

University of Pretoria etd - Delport, J P (2007)

# Link Layer Topology Discovery in an Uncooperative Ethernet Environment

by

Johannes Petrus Delport

Submitted in partial fulfilment of the requirements for the degree

**Magister Scientia (Computer Science)**

in the

**Faculty of Engineering, Built Environment and  
Information Technology**

at the

**University of Pretoria**

**May 2007**

University of Pretoria etd - Delport, J P (2007)

# Link Layer Topology Discovery in an Uncooperative Ethernet Environment

by

Johannes Petrus Delport

## Abstract

Knowledge of a network's entities and the physical connections between them, a network's physical topology, can be useful in a variety of network scenarios and applications. Administrators can use topology information for fault-finding, inventorying and network planning. Topology information can also be used during protocol and routing algorithm development, for performance prediction and as a basis for accurate network simulations.

Specifically, from a network security perspective, threat detection, network monitoring, network access control and forensic investigations can benefit from accurate network topology information.

The dynamic nature of large networks has led to the development of various automatic topology discovery techniques, but these techniques have mainly focused on cooperative network environments where network elements can be queried for topology related information.

The primary objective of this study is to develop techniques for discovering the physical topology of an Ethernet network without the assistance of the network's elements.

This dissertation describes the experiments performed and the techniques developed in order to identify network nodes and the connections between these nodes. The product of the investigation was the formulation of an algorithm and heuristic that, in combination with measurement techniques, can be used for inferring the physical topology of a target network.

**Keywords:** Network Topology Discovery, Network Mapping, Network Security, Ethernet, Network Management, Network Monitoring

**Supervisor:** Professor Martin S. Olivier

**Department:** Department of Computer Science

**Degree:** Magister Scientia

# Acknowledgements

I would like to thank the following people for helping me persevere and complete this dissertation.

- Monya, for her patience and encouragement.
- Professor Martin Olivier, for his considerate and clear guidance.
- The CSIR for continued financial support and provisions for study time.

# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>List of Figures</b>	<b>viii</b>
<b>1 Overview and Objectives</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Problem Statement . . . . .	2
1.3 Methodology . . . . .	2
1.4 Overview . . . . .	3
<b>2 Background and Context</b>	<b>5</b>
2.1 Introduction and Overview . . . . .	5
2.2 Layered Network Design . . . . .	5
2.3 Relevant Protocols . . . . .	9
2.3.1 Address Resolution Protocol (ARP) . . . . .	9
2.3.2 Simple Network Management Protocol (SNMP) . . . . .	11
2.4 Relevant Ethernet Concepts . . . . .	11
2.4.1 Data Link Sublayers . . . . .	12
2.4.2 Hardware Addresses . . . . .	13
2.4.3 Frame Layout . . . . .	13

<b>Table of Contents</b>	<b>iv</b>
2.4.4 Duplex Modes . . . . .	14
2.4.5 Hubs and Switches . . . . .	16
2.5 Active versus Passive Scanning . . . . .	17
2.6 Conclusion . . . . .	18
<b>3 Related Work</b>	<b>19</b>
3.1 Introduction and Overview . . . . .	19
3.2 Applications of Topology Knowledge . . . . .	20
3.2.1 Administration and Planning . . . . .	20
3.2.2 Performance Prediction . . . . .	21
3.2.3 Algorithm and Protocol Design . . . . .	21
3.2.4 Simulation . . . . .	21
3.2.5 Security . . . . .	22
3.3 Network Layer Topology Discovery . . . . .	23
3.4 Link Layer Topology Discovery . . . . .	24
3.5 Conclusion . . . . .	27
<b>4 Passive Network Node Discovery</b>	<b>28</b>
4.1 Introduction and Overview . . . . .	28
4.2 Experiment Design and Execution . . . . .	29
4.2.1 Purpose and Procedure . . . . .	29
4.2.2 Target Network Selection . . . . .	29
4.2.3 Capture Hardware . . . . .	30
4.2.4 Capture Software . . . . .	30
4.2.5 Database . . . . .	31
4.2.6 Experiment Execution . . . . .	32
4.3 Initial Processing . . . . .	32

<b>Table of Contents</b>	<b>v</b>
4.3.1 Packet Selection . . . . .	32
4.3.2 Address Resolution Protocol (ARP) Packets . . . . .	33
4.3.3 Browser Announce Packets . . . . .	36
4.3.4 Organisationally Unique Identifier (OUI) Data . . . . .	38
4.4 Node List Generation . . . . .	39
4.4.1 Unique MAC and IP Pairs . . . . .	40
4.4.2 Unique MAC and Browser Name Pairs . . . . .	40
4.4.3 Unique Nodes Merged with OUI Data . . . . .	40
4.5 Additional Processing . . . . .	41
4.5.1 Multiple IP Addresses or Names . . . . .	42
4.5.2 Packet Timestamps . . . . .	43
4.5.3 Communication Peers . . . . .	45
4.6 Conclusion . . . . .	48
<b>5 Internal Network Device Discovery</b>	<b>49</b>
5.1 Introduction and Overview . . . . .	49
5.2 Experiment Design and Preparation . . . . .	50
5.2.1 Motivation . . . . .	50
5.2.2 Considerations . . . . .	50
5.2.3 Packet Selection . . . . .	51
5.2.4 Operating System Selection . . . . .	53
5.2.5 Hardware and Device Drivers . . . . .	53
5.2.6 Software . . . . .	54
5.3 Experiment Execution . . . . .	55
5.3.1 Network Configurations . . . . .	55
5.3.2 Packet Configurations . . . . .	57

<b>Table of Contents</b>	<b>vi</b>
5.4 Single Packet Results . . . . .	58
5.5 Packet Group Results . . . . .	60
5.5.1 Pair of Large Packets . . . . .	61
5.5.2 Triplet of Packets (Small-Large-Small) . . . . .	62
5.5.3 Triplet of Packets (Large-Small-Large) . . . . .	63
5.6 Packet Delay Results . . . . .	63
5.7 Conclusion . . . . .	64
<b>6 Topology Inference</b>	<b>67</b>
6.1 Introduction and Overview . . . . .	67
6.2 Experiment Planning and Execution . . . . .	68
6.2.1 Design Concept and Objective . . . . .	68
6.2.2 Measurement Setup . . . . .	68
6.2.3 Execution . . . . .	69
6.3 Measurement Observations . . . . .	70
6.3.1 Round-Trip Time Distribution . . . . .	70
6.3.2 Viewpoint Dependent Variations . . . . .	72
6.4 Clustering Nodes using Signatures . . . . .	74
6.4.1 Data Processing . . . . .	74
6.4.2 Visualisation . . . . .	74
6.5 Quantifying Signature Similarity . . . . .	76
6.6 Topology Inference Using Signature Similarity . . . . .	78
6.6.1 Example Network . . . . .	79
6.6.2 Live Network . . . . .	79
6.7 Conclusion . . . . .	83

<b>Table of Contents</b>	<b>vii</b>
<b>7 Reconstruction Simulations</b>	<b>84</b>
7.1 Introduction and Overview . . . . .	84
7.2 Topology Generation . . . . .	85
7.2.1 Software Representation and Manipulation . . . . .	85
7.2.2 Creation of a Fully Connected Graph . . . . .	86
7.2.3 Pruning of Graph Edges . . . . .	87
7.3 Measurement Simulation . . . . .	88
7.4 Reconstruction and Comparison . . . . .	89
7.4.1 Viewpoint Similarity . . . . .	89
7.4.2 Reconstruction Heuristic . . . . .	92
7.4.3 Comparison . . . . .	92
7.5 Monte Carlo Method Results . . . . .	93
7.5.1 Perfect Measurements . . . . .	94
7.5.2 Measurement Errors . . . . .	95
7.5.3 Limited Viewpoints . . . . .	102
7.6 Conclusion . . . . .	103
<b>8 Conclusion</b>	<b>106</b>
8.1 Completed Work . . . . .	106
8.2 Limitations . . . . .	107
8.3 Future Work . . . . .	108
<b>Glossary of Abbreviations</b>	<b>110</b>
<b>Bibliography</b>	<b>113</b>



# List of Figures

1.1	Network Discovery Aspects. . . . .	3
2.1	The Open Systems Interconnection (OSI) reference model. . . . .	7
2.2	Network Layer Encapsulation. . . . .	8
2.3	Address Resolution Protocol (ARP) Packet Fields and Layout. . . . .	9
2.4	Ethernet 802.3 Medium Access Control (MAC) Frame Layout. . . . .	13
2.5	Example Ethernet Collision on a Shared Medium. . . . .	16
3.1	Early Hand-Drawn Map of the Internet. . . . .	19
4.1	Pseudo-Code for Parsing ARP Packets. . . . .	35
4.2	Example ARP Packet in PDML Syntax. . . . .	36
4.3	Example Browser Announce PDML File Snippet. . . . .	37
4.4	OUI Database Table Example. . . . .	39
4.5	Passively Discovered Node List Table Extract. . . . .	42
4.6	Nodes with Multiple IP Addresses Table Extract. . . . .	43
4.7	Desktop Node ARP Timestamps. . . . .	44
4.8	Possible Server Node ARP Timestamps. . . . .	45
4.9	Graph of Communication Peers. . . . .	47
4.10	Graph of Communication Peers (Zoomed In). . . . .	47

<b>List of Figures</b>	<b>ix</b>
5.1 Experimental (Controlled) Network Configurations. . . . .	56
5.2 Typical Single Packet Round-Trip Time Distributions. . . . .	59
5.3 Round-Trip Times for Various Packets Sizes. . . . .	59
5.4 Large-Large Packet Group Results. . . . .	61
5.5 Small-Large-Small Packet Group Results. . . . .	62
5.6 Large-Small-Large Packet Group Results. . . . .	65
5.7 Delayed Large-Large Packet Group Results. . . . .	65
6.1 Round-Trip Time Distribution in a Controlled Network. . . . .	71
6.2 Round-Trip Time Distribution in a Live Network. . . . .	71
6.3 Viewpoint Round-Trip Times to Node 37. . . . .	73
6.4 Viewpoint Round-Trip Times to Node 49. . . . .	73
6.5 Pseudo-Code for Calculating and Matching Node Signatures. . . . .	75
6.6 Nodes Clustered using Signatures from Four Viewpoints. . . . .	75
6.7 Example Network, Signatures and Similarity Values. . . . .	77
6.8 Example Recreated Network Graphs. . . . .	79
6.9 Reconstructed Network Graph – Threshold of 0.55. . . . .	81
6.10 Reconstructed Network Graph – Threshold of 0.8. . . . .	81
6.11 Reconstructed Network Graph – Threshold of 0.8. . . . .	82
6.12 Reconstructed Network Graph – Clusters Connected. . . . .	82
7.1 Pseudo-Code for Creating a Fully Connected Graph. . . . .	87
7.2 Example Generated Test Network. . . . .	88
7.3 Calculating Viewpoint Similarity – Perfect Measurements. . . . .	90
7.4 Pseudo-Code for Reconstructing a Network Topology. . . . .	92
7.5 Generated (left) and Reconstructed (right) Topology. . . . .	95
7.6 Calculating Viewpoint Similarity – Measurement Errors. . . . .	97

**List of Figures** **x**

---

7.7	Reconstruction Error at the Edge of a 10 Node Network. . . .	100
7.8	Reconstruction Errors at the Edge of a 50 Node Network. . . .	101
7.9	Reconstruction Errors when using Limited Viewpoints. . . .	104

# Chapter 1

## Overview and Objectives

### 1.1 Introduction

The word topology is derived from the Greek words *topos* meaning location or place and *logos* meaning study [1, 2]. In computer networking terms this *study of location* refers to the layout of and relationships between elements of a network. Topology discovery refers to the process whereby the topology information is extracted from the network. In particular, physical or link layer topology discovery concerns itself with finding the physical connections between network elements.

Topology discovery is also related to the act of mapping, as was performed by the cartographers of old. In this sense, as Dodge and Kitchin remark: “*Mapping is a process of creating, rather than revealing, knowledge, as a result, decisions are made about what to include and what to exclude, how the map will look, and what the map wants to communicate.*” [3, p. 75]

Network topology information can be valuable in a variety of situations, it can be used for network administration (including fault-finding [4, 5], inventorying and planning [6, 7, 5]), protocol and routing algorithm development [8], performance prediction [9] and monitoring as well as accurate network simulation [10]. From a network security perspective, topology information

## 1.2. Problem Statement

2

can find application in threat detection [11], network monitoring [12], network access control [13] and forensic investigations [14, 15].

Manual network mapping is becoming increasingly difficult [16] (if not impossible [17]) due to the size and dynamic behaviour of networks. Automatic topology discovery tools [5] and algorithms will therefore play an important role [18] in network security, management and administration.

Research efforts concerned with physical topology discovery have focused mainly on cooperative network environments [4] where it is assumed that network elements are intelligent and can be queried for topology related information.

## 1.2 Problem Statement

The purpose of the work presented in this dissertation is to discover the physical topology, that is the physical connections between elements, of an Ethernet network. Furthermore, the topology discovery is attempted in an uncooperative network environment where the existence of intelligent network elements and administrative access to these elements are not assumed.

Figure 1.1 shows that the focus of network discovery can be placed on either the connections between nodes of the network, or the node specific characteristics. Our efforts are directed towards the topology of the network and only consider node specifics to a limited extent.

## 1.3 Methodology

A literature survey was conducted at the outset of our investigation to identify research areas that have not received a lot of attention. A study of the relevant network concepts, that were needed during our investigation into topology discovery, was also performed.

The bulk of the work presented in this dissertation is the product of empirical experiments conducted on real-world Ethernet networks. The experiments

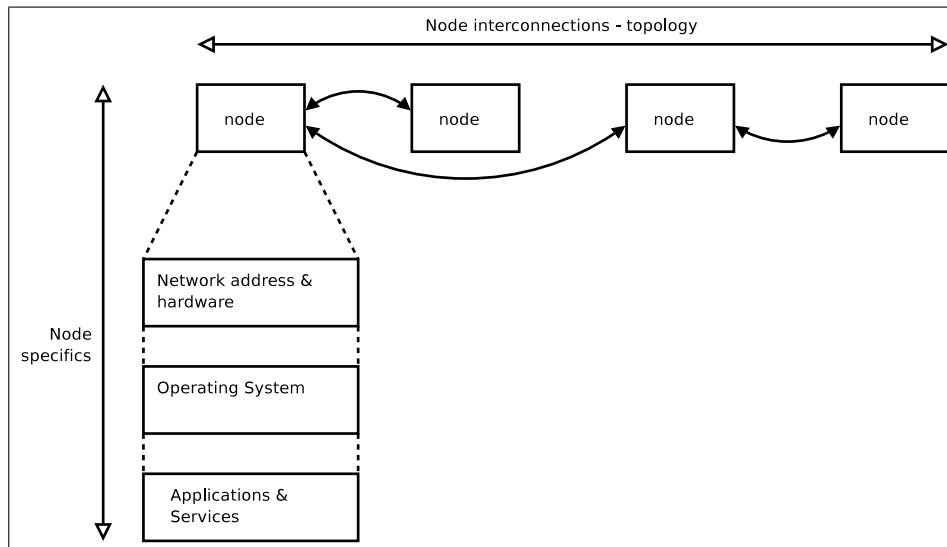


Figure 1.1: Network Discovery Aspects.

presented a hands-on environment for studying topology discovery techniques as well as a test bed for ideas.

The experiments conducted during the research proved extremely useful, but could not answer all of our questions. Computer simulations were employed to quantify experimental findings and to support the conceptual models and algorithms that were developed.

## 1.4 Overview

Background related to our research is presented in Chapters 2 and 3. Chapter 2 briefly discusses networking concepts and scanning techniques that are relevant in the context of our experiments, while Chapter 3 presents related research in the field of network topology discovery.

The following four chapters (Chapters 4 to 7) present results and discussions of our investigations.

Chapter 4 concentrates on identifying unique elements present in an Ethernet network without attempting to discover the connections between these

## 1.4. Overview

---

4

elements. Elements at the edge of the network as well as elements internal to the network are considered.

Internal Ethernet network element behaviour is explored in more detail in Chapter 5, where an attempt is made to discover the presence of these elements.

The knowledge gained and techniques developed during the investigations in Chapters 4 and 5 are combined in Chapter 6 to create an algorithm that can be used to infer the physical topology of a target network.

An extensive set of experiments to exercise the topology discovery algorithm presented in Chapter 6 proved impractical. Chapter 7 presents and discusses computer simulations that were performed to determine the accuracy of the algorithm.

Finally, Chapter 8 concludes the dissertation by reflecting on the work performed and discussing possible future research work.

## Chapter 2

# Background and Context

### 2.1 Introduction and Overview

This chapter introduces background topics relevant to both the related research and the experiments presented in later chapters. The discussions are not meant to be a thorough reference guide, but aim to set the context for the work presented later.

Our work is aimed at a specific network layer and Section 2.2 presents the layered network design concept. Section 2.3 introduces network protocols, while Section 2.4 focuses mainly on Ethernet concepts relevant to our work. Section 2.5 compares active and passive network information gathering methods and finally Section 2.6 concludes the chapter.

### 2.2 Layered Network Design

Networking protocols are normally designed using a layered approach [19, p. 1]. The set of protocols and layers are also commonly referred to as a network protocol stack [20, p. 20]. The layered design approach provides three advantages according to Keshav [21, p. 70]:



## 2.2. Layered Network Design

6

1. The complex networking problem is broken up into smaller, more manageable, pieces. Each layer focuses on a different aspect of communication and in this way complex services can be provided by combining simpler ones.
2. Implementation and specification are separated. The implementation details of a specific layer are abstracted and hidden from the other layers. This allows the implementation of a specific layer to change without influencing the other layers.
3. Functionality can be reused. Once a specific function is implemented in a network layer, all the layers above it can benefit from and use the functionality.

The layered design also has disadvantages. In the same way that computer ease of use and security present an engineering trade-off [22, p. 10], network design simplicity and security present a trade-off. Initially networks protocols were not designed with security in mind [23, p. 368]. When security is compromised at a specific network layer, the layers above it will be completely unaware of the problem [24, 25]. A lower network layer can therefore become the weakest link in the security chain [26].

The most common model for reasoning about network layers is the Open Systems Interconnection (OSI) reference model<sup>1</sup> developed by the International Organisation for Standardisation (ISO) [23, p. 82]. The OSI model consists of 7 layers with the physical layer being the lowest and the application layer being the highest. A representation of the OSI model is shown in Figure 2.1. In a logical sense, every network layer on one system communicates directly with the corresponding layer on another system; however, in practise, data passes from layer to layer on a single system and is transmitted only over the physical medium connecting the two systems [27, p. 35].

