, other delays — the delays that are causes by contention and nodes' activities — are discussed.

5.2.1 HELLO message jitter

As explained in section 2.1.2, in OLSR a random delay (jitter) can be added to HELLO message send-out times. Addition of jitter helps to prevent message collisions, ensuring that the probability of two nodes wanting to transmit a HELLO message at the same time is low. For a HELLO message interval set at K seconds, the jitter values can go up to $\frac{K}{4}$ seconds [12]. This jitter delay distribution is shown in Figure 5.7(a). From section 5.1, it is known that an addition of a random delay causes a PSD decline. HELLO message jitter is itself

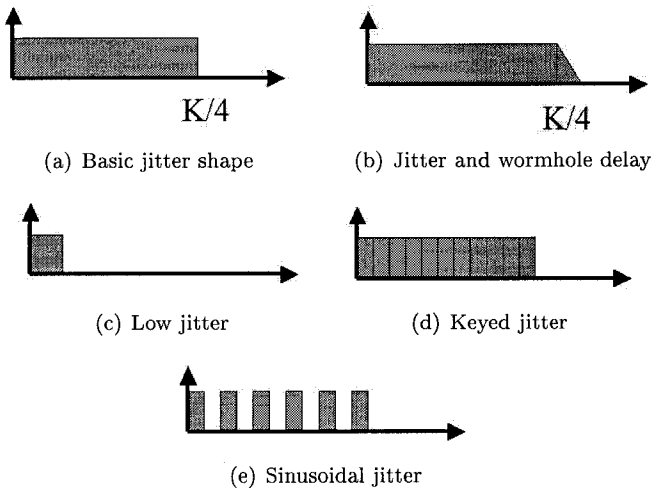
a delay, and potentially a large one. It may thus seems counterintuitive that even with a large delay – jitter – it is possible to observe the effect of a smaller delay – wormhole delay. First, this section explains that a wormhole attack should be detectable in the presence of a large jitter, although a large number of packets would be required for reliable wormhole detection. Next, changes to the jitter waveform that would make wormhole detection easier are introduced and explained.

What happens when a wormhole delay is added on top of the jitter? When two delays are added one after another, the resulting overall delay distribution is a convolution of the distributions of two delays that are involved [40]. When a wormhole delay is added on top of the HELLO jitter, the wormhole delay smooths out the 'edge' of the jitter delay and extends the overall delay range, as shown schematically in Figure 5.7(b). Since addition of a wormhole delay increases the delay randomness even with large jitter, a wormhole delay would still lead to the decline in the PSD, and should still be detectable. Unfortunately, as was shown in section 5.1.2, the higher the delay's randomness, the fewer high-frequency components the PSD has. When a jitter delay is large, the attenuation caused by a wormhole would have to be judged in low-frequency ranges of the PSDs, where the PSD decline introduced by a wormhole is not as large as it is in high-frequency ranges. To determine small, low-frequency declines in the PSD, a very precise characterization of the PSD would be needed, and that would require a large number of packets. So, while it would be possible to detect a wormhole delay in situations where the HELLO message jitter is large, this detection would require a large number of packets for reliable results. [14]

[14]It should be noted that the need to have a large number of packets is limiting in some cases, but not in others. Together with HELLO message interval K, the number of packets P_{req} that are required defines how fast a wormhole attack can be discovered. The formula for the wormhole discovery latency is very simple:

$$L_w = K * P_{req} \tag{5.18}$$

If $K = 2$ seconds, and $P_{req} = 600$, then L_w is 20 minutes, which could be a sufficiently low



(a) Basic jitter shape

(b) Jitter and wormhole delay

(c) Low jitter

(d) Keyed jitter

(e) Sinusoidal jitter

Figure 5.7: Figure 5.7(a) shows the basic jitter shape recommended by OLSR RFCs, where K is the HELLO message interval. Figure 5.7(b) shows that the addition of a wormhole delay on top of the jitter results in 'smoothing' of a distribution edge and in increasing the jitter range past $\frac{K}{4}$. The rest of the figures show the different jitter shapes proposed. Figure 5.7(c) demonstrates how the jitter's maximum value can be decreased. Figure 5.7(d) demonstrates the idea of keyed pseudo-random jitter: each of the intervals, shown in this figure, corresponds to a particular value of keyed function. Finally, Figure 5.7(e) shows a 'quantized' jitter distribution: instead of occupying complete range of 0 to $\frac{K}{4}$, jitter values within this range are distributed such that a sinusoid can be fitted to them.

Since collecting a high number of packets may not possible or desirable in all networks, other ways of dealing with jitter are necessary. First of all, jitter could be set to be low, just like it was set to a very low value in the testbed experiment described in 2.3. A possible jitter delay distribution for the low-jitter case is shown in Figure 5.7(c). Also, in some cases jitter use could be avoided altogether — jitter is recommended, but not required by OLSR

latency for some networks but not for others. If, for connectivity reasons, a network has a short K, then over the same timespan a higher number of packets can be obtained. Overall, the need to have a large number of packets is not limiting in the following cases:

- Low wormhole detection latency L_w is not required.
- Packet interval K is short.

specifications [12].[15] Also, since random *uniform* jitter is recommended but not required by OLSR RFCs, rather than changing the jitter maximum, the jitter distribution could instead be altered. One of the choices could be 'quantized' jitter which is shown graphically in Figure 5.7(e). The jitter of this form has a strong sinusoidal component which acts like a carrier frequency and introduces large PSD values at high frequencies. It is explained in more detail in section 5.2.2. As another approach, pseudo-random keyed jitter can be used: a jitter which appears random, but is actually generated by some function using a key known to all network nodes. This form of jitter is shown schematically in Figure 5.7(d). When this jitter is used, each node, knowing the keys that are used to generate jitter values, knows the jitter value that their neighbours are using to send HELLO messages and can simply subtract this known jitter value from the HELLO message time to get a clean undisturbed PSD. [16]

Finally, it should be mentioned that although this work mostly focuses on OLSR HELLO messages, instead of relying on OLSR HELLO messages, one may elect to create a separate 'wormhole discovery' protocol instead based on the same principles that are described in this chapter. Such protocol would, for example, test newly created links by sending small periodic precisely timed packets on a link. In this setting it would be possible to keep the message frequency high and to keep jitter to a minimum. The downside of using such a protocol is that it, unlike wormhole discovery based on HELLO messages, it would result in additional overhead.

[15] In this case, however, avoiding synchronization between nodes becomes a problem. To avoid synchronization, the nodes could start up at different times. For example, when the packet interval is K, prior to deployment each node could pick a random time delay r from $R = [0, K)$, wait r seconds after deployment, and send all its packets with perfect periodicity K after that. In addition, to avoid synchronization, each node could use a slightly different value for its HELLO message interval instead of a single global fixed K. For example, one network node could use the value $K + \Delta$, another — $K + 2\Delta$, where Δ is sufficiently small.

[16] This form of jitter is currently being investigated in a Masters project conducted at DRDC by Dan Lynch from Royal Canadian Military College (RMC)

5.2.2 Sinusoidal/quantized jitter

In section 5.2.1, it was noted that HELLO message jitter is problematic for the proposed wormhole attack detection technique as, when large jitter is implemented, the number of packets that need to be accumulated in order to reliably distinguish a 'valid' node from a malicious node is high. In this section an ingenuous scheme is described that makes large jitter values possible, and yet allows wormhole attack detection with few packets.

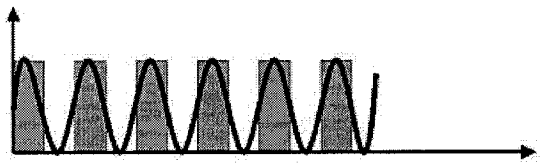


Figure 5.8: 'Sinusoidal' jitter waveform. Quantizing a jitter is this fashion introduces a high-frequency 'carrier' sinusoid to the jitter waveform.

A suggested jitter waveform was shown schematically in section 5.2.1 in Figure 5.7(e). When this jitter is used, the maximum jitter value is not changed (it is left at $\frac{K}{4}$), but the jitter distribution is altered. The idea behind quantized/sinusoidal jitter is the following: from signal processing and communications, it is known that a high-frequency sinusoid can act as a 'carrier component', 'carrying' low-frequency shape of the signal's description to higher, 'carrier', frequencies [40]. The sinusoidal jitter distribution is created in such a way that it incorporates a high-frequency sinusoid (shown schematically in Figure 5.8), which introduces high-frequency components to a PSD, easing wormhole attack detection (as will be shown).

Consider the following MATLAB experiment:

- HELLO message interval $K = 1$s.
- Sampling frequency $F_s = 500$ Hz.

- Maximum jitter on HELLO messages is $\frac{K}{4}$ (0.25 seconds).
- 200 samples (packets) are used to characterize a PSD.

The PSD that is obtained in this MATLAB experiment is shown in Figure 5.9.[17] Since jitter is large, the PSD of such signal is reduced to noise for the frequencies as low as $\sim$ 10 Hz. In this setting, the high-frequency attenuation a wormhole delay introduces would not be evident, as there is not enough high-frequency components in the signal's PSD, and the low-frequency characterization varies a lot over individual experiments due to a small number of packets.

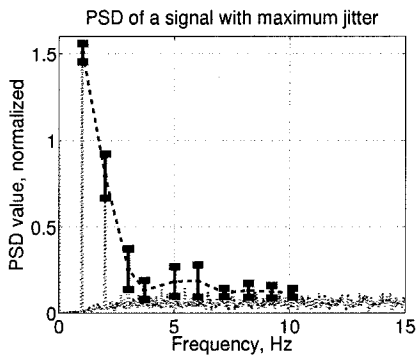


Figure 5.9: PSD of a signal with uniform jitter where maximum jitter value is $\frac{K}{4}$ and $P = 200$ packets are used to determine a PSD. The PSD of such signal does not have high-frequency components, and varies at lot in individual experiments (shown by long error bars) due to a small number of packets used to characterize a PSD. These two factors together – a small number of packets and the lack of high-frequency components – make wormhole attack detection in this setting difficult.

Consider what happens in this experiment if, instead of simply using random-uniform jitter, the jitter values are quantized to 20 uniformly distributed disjoint levels. Let us say that to create a uniform jitter, a random value is drawn from the range of $R_1 = [0, maxJ] = [0, 0.25]$s. Without quantization, the number

[17] In this section, all PSDs are shown with error bars. These bars are used to show the range of PSD values that were obtained by running each of the experiments independently 30 times

drawn from R_1 becomes the jitter value (i.e. the jitter distribution is random uniform with maximum of 0.25s). In order to perform jitter quantization, the range R_1 is separated into 20 sub-ranges of width 0.0125. When a number drawn from R_1 is under 0.0125, the jitter value is set to precisely 0.0125. If the number drawn from R_1 is between 0.0125 and 0.0250, the jitter value is set to precisely 0.0250, etc. After the quantization, there are 20 possible jitter values, with the highest jitter value at $\frac{1}{4}$th of the HELLO message interval. The PSD that was obtained with the above experimental setting with this quantized jitter (instead of a continuous uniform jitter) is shown in Figure 5.10(a). It can be seen in Figure 5.10(a) that when the jitter is quantized this way, a set of strong PSD components located at a high frequency values is obtained.

Quantization of jitter can be seen as an introduction of a high-frequency sinusoid. The frequency of such sinusoid is defined by the following. If the jitter range $R_1 = [0, maxJ]$ is divided into k values, the period T_{sin} of an introduced sinusoid is:

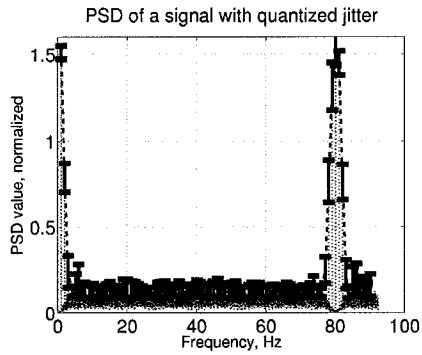
$$T_{sin} = \frac{maxJ}{k} \tag{5.19}$$

The frequency F_{sin} of a sinusoid is then

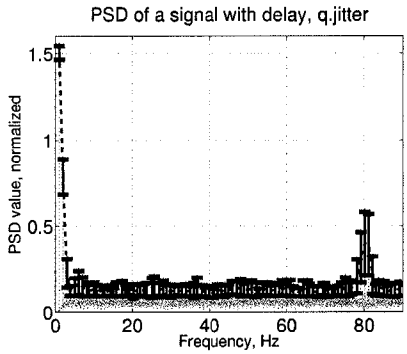
$$F_{sin} = \frac{1}{T_{sin}} = \frac{k}{maxJ} \tag{5.20}$$

In the example above, the maximum jitter value is $maxJ = 0.25$ seconds, and it is quantized to $k = 20$ levels. The frequency of the introduced sinusoid is thus $F_{sin} = \frac{20}{0.25} = 80$ Hz. With the introduction of such sinusoid, a high-frequency 'carrier component' is obtained, at which the baseband PSD is repeated (as can be seen, for the above example, in Figure 5.10(a)).

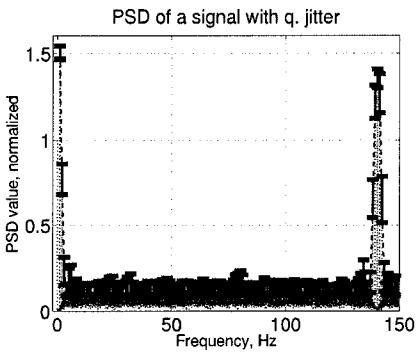
When such high-frequency components are present in the signal's PSD, detection of the wormhole delay becomes very simple. Consider Figure 5.10. Fig-



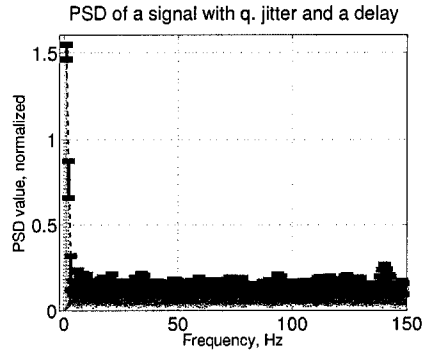
(a) PSD of a signal with quantized jitter. Jitter quantization adds a strong high-frequency component to the signal



(b) PSD of a signal with quantized jitter and a small (exponential, mean=4ms) wormhole delay. Presence of the delay is evident in the PSD



(c) PSD of a signal with quantized jitter. Because of the jitter quantization, strong frequency components around $F_{sin} = \frac{35}{0.25} = 140$ Hz are present in the PSD

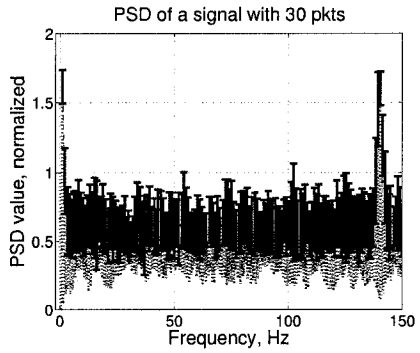


(d) PSD of a signal with quantized jitter and a small (exponential, mean=4ms) wormhole delay. The higher-frequency components of the PSD are reduced almost to the noise level

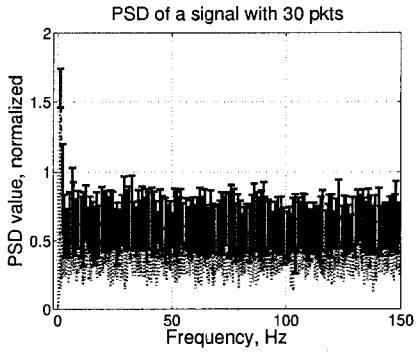
Figure 5.10: PSDs of signals with embeded quantized jitter

ure 5.10(a) shows PSDs of signals with the quantized jitter. Figure 5.10(b) show PSDs of signals with quantized jitter and a wormhole delay, where the wormhole delay is exponential with mean $0.004s$. It is clearly seen in Figure 5.10(b) that the introduction of a wormhole delay causes a significant decline in the PSD of such signal. If the jitter is quantized to more values, the 'carrier frequency' is increased, and the effect of a wormhole delay's effect can be seen with even greater clarity. Consider Figures 5.10(c) and 5.10(d). These figures refer to the same experiment as described in the beginning of this section, with jitter quan-

tized to more levels. Above, an example with $k = 20$ jitter quantization levels was examined; here, the jitter is quantized to $k = 35$ values, obtaining the sinusoidal frequency $F_{sin} = \frac{35}{0.25} = 140Hz$. The frequency components around this frequency are clearly seen in Figure 5.10(c); they are reduced almost to noise level with the introduction of a wormhole delay, as shown in Figure 5.10(d).



(a) PSD of a signal with quantized jitter, obtained with only 30 packets. Because of the jitter quantization, strong frequency components around $F_{sin} = \frac{35}{0.25} = 140$ Hz are obtained. Long error bars indicate high variability among individual experiments.



(b) PSD of a signal with quantized jitter and a small (exponential, mean=4ms) wormhole delay. Note that the PSD values in high-frequency range are lower than the low ends of the error bars in figure 5.11(a), indicating that it would be possible to detect a wormhole attack with this setup even if only 30 packets are used for PSD creation.

Figure 5.11: PSDs of signals with quantized jitter that are obtained using only 30 packets. From these figures, it is clear that a wormhole introduces a delay to a PSD that is apparent even if only 30 packets are used for PSD calculations.

Note that the above experiments refer to the case of 200 packets used to characterize a PSD. However, even fewer packets are sufficient to detect a wormhole in this scenario. Consider the previous experiment ($k = 35$) with only 30 packets instead of 200. If the OLSR HELLO message interval is set to its default value of 2 seconds, this number of packets would amount to exactly 1 minute of traffic. The PSDs that are obtained in this experiment, with and without a wormhole delay, are shown in Figure 5.11. In Figure 5.11, it is clearly seen that even with 30 packets a wormhole attack can be reliably discovered from

the attenuation of the high-frequency signal components. Note that although the error bars are long (which makes sense since there are only 30 packets used to characterize a distribution), the 'attenuated' PSD (with a wormhole) is much lower than the low points of the error bars of the non-declined PSD. Thus, even with 30 packets, the wormhole attack detection technique works for the case of quantized jitter.