The context of the research and experiments presented in this dissertation requires consideration of only the first three layers of the OSI model. The functions of the first three layers are briefly summarised below:

---

<sup>1</sup>ISO 7498

## 2.2. Layered Network Design

7

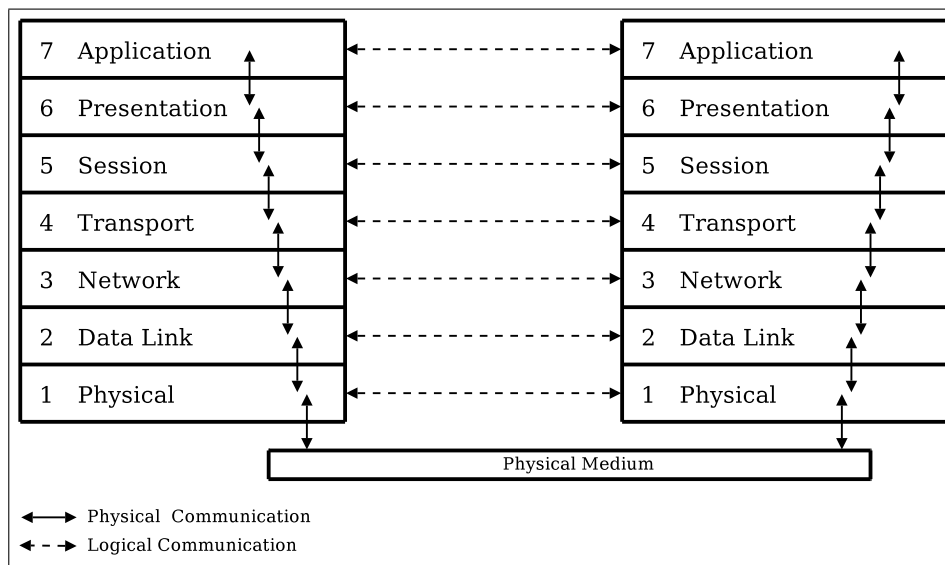


Figure 2.1: The Open Systems Interconnection (OSI) reference model.

The **physical layer** provides an abstraction of bit transmission between two systems connected to a single physical link and hides the details of the link technology used by the physical hardware in the system [21, p. 74]. The **data link layer** allows the transmission of an aggregation of bits (commonly called a frame [23, p. 104]) from one point on a physical link to another [27, p. 339]. In order to reliably deliver frames on a shared medium the layer also typically controls access to the medium, provides a means to uniquely address systems sharing the medium and optionally offers error and flow control [27, p. 176]. The data link layer is normally closely tied to the specific physical layer (and also the physical medium) used for communication [21, p. 75]. The **network layer** has the main function of moving (also called routing) data packets between two systems [27, p. 339] on the network. To this end it needs to utilise one or more physical links and it must be able to uniquely identify end systems in the wider network [21, p. 76].

The OSI model is a reference model in the sense that it provides abstract guidelines for constructing a network protocol stack. The model does not provide a specific implementation of the layers and concepts, but allows a multitude of network stacks to be developed [23, p. 83]. For our research

## 2.2. Layered Network Design

8

and experiments we are specifically interested in Ethernet as the physical network that carries Internet Transmission Control Protocol/Internet Protocol (TCP/IP) traffic. In this network scenario, Ethernet provides the physical medium and implements the physical and data link layers, while IP<sup>2</sup> operates at the network layer.

The layers of the OSI model can also be visualised hierarchically. From this perspective, data from higher layers are seen as encapsulated<sup>3</sup> within the structures provided by the lower layers [23, p. 95]. Figure 2.2 illustrates the idea of encapsulation. The IP packet (layer 3) with all its related data is encapsulated within an Ethernet frame (layer 2). The data link layer mechanisms employed to deliver the Ethernet frames require no knowledge of the data encapsulated within the frame.

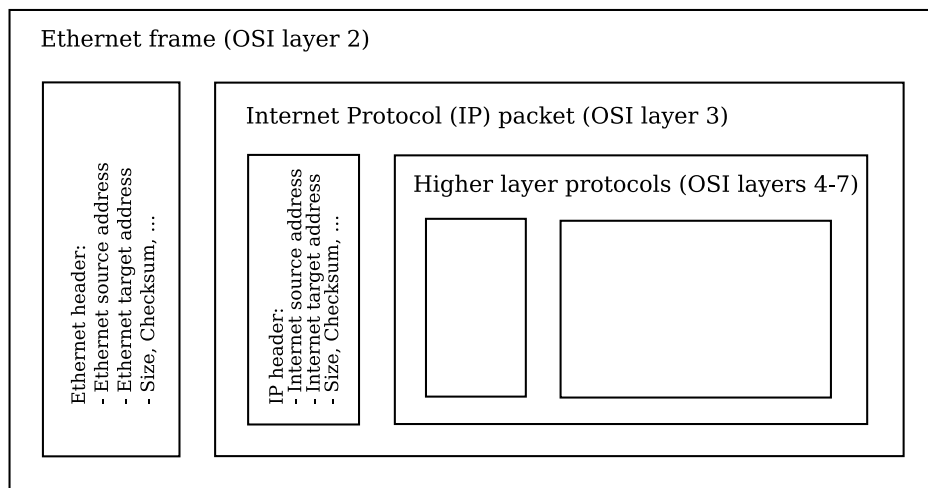


Figure 2.2: Network Layer Encapsulation (IP Packet Inside Ethernet Frame).

The brief network layer discussion showed that a system has an address at both the data link and network layers. In particular, delivery of data at the data link layer depends on a correct data link layer target address. The mechanism for converting a network layer address into a data link layer address is the focus of Section 2.3.1.

<sup>2</sup>In particular, we focus on Internet Protocol version 4 (IPv4).

<sup>3</sup>An analogy that is often used is that of placing a letter (higher layer data) into an envelope (the lower layer structure).

## 2.3 Relevant Protocols

### 2.3.1 Address Resolution Protocol (ARP)

The process whereby network layer addresses are resolved into data link layer addresses is known as address resolution [23, p. 212]. As mentioned earlier, we are primarily concerned with IP network layer addresses and Ethernet data link layer addresses. The TCP/IP Address Resolution Protocol (ARP), as described in the Internet Request For Comments (RFC) 826, [28]<sup>4</sup> was specifically devised for tackling the problem of converting IPv4 network addresses into Ethernet hardware addresses.

Figure 2.3 shows the fields and layout of an ARP packet.

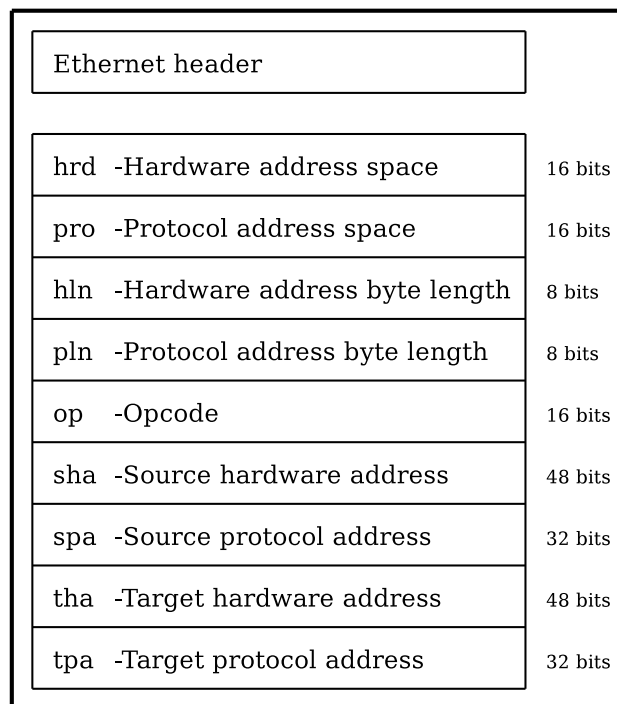


Figure 2.3: Address Resolution Protocol (ARP) Packet Fields and Layout.

<sup>4</sup>It is interesting to note that even the title of the RFC (“An Ethernet Address Resolution Protocol”) mentions Ethernet, even though the protocol can be used for translation of various types of network and data link layer addresses.

### 2.3. Relevant Protocols

10

Details of the specific fields, as used during IPv4 to Ethernet address translation, are provided below. The field names are taken from the RFC [28].

The hardware and protocol types are stored in the **hardware address space (hrd)** and **protocol address space (pro)** fields respectively. Examples for hardware types are Ethernet (with a value of 1 or 6) and Fibre Channel (with a value of 18). For Ethernet, the protocol type matches the Ethernet type field [29, p. 40] in the Ethernet frame. For the Internet Protocol (IPv4) a value of 0x0800 is used.

The **hardware address byte length (hln)** and **protocol address byte length (pln)** fields contain values of 6 and 4 for Ethernet and IPv4 addresses respectively. Ethernet addresses have a length of 48 bits (6 bytes) and IPv4 addresses have a length of 32 bits (4 bytes).

The **opcode (op)** field indicates the type of ARP message. A 1 indicates an ARP request and a 2 indicates an ARP reply message.

Ethernet and IPv4 addresses of the ARP packet sender are stored in the **source hardware address (sha)** and **source protocol address (spa)** respectively.

The **target hardware address (tha)** and **target protocol address (tpa)** contain the Ethernet and IPv4 addresses of the ARP target.

The total size of the ARP payload in the case of IPv4 to Ethernet address translation is 28 bytes [23, p. 218].

In its simplest form, the ARP is a simple request and reply process<sup>5</sup>. A system (source) on the network wants to send an IP packet to another system (target). The source system knows the IP address of the destination, but not its hardware address. The source system fills the fields of an ARP request packet<sup>6</sup> and broadcasts it to the local network. Systems on the local network receive the ARP packet and examine it to determine if it is their own IP address mentioned in the target protocol address (tpa). If it is, the target

---

<sup>5</sup>This simple description of the ARP protocol, even though completely valid, omits much of the practical implementation details; however, it is sufficient for setting the context for the use of ARP packets in the dissertation.

<sup>6</sup>All but the target hardware address.

## 2.4. Relevant Ethernet Concepts

11

system enters its own hardware address into the packet, swaps the source and target addresses in the packet, sets the opcode to reply and sends the packet back to the original sender. The original sender receives the ARP reply packet and can then use the hardware address contained therein.

The ARP protocol was designed to be efficient at a point in time when everyone on a network was reasonably trustworthy [30]. This implicit trust between the source and target nodes allows malicious use of the protocol [25], but our research does not depend on any of the protocol's vulnerabilities.

### 2.3.2 Simple Network Management Protocol (SNMP)

The Simple Network Management Protocol (SNMP) is not directly used in our research, but is often employed in related work. An extremely simplified overview is provided.

SNMP provides a systematic way of monitoring and managing a computer network [27, p. 630] and is part of the Internet protocol suite [19, p. 359]. Network management is performed by **network management stations** that communicate with **network elements** [19] (also called **managed devices** [23, p. 1072]).

The **managed devices** contain software (called an **agent** [23, p. 1073]) for communicating with the **network management stations** and a virtual information store [31] called a **Management Information Base (MIB)**.

The **MIB** defines the types of information stored about the **managed device** that can be queried by the **network management stations** [19, p. 359] and defines variables that can be used by the **network management stations** to control the **managed device** [23, p. 1073].

## 2.4 Relevant Ethernet Concepts

Ethernet was conceptualised at the Xerox Palo Alto Research Center [32] in 1973 [33, 34]. One of the inventors, Robert Metcalfe, was inducted into the

## 2.4. Relevant Ethernet Concepts

12

Inventors Hall of Fame<sup>7</sup> in 2007 [35]. Ethernet has seen explosive growth [21, p. 42] into a truly ubiquitous [35] Local Area Networking (LAN) technology. It was estimated that 85 percent of the world's personal computers and workstations used Ethernet as a connection standard in the year 2000 [36, p. 7-1].

A draft Ethernet standard was approved by the Institute of Electrical and Electronics Engineers (IEEE) 802.3 working group in 1983 after which an official Ethernet standard was published in 1985 (ANSI/IEEE Std. 802.3-1985) [36, p. 7-2]. The term Ethernet as used in this dissertation refers to the 802.3 standard (especially the 2002 incarnation of the standard [29, 37]).

It is impossible to provide a detailed description of Ethernet in this section, but concepts and details relevant to further discussions and experiments are presented.

### 2.4.1 Data Link Sublayers

The Ethernet standard splits the data link layer (as defined in Section 2.2) into two sublayers: The Logical Link Control (LLC) sublayer<sup>8</sup> forms the upper part and the Medium Access Control (MAC) layer<sup>9</sup> forms the lower part.

Typical functions of the LLC layer include the ability to retransmit corrupted data (error control) and to pace the rate at which data is sent (flow control) [21, p. 75].

The MAC sublayer concerns itself with the mechanisms used to allow multiple stations to access and use a shared communication medium. The problem of sharing a medium between multiple stations is called the *multiple-access* problem [21, p. 117].

---

<sup>7</sup>The Hall of Fame has 331 members that include the likes of Alexander Graham Bell and the Wright brothers [35].

<sup>8</sup>IEEE standard 802.2.

<sup>9</sup>IEEE standard 802.3 in our case.

## 2.4. Relevant Ethernet Concepts

13

### 2.4.2 Hardware Addresses

Ethernet stations on a LAN are uniquely identified using a 48-bit address. The address is directly used by the MAC sublayer and is therefore also referred to as the MAC address. The addresses are normally split up into 8 bytes<sup>10</sup> represented as hexadecimal numbers for presentation or discussion, for example 00:15:F2:1F:4D:3A.

A unique address used for sending data to a single destination is called a **unicast** address. Ethernet addresses also exist for sending data to multiple stations at once and are called **multicast** addresses. A special address, with all the bits in the address set to one (FF:FF:FF:FF:FF:FF), is used to send data to all the stations connected to the Ethernet LAN and is called the **broadcast** address.

The first 24 bits of a MAC address are assigned to companies or vendors by a registration authority. In the case of Ethernet the registration authority is the IEEE [29]. This globally unique number is referred to as the Organisationally Unique Identifier (OUI). Vendors combine the OUI with another 24-bit number to form unique device addresses.

### 2.4.3 Frame Layout

The 802.3 MAC frame structure is shown in Figure 2.4.

Preamble	Start of frame delimiter	Destination address	Source address	Length or type	Data	Pad	Frame check sequence
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	0 to 1500 bytes	0 to 46 bytes	4 bytes

Figure 2.4: Ethernet 802.3 Medium Access Control (MAC) Frame Layout.

The fields of the frame, as defined in the standard [37], are briefly discussed below:

The **preamble** and **start of frame delimiter** help with the framing and synchronisation when two stations communicate. These values are defined by the standard and cannot be altered by the user through software.

<sup>10</sup>The standard refers to bytes as octets to clearly show that they consist of 8 bits.



## 2.4. Relevant Ethernet Concepts

**Destination and source addresses** are MAC addresses as defined in Section 2.4.2. The destination address might contain a unicast, multicast or broadcast address.

The **length** or **type** field allows two types of frames to co-exist on the same network. When the value is 1500 or less it indicates the length of the data field. If the value is larger than 1536 (hexadecimal 0x0600) it indicates that a specific type of frame is present, for example this field contains the hexadecimal value 0x0806 for an ARP packet.

Actual data for higher level protocols are contained in the **data** field. The data field can contain a maximum of 1500 bytes. The **pad** field is optional and is used to make sure that the Ethernet frame has a minimum size of 64 bytes (counting from the destination address to the frame check sequence). This field can therefore have a length from 0 to 46 bytes. A minimum length is required for proper collision detection and is explained in the following section.

The **Frame Check Sequence (FCS)** is a 32-bit Cyclic Redundancy Check (CRC) value calculated over the byte range from the destination address to the pad field. The FCS value is used for error detection.

### 2.4.4 Duplex Modes

The Ethernet 802.3 standard [37] makes provision for both half duplex and full duplex modes of operation.

#### Half Duplex

In half duplex mode two or more stations share a common communication medium and the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) medium access control method is used to make sure that only a single station transmits on the medium at any one time [36, p. 7-8]. Any active station in half duplex mode can either be busy transmitting or receiving data, but cannot perform both functions at the same time [23, p. 42].

## 2.4. Relevant Ethernet Concepts

**Carrier sense** means that stations connected to the shared medium can sense when the medium is being used. Stations can therefore defer their own transmission until they sense that the medium is available.

Even though stations can detect availability of the medium it does not automatically guarantee that two or more stations will not start transmitting at the same time. **Collision detection** refers to the ability of Ethernet stations to detect simultaneous use of the medium. In order for a station to detect a collision, a message from another transmitting station must have propagated along the wire towards it. An engineering trade-off therefore has to be made between the minimum frame size<sup>11</sup> and the maximum cable length<sup>12</sup> allowed for an Ethernet network.

The following example [27, p. 282], graphically shown in Figure 2.5, illustrates the trade-off problem. Consider two Ethernet stations (A and B) placed at the maximum allowable distance from each other. Assume that the time it takes for a bit sent from A to reach B is equal to  $\tau$ . Station A starts to transmit a packet at time 0 (a). Just before the frame reaches the other end of the cable ( $\tau - \epsilon$ ), station B starts to transmit (b). Station B detects the collision and sends a 48-bit noise burst to the network (c). The noise burst only reaches A after time  $2\tau$  (d). The minimum frame size therefore has to be large enough so that station A receives the noise burst before it has finished its own transmission. The minimum frame size allowed for an Ethernet network operating at 100 megabits per second (Mbps) over Unshielded Twisted Pair (UTP) cables is 64 bytes and the maximum cable length allowed is 100m.

After two stations have determined that a collision has occurred, they have to wait before retrying the transmission. If the stations wait the same amount of time, a collision will occur again, and to avoid this situation Ethernet employs a **binary exponential backoff algorithm** [21, p. 280]. Details of the algorithm are not important for our investigation.

---

<sup>11</sup>The frame size directly translates into time on the wire when considering the bit rate at which the network operates.

<sup>12</sup>The cable length determines the time a frame spends on the wire when considering the electrical propagation delay.