Overall, the use of 'quantized' ('sinusoid') jitter allows to use a large, well-distributed number of values for the HELLO message jitter while also introducing high-frequency components to the PSDs that can be used for wormhole attack detection. With quantized jitter, the number of packets required to reliably detect a wormhole attack becomes low and the wormhole discovery latency is reduced.

5.2.3 Other causes of HELLO message delays

In addition to jitter, there could be other network events that cause delays in HELLO message transmissions: contention, slow nodes, etc. Unfortunately, unlike HELLO message jitter, these delays cannot be easily characterized mathematically.

First of all, it should be noted that due to unpredictable nature of such delays, in order to account for them a node may choose to compare the PSDs of its neighbours not to a theoretically derived 'baseline' PSD, but to a PSD derived from node's own HELLO messages. To derive this PSD, a node can use the timestamps on its own outgoing HELLO messages.[18] Also, it should be noted that a wormhole link only adds delays and cannot remove delays that are present on a network. If there are natural delays on a network, packets that travel through a wormhole would be delayed even further. The naturally occurring

[18]Note that in this case the node should not rely on the times when its packets are generated, but should use message sent-out times provided by node's wireless card.

delays that are described in this section are important from the perspective of minimizing false positives rather than from the perspective of characterizing a wormhole itself.

At the physical layer, contention may cause delays, as the IEEE 802.11 MAC standard uses contention-based media access. When a node wants to send a packet, it listens to the media. If the media is free, the node sends a packet. If the media is busy, the node waits until the media becomes free, and then waits a random amount of time before transmitting [7]. Thus if the media is busy, contention-based techniques add a delay to message times. Contention delays are hard to characterize statistically as they depend on a large number of parameters (number of nodes in an area, load of each node, contention window size, protocol version, modulation scheme used, etc). It should be noted, first of all, that, unless congestion is severe, the physical-layer delays should be small compared to wormhole delays (which involve not only physical layer delays, but also processing delays). In a severely congested scenario, the wormhole attack detection technique described in this chapter may suffer, but in the case of severe congestion secondary checks could be use to 'diagnose' congestion and ease wormhole attack detection. Note, also, that if node's own PSD is used as a 'baseline', then certain aspects of congestion factor into the baseline. Finally, it should be mentioned that minimization of congestion delays in 802.11-based MAC protocol is currently a significant research area. While this current work is done within the framework of a standard IEEE 802.11 protocol, the protocols that minimize congestion delays would help this technique.

Another cause of delays could be a node being too slow in processing its own packets. For example, a node could be overloaded with traffic, or overloaded with computational tasks. These delays are also hard to characterize mathematically, as they depend on many factors. The effect of these delays could be

minimized if HELLO messages are treated as priority messages. Note that if a node is overloaded with traffic, its neighbours can detect that a large number of packets are coming in and out of a node, and can use this traffic amount as a secondary check.

While it is hard to characterize these delays mathematically, their effect can be studied, to a certain extent, in a network simulator. Section 5.3 describes NS-2 results that were obtained when network nodes were handling a relatively large amount of traffic, and demonstrates that with the amounts of traffic studied, the wormhole detection technique does not break down. Also, note that these delays are inherently related to MANET QoS: the smaller the valid delays are (the objective of QoS), the easier it is to spot the invalid wormhole delay. Since MANET QoS is currently an active research area, it is quite likely that these delays will become less of a factor as MANET QoS techniques are being developed and integrated.

5.3 Experimental results

Previous sections describe how wormhole attackers can be discovered by assessing the PSDs of signals created by the HELLO messages coming from network nodes. This section explicitly demonstrates the application of this wormhole attack discovery technique to the traffic generated by:

- A testbed wormhole attack experiment described in section 2.3. These results are presented in section 5.3.1
- An NS-2 wormhole attack simulation described in chapter 3 (section 3.2). These results are presented in section 5.3.2.

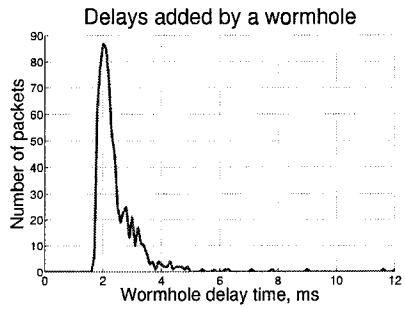


Figure 5.12: Profile of the experimental time delay resulting from packet traveling through a wormhole. The delay is measured by comparing the sniffer timestamps of packets broadcast by the legitimate node and the same packet after it is rebroadcast at the end of the wormhole.

5.3.1 Testbed wormhole attack implementation results

This section presents the application of frequency-based wormhole attack discovery technique to the experimental traffic generated by a testbed wormhole attack implementation.

In the testbed experiment described in section 2.3, a 9-node MANET was implemented in a testbed, and two fully functional wormhole attackers were introduced to the MANET. The MANET traffic was recorded with a wireless sniffer and was then processed with the NTA-MANET traffic analyzer suite created in this thesis and described in chapter 3. A salient feature of NTA-MANET is the ease of creating processing functions for analysis like this: having basic functionality (uploading of wireless packets, parsing of 802.11 headers, handling of OLSR) in place, higher-level analytical processing can be done with only a few lines of code.

Using the NTA-MANET tools, it is possible to easily parse the captured testbed traffic and calculate wormhole delay times. This is done by identifying packet repetitions (the signature of a wormhole) and subtracting the capture timestamps. Figure 5.12 shows the histogram of the measured wormhole delay that the testbed wormhole attackers introduced to the HELLO messages coming

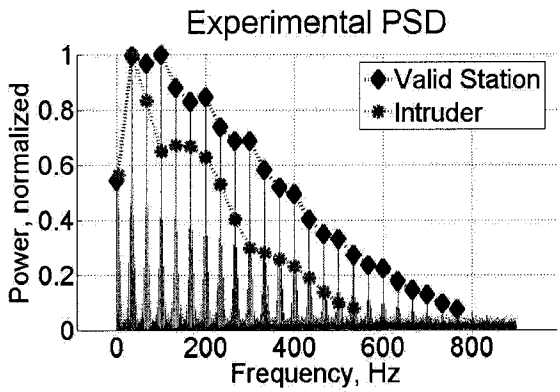


Figure 5.13: The PSDs calculated based on the data obtained in a testbed experiment. The 'diamonds' show the PSD of a valid station; the 'circles' show the PSD generated by the HELLO messages that were processed by wormhole intruders. The PSD of a signal formed by wormhole-processed packets exhibits a markedly more rapid fall-off than the PSD of a 'valid' signal.

from valid network nodes (where the delays were calculated by assesing the timestamps of original and wormhole-processed packets). It is interesting to note that this delay distribution is well approximated by a Rayleigh distribution, with its mean around $2ms$. Note that this delay is so low that it does not affect network connectivity or network performance. In fact, as noted in section 2.3, the jitter on HELLO mesages in this experiment was 0.315 seconds (on average), so this wormhole delay is much smaller than jitter, and would not be detected by timing analysis of individual messages.

Now that it has been shown that the wormhole *does* introduce statistical delays to each packet, the analysis techniques developed and described previously can be applied to the testbed traffic. Figure 5.13 shows the PSD profiles of the valid node ('diamonds') and the wormhole attacker ('circles') that were obtained from the experimental data.[19] It is clear in Figure 5.13 that the PSD of an at-

[19]These results were obtained using sampling frequency F_s = 2000 Hz, and using 300 packets to generate a PSD

tacker is attenuated with respect to the PSD of a valid station, and therefore a wormhole attacker in this testbed experiment can be identified through the frequency-based analysis of HELLO message times. This result agrees nicely with the theoretical derivations presented in section 5.1.

Note that the experimental setup that was used in the testbed – two attackers connected by a direct wire – is perhaps the best-case scenario for the wormhole attackers (and the worst-case scenario for attack detection), requiring the least amount of processing. It is encouraging that the PSD-based technique is able to spot a wormhole attacker in this worst-case scenario.

5.3.2 NS-2 wormhole attack simulator results

In this section the PSDs obtained from the NS-2 wormhole attack simulation are considered. As noted in section 5.2.3, it is very difficult to assess theoretically the impact of network-related delays, such as traffic loads and nodes' computational loads, on the PSDs. These issues are also difficult to access within a testbed scenario as they require flexibility and numerous experiments with varying parameters. It is, however, relatively easy to run several different experiments with different traffic loads within a simulator environment. This section demonstrates the effect of increasing traffic load on PSD profiling with the help of the NS-2 wormhole attack simulation described in section 3.2.