## 2.4. Relevant Ethernet Concepts

16

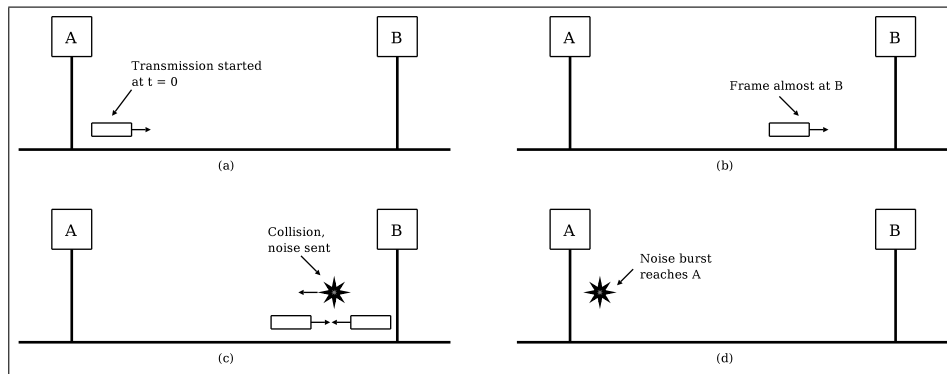


Figure 2.5: Example Ethernet Collision on a Shared Medium.

### Full Duplex

In full duplex mode two stations can communicate simultaneously (both can transmit and receive at the same time). Full duplex mode requires a point-to-point dedicated channel [36, p. 7-10]. Since there is no contention for using the medium there is no need for the CSMA/CD medium access control method.

A dedicated channel is achieved by connecting stations to a switch or bridge where each port on the switch effectively forms a dedicated LAN with the station connected to that port.

### 2.4.5 Hubs and Switches

A hub, also called a repeater [37], connects multiple stations together to form a larger shared medium. Hubs are often used to connect stations into a star type network. A hub essentially electrically mirrors traffic received on one of its ports onto all of its output ports [38, p. 77].

Switches, also called bridges, perform the same function as hubs, but are more intelligent. They normally consist of a processor and some memory [38, p. 77]. A switch performs some analysis of Ethernet frames received on its ports. MAC addresses of stations connected to each port is placed into a list and when a frame arrives it can be routed directly to the appropriate

## 2.5. Active versus Passive Scanning

17

port or ports. In order to analyse an Ethernet frame and decide onto which port to output the frame, the switch has to receive at least part and in many cases the whole frame [39, p. 166]. Switches are therefore also called store-and-forward elements and introduce a delay when delivering frames between ports [39, p. 166].

Stations connected to hubs can only function in half duplex mode. Hubs are not normally addressable in the network, that is, they do not have MAC addresses. By contrast, stations connected to a switch can operate in full duplex mode and switches can be addressable in the network, that is, they can have MAC addresses assigned.

## 2.5 Active versus Passive Scanning

Network monitoring or scanning can be performed either actively or passively [40]. Active scanning sends out data (often called probe packets [41]) to the network and evaluates responses. Passive scanning or monitoring uses captured, naturally occurring, network traffic data in order to gather information about a network and its elements [40]. The two techniques are contrasted in the following discussion.

Passive techniques are more **covert** and **stealthy** than active techniques [38, p. 152]. By nature, active scanning techniques transmit data to the network or node being investigated and this traffic can be detected by another party.

**Target** selection and **timing** can be **controlled** during active scanning, that is, an investigator can decide what to scan and when do to a scan [41]. By contrast passive scanning relies on network traffic where sources, targets and timing cannot be controlled [42]. Passive mapping can however be used to infer the **usage patterns** of the network, because timing information is received from the network and not generated by the scanning tool as in active mapping [40].

Active scanning **influences** the target network (or target node) when sending data, while passive scanning does not directly have an **impact** on the network

infrastructure [43, 42].

Active probing attempts might explicitly be **blocked** or **filtered** on a network to hide the presence of nodes, but if these nodes communicate at all the communication could be detected by passive means [42].

Active mapping can provoke responses from any **location** in the network, whereas passive mapping's success depends a great deal on the location in the network from where information is gathered [43, 42, 40].

From the discussion it is clear that passive and active techniques have different strengths and weaknesses and could be used to complement each other [43, 40].

## 2.6 Conclusion

The topics and information presented in this chapter provide the background and context for the work to follow. Chapter 3 presents related research and Chapters 4 to 7 present experiments that build on the concepts presented here.

# Chapter 3

## Related Work

### 3.1 Introduction and Overview

Internet maps are as old as the Internet itself and were initially drawn by hand [3, p. 108]. Figure 3.1 shows an early conceptual map of the Internet drawn by Larry Roberts while working at the Advanced Research Projects Agency (ARPA) in the 1960's [44, p. 49].

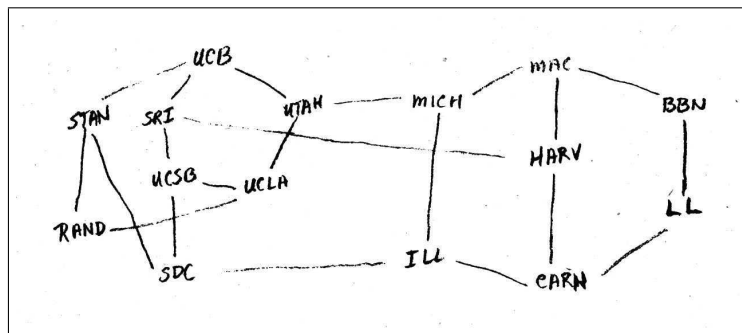


Figure 3.1: Early Hand-Drawn Map of the Internet [3, p. 108].

The manual effort required to generate accurate maps of networks' infrastructure increased significantly [3, p. 94] as the early computer networks grew. Research was therefore spurred on in the areas of network topology discovery and the automatic creation of network maps [16].

## 3.2. Applications of Topology Knowledge

20

This chapter only samples and discusses more recent work in the area of topology discovery related to our research. For a more historic perspective the reader is encouraged to consult the excellent text by Dodge and Kitchin [3].

The related work is divided into three main sections. Section 3.2 presents related work from the viewpoint of the application areas of topology information. Section 3.3 presents mapping efforts at the network layer (OSI layer 3), without going into detail about the actual methods employed. Finally, Section 3.4 discusses techniques that were developed to determine the physical (OSI layer 2) topology of a target network.

Conclusions surrounding the related work are presented in Section 3.5.

## 3.2 Applications of Topology Knowledge

Motivations for research into topology discovery techniques are often found in the areas where the resulting topology knowledge can practically be applied [45]. This section highlights five of the application areas of network topology information with a bias towards security applications.

### 3.2.1 Administration and Planning

Network administrators are often faced with network problems where fault finding needs to be performed [4, 46]. In order to troubleshoot network problems, a topology map of the network can effectively be used to isolate the problem area [4, 47]. The topology map can also help identify infrastructural vulnerabilities [48, 49] and the network can then be adapted to provide more redundancy.

Network expansion planning [5] and decisions regarding the placement of new infrastructure [50] are also aided by accurate knowledge of the network topology.

## 3.2. Applications of Topology Knowledge

21

Network Management Systems (NMS) also employ topology information [51] to help with network administration. The most notable [52] systems include IBM's Tivoli<sup>1</sup>, Hewlett-Packard's OpenView<sup>2</sup> and the open source OpenNMS<sup>3</sup>.

### 3.2.2 Performance Prediction

In a second application area, that of performance prediction, topology information can be used to optimise the performance of network aware applications [53] as well as the performance of distributed, either grid or cluster, applications [54, 55].

Topology knowledge can help determine if a given network would provide a certain Quality of Service (QoS) [50, 56]. As an example, in order to determine if a network would support multimedia technologies such as Voice over IP (VOIP), knowledge of the network topology is essential [57, 58].

### 3.2.3 Algorithm and Protocol Design

Algorithm and protocol design represents a third application area of topology knowledge. A network's topology influences the dynamics of routing protocols [59, 60] and should therefore be taken into account during the design of the protocols [8, 17].

Large network topology visualisation has proved to be a challenging task and algorithms have been developed for effectively presenting the topology information [61, 62, 8].

### 3.2.4 Simulation

The accuracy of network simulations, a fourth application area, depends on realistic and accurate network topologies [10]. Generated topologies for use

---

<sup>1</sup><http://www.ibm.com/tivoli/>

<sup>2</sup><http://openview.hp.com/>

<sup>3</sup><http://www.opennms.org/>



## 3.2. Applications of Topology Knowledge

22

in simulation do not always match real-world topologies [60] and create the need for accurately [63] measuring real-world network topologies.

Network simulation can not only help researchers understand the current behaviour of a network, but also the effects of possible future changes to the network [64].

### 3.2.5 Security

The fifth and final application area that we address is that of network security.

Firewalls have traditionally been placed at the network edge to protect against external threats [65, p. 4]. This creates a situation where the network has a “crunchy outside with a soft, chewy center” [66]. Insider threats to networks have become more common and it is estimated that they account for around 30% of security incidents [67]. These security incidents also lead to significant financial losses [68, 13]. Insiders do not only include disgruntled employees, but even loyal workers can become *unwitting insiders* [69] by unknowingly executing malicious software. Firewall placement and the management of a network security policy should therefore be influenced by the network topology [70, 69].

Another perimeter defence mechanism, Intrusion Detection Systems (IDS), can also benefit by taking network topology information into account [71]. If an IDS is not placed correctly it could generate both false positives and false negatives [71, 30, 72].

The problems with firewall and intrusion detection systems have generated research interest in the areas of Network Access Control [73] (NAC, also called Network Admission Control [13]). These systems are proactive and attempt to enforce a network security policy at either layer 2 or 3 of the network [13]. Devices that do not conform to the security policy can for example be denied access to the network infrastructure by physically disabling switch ports [31]. A lack of knowledge about the network’s topology and the connected devices can however seriously hamper the effectiveness of NAC solutions [72].

### 3.3. Network Layer Topology Discovery

23

Another proactive approach to network security, that augments firewalls and intrusion detection systems, exists in the form of network forensics [14, 15]. Firewalls and intrusion detection systems focus mainly on attack prevention and detection [14, 13]. By using topology information, forensic tools attempt to trace network attacks to their source and actively respond to these attacks [14].

In order to construct a proper security model of a network, to perform security analysis [12] and to analyse network attacks [74], knowledge of a network's topology is needed [11]. Network security models and simulation can also be used to predict the spread of worms and other malware [75].

## 3.3 Network Layer Topology Discovery

In the late 1990's Hal Burch and Bill Cheswick performed research in the field of Internet mapping [16, 49]. Their research culminated in the creation of the "Internet Mapping Project" in 1998 [76]. The project and research focused on gathering topological Internet data over an extended period of time in order to study routing problems, routing changes, Distributed Denial of Service (DDoS) attacks and graph theory [76].

The motivation for initially collecting topological Internet data [49] was voiced clearly by the Rand Corporation: "Network protection, and incident analysis, is at times hampered by a lack of understanding of the topology of the network, and an understanding of the critical nodes for its operation." [77]. "Incidents" can refer to both on-line and physical events, for example, Internet maps allowed Burch and Cheswick to detect loss of power to routers in Yugoslavia after NATO bombing of the country [49].

Active Internet mapping research is also performed by the Cooperative Association for Internet Data Analysis (CAIDA)<sup>4</sup>. CAIDA's work has many applications including Internet traffic engineering, capacity planning and security breach detection [8]. Assessment of a network security situation can benefit

---

<sup>4</sup><http://www.caida.org/>

### 3.4. Link Layer Topology Discovery

---

from network topology information in that network sensors and probes can be placed at optimal positions in the network to reduce system load and network traffic [78].

Topology discovery has also been performed in newer IPv6 networks by Waddington et al [17]. The research supports architectural design decisions related to address allocation and distribution schemes [17].

Another research area related to network topology discovery is that of network tomography. In contrast to topology discovery, where measurements can be performed from any location in the network, network tomography is a process for inferring internal network characteristics by using end-to-end measurements from the edge of the network [79]. The advantage of using edge based measurements is that cooperation from the internal network elements is not required [80].

Network tomography requires knowledge of the network topology [79, 81], but the techniques can also be applied to gain knowledge about the network topology [79].

Ideas from network layer tomography research, such as the use of similarity metrics [82, 18] to find shared paths between edge nodes, were applied during our investigations into layer 2 topology discovery.

## 3.4 Link Layer Topology Discovery

A network's physical topology can potentially correspond to many logical topologies depending on the level of abstraction used [50, 45]. In 2000 Breitbart et al [83] realised that network management tools as well as previous research efforts focused on layer 3 topology discovery and ignored the connectivity of layer 2 network elements. Where layer 2 topology discovery tools did exist, they were found to specifically target single vendor products [83]. Breitbart et al therefore developed algorithms that could perform layer 2 topology discovery in multi-vendor (heterogeneous) networks by using standard Simple Network Management Protocol (SNMP, refer to Section 2.3.2) Management

### 3.4. Link Layer Topology Discovery

Information Base (MIB) data [83]. Hwa-Chun Lin et al [6] developed the same techniques as Breitbart et al; however, they did not formally design a discovery algorithm, but only demonstrated a possible discovery process by example.

The initial algorithm developed by Breitbart et al [83] depended on perfect Address Forwarding Table (AFT) data collected from every single element in the network. Their research was refined to relax this dependency on complete information [84, 7, 85]. The newer algorithms could successfully discover the target network topology, provided that the network was uniquely described by the SNMP MIB data obtained [84].

Lowekamp et al [9] also extended the initial work by Breitbart et al [83] and developed a topology discovery algorithm that performed well with only limited AFT data collected from SNMP enabled network elements. Yantao Sun et al [86] proposed an enhanced algorithm that made more efficient use of the AFT data than did Lowekamp et al's [9].

Further work by Bejerano [87] showed that the algorithms developed by Lowekamp et al [9] and Breitbart et al [84, 7, 85] did not perform well in multi-subnet networks or in the presence of uncooperative switches and hubs. Uncooperative elements do not speak SNMP (unmanaged elements), do not allow access or do not even have layer 2 addresses. Bejerano [87] developed an algorithm that could more accurately determine the network topology than the previous attempts.

The work of Yuzhao Li et al [88] again concentrated on AFT data obtained from SNMP enabled network elements, but they augmented incomplete AFT data with the analysis of port traffic data (also obtained using the SNMP) to create a topology of a single subnet network.

Research by Stott [56] also employed SNMP MIB data, but instead of using forwarding table data, the algorithm used data from the Bridge-MIB [89]. A white paper by Hewlett-Packard [90] describes the same technique. Stott's algorithm neglects any elements that do not support SNMP and further requires that all switches use the spanning tree algorithm [56]. The work of

### 3.4. Link Layer Topology Discovery

Stott was further refined to become part of a tool for assessing the readiness of networks for IP Telephony [58].

The Internet Engineering Task Force (IETF) attempted to create a standard for SNMP topology discovery by creating the Physical Topology MIB [91], but adoption of the proposal was hampered by the fact that it did not include details on how to actually populate the required MIB objects [86]. To remedy the situation, the IEEE developed the Link Layer Discovery Protocol (LLDP) as part of the 802.1AB-2005 standard [92]. The LLDP allows neighbouring devices to become aware of each other and populate their Physical Topology MIBs [87, 93]. The efforts surrounding LLDP clearly shows an industry need for topology discovery in heterogeneous networks at layer 2; however, LLDP cannot easily be deployed on legacy equipment [87].

All the layer 2 topology discovery techniques and algorithms discussed thus far depend on SNMP enabled network elements. The reliance on SNMP can prove problematic in quite a number of network environments. As networks grow and management becomes decentralised it cannot be assumed that SNMP would be enabled or that administrative SNMP access would be granted [50, 17]. A lot of small business, home office and branch office networks are built using consumer-grade network equipment that do not even support SNMP [4]. A need for topology discovery techniques that do not require network cooperation [81, 51] and for tools that can augment SNMP-based techniques [4, 47] therefore exists.

A technique for layer 2 topology discovery without network element cooperation has been implemented by Black et al [4]. The technique exploits the packet forwarding properties of network elements, specifically those of switches. The algorithm requires specialised software on many edge nodes (hosts) that are controlled from a master node to execute the distributed discovery algorithm [4]. Cooperating hosts *train* switches they are connected to in order to only pass packets with specific addresses. The master node then instructs other hosts to send probe packets with the specific addresses. Depending on where the probe packets are delivered to (or not), a picture of the network internals can be formed [4].

Other efforts worth mentioning are proprietary protocols by network vendors used prior to the standardisation of the LLDP. These include the Cisco Discovery Protocol (CDP), Enterasys Networks' Cabletron Discovery Protocol (also CDP), Extreme Networks' Extreme Discovery Protocol (EDP) and Nortel Networks' Nortel Discovery Protocol (NDP) [94, 93].

The use of Ethernet as an access technology, especially in the telecommunication industry, has also led to efforts to add and standardise Ethernet capabilities for Operational, Administration and Maintenance (OAM) management [95]. The main operational issues addressed are discovery, link monitoring, fault signalling and remote loopback [95]. The added functionality is not aimed specifically at topology discovery in enterprise networks, but could potentially be used.

## **3.5 Conclusion**

In this chapter we have demonstrated some uses of network topology information and we have shown how these applications drive network topology research. Topology discovery at the network layer, especially Internet topology discovery, represents a vast research area and we have only sampled some of the related research. The techniques used for network layer topology discovery cannot directly be applied at the data link layer, but some of the concepts proved to be useful guidelines.

In comparison to network layer topology discovery, link layer topology discovery has not received as much attention and most of this attention has been focused on cooperative network environments. Research into techniques that can be applied in an uncooperative network environment can therefore prove useful in itself and can also be used to augment the other techniques developed thus far.

# Chapter 4

## Passive Network Node Discovery

### 4.1 Introduction and Overview

As a first step in gaining knowledge about an Ethernet network, it was decided to focus on the process of identifying unique network nodes, both at the edge and internal to the network, at the data link layer. Furthermore, a choice was made to attempt the identification in a passive manner, that is by only listening to network traffic (refer to Section 2.5). The nodes identified during the passive discovery phase could be used as targets for further active discovery techniques.

Nodes in an Ethernet network are uniquely identified by their MAC addresses at layer 2 (refer to Section 2.4.2), but this raw number by itself does not provide a lot of information about the node. Other sources of information were therefore combined, where possible, with this number to provide more information about each node.

The layout of this chapter is as follows. The design, preparation and execution of the experiments are discussed in Section 4.2. The initial execution of the experiments provided raw data and the processing of the data is presented in Sections 4.3 and 4.4. Section 4.3 discusses initial processing of

## 4.2. Experiment Design and Execution

29

the raw data, where decisions were made as to what elements of the data to consider. Section 4.4 focuses on the generation of the network node list. Section 4.5 presents interesting processing results that were not directly used in the creation of the node list. Finally, Section 4.6 concludes the chapter.

## 4.2 Experiment Design and Execution

This section details the design of the experiments, the preparations in terms of hardware and software and the execution of the experiments performed during node discovery.

### 4.2.1 Purpose and Procedure

The primary goal of the passive node discovery experiments was the generation of a list of network nodes. Information obtained about node specifics and network internals was seen as a secondary objective.

The procedure envisioned in order to generate a network node list consisted of the following activities:

1. Capture traffic data from the network in a passive manner.
2. Identify relevant packets and protocols and store the raw data in an easy to query format.
3. Perform queries and processing on the data to generate a node list.
4. Correlate data in the list with other external data sources where possible.

### 4.2.2 Target Network Selection

A 100Mbps corporate Ethernet network using standard UTP network cable was identified as the target for the data capture step of the experiment. The



## 4.2. Experiment Design and Execution

30

passive nature of the data capture process ensured that the network was not affected during the experiment.

### 4.2.3 Capture Hardware

A Personal Computer (PC) with a standard full duplex 100Mbps network card was used for capturing data. The PC was connected to the network as a normal workstation. The PC was placed into an unused office and was equipped with enough storage to allow for unassisted capture of network data for extended periods of time (typically in the order of a week or two).

### 4.2.4 Capture Software

The problem of capturing network data is quite common and many software packages, both proprietary and open source, exist for this purpose. There was therefore no need to write custom software for the capture step of the experiment. The Ethereal<sup>1</sup> application was identified as an easy to use capture application and was installed on a Debian GNU/Linux<sup>2</sup> based PC.

In order to limit the number of packets stored to the hard disk, Ethereal can filter packets as they are captured. The filter syntax is the same as that of the ubiquitous tcpdump<sup>3</sup> program and is described in detail in the tcpdump manual page [96]. The following filter was used to limit the capture of packets to Ethernet or IP multicast or broadcast traffic:

```
ether broadcast or ether multicast or  
ip broadcast or ip multicast
```

The reason for capturing only broadcast and multicast traffic becomes clear when we remember how a switch forwards frames in an Ethernet network

<sup>1</sup><http://www.ethereal.com/>. In June 2006 Ethereal was renamed to Wireshark, also see <http://www.wireshark.org/>.

<sup>2</sup><http://www.debian.org/>

<sup>3</sup><http://www.tcpdump.org/>

## 4.2. Experiment Design and Execution

31

(refer to Section 2.4.5). Traffic destined for a specific destination node would only reach our capture node if the destination node shares a switch port with our capture node. Since we were not interested in this special case, the filter helped minimise the packets captured.

Captured packet data was saved in the `libpcap`<sup>4</sup> file format. Ethereal also allows for splitting the captured data into multiple output files while capturing. The files can be split by either time, packet count or file size. The file size option proved useful to keep the files small enough for our processing scripts, while the time option allowed for easy identification of files captured over an extended time period.

### 4.2.5 Database

It was decided to place all captured data into a database to allow for simplified queries and correlation of the data. The database could also be used to store processed data. Furthermore, a database provides a unified interface to the data, whereas a mixture of data formats would require separate tools to process the data.

The SQLite<sup>5</sup> database engine was chosen for the following reasons:

SQLite supports a **standard query format** in that most of the Structured Query Language (SQL), SQL92, standard is implemented [97]. The database engine is **cross platform** and can run on Windows and Linux among others. This enables tools running on different operating systems to interact with the data. The database is **easy to install** as the whole SQLite engine is contained in a single executable or library file. No system services or special permissions are needed to use the engine.

The database contents are **self contained** and saved into a single file that can easily be transported across PCs. Various **programming languages** can interface with the database and hooks to query and manipulate the database

---

<sup>4</sup><http://www.tcpdump.org/>

<sup>5</sup><http://www.sqlite.org/>

### 4.3. Initial Processing

32

exist for C, Perl and Python among others. The choice of language used to implement a processing tool or script is therefore not constrained.

#### 4.2.6 Experiment Execution

Once all the tools were set up, capturing experiments were executed numerous times by starting the capture application and allowing it to record data for various periods of time. Initially, when exploring the range of packets available, the capture times were kept short.

The data used for the processing results presented in this chapter was recorded continuously over a two week period from a single network location.

## 4.3 Initial Processing

The data acquisition process resulted in numerous binary data files containing captured data. The initial processing step focused on getting this data decoded and placed into the database.

The Organisationally Unique Identifier (OUI), as discussed in Section 2.4.2, was identified as another source of information that could be used to identify nodes on the network. The procedure for obtaining OUI data and inserting it into the database is discussed in Section 4.3.4.

### 4.3.1 Packet Selection

A decision had to be made as to which packets, from the variety of protocols carried on the Ethernet network, to use for the construction of the node list. ARP packets (refer to Section 2.3.1) were the obvious choice considering the following preferences:

- We would like to create a list of nodes at the data link OSI layer.
- We would like to use packets that occur on the network quite frequently.

The processing of the ARP packets are discussed in the following section.

After reviewing some of the captured data it was realised that the second most common broadcast packets<sup>6</sup>, after ARP packets, on the network were Server Message Block (SMB) browser announce packets. The SMB protocol is commonly used to share printers and files across a LAN and all Windows machines and Unix-like machines running Samba<sup>7</sup> broadcast these packets to make the rest of the network know about their existence.

The browser announce packets are carried inside User Datagram Protocol (UDP) packets at the transport layer of the OSI reference model (refer to Section 2.2). They are however delivered using the Ethernet broadcast address and this makes them relevant to our list construction process. The browser announce packets make it possible to associate human readable names with raw MAC addresses. The processing of browser announce packets is discussed in Section 4.3.3.

### 4.3.2 Address Resolution Protocol (ARP) Packets

The captured ARP packets were placed into a database table in order to enable queries and cross correlation with the other captured data. Two alternative procedures were investigated and will be discussed: The recorded binary data file can either be read, interpreted and inserted directly into the database by an application, or the file can be converted to a more parse-friendly intermediate format before it is inserted into the database.

#### ARP Database Table

The ARP database table (named “arp”), created during the data capture process, consisted of the following fields:

- timestamp – Time in seconds since January 1 1970 when the packet was captured from the network.

---

<sup>6</sup>Packets with an Ethernet destination address of FF:FF:FF:FF:FF:FF (see also Section 2.4.2).

<sup>7</sup><http://www.samba.org/>

### 4.3. Initial Processing

34

- eth\_src\_mac – The source MAC address of the received Ethernet frame.
- eth\_dst\_mac – The destination MAC address of the received Ethernet frame.
- opcode – Whether the ARP packet was a request or a reply.
- arp\_src\_mac – The hardware (MAC) address of the source node.
- arp\_src\_ip – The protocol (IPv4) address of the source node.
- arp\_dst\_mac – The hardware (MAC) address of the destination node.
- arp\_dst\_ip – The protocol (IPv4) address of the destination node.

#### Binary File to Database

A Perl script was written that understood the `libpcap` capture file format and placed the processed data into the database table. The main disadvantage of this direct approach is that the parser script needs to understand the binary (as on the wire) format of the packets it needs to process. For simple packets, such as ARP, this is possible, but the complexity quickly increases for larger multi-layer packets. It is therefore advisable to use a common well tested packet dissection library, rather than reinventing one's own packet parsing code.

The `NetPacket`<sup>8</sup> Perl library was available that could parse the binary ARP packet data and was used in the script. Simplified pseudo-code for the script is shown in Figure 4.1.

#### Intermediate File to Database

After some investigation into file formats used for storing captured network traffic data, the choice fell on the Packet Details Markup Language (PDML) [98]. PDML uses an eXtensible Markup Language (XML) file format that contains details of decoded network packets.

---

<sup>8</sup><http://www.cpan.org/modules/by-module/NetPacket/>

### 4.3. Initial Processing

35

```
open the capture file
open a connection to the database
while there are frames in the capture file:
    read a single frame
    if the frame contains an ARP packet:
        decode the fields of the frame and the ARP packet
        build up an SQL instruction to add the data to the database
        execute the SQL statement
close the capture file and database connection
```

Figure 4.1: Pseudo-Code for Parsing Address Resolution Protocol (ARP) Packets.

The binary capture files were converted using the `tethereal`<sup>9</sup> utility. Tethereal supports further filtering<sup>10</sup> of packets as they are read from the capture file. The output PDML file can therefore be constructed so that it only contains the packets we are interested in (in this case the ARP packets).

Figure 4.2 shows a single packet excerpt from an ARP PDML file. Note how the packet fields can easily be identified from the XML file.

The disadvantage of converting the binary dump file to an XML format is that the resulting file is an order of magnitude larger than the original file and therefore requires a lot of disk space. As an example a 64MB capture file can easily result in a 650MB ARP PDML file.

A Python script was written to parse the PDML file and place the ARP packet data into the ARP database table. The formatting of the text fields were kept consistent with that of the Perl script used earlier.

<sup>9</sup>The utility is part of the larger Ethereal package.

<sup>10</sup>The syntax of the filter is different from that used for the packet capture. The syntax can be found in the Ethereal documentation [99].

### 4.3. Initial Processing

36

```

<proto name="arp" showname="Address Resolution Protocol (request)"
size="28" pos="14">

  <field name="arp.hw.type" showname="Hardware type: Ethernet (0x0001)"
size="2" pos="14" show="0x0001" value="0001"/>

  <field name="arp.proto.type" showname="Protocol type: IP (0x0800)"
size="2" pos="16" show="0x0800" value="0800"/>

  <field name="arp.hw.size" showname="Hardware size: 6"
size="1" pos="18" show="6" value="06"/>

  <field name="arp.proto.size" showname="Protocol size: 4"
size="1" pos="19" show="4" value="04"/>

  <field name="arp.opcode" showname="Opcode: request (0x0001)"
size="2" pos="20" show="0x0001" value="0001"/>

  <field name="arp.src.hw_mac" showname="Sender MAC address: 146.64.246.1 (00:0a:04:9b:cf:80)"
size="6" pos="22" show="00:0a:04:9b:cf:80" value="000a049bcf80"/>

  <field name="arp.src.proto_ipv4" showname="Sender IP address: 146.64.240.2 (146.64.240.2)"
size="4" pos="28" show="146.64.240.2" value="9240f002"/>

  <field name="arp.dst.hw_mac" showname="Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)"
size="6" pos="32" show="00:00:00:00:00:00" value="000000000000"/>

  <field name="arp.dst.proto_ipv4" showname="Target IP address: 161.71.15.126 (161.71.15.126)"
size="4" pos="38" show="161.71.15.126" value="a1470f7e"/>

</proto>
  
```

Figure 4.2: Example ARP Packet in PDML Syntax.

#### 4.3.3 Browser Announce Packets

As was done with the ARP packets, all captured browser announce packets were placed into a database table. The procedure is discussed below.

##### Browser Announce Database Table

The browser announce database table (named “browser”), containing the captured browser announce packets, consisted of the following fields:

- timestamp – Time in seconds since January 1 1970 when the packet was captured from the network.
- src\_mac – The MAC address of the sender as per the Ethernet frame.
- src\_ip – The IP address of the sender as per the UDP packet.
- name – The name that the node uses for itself.

### 4.3. Initial Processing

37

- comment – A descriptive string to used by the node for further identification.
- os\_major – The major version number of the operating system.
- os\_minor – The minor version number of the operating system.

#### Packet Parsing

The PDML [98] file format was again used as an intermediate format to transfer the raw packet data from capture files into the database. A filter was used during creation of the PDML file to make sure that it only contained browser announce packets.

An excerpt of a PDML file showing the relevant fields of a browser announce packet is shown in Figure 4.3.

```

<proto name="browser" showname="Microsoft Windows Browser Protocol" size="33" pos="210">
  <field name="browser.command" showname="Command: Host Announcement (0x01)"
  size="1" pos="210" show="0x01" value="01"/>

  <field name="browser.update_count" showname="Update Count: 0"
  size="1" pos="211" show="0" value="00"/>

  <field name="browser.period" showname="Update Periodicity: 12 minutes"
  size="4" pos="212" show="720000" value="80fc0a00"/>

  <field name="browser.server" showname="Host Name: SLEDIGA1"
  size="16" pos="216" show="SLEDIGA1" value="534c4544494741310000000000000000"/>

  <field name="browser.os_major" showname="OS Major Version: 5"
  size="1" pos="232" show="5" value="05"/>

  <field name="browser.os_minor" showname="OS Minor Version: 0"
  size="1" pos="233" show="0" value="00"/>

  <field name="browser.sig" showname="Signature: 0xaa55"
  size="2" pos="240" show="0xaa55" value="55aa"/>

  <field name="browser.comment" showname="Host Comment: "
  size="1" pos="242" show="" value="00"/>

</proto>

```

Figure 4.3: Example Browser Announce PDML File Snippet.

A Python script was written to parse the PDML fields and insert the data into the database table. The script loaded the whole PDML file into memory



### 4.3. Initial Processing

38

and this became a limitation when large capture files were used. Instead of rewriting the parsing code it was possible to split the binary input file using either the `tcpsplit` or `editcap` utility.

#### 4.3.4 Organisationally Unique Identifier (OUI) Data

OUI data allows for the matching of MAC addresses to Ethernet vendors (refer to Section 2.4.2). This data was also placed into a database table.

##### OUI Database Table

The OUI database table (named “oui”) consisted of the following fields:

- code – The hexadecimal value of the OUI entry.
- info – The vendor name associated with the entry.

##### Data Parsing

The OUI data can be obtained directly from the IEEE website<sup>11</sup> as a flat text file. An excerpt from the file is shown below:

00-03-E4	(hex)	Cisco Systems, Inc.
0003E4	(base 16)	Cisco Systems, Inc. 170 West Tasman Dr. San Jose CA 95134 UNITED STATES
00-03-E5	(hex)	Hermstedt SG
0003E5	(base 16)	Hermstedt SG Carl-Reuther - Str. 3 D-68305 Mannheim GERMANY

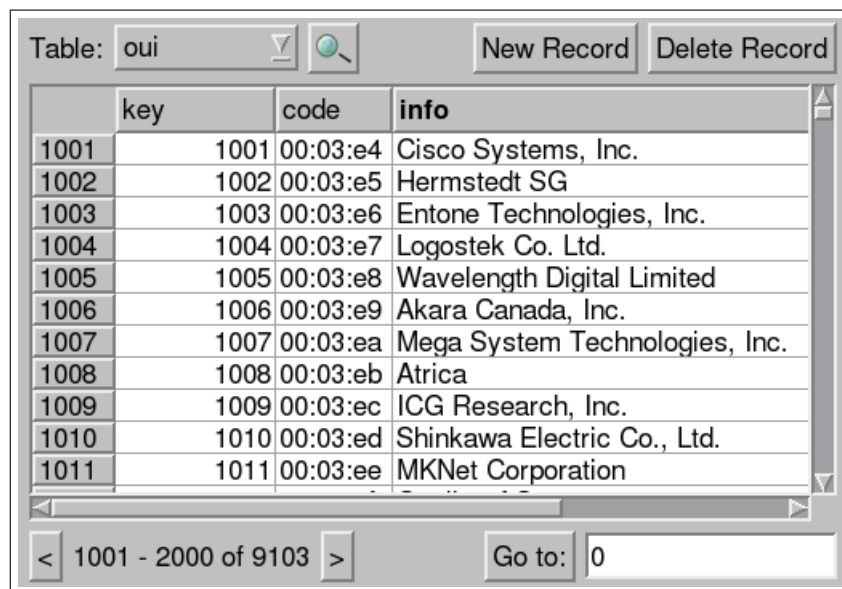
<sup>11</sup><http://standards.ieee.org/regauth/oui/index.shtml>

#### 4.4. Node List Generation

39

For our purposes, the first line of each entry contained sufficient information to place into the database. A Perl script was used to parse the OUI text file and extract the first three octets and the company name from the data. In order to make the data consistent, the first three octets were converted to lowercase and separated by colons.

Figure 4.4 shows an excerpt of the created database table. At the time of executing the script a total of 9103 entries were added to the database.



	key	code	info
1001	1001	00:03:e4	Cisco Systems, Inc.
1002	1002	00:03:e5	Hermstedt SG
1003	1003	00:03:e6	Entone Technologies, Inc.
1004	1004	00:03:e7	Logostek Co. Ltd.
1005	1005	00:03:e8	Wavelength Digital Limited
1006	1006	00:03:e9	Akara Canada, Inc.
1007	1007	00:03:ea	Mega System Technologies, Inc.
1008	1008	00:03:eb	Atrica
1009	1009	00:03:ec	ICG Research, Inc.
1010	1010	00:03:ed	Shinkawa Electric Co., Ltd.
1011	1011	00:03:ee	MKNet Corporation

Figure 4.4: OUI Database Table Example.

#### 4.4 Node List Generation

The initial processing steps placed all the data needed for the generation of the network node list into the database. In order to complete the list the data from the various tables had to be manipulated and merged.

The ARP and browser announce tables contained data directly converted from the capture files and potentially had many duplicate entries. As a first step unique nodes had to be identified, after which data from the unique nodes were merged with the OUI table.

## 4.4. Node List Generation

40

### 4.4.1 Unique MAC and IP Pairs

It is possible for a single node in the database to have multiple IP addresses for its specific MAC address. The node can either have multiple IP addresses assigned (either manually or automatically through the use of the Dynamic Host Configuration Protocol (DHCP)) over the course of the data acquisition period. In order to allow for these cases, nodes had to be grouped not only according to unique MAC addresses, but according to unique pairs of MAC and IP addresses.

The SQL “GROUP BY” [100] command was used to create a new database table named “uniq\_mac\_ip” that contained unique entries with respect to MAC and IP address pairs. The command executed was:

```
CREATE TABLE uniq_mac_ip AS SELECT * FROM arp  
GROUP BY arp_src_mac, arp_src_ip
```

### 4.4.2 Unique MAC and Browser Name Pairs

As with the MAC and IP pairs, a situation can occur where a single MAC address can relate to multiple browser names. The most obvious example occurs when a node’s name is changed during the data acquisition period.

A new database table named “uniq\_mac\_browse” was created using the following SQL command:

```
CREATE TABLE uniq_mac_browse AS SELECT * FROM browser  
GROUP BY src_mac, name
```

### 4.4.3 Unique Nodes Merged with OUI Data

The unique node database tables as created earlier in this Section were now merged with the OUI database table (as created in Section 4.3.4) to form the final network node list.