For this analysis, the network under consideration was a 'star' network, and wormhole attackers were connecting opposite star rays, as shown in Figure 5.14. Five separate experiments were conducted, where the traffic load was increased in each subsequent experiment. Figure 5.15 demonstrates the normalized histogram of wormhole delays obtained in an experiment with no traffic. The PSDs obtained in these experiments are presented in Figure 5.16.[20]

[20]In this simulation, the following parameters were used:

- OLSR HELLO message interval $K = 1$ second

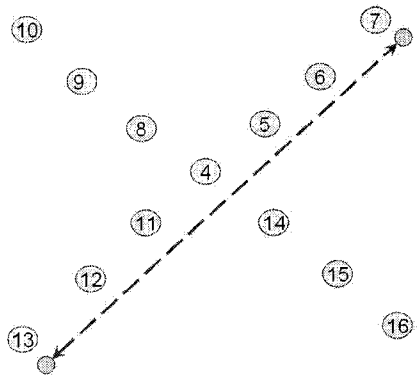


Figure 5.14: The NS-2 MANET used in this experiment has a 'star' shape, as shown here. The wormhole attackers 'connect' opposite star rays (nodes 13 and 7)

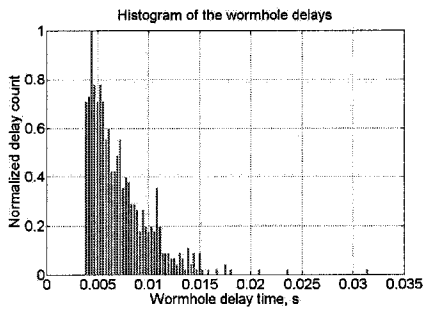
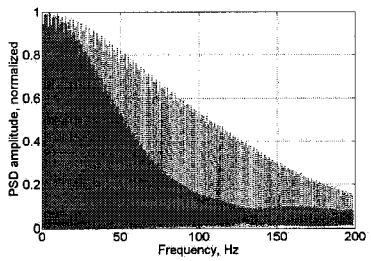
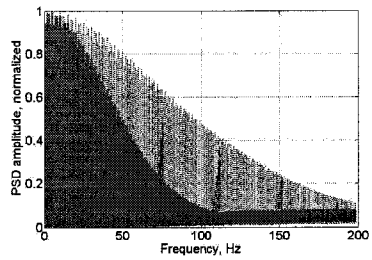


Figure 5.15: Normalized histogram of the wormhole delays obtained in an NS-2 experiment (without traffic)

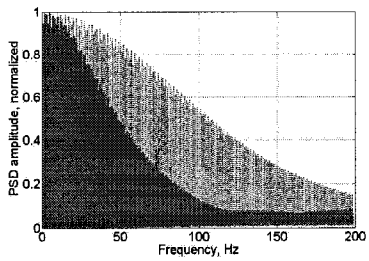
- Maximum HELLO message jitter $R_{max} = \frac{K}{100}$ seconds
- Wired links that connect wormhole attackers have a bandwidth of $1Mb$ and a fixed pre-set latency of $1ms$.
- The delay introduced by a wormhole is exponentially distributed with mean 0.003 seconds. The total wormhole delay is the sum of link latency and the random wormhole delay.
- 150 packets were used to create each individual PSD.
- Sampling frequency $F_s = 1000$ Hz was used for PSD profile creation.



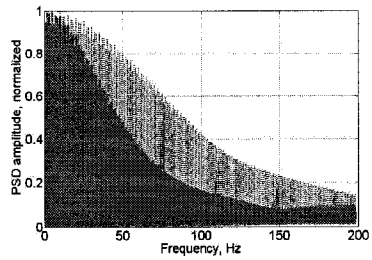
(a) The PSDs obtained in the 'baseline' experiment with no traffic load on the network (except for routing messages)



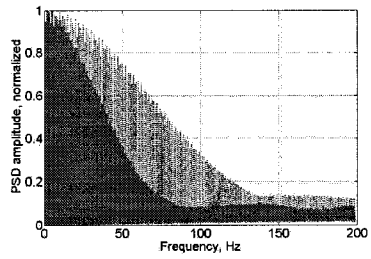
(b) The PSDs obtained in the experiment with 3 traffic flows of 0.01Mb each



(c) The PSDs obtained in the experiment with 5 traffic flows of 0.01Mb each



(d) The PSDs obtained in the experiment with 3 traffic flows of 0.1 MB each and 2 traffic flows of 0.01Mb each



(e) The PSDs obtained in the experiment with 5 traffic flows of 0.1 MB each

Figure 5.16: The PSDs obtained from the NS2 wormhole attack simulation data. The dashed line shows the PSDs of valid stations, while the thick line shows the intruders PSD. Figure 5.16(a) shows the 'baseline' case. In this experiment, no additional traffic (in addition to the routing traffic) was introduced on the network. The other figures show the PSDs that were obtained when traffic was introduced to the network. It can be seen that as traffic increases, the PSD profiles of both valid nodes and intruders decrease. However, the PSD profile of an intruder is always attenuated with respect to the PSD profile of a valid station.

Figure 5.16 shows the PSDs that were obtained in the NS2 experiments with different traffic loads (increasing loads from one experiment to the next). Figure 5.16(a) shows the least amount of traffic (routing traffic only), while Figure 5.16(e) demonstrates the PSDs that were obtained when 5 UDP flows of 0.1 Mb each were introduced to the network. It is clear in Figure 5.16 that the addition of traffic does cause PSD profiles of valid nodes to decline. However, note that with the highest traffic load considered (Figure 5.16(e)), the PSD profile of a valid station is less attenuated than the PSD profile of an intruder with the least amount of traffic (Figure 5.16(a)). However, since traffic does cause the decline in the PSD profiles of valid nodes, secondary checks may be necessary to distinguish the wormhole attackers from valid nodes that are burdened by heavy traffic loads.

It is interesting to note, also, that the intruder's PSD declines with the increase in traffic load. This makes sense: the more data a wormhole has to tunnel, the more delays it ends up introducing, and the more random these delays become.[21]

5.4 Detection avoidance

It is well known that security based on obscurity is bad security [8]. When implementing a security technique, one cannot rely on attackers being ignorant (i.e. not using the fact that a given security mechanism is in place). In this section, it is assumed that wormhole attackers trying to attack a network know that they are being monitored with the techniques described in this chapter. In this section, strategies that attackers may try to avoid being detected are described. Section 5.4.1 discusses what implementation choices would be most

[21] In [42], Chiu and Lui also noticed that wormhole delay increases as wormhole attracts more traffic.

suitable for attackers. Section 5.4.2 shows how attackers may try to avoid being detected with 'clever' mathematical manipulations.[22]

5.4.1 Changing attack implementation strategy

Knowing that random offsets in packet periodicity are being watched, attackers could change their attack implementation strategy in order to minimize the delay and/or minimize the randomness on the delay. Within a wormhole, roughly two packet-processing steps can be identified: packet transmission and packet processing. Below, the strategy that would help attackers to minimize the delays in these steps is provided.

To minimize randomness on packet transmission, attackers could take a great care in selecting the kind of link they use to communicate between each other. Previously, it was mentioned that if attackers are sending the packets, encrypted, over a covert multi-hop tunnel within the network itself (a so-called 'in-band' wormhole), then the randomness of the wormhole delay is increased significantly as in this case there are many more nodes processing wormhole packets. Thus wormhole attackers, trying to avoid being detected, may avoid an 'in-band' link, opting out for an off-channel link instead. In the experiments described in section 5.3, the attackers used a direct wired link to communicate. In a real network, however, a link between attackers is not likely to be a direct wire running from one attacker to the other (that could, after all, require a very long wire) except for some special cases of 'short-distance' wormholes or wormholes in certain sensor network scenarios. As running a wire is not a practical solution in many cases, attackers could choose other forms of links instead. It may be possible for attackers to use the Internet (or some other wired network), a

[22] When a paper based on this thesis [39] was refereed, the questions about the possibility of attackers avoiding being detected by 'clever' manipulation of their processing delays were raised. This section addresses — and dismisses — these concerns.

satellite connection, or a high-power off-band wireless connection. If a satellite or an Internet link is used, there would be additional processing of packets at each of the intermediate nodes (on the satellite or on each Internet router), and the randomness on the delay would be much higher than what can be achieved with a direct link between two attackers. A high-power off-band wireless link would be a better choice for attackers as no additional processing of packets would be involved.

When choosing a link, attackers not only select the medium for their tunnel, they also select link parameters. Obviously, it would be better for attackers to have a dedicated, rather than shared and contested, link, and a full-duplex link (where both connected nodes can transmit at the same time) would be better than a half-duplex (where only one of the nodes transmits at any given time). A high quality, high-bandwidth link would help attackers to minimize the transmission delays in cases where the attackers end up forwarding large amounts of traffic. When attackers cannot afford to have a dedicated high-bandwidth link, they may use QoS techniques (if possible) to ensure that their messages in general and HELLO messages in particular are treated as priority. [23]

In terms of minimizing packet processing, first of all, a dedicated attacker – one that only does wormhole-related activities, and nothing else – is better than an attacker that has other tasks. Also, extremely resourceful attackers can minimize packet processing by avoiding the use of TCP/IP when communicating with each other. If they are connected by a dedicated link, they may perhaps come up with a simplified protocol that would not require packet encapsulation and decapsulation, or even with a protocol that will allow them to send each packet bit-by-bit [4].

[23]Note that HELLO messages may be treated as priority only if HELLO messages can be identified by attackers. Further discussion on identifying HELLO packets is provided in chapter 4.

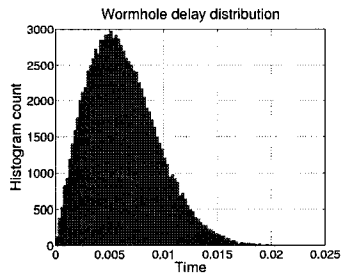
With these techniques attackers can minimize, but not fully remove, delay randomness. Also, a need to implement these attack improvement measures further increases the cost of an already costly and complicated wormhole attack [6].

5.4.2 Mathematical approaches to wormhole compensation

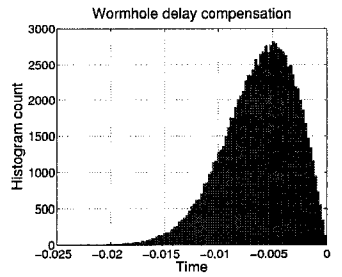
In addition to the network-related approaches described above, attackers could also use mathematical techniques to try to defeat the proposed attack detection scheme, that is, they could try to minimize the wormhole delay randomness by mathematical rather than physical means. When attackers have processed a packet, the time has passed, and cannot be turned back. By the time a second intruder receives a packet and is ready to send it back to the network, the packet is already delayed, the intruder cannot turn back the time.

Attackers can wait for an interval equivalent to a HELLO message period (K) and release a packet only after that interval has passed. This kind of processing would give the attackers some maneuvering space as they can 'undelay' a packet by μ if they send it $K - \mu$ seconds after the packet reception. Note, however, that holding a particular packet for that long would be immediately obvious from packet sequence numbers, and would be apparent even with loose synchronization of the nodes. Nonetheless, it is still interesting to see how the attackers could try to defeat this wormhole attack detection technique. [24]

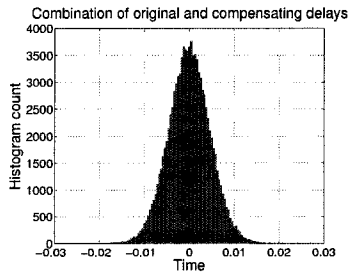
[24] This is done for completeness. For example, if a different wireless protocol is in use that does not rely on packet sequence numbers.



(a) A wormhole delay histogram. The delay this wormhole introduces is Rayleigh, with mean 0.005 seconds and variance $\sim 0.0106 * 10^{-3}$.



(b) A histogram of the 'compensating' delay. Trying to avoid being detected, the attackers, knowing that the delay they are introducing is Rayleigh with the specified mean and variance, can add this 'negative' delay, also Rayleigh with the same parameters, to the times they release their packets.



(c) The histogram of the delay that is achieved by combining a wormhole delay and a compensating delay presented above. The total delay is approximately Gaussian, with mean 0, and variance $\sim 0.0212 * 10^{-3}$.

Figure 5.17: Histograms of a sample 'wormhole delay', a compensating delay (a delay with the same parameters as the original delay, but negative), and the resulting combined delay. Since the attenuation of the PSD is caused by the 'randomness' of the delay rather than the actual delay mean, the attackers would not be able to avoid detection by 'subtracting' the delay distribution from the packet times.

Let us say that wormhole attackers are aware that they are introducing a delay, and know the statistical characteristics of this delay. If the delay mean is α, the attackers may try to release a packet $K - \alpha$ seconds after they receive it. This, however, would be pointless as the delay shape would stay exactly the same, only the delay mean would get shifted around (and the *mean* does not matter in PSD comparisons). For the next level of sophistication, attackers may try to compensate for their delay by introducing a 'negative' delay with the same distribution. Let us say that the attackers know that their distribution is $\sim D(\mu)$. The second attacker can then hold on to the packets it receives, and release them at $K - D(\mu)$ seconds after the reception time. In this way, the attacker would 'subtract' the distribution. However, this does not remove randomness. This processing changes the shape of the distribution and makes the mean of the resulting distribution zero, but does not get rid of the statistical distribution. In fact, the variance – which could be used as a measure of 'randomness' – of the combined delay is higher than the variance of the original delay (as the variance of the sum of two independent random variables is equal to the sum of their variances). An example of this is shown in Figure 5.17. The original wormhole delay shown in Figure 5.17(a) is Rayleigh with mean 0.005 seconds. The 'compensating' distribution is a Rayleigh with the same parameters, but it is negative (a packet is released $K - R(0.005)$ seconds later). The total delay of this processed packet (minus K) is shown in Figure 5.17(c). This resulting delay is no longer Rayleigh (it is actually well-approximated by a Gaussian), and its mean is zero. However, the resulting delay is more distributed. While the variance of the original Rayleigh is $\sim 0.0106 * 10^{-3}$, the variance of the combined delay is $\sim 0.0212 * 10^{-3}$. [25] It is clear that the subtraction of the statistically equal delay would not help attackers to avoid being detected.

[25] These results were obtained with a MATLAB simulation, which used $n = 100,000$ measurements to calculate the variance of a sample

So, this way of compensation does not help attackers: once the randomness is introduced to the packets, it is difficult to remove it. Knowledge of the delay distribution does not help attackers to remove the randomness. However, if the attackers know the actual delay they are introducing to a particular packet (which could be possible if attackers are precisely synchronized), then they can remove the randomness. Suppose the first attacker receives a packet at time t_0. When sending the packet on to the second attacker, the first attacker would include the timestamp t_0 with the packet. The second attacker, having processed the packet, would then be able to send it to the network exactly K seconds after t_0. This requires precise synchronization between attackers, and, as noted above, would still be evident if sequence number checking or some loose time synchronization is enabled on the network.

5.5 Discussion

In this chapter, a novel wormhole attack detection technique based on frequency analysis of periodic HELLO messages is described. This technique is based on the fact that wormhole attackers add small random delays to the HELLO messages they are retransmitting and the presence of such small random delays can be easily detected with frequency analysis of HELLO message periodicity.

With this technique, it is possible to detect wormhole attackers that do not drop traffic and do not disrupt network operations – so-called 'dormant' wormholes. Moreover, note that this technique does not rely on distant physical separation of wormhole attackers like the GPS-based techniques. The GPS-based techniques detect that a packet has travelled a long distance; this technique detects that a packet has been additionally processed. Therefore, unlike GPS-based techniques, this technique is capable of detecting 'short-range' wormholes [5]. It is believed that this is the first published technique that can

detect both short-range and long-range wormholes (i.e. range-independent), and also first that can detect dormant wormholes without introducing overhead or changes to the existing protocols.

In section 5.4, it was suggested that the attackers that are tightly synchronized may avoid being detected by this technique if they hold on to the HELLO messages so that the periods between the HELLO messages they are releasing are exactly what they should be. It should be stressed that having to be that tightly synchronized is costly for attackers. However, it is possible. Fortunately, these actions of attackers are easily detectable – and are, indeed, immediately obvious – by either packet sequence number checking or by having network nodes loosely synchronized and aware of possible wormhole-induced delays on HELLO messages. Therefore, one of these techniques should be incorporated in a network that relies on this wormhole attack detection technique. Sequence number checking is trivial and is very easy to implement; if network nodes are loosely synchronized, the implementation of delay detection is trivial as well.

In section 5.2.2, the sinusoidal jitter idea was discussed. This form of jitter acts as a carrier frequency, introducing high-frequency components to the PSDs of valid stations. With this jitter, the wormhole delay would be immediately obvious from the PSDs, even if very few packets are used for PSD creation. The sinusoidal jitter appears to be a promising research direction. Such jitter can be easily implemented within an NS-2 simulation where its effect on network performance can be studied, and its effectiveness in discovery of wormhole attacks can be demonstrated. Further work, built on the foundation of the tools and the analysis preseted in this thesis, is being pursued at DRDC.

Section 5.3.2 described the PSDs that were obtained in an NS-2 simulator with some traffic load. These NS-2 experiments demonstrate that the PSD profiles of valid stations do decline when the amount of traffic on a network is

increased, but the PSD profiles of intruders decline even further as well. This demonstrates that simple secondary checks may be required to distinguish a node overloaded with traffic and a node affected by wormhole attackers.

An extension of the NS-2-based study of this wormhole attack detection technique is an important future work direction. The NS-2 simulations with more traffic, and more diverse traffic, can be considered. Within the simulator, it would be possible to implement and study the secondary checks and their effectiveness in distinguishing an overloaded node and a wormhole attacker. Also, it can be examined whether assigning HELLO messages a 'priority' status would help deal with the attenuation of PSD profiles caused by traffic loads.

Chapter 6

Summary and future work directions

This thesis deals with wormhole attack discovery in mobile ad hoc networks. The contributions of this thesis to the field of ad hoc network security research are the tools that were created as part of this work and the wormhole attack discovery techniques. The developed wormhole attack discovery techniques do not rely on specialized hardware and do not require clock synchronization. Moreover, these techniques are local, and do not introduce overhead; instead, they work with routing messages that are already present on a network.