## 4.5. Additional Processing

41

The unique node tables are joined by comparing the MAC address of the nodes to form a new table named “mac\_ip\_name”. The SQL command is shown below:

```
CREATE TABLE mac_ip_name AS
SELECT arp_src_mac, arp_src_ip, name FROM
uniq_mac_ip LEFT JOIN uniq_mac_browse ON
arp_src_mac = uniq_mac_browse.src_mac
ORDER BY arp_src_mac
```

The list was completed by adding the vendor name from the OUI database table. The vendor code in the OUI table is compared to a substring<sup>12</sup> of the MAC address from the “mac\_ip\_name” table created above. The SQL statement to add to OUI data and create a new table named “mac\_ip\_name\_vendor” is shown below:

```
CREATE TABLE mac_ip_name_vendor AS
SELECT mac_ip_name.*, oui.info FROM
mac_ip_name, oui WHERE
SUBSTR(mac_ip_name.arp_src_mac,1,8) = oui.code
ORDER BY arp_src_ip
```

The final list contained the MAC address of the node, its IP address, the browser name (if present) and the vendor name of the network card used by the node. Figure 4.5 shows a few rows from the final table.

## 4.5 Additional Processing

The network node list as generated in Section 4.4 already contains a lot of information and is useful in itself, but further queries performed on the list can reveal even more information. This section explores some processing steps that revealed more information from the generated list as well as steps that revealed additional information from the original database tables.

---

<sup>12</sup>The substring length was chosen to span the first 3 octets of the MAC address.

## 4.5. Additional Processing

42

Table: mac_ip_name_vendor				
arp_src_mac	arp_src_ip	name	oui.info	
180	00:00:e2:99:47:46	146.64.246.75	DEXTER	ACER TECHNOLOGIES CORP.
181	00:c0:df:07:57:81	146.64.246.77	IWSUPER2	KYE Systems Corp.
182	00:04:76:51:79:12	146.64.246.81	CVDMERWE1	3 Com Corporation
183	00:04:76:4f:82:36	146.64.246.88	ASMIT-NB1	3 Com Corporation
184	00:01:02:38:2d:74	146.64.246.89	FLEROUX2	3COM CORPORATION
185	00:04:76:26:0f:f5	146.64.246.93	PHANTOM	3 Com Corporation
186	00:0b:db:1a:b5:f7	146.64.248.118	ADEZINGER-NB1	Dell ESG PCBA Test
187	00:02:b3:0f:04:55	146.64.248.16	PA-B44-A122-HP5	Intel Corporation
188	00:0a:04:9b:cf:80	146.64.248.2		3Com Europe Ltd
189	00:0b:db:1a:b6:49	146.64.248.24	RPELSER-NB1	Dell ESG PCBA Test
190	00:60:08:0e:84:cd	146.64.248.3		3COM CORPORATION
191	00:0a:04:9b:cf:80	146.64.249.1		3Com Europe Ltd
192	00:0c:f7:11:80:fe	146.64.254.1		Nortel Networks

Figure 4.5: Passively Discovered Node List Table Extract.

### 4.5.1 Multiple IP Addresses or Names

Nodes with multiple IP addresses for a specific MAC address are often interesting. If multiple IP addresses from separate IP subnets are observed in a short space of time, it indicates that the specific node functions as a router between the subnets. The probability of the node being a router can be increased by looking at the vendor attached to the specific node's MAC address in the list created in Section 4.4. If the IP address changes more infrequently, it can either indicate that the node gets an address through the use of DHCP or that someone is manually changing the node's static IP address.

DHCP address allocation occurs mostly when a node is switched on, so these type of IP address changes can be identified by looking for the presence of ARP packets in the time vicinity of the change. If the IP address change is not preceded by other ARP packets, the node has probably been switched on and has received an IP address using DHCP. The subnet served by the DHCP server can also be identified by observing these types of changes. If the IP address change is preceded in time by other ARP packets using the same MAC address, the IP has more likely been changed manually.

Multiple node names are not as interesting as multiple IP addresses, but can provide some history on the names used by the node.

## 4.5. Additional Processing

43

As a first step, we identified nodes with multiple IP addresses or names by counting.

```
CREATE TABLE mult_ip AS
SELECT *, COUNT(*) AS n FROM mac_ip_name_vendor
GROUP BY arp_src_mac HAVING n > 1
```

We then created a new table called “mult\_ip\_list” where the multiple IP addresses are listed by matching MAC addresses as follows:

```
CREATE TABLE mult_ip_list AS
SELECT mac_ip_name_vendor.*
FROM mac_ip_name_vendor, mult_ip
WHERE mac_ip_name_vendor.arp_src_mac=mult_ip.arp_src_mac
ORDER BY mac_ip_name_vendor.arp_src_mac
```

Figure 4.6 shows a selection of rows from the resulting multiple IP table.

Table: mult_ip_list					New Record	Delete Record
	arp_src_mac	arp_src_ip	name	oui.info		
40	00:04:76:e3:04:6f	146.64.245.227	COMPUTER1	3 Com Corporation		
41	00:04:76:e3:04:6f	146.64.245.227	EDDIE	3 Com Corporation		
42	00:04:76:e3:04:6f	146.64.245.227	EDDIE1	3 Com Corporation		
43	00:04:76:e3:04:6f	146.64.245.227	JSB	3 Com Corporation		
44	00:06:5b:6e:f8:9a	0.0.0.0	PSMIT-WS2	Dell Computer Corp.		
45	00:06:5b:6e:f8:9a	146.64.254.30	PSMIT-WS2	Dell Computer Corp.		
46	00:06:5b:6e:f8:9a	146.64.254.91	PSMIT-WS2	Dell Computer Corp.		
47	00:0a:04:9b:cf:80	146.64.240.2		3Com Europe Ltd		
48	00:0a:04:9b:cf:80	146.64.243.1		3Com Europe Ltd		
49	00:0a:04:9b:cf:80	146.64.245.1		3Com Europe Ltd		
50	00:0a:04:9b:cf:80	146.64.246.1		3Com Europe Ltd		
51	00:0a:04:9b:cf:80	146.64.248.2		3Com Europe Ltd		
52	00:0a:04:9b:cf:80	146.64.249.1		3Com Europe Ltd		
53	00:0a:04:9b:cf:80	146.64.254.10		3Com Europe Ltd		

Figure 4.6: Nodes with Multiple IP Addresses Table Extract.

### 4.5.2 Packet Timestamps

The ARP packet timestamps reveal if a node on the network was active on the network at a particular time. Desktop machines are more likely to be

#### 4.5. Additional Processing

switched off daily than server machines and it was hoped that the timestamp data would be able to show this. It was decided to use plots of the timestamp data to investigate node activity.

In order to generate a plot of ARP packet timestamps a node address was chosen manually from any of the tables created earlier. A simple SQL query retrieved all the ARP packets for the specific node, after which the data was exported as a comma delimited text file and plotted using Matlab.

Figure 4.7 shows the ARP activity of a node over the period of a week. This node was clearly switched on and off daily. Figure 4.8 shows the ARP activity of a node that remained on during the night (using the same timescale as Figure 4.7).

The ARP packet timestamps therefore help in distinguishing desktop machines from possible server machines. Note also how (in a possible privacy invasive way) a desktop user's habits in terms of working hours can be determined from the PC's ARP timestamps.

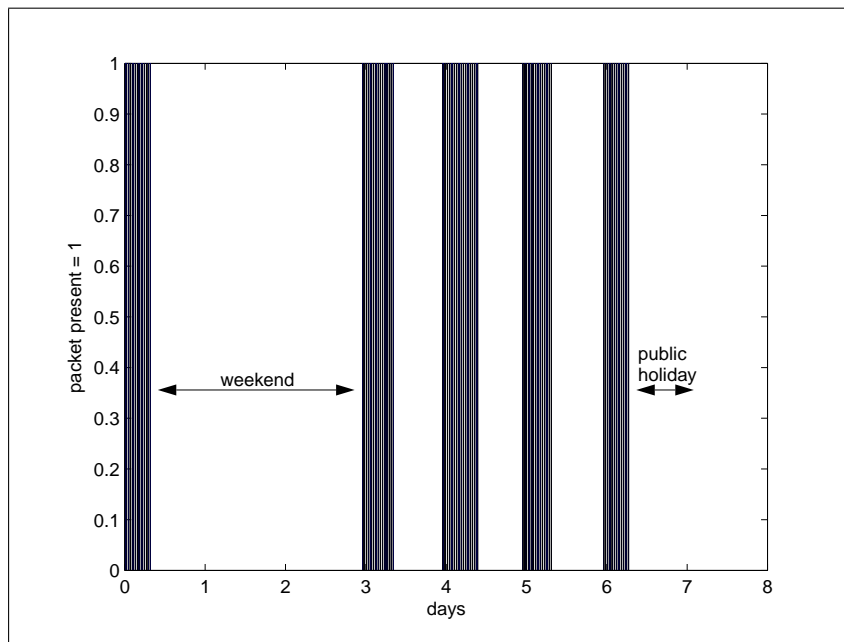


Figure 4.7: Desktop Node ARP Timestamps.

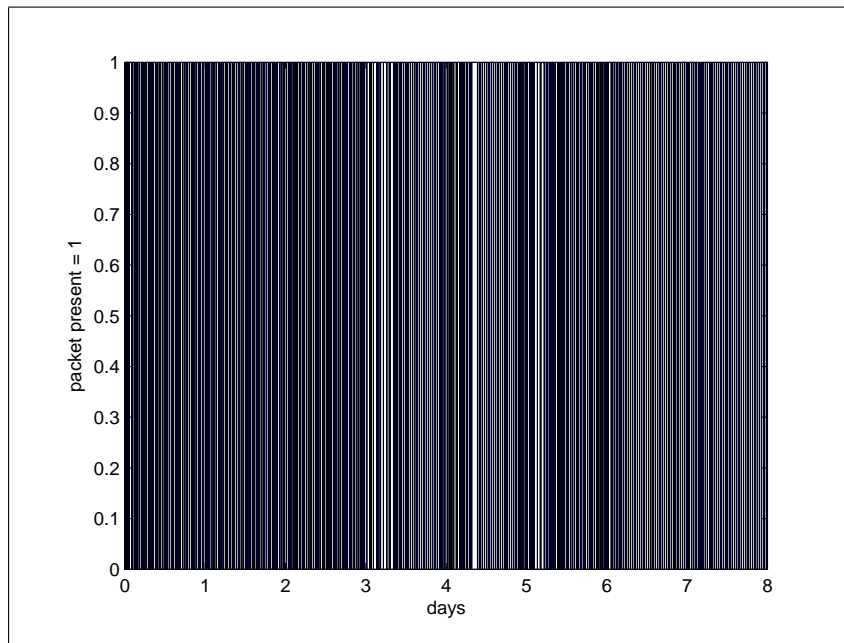


Figure 4.8: Possible Server Node ARP Timestamps.

### 4.5.3 Communication Peers

The ARP traffic present on the network also gives an indication of all the nodes a specific node tries to communicate with (refer to Section 2.3.1). An attempt was made to visually show this relationship.

A new table was created that counted the number of times a certain node (identified by its MAC address) requested a specific IP address.

```
CREATE TABLE ip_req_count AS
SELECT *, COUNT(*) as n FROM arp
GROUP BY arp_src_mac, arp_dst_ip having n > 1
ORDER BY arp_src_mac
```

The requested IP addresses were then converted back to MAC addresses by using the unique MAC and IP pair table created in Section 4.4.1. Note that a specific IP address can possibly relate to multiple MAC addresses, but for this experiment only the first match was chosen.



## 4.5. Additional Processing

46

```
CREATE TABLE arp_peers AS
SELECT n, ip_req_count.arp_src_mac AS src,
uniq_mac_ip.arp_src_mac AS dst FROM
uniq_mac_ip INNER JOIN ip_req_count
ON uniq_mac_ip.arp_src_ip = ip_req_count.arp_dst_ip
ORDER BY src
```

The created “arp\_peers” table now contained a source MAC address, a guessed destination MAC address (using the reverse look up) and a count approximating the number of times the source wanted to communicate with the destination. It was decided to plot the communication relationships as a force directed graph [101]<sup>13</sup> with the force between the nodes related to the request count.

An example of an output graph is shown in Figure 4.9. After examining the graph in more detail it was noted that the nodes belonging to a specific subnet were grouped together in a cluster. Figure 4.10 shows a close-up of the cluster of nodes in the top-left corner of Figure 4.9 and it can be seen that they belong to the same subnet when looking at the IP addresses.

The router between the different subnets was also clearly distinguishable in the middle of the graph. Nodes that sit in between the clusters could have moved between subnets during the course of the data capture or they might have improper IP network masks [47] set.

---

<sup>13</sup>The `fdp` utility from the GraphViz package was used to generate the graphs.

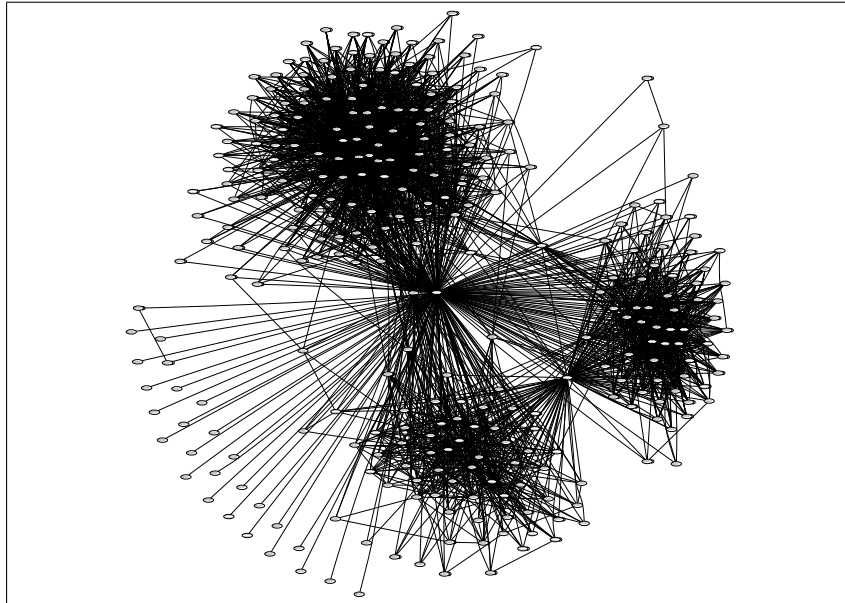


Figure 4.9: Graph of Communication Peers.

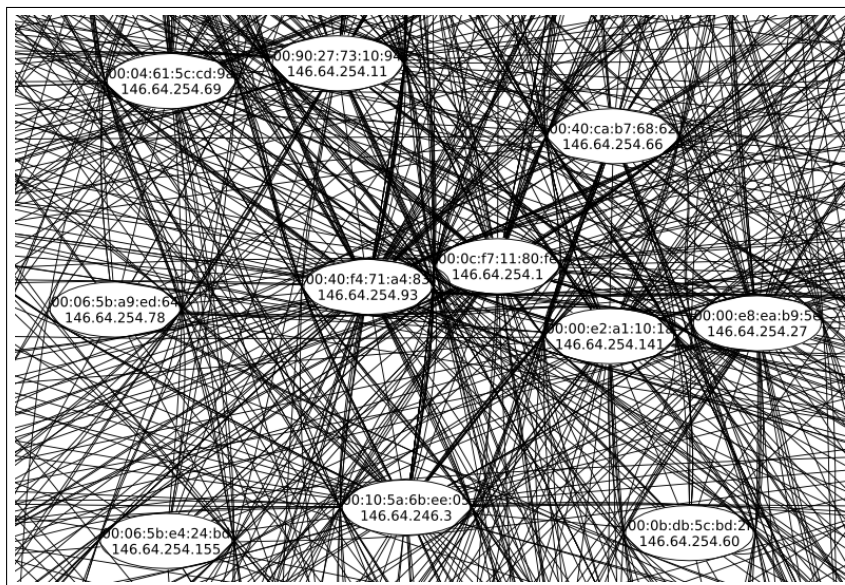


Figure 4.10: Graph of Communication Peers (Zoomed In).

---

## 4.6 Conclusion

The main goal of the chapter, the generation of a network node list, was achieved by the experiments and processing presented. The generated list represents all the nodes whose presence could be determined passively by means of their broadcasting of ARP request packets.

The generated list can also be seen as an inventory of the elements on the network. If this list is generated periodically it can be compared to snapshots of the network at different times in order to reveal changes to the network.

The processing described in Section 4.5 also showed that it is possible to identify routers in the network not only by means of their MAC addresses, but also through their communication patterns.

The list of network nodes contains initial knowledge about a target network and is used in further experiments conducted during the course of our research.

## Chapter 5

# Internal Network Device Discovery

### 5.1 Introduction and Overview

The results of the experiments and data processing presented in Chapter 4 provided information about the nodes of the network, but not a lot of information about the connections between these nodes. The focus of this chapter is on finding the internal elements of the network as a step towards identifying and characterising the links between nodes of the network.

The chapter is organised as follows. Details about the motivation, considerations and design of the experiments are discussed in Section 5.2. Section 5.2 also provides details concerning the hardware and software used during the experiments. The execution of the experiments is discussed in Section 5.3. Results of the experiments are presented in Sections 5.4, 5.5 and 5.6. Finally some conclusions are drawn in Section 5.7.

## 5.2 Experiment Design and Preparation

### 5.2.1 Motivation

Internal network elements such as switches and hubs (refer to Section 2.4.5), even though transparent to the network layer, influence the delivery times of packets in the network. Store-and-forward delays [39, p. 166], carrier-sense delays as well as collisions [27, p. 282] influence the time it takes for an Ethernet frame to reach a receiver on the network.

The experiments in this chapter were constructed to explore the feasibility of detecting store-and-forward, carrier-sense and collision delays experienced by an Ethernet frame as it travels across the network. It was hoped that the temporal influence of internal elements on packet delivery times would reveal the existence of the elements themselves.

### 5.2.2 Considerations

Proper preparation for the experiments required consideration of two problems: What is the magnitude of packet delays we can expect and how can we measure these delays?

The delays experienced by a packet as it travels across an uncongested Ethernet network are quite small. A rough calculation (Equation 5.1) showed that a 1000-byte packet takes approximately 80 microseconds to transmit on a 100Mbps Ethernet network. This is also the typical delay a packet would experience as it traverses a store-and-forward element in the network (refer to Section 2.4.5).

$$\frac{1000 \text{ bytes}}{100 \text{ Mbps}} = \frac{(1000 * 8) \text{ bits}}{100 \times 10^6 \text{ bits}} s = 80 \times 10^{-6} s \quad (5.1)$$

In order to measure the delay experienced by a packet, two methods were considered:

## 5.2. Experiment Design and Preparation

51

1. Send a packet from one network node to another and record the sending time on the sender and the reception time on the receiver. This method allows calculation of the one-way delay the packet experiences, but requires accurate clock synchronisation between the nodes as well as the cooperation of the target node.
2. Send a packet that evokes a response from the target node and record the sending time as well as reception time of the response on a single node. This method measures the round-trip time of the packet and is simpler in terms of time-stamping and clock synchronisation [79], but it has to contend with the uncertainty in the target node's response time.

The second method was chosen for the experiments in this chapter because of its simplicity. It could also potentially be used in a live network environment if suitable packets were used. The selection of probe packets is addressed in the following section. The infrastructure, in terms of hardware and software, required to measure the small round-trip delays are discussed in Sections 5.2.4, 5.2.5 and 5.2.6.

### 5.2.3 Packet Selection

The round-trip technique for measuring physical link delays, chosen for its time-stamping simplicity as described in Section 5.2.2, relies on packets that evoke a response from a target node. Two types of packets were considered, namely: Internet Control Message Protocol (ICMP) echo request/reply messages and ARP packets.

ICMP is a network layer (OSI layer 3) protocol [23, p. 508] used for sending diagnostic and notification messages for other protocols in the TCP/IP suite [38, p. 134]. The ICMP request/reply messages are used as the most basic test to see if two stations can send IP packets to each other [23, p. 536]. The fact that ICMP packets are **routed**, in the same way as IP packets, disqualifies them for use in our experiments.

## 5.2. Experiment Design and Preparation

Routing at layer 3 can provide misleading information about the layer 2 topology of the network, for example: two stations connected to the same switch but with different network masks<sup>1</sup> would communicate through a router (that could be connected through a number of other layer 2 network elements) even though they are in fact directly connected. ICMP request packets are also quite often **filtered** either at the target host or at intermediate routers [102, 80].

In comparison to ICMP, ARP packets (refer also to Section 2.3.1) have the following useful properties:

Probing **targets are abundant**, because any host on an Ethernet LAN that wishes to communicate using the TCP/IP protocol has to listen for ARP broadcasts requesting its IP address [21, p. 283]. If the host receives a valid ARP request, it should send a targeted reply to the sender containing its own MAC address. There is also a good chance that ARP request packets will not be filtered by personal firewall software operating at the network layer.

Secondly, packets are **easy to process**. ARP request packets are easy to generate and ARP reply packets are easy to parse by host software.

Thirdly, ARP packets are normally handled at a low level in the network stack of a target node, which increases the predictability of reply times and makes **reply times more consistent**.

Most importantly, **various packet sizes** can be used. ARP request packets have a specific size (refer to Section 2.3.1), but during experimentation it was found that they remain valid even if encapsulated within Ethernet frames of various sizes. Variations in Ethernet frame sizes cause variable delays when an ARP request packet moves through a store-and-forward element [39, p. 166].

By virtue of the properties mentioned above, ARP packets were chosen for the experiments presented in this chapter.

---

<sup>1</sup>The two stations are on different IP subnets.

## 5.2. Experiment Design and Preparation

53

### 5.2.4 Operating System Selection

In order to measure the sub-millisecond delays introduced by Ethernet network elements, the transmission and time-stamping of packets had to be controlled very accurately. The use of a real-time operating system was investigated and it was decided to use RTLinux/GPL [103] for the experiments.

RTLinux is a hard real-time operating system that runs the standard Linux kernel as its lowest priority thread [104]. RTLinux provides low (at the interrupt and IO port) level access to devices, but still allows one to use all the normal GNU/Linux tools for non-real-time tasks.

### 5.2.5 Hardware and Device Drivers

The RTLinux distribution by itself does not provide real-time network drivers for Ethernet cards, but relies on the non-real-time Linux drivers; however, the RTLinux Ethernet Device Drivers (REDD) project [105, 106] has done some work to produce real-time Ethernet drivers.

The REDD project provided a driver for the RTL8139 Ethernet chipset [107] used by a lot of inexpensive Peripheral Component Interconnect (PCI) network cards. An 80x86 PC equipped with an RTL8139 based network card was used for the execution of our experiments.

#### Device Driver Modifications

The device driver provided by the REDD project could be used to send raw Ethernet frames, but did not have the ability to time-stamp the transmission or reception of packets. The driver was modified using information about the design and architecture [105, 107, 108] in order to record the time<sup>2</sup> when a packet was fully transmitted by the RTL8139 chip (the packet was on the wire) and the time when a complete packet was received.

---

<sup>2</sup>The RTLinux clock resolution is well below  $1\mu s$ , but worst case interrupt latency can be up to  $40\mu s$  [109].



## 5.2. Experiment Design and Preparation

54

In order to measure round-trip times, a packet is sent to a target host and time of transmission is recorded. When the reply from the target arrives the time-stamp of the reply packet is again recorded, but identifying the reply packet amongst all the other traffic on the network proved problematic. A filter had to be added to the device driver to only consider and time-stamp received packets matching the target node's MAC address.

### 5.2.6 Software

RTLinux applications consist of two parts: a real-time kernel module and a userspace application [104]. The module is responsible for the real-time tasks and is kept as small, fast and deterministic as possible. The module runs in kernel-space and can use only a limited set of RTLinux and Linux kernel system calls. The userspace application can use all available GNU/Linux tools and libraries and communicates with the real-time module using RTLinux First-In-First-Out (FIFO) buffers.

#### Kernel Module

A kernel module was written to send ARP packets (or groups of packets) using a variety of parameters. The module was also responsible for collecting the time-stamp data from the device driver and for passing all the data to the userspace application.

The parameters passed to the module included: the target (destination) MAC address, the target (destination) IP address, the number of ARP packets to send in close succession (a maximum of 10 was allowed), an array containing the packet lengths (in bytes) for each of the packets and an array containing the delay in microseconds between packets  $n$  and  $n + 1$ .

The kernel module main loop contained the following steps:

1. Wait for parameters (as described above) from the userspace application.

### 5.3. Experiment Execution

55

2. Send ARP packets using these parameters using the device driver.
3. Wait for reply packets or time-out.
4. Collect time-stamp data from the device driver and push into the FIFO buffers.

#### Userspace Application

The kernel module provided the flexibility for conducting a wide range of experiments and all the userspace application had to do was provide the module with appropriate parameters and save the data received from the kernel module to disk. The application therefore changed depending on the specific experimental needs.

The data was saved in a simple space delimited text file that allowed for easy reading into processing applications.

## 5.3 Experiment Execution

The experimental setup as discussed in the previous sections allows for various experiments to be performed. This section provides specific details about the execution environment and parameters used during the experiments.

### 5.3.1 Network Configurations

Initially a few simple tests were performed on a live corporate network, but it was soon realised that the number of variables were too many to make sense of the recorded data. It was therefore decided to perform experiments in a more controlled environment.

A range of network configurations, as shown in Figure 5.1, were constructed to determine the effect of specific elements on the round-trip time of ARP packets.

5.3. Experiment Execution

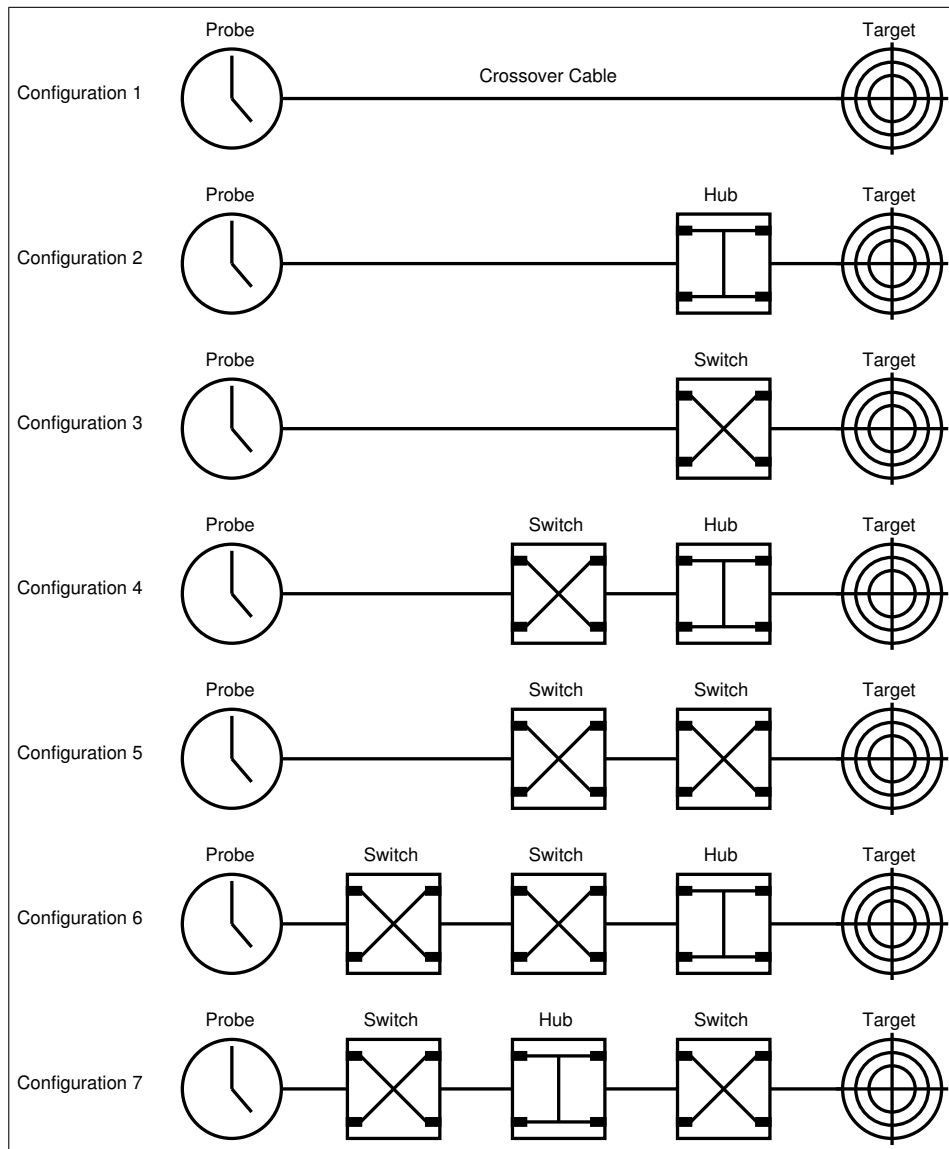


Figure 5.1: Experimental (Controlled) Network Configurations.

### 5.3. Experiment Execution

57

The network configurations consisted of a mixture of two unmanaged network switches (both 100Mbps, but from different manufacturers) and a 100Mbps network hub. The target node was a laptop running the Debian GNU/Linux distribution, while the probing host was a PC with a PCI RTL8139 based network card as discussed in Section 5.2.5.

#### 5.3.2 Packet Configurations

A set of four experiments were conducted for each of the experimental network configurations described above. The experiments differed in the cocktail of packets that were sent to the target node. The ARP request packets (or sequence of packets) sent during each of the experiments consisted of the following:

1. Single packets with sizes of 60, 120, 240, 480 and 960 bytes 10 milliseconds apart.
2. A pair of packets (both 960 bytes in length) as close together in time as possible.
3. A triplet of packets (60, 960 and 60 bytes in length) as close together in time as possible.
4. A triplet of packets (960, 60 and 960 bytes in length) as close together in time as possible.

It was hoped that the combinations of packets sent close together in time would reveal the half-duplex or full-duplex (refer to Section 2.4.4) mode of the target node.

Even though we have control over the ARP request packet size, it should also be noted that the size of the ARP reply packets generated by the target node remains constant (60 bytes<sup>3</sup>).

---

<sup>3</sup>This includes 14 bytes of the Ethernet header (excluding the preamble and start of frame delimiter), the ARP payload of 28 bytes and an 18 byte pad. Together with the frame check sequence of 4 bytes, the minimum Ethernet frame size of 64 bytes is reached.

## 5.4 Single Packet Results

Single packets with various sizes (60, 120, 240, 480 and 960 bytes) were sent to the target node a total of 1000 times for each packet size and network configuration. The minimum size of an ARP packet is 60 bytes (refer to Section 2.3.1) and the packet sizes were chosen by starting with the minimum value and doubling it while the packet size remained valid for 100Mbit Ethernet.

The value of 1000 for the number of iterations was chosen arbitrarily, but was found to provide a good statistical distribution of round-trip times. The distribution of round-trip times for the 960-byte case for two of the network configurations are shown in Figure 5.2.

The noise in the data is probably due to the processing delay on the target node. Even though the network and target node was idle in our case, the differences could still occur because of operating system scheduling jitter and delays. In a live network the noise would also increase because of node and network activity and load.

It was decided to use the median of the 1000 samples of each experimental run for further processing and plots. Figure 5.3 shows the median of the round-trip times for all the packet sizes and network configurations used during the single packet experiments.

The following paragraphs discuss observations that were made from Figure 5.3.

The number of switch (store-and-forward) elements has a large influence on the round-trip times of the packets, while it seems that the hub has a negligible influence. This can be seen from the fact that the results of configuration 1 and 2, configuration 3 and 4 as well as configuration 5 to 7 are closely grouped together. Hubs could therefore not be identified using these single packets and some other method had to be investigated.

The extra delay caused by a store-and-forward element is very close to the theoretically calculated time it takes to transmit a packet. The round-trip

### 5.4. Single Packet Results

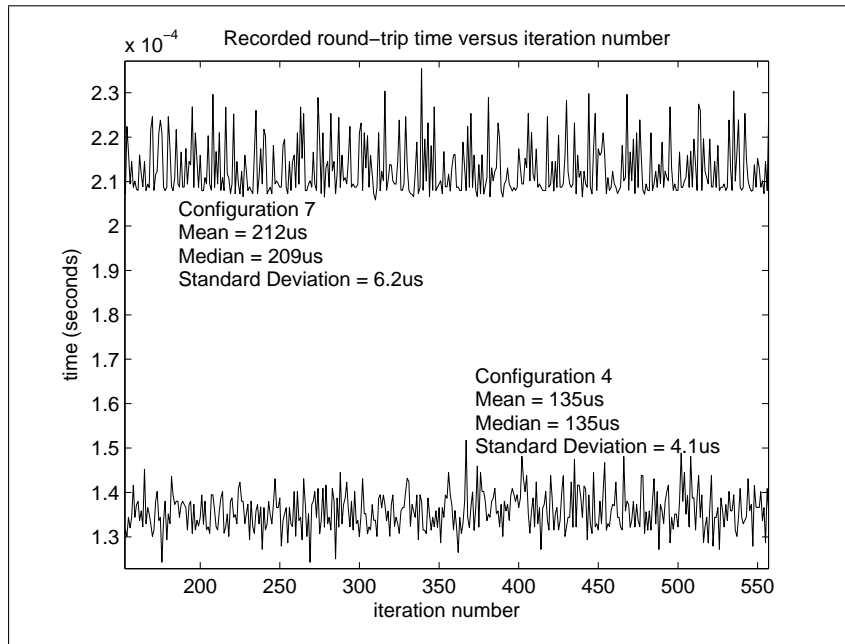


Figure 5.2: Typical Single Packet Round-Trip Time Distributions.

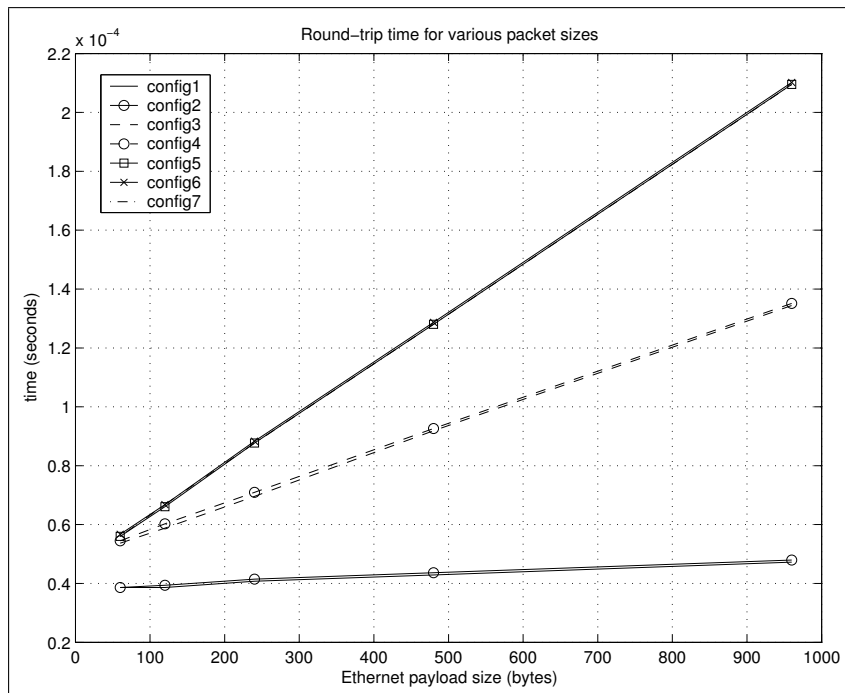


Figure 5.3: Round-Trip Times for Various Packets Sizes. The lines representing configurations 1 and 2 are close together, as are the lines of configurations 3 and 4, and also the lines of configurations 5 to 7.

## 5.5. Packet Group Results

60

delay of the 960-byte packets in Figure 5.3, of around 80 microseconds, compared very well with the transmit time calculated for a 1000-byte packet (Equation 5.1) in Section 5.2.2. Using this knowledge it should therefore be possible to determine if switching elements were added or removed between a probe and target node provided that timing information was saved earlier.

One can also guess<sup>4</sup> at the number of switches between the probing host and the target node by subtracting the round-trip time for the 60-byte packets from that of the 960-byte packet and dividing the answer by the delay added by a store-and-forward element for a 960-byte packet. The 60-byte round-trip time roughly represents the reply delay of the target host while the 960-byte round-trip time is mostly influenced by the store-and-forward elements present on the delivery path. A rough calculation for configuration 3 gives:

$$\frac{135\mu s - 55\mu s}{80\mu s} = \frac{80\mu s}{80\mu s} = 1$$

and for configuration 5 gives:

$$\frac{210\mu s - 55\mu s}{80\mu s} = \frac{155\mu s}{80\mu s} \approx 2$$

There is a linear (or at least very close to linear) relationship between ARP request packet size and the round-trip time. This makes sense if we remember that we control only the ARP request size. It is also easier to spot timing differences using the larger packet sizes and it was decided to use larger packets where possible in future experiments.

## 5.5 Packet Group Results

Section 5.4 showed that we could distinguish between three groups of network configurations using single packets. It was hoped that by using groups of packets we could find further round-trip time differences between the experimental network configurations.