One of the tools that was developed in this thesis is a network traffic analyzer's MANET suite, which allows the network traffic analyzer to easily work with 802.11 data and OLSR routing messages, and also has some ad-hoc-network-specific functionality. This tool will allow researchers wanting to create their own attack detection or attack analysis modules to do so easily, without having to implement low-level functionality related to ad hoc networks. In chapter 3, the functionality of this tool is described, and a small case study

demonstrating its effectiveness is discussed.

In this thesis, a realistic NS-2 wormhole attack simulation was also developed. Although a number of researchers have been working with wormhole attacks, a wormhole attack simulation was neither presented nor described by any of them. The developed NS-2 wormhole attack simulation is described in chapter 3; its effectiveness is demonstrated in appendix A. This wormhole attack simulation will allow researchers to easily modify wormhole attack parameters and to test their own solutions to wormhole attacks.

A wormhole attack detection technique suitable for detecting wormholes that drop network traffic is developed in this thesis, and is presented in chapter 4. It is a simple, ready-to-implement technique that is based on detection of wormhole-induced loss of periodic network messages. It introduces no overhead, does not require changes to routing protocols, and has no specialized hardware requirements. This technique is similar in spirit to the attack detection approaches based on counting the number of packets that go through a link. However, unlike these techniques, the protocol-breaking technique presented in chapter 4 uses protocol-implied number of routing messages as side information, therefore eliminating the need for acknowledgements, cooperation between nodes, overhearing other node's forwarding, etc.

This thesis also presents an approach to wormhole attack detection in wireless ad hoc networks based on frequency analysis of periodic routing messages. This approach is novel: this technique discovers that a packet has been 'over-processed', while other wormhole attack discovery techniques work by discovering that a packet has travelled too far/too fast. This approach, presented in chapter 5, works because wormhole attackers inevitably add small random delays to the messages they are processing. Like the protocol-breaking technique, this technique relies on periodic routing messages that are readily available on

the network, and does not generate any extra traffic. In chapter 5, this technique was examined mathematically, and was shown to work on testbed experimental data as well as on the data generated by the NS-2 wormhole attack implementation. Different ways for attackers to avoid being detected were discussed in chapter 5, and it was shown that the described avoidance mechanisms would not be able to defeat this attack detectiom technique. This is a powerful detection method because it can detect dormant wormholes and because it does not couple intrusion detection to connectivity/network performance.

6.1 Future work directions

Certain work can be done in terms of extending the tools – the network traffic analyzer and the wormhole attack simulation – that were developed in this thesis. The network traffic analyzer tool was made capable of working with IEEE 802.11 and with OLSR; it can be extended to work with other MAC-layer and routing-layer protocols. The NS-2 implementation of the wormhole attack can be extended to include different links, to work with different routing protocols, etc.

Chapters 4 and 5 present a framework, the techniques that can be used for wormhole attack detection. Real-life implementation of these techniques would be a very interesting project. Such an implementation would allow one to set specific thresholds and specify exact number of packets required for reliable wormhole attack detection, examine the false-positive and false-negative wormhole attack detection results, and study the effectiveness of secondary checks in reducing false-positives and false-negatives.

In chapter 5, the idea of using sinusoidal jitter to ease wormhole attack detection was presented. It seems promising because it allows one to discover wormholes using very few HELLO packets – that is, achieves low wormhole

attack discovery latency. Further mathematical analysis (as well as a real-world implementation) of this technique would be an exciting research direction.

Finally, the work in the area of wormhole attack detection was done with proactive routing protocols that rely on periodic HELLO messages. When such protocols are used, there is no need to send any additional messages for wormhole attack discovery. However, the work described in chapters 4 and 5 equally applies to any other periodic messages on a network. For the networks that use reactive routing protocols, a separate wormhole attack discovery protocol may be created. Note, that such protocol can be based on simple ICMP *ECHO REQUESTs* – pings, that are, by default, periodic with period 1 second, and their periodicity can be set by a user [43]. Creation of a separate wormhole attack discovery protocol independent of routing protocol would be another interesting project; the framework for such protocol is presented in this thesis and the implementation should be relatively simple.

Appendix A

NS-2 wormhole in action

Chapter 3 describes the NS-2 wormhole attack simulation that was designed and developed in this thesis. This appendix shows with trace files and network nodes' routing tables how the wormhole attack simulation works. Since the wormhole is 'observed' through simulation trace files, the NS-2 trace formats are first briefly explained in section A.1. Section A.2 shows how the routing tables of network nodes are affected when a wormhole is introduced to the network. Finally, section A.3 demonstrates how a packet travels through a wormhole (as observed in the NS-2 traces).

A.1 NS-2 trace format

There are several trace formats that NS-2 uses, often simultaneously. There are 'old' and 'new' trace formats, as well as changes to formats from one NS-2 release to the next. In addition, different trace formats are used in wired and wireless simulations [44]. In this wormhole attack simulation, where both wired and wireless packets are used, both wired and wireless trace formats are encountered. In this section, the NS-2 traces that are relevant to the wormhole

attack simulation are discussed; for more details, readers are encouraged to consult [44].[1]

The following is an example of a 'wireless' NS-2 trace line:[2]

```
s 22.002612476 _5_ MAC  --- 340 OLSR 240 [0 ffffffff 5 800]
                        ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
```

In order of appearance, the following information can be obtained from this trace:

- **s**: The first entity in the trace shows the 'event' that happened. In the line above, the event is 's', which stands for 'send'. Other possible events are 'r', 'f', and 'D', which stand, respectively, for 'receive', 'forward', and 'drop' [44].

- **22.002612476**: The second entry is the timestamp of when an event has happened (where the time is in seconds). These are simulated, NS-2 event timestamps, and they are not related to the time it took the simulator to get to an event.

- **_5_**: The unique address of the NS-2 node that generated this event. In NS-2, the unique ID of a node is the same as the node IP address. It may be interesting to note that this NS-2 unique ID of a node is not assigned by a user in a script — rather, NS-2 itself assigns these addresses.

- **MAC**: This entry indicates the level on which the event is recorded. In NS-2, when a test script is created, the activated tracing level is selected, for example, MAC (Medium Access Control) or RTR (routing layer). This field is important and cannot be ignored. For example, a packet 'received' on a MAC level is not the same as the packet being 'received' by a routing

[1] Further information on the OLSR trace format is provided in [45]
[2] This is an 'old' wireless trace format [44]

agent, as a packet received on a MAC level won't necessarily go up to the router.

- **340**: A unique packet ID, assigned to a packet by NS-2. This packet ID stays the same no matter how much a packet is altered when it goes through different layers of different nodes.
- **OLSR**: This is an OLSR packet.
- **240**: Packet size, in bytes.
- **[0 ffffffff 5 800]**: MAC-level packet information, in hexadecimal format: the packet MAC-duration, the Ethernet address of the destination (broadcast in this case), the Ethernet address of the packet source, and finally the packet type (where 800 in hexadecimal stands for IP).
- **[5:255 -1:255 32 0]**: IP-level packet information: IP address of the source, source port address, IP destination address (where -1 stands for broadcast), IP destination port, packet's Time To Live, and next hop destination (where applicable; here it is zero as it is not applicable).
- **[1 30 [HELLO 5 0 14]]**: OLSR-specific information: the number of OLSR messages in this OLSR packet (1), the sequence number of the OLSR packet (30), and the message information: this is an OLSR HELLO message, sent out by node 5, with hop count zero and sequence number 14 [45].

In this work, the 'wired' trace format is also encountered. An example of an NS-2 trace line in the wired format is presented below:

```
r 22.015102 2 3 OLSR 188 ------- 0 5.255 -1.255 -1 340
```

In this line, the following information can be identified, in order of appearance:

- **r**: An event that had happened, where 'r' stands for 'receive', similar to the wireless trace. In addition to the events indicated for wireless traces, the 'event' entry could also be '+' and '-' — the signs indicating a packet being enqueued/dequeued on a wired link.
- **22.015102**: The timestamp of the event
- **2 3**: Shows to whom and from whom the packet came. In this case, node 3 has received a packet that was sent by node 2. Node 2 is not necessarily an originator of this packet — rather, its the packet's previous-hop.
- **OLSR**: This is an OLSR packet.
- **188**: Packet size, in bytes.
- **0**: Packet flow ID.
- **5.255 -1.255**: The IP-level packet information. This packet was sent from port 255 of node 5 to the port 255 of a broadcast destination (IP address -1).
- **-1** : Packet sequence number.
- **340**: This is the unique packet ID. In wireless traces, the packet ID is mixed in the middle; in wired traces, the packet ID is at the very end of the line.

A.2 Wormhole's effect on routing

The wormhole attack was tested on a number of network layouts and it was observed that the wormhole was working correctly: with a wormhole present, nodes that were far away considered themselves to be neighbours. In this section, a wormhole's effect on a particular network setup is demonstrated.

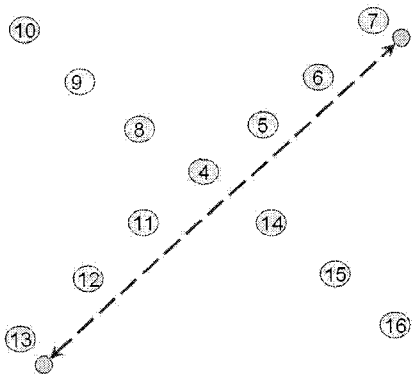


Figure A.1: A 'star' network layout with a wormhole connecting opposite ends of rays. A network, created with NS-2, has the 'star' shape with 4 rays. A wormhole 'connects' nodes 13 and 7.

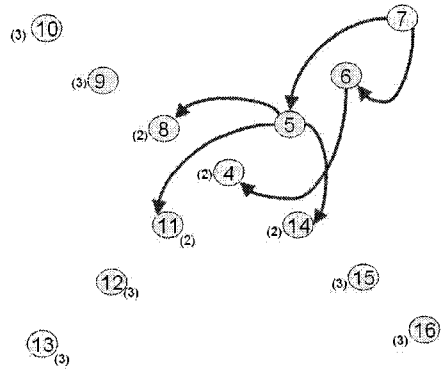


Figure A.2: Graphical representation of node 7 routing table (no wormhole present). The number of hops is shown in brackets; links in the two-hop region of node 7 are shown by arrows

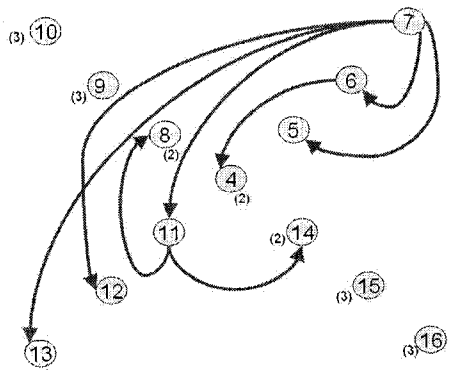


Figure A.3: Graphical representation of node 7 routing table with a wormhole 'connecting' nodes 7 and 13. Nodes as distant as 13, 12, and 11, node 7 considers to be direct neighbours. Note, that communication with nodes 14 and 8 also goes through the wormhole, as node 7 considers them to be 2 hops away (going through node 11).

To study the effect of a wormhole attack, a wormhole is introduced to a 'star' layout network, which is shown in Figure A.1. In this layout, the nodes are far enough from each other to necessitate multi-hop communication. For example, when the wormhole is not activated, the routing table of node 7 (shown in Table A.1 and graphically represented in Figure A.2) shows that node 7 is 3 hops away from nodes 13, 10, and 16.

Destination	Next	Distance
4	6	2
5	5	1
6	6	1
8	5	2
9	6	3
10	5	3
11	5	2
12	6	3
13	5	3
14	5	2
15	6	3
16	5	3

Table A.1: Routing table of node 7 (from Figure A.1), without a wormhole. Graphical representation is shown in Figure A.2.

Destination	Next	Distance
4	6	2
5	5	1
6	6	1
8	**11**	2
9	6	3
10	**11**	3
11	**11**	**1**
12	**12**	1
13	**13**	**1**
14	11	2
15	6	3
16	11	3

Table A.2: Routing table of node 7 with a wormhole present. Affected routes are shown in bold. Graphical representation is shown in Figure A.3

As shown in Figure A.1, the created wormhole 'connects' nodes 13 and 7. The routing state of node 7 with wormhole activated is shown in Table A.2 and graphically in Figure A.3. With a wormhole present, node 7 now believes nodes 12 and 13, which are both 3 hops away in reality, to be its direct neighbours. This demonstrates that the NS-2 wormhole attack does 'work' in a sense that it results in predicted routing changes.

A.3 A packet going through the wormhole: NS-2 trace

Previously, the schematics of NS-2 wormhole attack model were explained (section 3.2.2). This section shows in detail how a packet travels through a wormhole; a packet's path is demonstrated in the NS-2 trace.

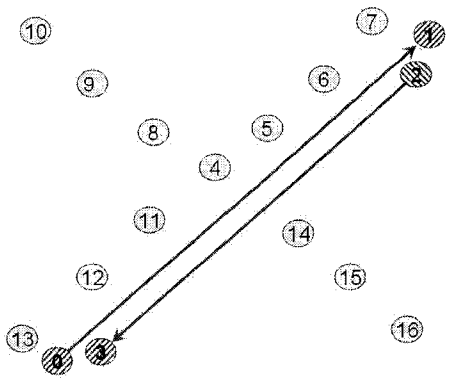


Figure A.4: A star topology from figure A.1 with wormhole attackers identified. In routing tables coming out of the star topology (figures A.1, A.3, A.2) attacking nodes are not present. They are, in essence, invisible. In the NS-2 trace, however, the attackers can be observed. In this star topology, the attackers 0 and 3(a sink and a source) are placed next to node 13, while the other sink and source (nodes 1 and 2) are placed next to node 7.

Below, a part of NS-2 trace is provided, which shows how a packet is transferred through the developed NS-2 wormhole. This trace was obtained from the 'star topology' network simulation described in section A.2, with the wormhole attack activated. The star topology, including the intruders present on the network, is shown in Figure A.4. Figure A.4 shows the intruders: nodes 0 and 2 are sinks, they collect the information from the network and send it on a wired link. Nodes 1 and 3 are the sources that replay the information to the network. The trace below shows how a HELLO message with unique ID 340, created by node 5, is captured by wormhole attackers and replayed at the other end of the

network:

```
[1]   s 22.002612476 _5_ MAC  --- 340 OLSR 240 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[2]   r 22.003572869 _4_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[3]   r 22.003572870 _6_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[4]   r 22.003573032 _8_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[5]   r 22.003573036 _14_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[6]   r 22.003573269 _7_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[7]   r 22.003573272 _11_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]

[8]   D 22.003573306 _1_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[9]   r 22.003573307 _2_ MAC  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[10]  r 22.003597869 _4_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[11]  r 22.003597870 _6_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[12]  r 22.003598032 _8_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[13]  r 22.003598036 _14_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[14]  r 22.003598269 _7_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[15]  r 22.003598272 _11_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[16]  + 22.003598 2 3 OLSR 188 ------- 0 5.255 -1.255 -1 340
[17]  - 22.003598 2 3 OLSR 188 ------- 0 5.255 -1.255 -1 340
      .
[18]  r 22.015102 2 3 OLSR 188 ------- 0 5.255 -1.255 -1 340
[19]  r 22.015102307 _3_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 31 0] [1 30 [HELLO 5 0 14]]
[20]  f 22.015102307 _3_ RTR  --- 340 OLSR 188 [0 ffffffff 5 800] ------- [5:255 -1:255 31 0] [1 30 [HELLO 5 0 14]]
[21]  s 22.015177307 _3_ MAC  --- 340 OLSR 240 [0 ffffffff 3 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[22]  D 22.016137306 _0_ MAC  --- 340 OLSR 188 [0 ffffffff 3 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[23]  r 22.016137370 _13_ MAC  --- 340 OLSR 188 [0 ffffffff 3 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[24]  r 22.016137767 _12_ MAC  --- 340 OLSR 188 [0 ffffffff 3 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[25]  r 22.016162370 _13_ RTR  --- 340 OLSR 188 [0 ffffffff 3 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
[26]  r 22.016162767 _12_ RTR  --- 340 OLSR 188 [0 ffffffff 3 800] ------- [5:255 -1:255 32 0] [1 30 [HELLO 5 0 14]]
```

Lines 1-26 above show a single packet traveling through a wormhole. The first line shows node 5 sending a HELLO message with NS-2 unique ID 340. Then, valid nodes 4,6,8,14,7,and 11 receive the message on the MAC level - i.e, their wireless cards register that message. Line 8 indicates that node 1 (one of the source intruders), also receives this message, but the message is dropped (indicated by letter *D* at the beginning of line 8), as it should be, since node 1 is an intruder source node, and source nodes are supposed to drop all packets they receive wirelessly. Line 9 shows that node 2 (a wormhole sink intruder) has received the HELLO message sent by node 5. The following lines (10-15), demonstrate that nodes that previously registered the message on the MAC level, transfer it to the router (OLSR) level.

Lines 16 through 18 use wired trace formats. Line 16 is an enqueuing event:

it indicates that a packet was added to a queue on the link between nodes 2 and 3. Line 17, respectively, shows that a packet was removed from that link. Line 18 shows that the packet was received by node 3 (the source intruder on the opposite network end) on the wired interface. Note, that in the process of capturing and retransmitting this packet, its IP-level (and higher-level) information was not altered — as noted before, the wormhole has to resend the packets unaltered.

Once the intruder source node 3 receives the message on the wired side, it has to forward it wirelessly. Line 19 indicates that a packet was transferred to the routing level of node 3, while line 20 shows node 3 forwarding this message (indicated by letter f), as it should. Line 21 shows that node 3 sends this message, and the following lines show that the message is received by nodes 0, 13, and 12. Note, that while nodes 13 and 12 forward the packet to the router level (lines 25 and 26), an intruder sink node 0 drops it (line 22), as it should.