---

<sup>4</sup>If we assume that all the store-and-forward elements operate at the same speed.

## 5.5. Packet Group Results

61

### 5.5.1 Pair of Large Packets

In this experiment a pair of 960-byte packets (closely spaced in time) were sent to the target node a total of 1000 times. The median of the round-trip times were recorded for both of the packets. Figure 5.4 shows the results for all the experimental network configurations.

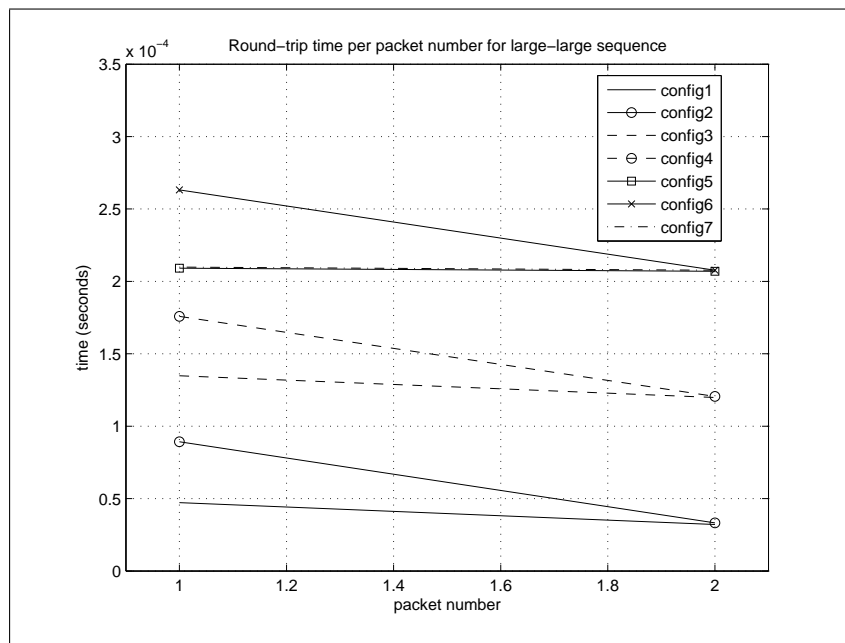


Figure 5.4: Large-Large Packet Group Results.

It can be seen that the first packet of the group experiences a greater delay than the second when a hub is the final element en-route to the target node. The additional delay can be explained by remembering the half-duplex nature of the hub and the CSMA/CD principles as explained in Section 2.4.4. The two large packets are sent back-to-back and the Ethernet carrier-sense [21, p. 139] circuitry in the target node forces it to wait until the second of the packets has been completely received before it can reply to the first packet.

The pair of large packets allows us to differentiate between network configurations 1 and 2, 3 and 4 as well as 5 and 6; however, configuration 7 still produces the same results as configuration 5.



## 5.5. Packet Group Results

62

### 5.5.2 Triplet of Packets (Small-Large-Small)

A triplet of packets with sizes of 60, 960 and 60 bytes respectively were sent to the target node a total of 1000 times and the median of the round-trip times for each of the packets was recorded. Figure 5.5 shows the results for all the experimental network configurations.

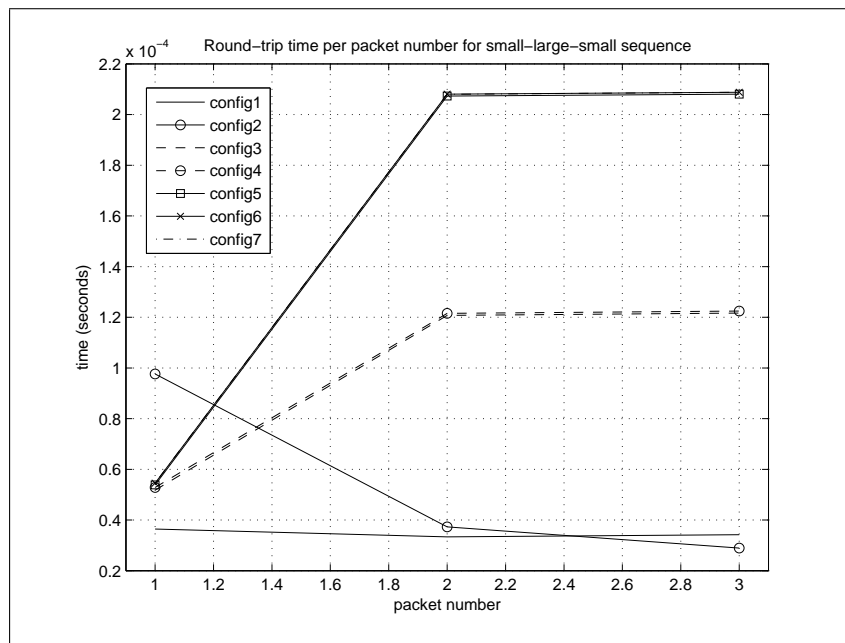


Figure 5.5: Small-Large-Small Packet Group Results.

It can be seen that this group of packets did not provide as good a means of differentiation between the different network configurations as the pair of packets in the previous (large-large packet) experiment. It is however included in our discussion to indicate the thought processes involved and the order in which the experiments were conducted.

The difference in behaviour of the first packet of network configuration 2 versus the other configurations is notable. One would expect the large packet (second one sent) to cause a delay in the reply to the first (small) packet whenever a hub is the final element en-route to the target (as in our previous experiment), but it occurs only for configuration 2. The apparent anomaly is explained by realising that the delay a packet experiences in a store-and-

## 5.6. Packet Delay Results

63

forward element is proportional to its size. The first small packet experiences only a short delay en-route to the target and the target has enough time to respond while the second larger packet is being received<sup>5</sup> by one of the store-and-forward elements. When no store-and-forward elements are present, as is the case for configuration 2, the target itself has to wait until it fully receives the larger second packet before the response to the first packet can be sent.

### 5.5.3 Triplet of Packets (Large-Small-Large)

For this experiment a triplet of packets was again used. Packets with sizes of 960, 60 and 960 bytes respectively were sent to the target node a total of 1000 times and the median of the round-trip times for each of the packets was recorded. Figure 5.6 shows the results for all the experimental network configurations.

In this case, whenever a hub is the final element en-route to the target, the target node has to fully receive the final large packet before it can send replies for the first two packets. The explanation is the same as that for the pair of large packets experiment discussed earlier (refer to Section 5.5.1). The triplet of large-small-large packets did not provide any additional information compared to the pair of large packets. The pair of large packets therefore remains the best way investigated so far to determine the presence of a hub as the last element en-route to the target node.

## 5.6 Packet Delay Results

The experiments discussed in Section 5.4 and Section 5.5 allowed for differentiation between 6 of the 7 experimental network configurations. Knowledge gained during these experiments about collisions and carrier-sense behaviour on the half-duplex part of a network link suggested a procedure for distinguishing the final network configuration from the rest.

---

<sup>5</sup>During the reception of the packet by the store-and-forward element the half-duplex link on its other port is not utilised.

If the second packet in a large packet pair is delayed by just the right amount, the carrier-sense circuitry of an element in half-duplex mode, anywhere between the probing host and target node, can be triggered. An experiment was set up to again send pairs of large packets, but the second packet of the pair was delayed by various amounts of time. The experiment was conducted on network configuration 5 and 7. Figure 5.7 shows the round-trip time for the first packet versus the delay before sending of the second packet.

It can be seen how the round-trip time for the first packet starts to increase when the second packet is delayed by around 120 microseconds. In this case, the second packet has barely made it through the half-duplex part of the link between the probing host and target. If the second packet is delayed further, the final switch en-route to the target has to wait a little bit longer for the second packet to make it through the half-duplex part of the link.

The maximum delay is achieved when the second packet reaches the half-duplex link just as the final switch en-route to the target node is about to send the reply to the first packet (between 200 and 220 microseconds in this case). This maximum delay is directly related to the time it takes to transmit the second packet. The value of around 76 microseconds for the maximum delay obtained by examining Figure 5.7 correlates well with the calculated time of around 80 microseconds to transmit a 1000-byte packet (Equation 5.1 in Section 5.2.2).

## 5.7 Conclusion

This chapter introduced a set of experiments with the goal of determining the presence of internal network elements in an Ethernet network. The results of the experiments conducted in a controlled network environment showed that it is definitely possible to determine when internal elements are added or removed from the network by examining the changes in the round-trip time of packets. Round-trip time information gathered at different dates can therefore be compared to determine if changes occurred to the internals of a network.

## 5.7. Conclusion

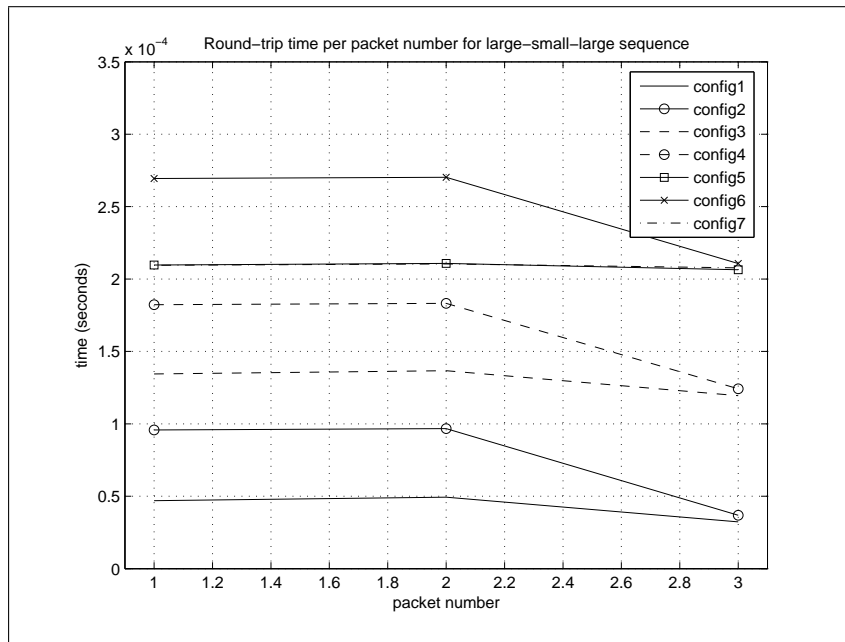


Figure 5.6: Large-Small-Large Packet Group Results.

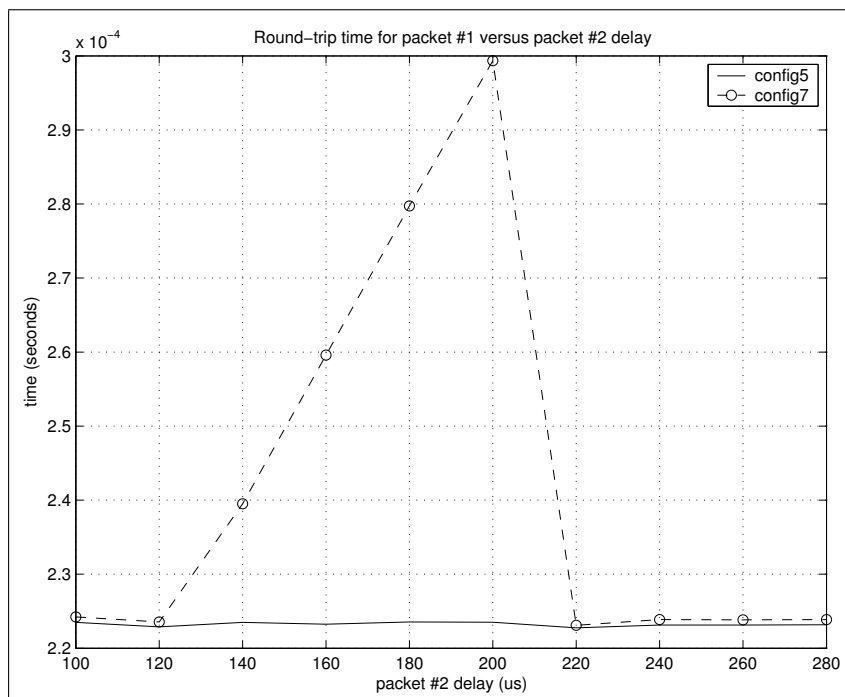


Figure 5.7: Delayed Large-Large Packet Group Results.

## 5.7. Conclusion

66

The round-trip time results for single packets with varying sizes (Section 5.4) presented a method for estimating the number of store-and-forward (switch) elements present between two nodes in the network, while results from pairs of large packets (Sections 5.5.1 and 5.6) showed that it is possible to detect the presence of hubs between two nodes in the network.

The work presented in this chapter was also delivered in a paper at the Southern African Telecommunication Networks and Applications Conference (SATNAC) in 2005 [110].

# Chapter 6

## Topology Inference

### 6.1 Introduction and Overview

Work presented in earlier chapters showed that it is possible to identify, and gain some knowledge about, network nodes (Chapter 4) and that it is possible to determine when network elements are added or removed from the internal network infrastructure (Chapter 5). This chapter builds on techniques developed earlier in an attempt to infer the internal structure of a live target network.

The design, planning and execution of the live network experiments are presented in Section 6.2. Observations related to initial processing of the measurement data is presented in Section 6.3. The initial observations are further explored in Section 6.4 to create the concept of node signatures and these signatures are then employed to create clusters of similar nodes. The similarity between clusters of nodes is then exploited in Section 6.6 to infer the connectivity between clusters of nodes. Finally, Section 6.7 presents conclusions drawn and future work identified during the experiments and data processing.

## 6.2 Experiment Planning and Execution

Empirical experiments were again used as the preferred method for tackling the topology inference problem. This section provides details on the design, setup and execution of the experiments performed on a single live corporate Ethernet network. The tools developed during earlier experiments provided a suitable starting point for the live network experiments.

### 6.2.1 Design Concept and Objective

Round-trip packet delay measurements proved successful at identifying the addition or removal of network elements between a probing host and a target node in a controlled network environment (refer to Chapter 5). In a live network this addition or removal of network elements can be mimicked by performing measurements from various network locations or viewpoints.

The number of network elements between a specific network viewpoint and a target node can potentially change as the viewpoint changes [111]. A change in the number of internal elements affects the round-trip delay to the target node and it was hoped that information about the internal layer 2 structure of the network would be revealed by comparing the measurement results from various viewpoints.

### 6.2.2 Measurement Setup

The experimental setup as used during the internal network element identification, as discussed in Chapter 5, was employed for the multiple viewpoint experiments. A short overview, with the differences between the new and old experiments, is presented below<sup>1</sup>:

ARP packets were again used for probing target nodes, but only a pair of 960-byte packets was used during the new experiments. A PC (with an RTL-8139 network card) running RTLinux was again used as the probing host,

---

<sup>1</sup>For a detailed discussion please refer to Section 5.2

## 6.2. Experiment Planning and Execution

but a laptop was used for the new experiments in order to increase mobility around the live network. The drivers and kernel module developed earlier was used without modification, but the userspace application was adapted to read a target list from a file.

### Target List

During the internal network element identification experiments, a single target node with known MAC and IP addresses was used for all the experiments. In order to probe target nodes on a live network it is necessary to create a list of target node IP addresses to be used in the ARP request packets.

The procedure developed for passive node discovery as discussed in Chapter 4 came in handy and could be used for creating the target node list.

### 6.2.3 Execution

The probing PC was connected to a live corporate Ethernet LAN and as a first step, passive node discovery was performed for around 2 hours. A database table consisting of 155 unique MAC addresses was created as described in Section 4.4. The database table was then exported to a text file that served as a target node list for the new measurement application.

Packets were sent from the probing PC to all targets in the list from thirteen physically distinct Ethernet endpoints in the network. The viewpoints were manually chosen by simply moving around the building and plugging the PC into unused Ethernet sockets. A total of 100 pairs of packets were sent to each of the target nodes from each network viewpoint.

The probing experiment was conducted in as short a time as possible (around 3 hours) to minimise the possibility of changes occurring to the internal network or target nodes. The round-trip times for each of the target nodes and each of the viewpoints were again saved into text files for further processing.



## 6.3 Measurement Observations

The experiments discussed in Chapter 5 were conducted in a very controlled network environment where neither the network nor target nodes were loaded in any way. A difference between the round-trip times for the controlled versus the live network was expected, but the extent of the variation was unknown.

Round-trip time variations were also expected in measurements conducted from different viewpoints, but it was unclear whether the variations would be significant enough to be detected. The following sections address these uncertainties by looking at the low-level measurement data.

### 6.3.1 Round-Trip Time Distribution

Figure 6.1 shows round-trip time data measured for two configurations of the controlled network environment used in earlier experiments (refer to Section 5.3.1). Figure 6.2 shows data for a single target node measured from two different network viewpoints on the live network. In each figure the round-trip results from 100 probe packets are shown. Even though disturbances in the live network data can be observed (probably due to either network or target node activity), it can be seen that the data from the distinct viewpoints can be clearly separated.

### 6.3. Measurement Observations

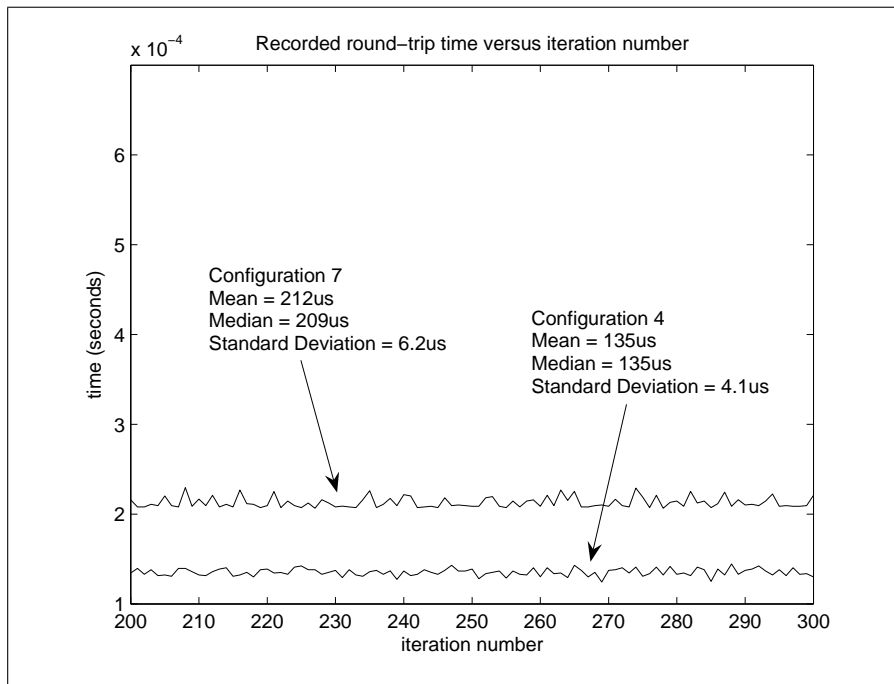


Figure 6.1: Round-Trip Time Distribution in a Controlled Network.