Appendix B

List of acronyms

AES	Advanced Encryption Standard
AODV	Ad-hoc On-Demand Vector routing
ARP	Address Resolution Protocol
CPU	Central Processing Unit
CRC	Communications Research Centre
DRDC	Defence Research and Development Canada
FTP	File Transfer Protocol
GPS	Global Positioning System
HMTI	HELLO Message Time Interval
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers

IP	Internet Protocol
LAGN	Location-Aware Guard Node
LL	Link Layer
LLC	Logical Link Control
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
MNE	Mobile Network Emulator
MPR	Multipoint Relay
NIO	Network Information Operations
NTA	Network Traffic Analyzer
OLSR	Optimized Link-State Routing
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PSD	Power Spectral Density
QoS	Quality of Service
RTT	Round Trip Travel Time
SIFS	Short Interframe Space
TC	Topology Control
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

WEP Wired Equivalent Privacy

WPA Wi-Fi Protected Access

Bibliography

[1] H. Yang, H. Y. Luo, F. Ye, S. W. Lu, L. Zhang, Security in Mobile Ad Hoc Networks: Challenges and Solutions, IEEE Journal of Wireless Communications, Vol. 11, Issue 1, pp. 38-47, February 2004

[2] Y.-C. Hu, A. Perrig, A Survey of Secure Wireless Ad Hoc Routing, IEEE Security and Privacy Magazine, Vol. 2, pp. 23-39, May 2004

[3] A. Mishra, K. Nadkarni, A. Patcha, Intrusion Detection in Wireless Ad Hoc Networks, IEEE Wirelss Communications Magazine, Vol. 11, pp. 48-60, February 2004

[4] Y.-C. Hu, A. Perrig, D. B. Johnson, Packet leashes: a defense against wormhole attacks in wireless networks, Proceedings of 22 Annual Joint Conference of the IEEE Computer and Communication Societies INFOCOM 2003, Vol. 3, pp. 1976-1986, April 2003

[5] Y.-C. Hu, A. Perrig, D. B. Johnson, Wormhole Attacks in Wireless Networks, IEEE Journal on Selected Areas of Communications, Vol. 24, Issue 2, pp. 370-380, February 2006

[6] P.Ebinger, T. Bucher, Modelling and Analysis of Attacks on the MANET Routing in AODV, Proceedings of Ad Hoc Now 2006, pp. 294-307, Ottawa, Ontario, Canada, August 2006

[7] Medium Access Control (MAC) and Physical (PHY) Specifications. ANSI/IEEE Std 802.11, 1999 Edition.

[8] B. Schneier, Secrets and Lies: Digital Security in a Networked World, John Wiley & Sons, 2000

[9] Ethereal: A Network Protocol Analyzer, `www.ethereal.com/`, acessed on April 02, 2007

[10] WildPackets(tm) AiroPeek Family Overview, `www.wildpackets.com/products/airopeek/overview`, acessed on April 02, 2007

[11] Ad Hoc On-Demand Distance Vector (AODV) routing; available at `www.ietf.org/rfc/rfc3561.txt`, acessed on April 02, 2007

[12] IETF draft, Optimized Link State Routing (OLSR) protocol, RFC 3626.

[13] Optimized Link State Routing Protocol, Luceor S.A.S.; available at `www.luceor.com/article.php3?id_article=48`, acessed on April 02, 2007

[14] OLSR networks– a (slightly) biased MANET tutorial, T. H. Clausen; available at `www.soi.wide.ad.jp/class/20030000/slides/05/index_73.html`, acessed on April 02, 2007

[15] NS-2, `www.isi.edu/nsnam/ns`, acessed on April 02, 2007

[16] OPNET, `www.opnet.com`, acessed on April 02, 2007

[17] GloMoSim, pcl.cs.ucla.edu/projects/glomosim, acessed on April 02, 2007

[18] IPAM Tutorial: Network Modeling and Traffic Analysis with ns-2, J. Heidemann, P. Huang; available at www.isi.edu/nsnam/ns/ns-tutorial/ipam_0203_john.pdf, acessed on April 02, 2007

[19] NS Simulator for beginners, E. Altman, T. Jimenez; available at www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf, acessed on April 02, 2007

[20] Developing the Next-Generation Open-Source Network Simulator (ns-3), T.R.Henderson and Sumit Roy; available at www.isi.edu/nsnam/ns-3/ns-3-cri-workshop-2006.pdf, acessed on April 02, 2007

[21] The NS manual, K. Fall and K. Varadham; available at www.isi.edu/nsnam/ns/doc/ns_doc.pdf, acessed on April 02, 2007

[22] W. Wang, B. Bhargava, Y. Lu, X. Wu, Defending against Wormhole Attacks in Mobile Ad Hoc Networks, Wireless Communication and Mobile Computing, Vol. 6, Issue 4, pp. 483-503, June 2006

[23] S. Capkun, L. Buttyan, J.-P. Hubaux, SECTOR: Secure Tracking of Node Encounters in Multi-Hop Wireless Networks, Processings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks, October 2003

[24] Y.-C. Hu, A. Perrig, D. Johnson, Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols, Proceedings of

the 2003 ACM workshop on Wireless security (WiSe 2003), San Diego, California, September 2003

[25] F. Hong, L. Hong, C. Fu, Secure OLSR, Proceedings of International Conference On Advanced Information Networking and Applications (AINA 2005) Vol. 1, pp. 713-718, March 2005

[26] T. Korkmaz, Verifying Physical Presence of Neighbours against Replay-based Attacks in Wireless Ad Hoc Networks, Proceedings of the International Conference On Information Technology: Coding and Computing (ITCC 2005), pp. 704-709, Las Vegas, Nevada, April 2005

[27] A. Baruch, R. Curmola, C. Nita-Rotaru, D. Holmer, H. Rubens, On the Survivability of Routing Protocols in Ad Hoc Wireless Networks,Converence on Security and Privacy for Emerging Areas in Communications (SecureComm 2005), September 2005

[28] C. Chigan, R. Bandaru, Secure Node Misbehaviors in Mobile Ad Hoc Networks, Proceedings of the 60th Vehicular Technology Conference (VTC 2004), Vol. 7, pp. 4730-4734, September 2004

[29] L. Hu, D. Evans, Using Directional Antennas to Prevent Wormhole Attacks, In Proceedings of the 2004 Symposium on Network and Distributed Systems Security (NDSS 2004), February 2004

[30] L. Loukas, P. Radha, Serloc: Secure Range-Independent Localization for Wireless Sensor Networks, Proceedings of the ACM Workshop on Wireless Security, October 2004

[31] W. Wang, B. Bhargava, Visualization of wormholes in sensor networks, Proceedings of the ACM workshop on Wireless Security, October 2004

[32] L. Lazos, R. Poovendram, C. Meadows, P. Syverson, L.W. Chang, Preventing Wormhole Attacks on Wireless Ad Hoc Networks: a Graph Theoretical Approach, Proceedings of IEEE Wireless Communications and Networking Conference, 2005

[33] I. Khalil, S. Bagchi, N. B. Shroff, A Lightweight Countermeasure for the Wormhole Attack in Multihop Wireless Networks, Proceedings of the International Conference on Dependable Systems and Networks (DSN'05), 2005

[34] N. Song, L. Qian, X. Li, Wormhole Attack Detection in Wireless Ad Hoc Networks: a Statistical Analysis Approach, 2005, Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium IPDPS'05, April 2005

[35] Communictions Research center Canada (CRC), `www.crc.ca`, acessed on April 02, 2007

[36] Defence Research and Development Canada (DRDC) `www.drdc-rddc.gc.ca`, acessed on April 02, 2007

[37] The libpcap project; available at `sourceforge.net/projects/libpcap/`, acessed on April 02, 2007

[38] G. A. Di Caro, Analysis of simulation environments for mobile ad hoc networks,Technical Report No. IDSIA-24-03, December 2003, Dalle Molle Institure for Artifical Intelligence; available at `www.`

idsia.ch/idsiareport/IDSIA-24-03.pdf, acessed on April 02, 2007

[39] M.A. Gorlatova, P.C. Mason, M. Wang, L. Lamont, R. Liscano, Detecting Wormhole Attacks in Mobile Ad Hoc Networks Through Protocol Breaking and Packet Timing Analysis, in Proceedings of IEEE Military Communications (MILCOM), Washington, DC, October 2006

[40] Leon W. Couch, Digital and Analog Communication Systems, sixth edition, 2001, Prentice Hall Inc.

[41] C. M. Grinstead and J. L. Snell, Introduction to Probability, Second Edition, available at www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html, acessed on April 02, 2007

[42] H.S. Chiu, K.-S. Lui, DelPHI: Wormhole Detection Mechanism for Ad Hoc Wireless Networks, 1st International Symposium on Wireless Pervasive Computing, January 2006

[43] Linux Man page: ping; available at www.die.net/doc/linux/man/man8/ping.8.html, acessed on April 02, 2007

[44] NS-2 Trace Formats; available at nsnam.isi.edu/nsnam/index.php/NS-2_Trace_Formats, acessed on April 02, 2007

[45] Using UM-OLSR code; available at masimum.dif.um.es/?Software:UM-OLSR:Using, acessed on April 02, 2007