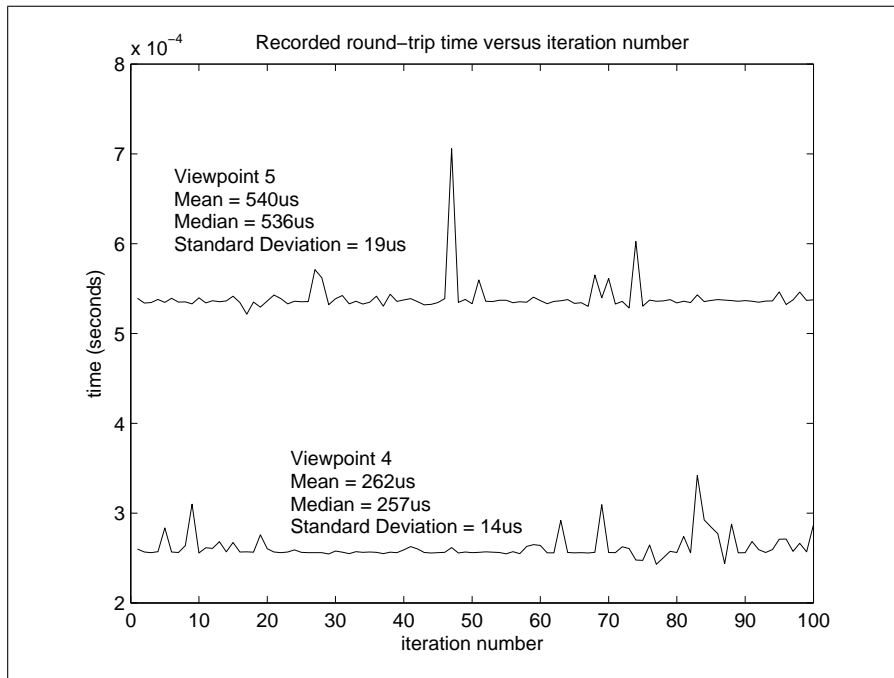


Figure 6.2: Round-Trip Time Distribution in a Live Network.

### 6.3.2 Viewpoint Dependent Variations

The mean of the round-trip times were calculated for each node, packet sequence number (first or second packet) and network viewpoint. Plots of the calculated values were initially used for visually comparing the round-trip time variations between viewpoints.

Figure 6.3 and Figure 6.4 show plots for two target nodes where data from seven viewpoints were processed per node and packet sequence number. It can be seen how the round-trip time varies for each of the nodes depending on the network viewpoint that was used during the measurement.

From the two figures it is also evident that the order in which the viewpoints appear in the graphs (in terms of round-trip time) are different. It was decided to use this order of viewpoint occurrence as a **signature** for the specific node. If it is assumed that the internal node response time does not change during the measurements; the signature depends only on the store-and-forward network elements present between the measurement node and the target node. The round-trip time decreases as the number of store-and-forward elements between the node and network viewpoint decreases.

### 6.3. Measurement Observations

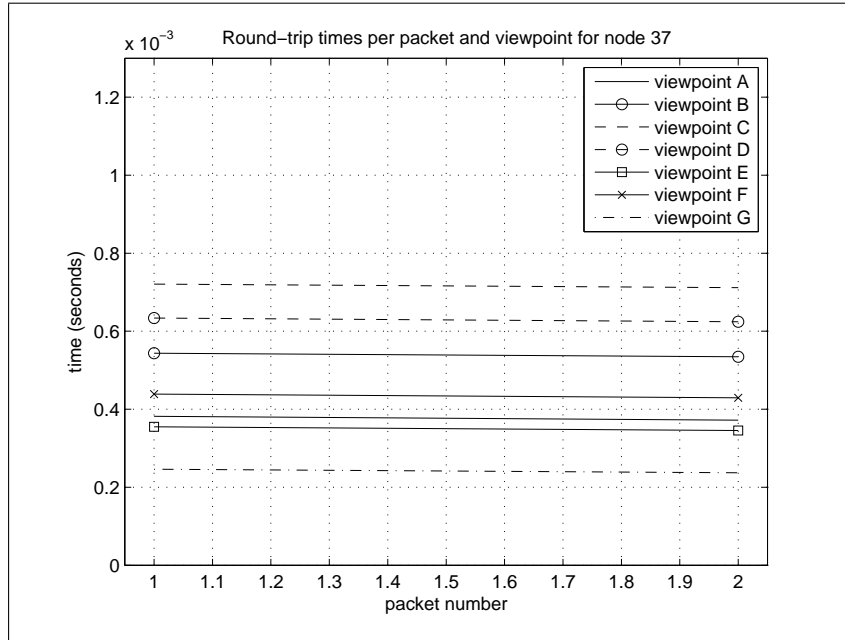


Figure 6.3: Viewpoint Round-Trip Times to Node 37.

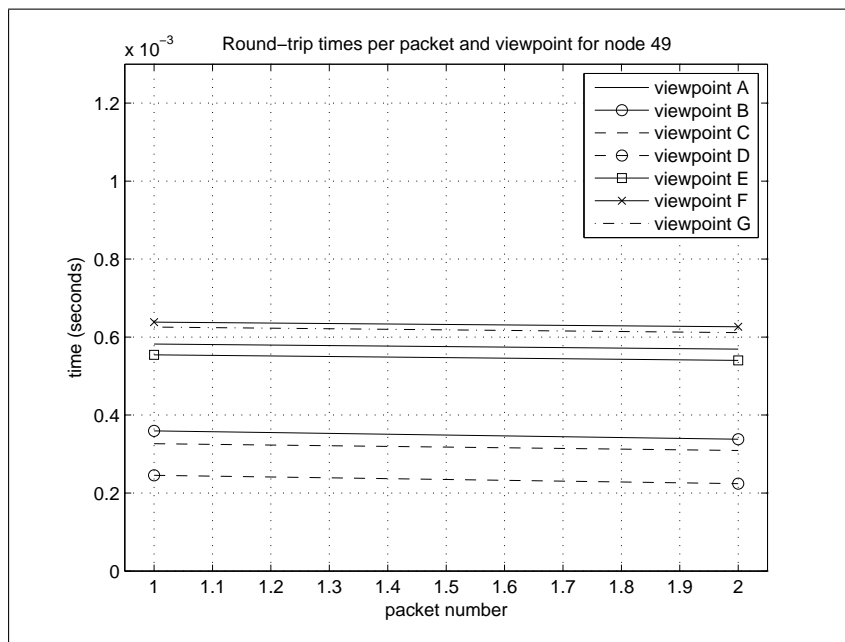


Figure 6.4: Viewpoint Round-Trip Times to Node 49.

## 6.4 Clustering Nodes using Signatures

In the previous section it was shown how the temporal order of round-trip data from various viewpoints can be used as a signature for the specific target node. If we calculate the signature for every viewpoint to all the target nodes, we can then compare the signatures and group nodes with similar signatures. The nodes with similar signatures should be close to each other in terms of the network's layer 2 topology.

A tool was created to automate the data processing task in order to experiment with different numbers of viewpoints and viewpoint combinations. The following sections explain the procedure used for processing and visualising the collected round-trip time data.

### 6.4.1 Data Processing

A program was written to automatically calculate node signatures based on data files from a number of viewpoints. The pseudo-code for the program is presented in Figure 6.5. In essence, the program sorts the viewpoint results per round-trip time for each node, forming a signature for the node, and then creates clusters of nodes by comparing and matching the signatures.

The result of the program is written to a text file that indicates to which group each node belongs.

### 6.4.2 Visualisation

The output data of the program presented in Section 6.4.1 was fed directly to the `neato` [112] application of the GraphViz package [113]. Nodes with similar signatures were connected using graph edges to a common group node. Figure 6.6 shows a graph generated by using data from four distinct viewpoints.

The group nodes created by the clustering application are numbered from 500 upwards. The other numbers in the figure identifies the target node used

## 6.4. Clustering Nodes using Signatures

```

for each viewpoint data file:
  calculate the mean of round-trip times per node
for each node:
  if the node is visible from all viewpoints:
    create a node signature by sorting viewpoints per round-trip time
    if the signature is unique:
      add to unique signature list
for each unique signature:
  create a group node
  find nodes with matching signatures and attach them to the group node
write out the groups
  
```

Figure 6.5: Pseudo-Code for Calculating and Matching Node Signatures.

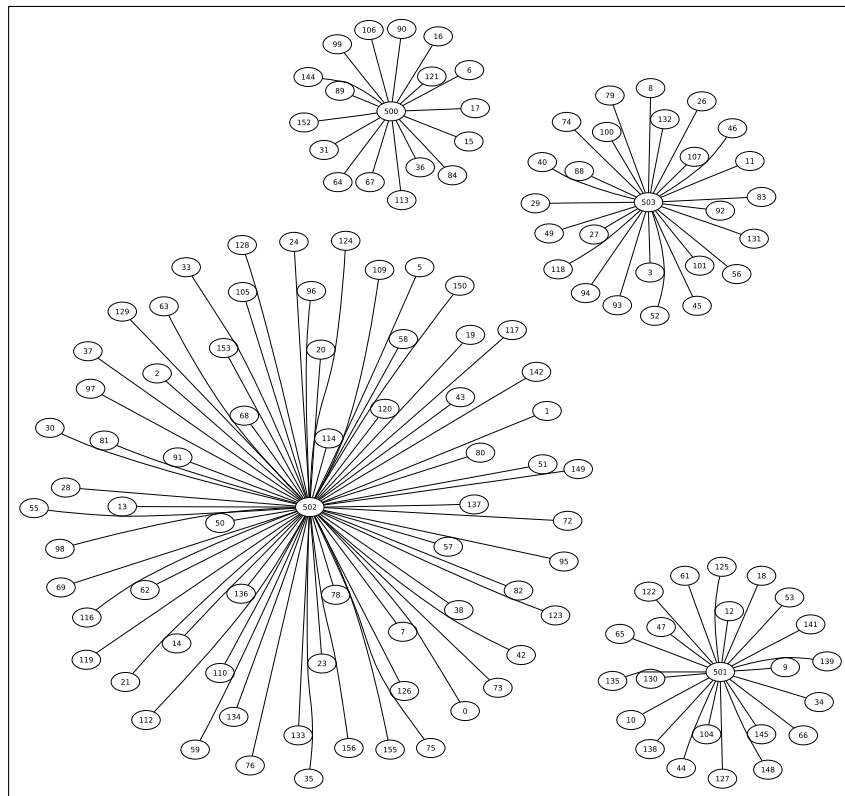


Figure 6.6: Nodes Clustered using Signatures from Four Viewpoints.

## 6.5. Quantifying Signature Similarity

76

for probing. The visualisation technique immediately shows the different signature groups and also provides a spatial dimension to the raw data.

It should be noted that the number of clusters formed does not necessarily match the number of viewpoint data sets considered during the processing. Physically distinct Ethernet endpoints might be connected to the same switch and would not result in a greater number of unique signatures. Adding more viewpoints that are not unique would only result in longer signatures that contain redundant data.

## 6.5 Quantifying Signature Similarity

Nodes clustered together as per Section 6.4 have a very good chance of being in close proximity at the data link network layer. Proximity in this case means that they are either connected to the same network switch or that they share a common switch close to both of them. In order to determine the proximity of one cluster to another a method had to be found to quantify the signature similarity between clusters of nodes.

A metric similar to the RSIM metric presented by Hu and Steenkiste [114] was constructed. RSIM delivers a route similarity value (between 0 and 1) for two nodes based on the Internet paths between a set of network endpoints and the two nodes. The RSIM similarity metric increases if the number of shared links on a path between the two nodes and a specific endpoint increases. In our case, because there is initially no clear indication of shared links, viewpoint pairs in the node signatures were used as a substitute for the shared link concept.

Let  $sig(c)$  be an ordered set of viewpoint names sorted by round-trip time (from smallest to largest in time), where  $c$  represents a cluster of nodes. As an example, let  $c_B$  represent the cluster of nodes connected to switch B in Figure 6.7. If we assume that only switches influence packet round-trip times, that the switches delay packet transmission by the hypothetical duration indicated in Figure 6.7 and that packets to and from the target node

## 6.5. Quantifying Signature Similarity

77

experience the same delay, the round-trip times from the set of viewpoints to any node in  $c_B$  would be: viewpoint (VP) A =  $(1 \times 2 + 1 \times 2) = 4$  units, B =  $(1 \times 2) = 2$  units, C =  $(1.1 \times 2 + 1 \times 2) = 4.2$  units, D =  $(1.2 \times 2 + 1.1 \times 2 + 1 \times 2) = 6.6$  units and E =  $(1.3 \times 2 + 1.1 \times 2 + 1 \times 2) = 6.8$  units. Therefore, the signature would be defined as  $sig(c_B) = \langle B, A, C, D, E \rangle$ .

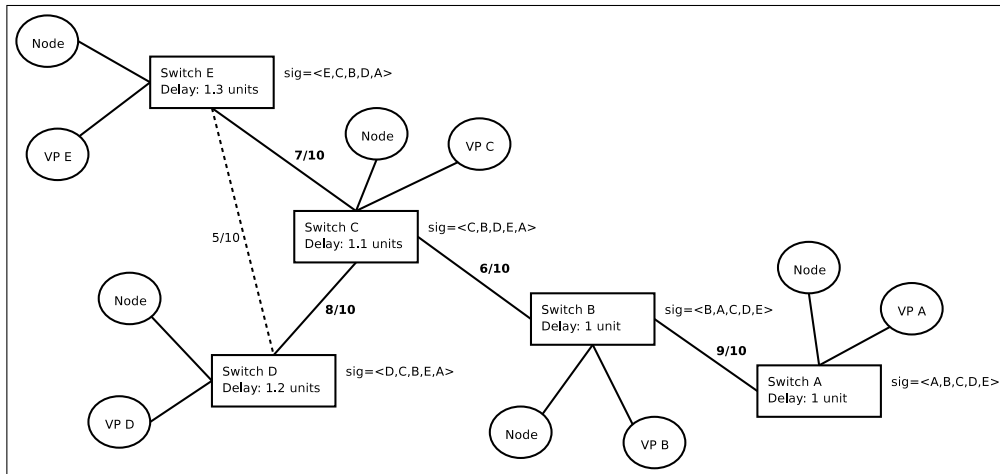


Figure 6.7: Example Network, Signatures and Similarity Values.

Further, let  $shared(c_1, c_2)$  be the number of unique viewpoint pairs that occur in the same order in both  $sig(c_1)$  and  $sig(c_2)$ . Let  $pos(x, c)$  be the position (counting from 1) of  $x$  in  $sig(c)$ , for example  $pos(C, c_B) = 3$ , then  $shared(c_1, c_2)$  can be represented as:

$$shared(c_1, c_2) = \sum_{x,y \in sig(c_1)} pairshared(c_1, c_2, x, y) \quad (6.1)$$

$$pairshared(c_1, c_2, x, y) = \begin{cases} 1 & \text{if } pos(x, c_1) < pos(y, c_1) \text{ and} \\ & pos(x, c_2) < pos(y, c_2) \\ 0 & \text{otherwise} \end{cases} \quad (6.2)$$

As an example, let  $c_B$  and  $c_C$  represent the clusters of nodes attached to switches B and C in Figure 6.7 respectively. The function  $shared(c_B, c_C)$  would increment by one for the signature pairs  $\{B, A\}$ ,  $\{B, D\}$ ,  $\{B, E\}$ ,



## 6.6. Topology Inference Using Signature Similarity

78

$\{C, D\}$ ,  $\{C, E\}$  and  $\{D, E\}$ . Therefore  $shared(c_B, c_C) = 6$ .

The maximum value of the *shared* function is attained when it is used to compare identical signatures. In this case the sum takes on the form:

$$1 + 2 + 3 + \dots + n$$

where  $n$  is one less than the number of viewpoints present in the signature. We define this sum as  $total(c_1, c_2)$  and it can be rewritten as:

$$total(c_1, c_2) = \frac{n(n+1)}{2} \quad (6.3)$$

The *total* function value for the example network in Figure 6.7 would be  $\frac{4(4+1)}{2} = 10$ .

Finally, we can define our similarity metric as:

$$similarity(c_1, c_2) = \frac{shared(c_1, c_2)}{total(c_1, c_2)} \quad (6.4)$$

The similarity values for some cluster pairs in the example network are indicated next to graph edges in Figure 6.7. The higher the similarity value between clusters, the better the chances are that the two clusters are directly connected.

## 6.6 Topology Inference Using Signature Similarity

The metric presented in Section 6.5 can be calculated for all possible pairs of node clusters (as identified in Section 6.4). The similarity value between pairs of clusters can then act as a guideline to indicate if two clusters should be connected to recreate the network topology. An application was written to programmatically recreate and visualise a network topology from signature similarity values.

## 6.6. Topology Inference Using Signature Similarity

79

The following sections describe the thought process followed in the attempt to firstly recreate the topology of the test network shown in Figure 6.7 and to secondly infer the topology of the measured live network.

### 6.6.1 Example Network

If one blindly connects all node clusters of the test network shown in Figure 6.7 the result is as shown in the left graph in Figure 6.8. Since an Ethernet LAN is not allowed to contain loops [32], a method had to be found to eliminate some of the edges in the graph. A threshold value was introduced to ignore edges (representing connections between clusters) with a similarity value less than the threshold.

The graph to the right of Figure 6.8 shows the recreated topology of the test network with a similarity threshold value of 0.5. It can be seen that the graph matches the topology of the example network in Figure 6.7.

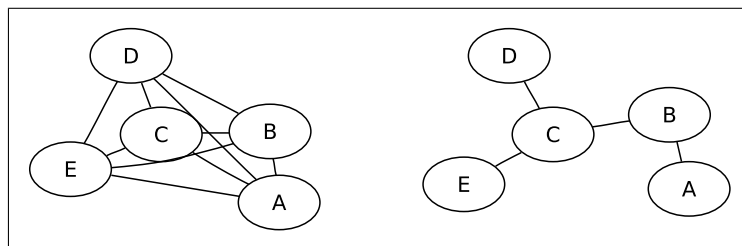


Figure 6.8: Example Recreated Network Graphs.

At this stage it was hoped that a single threshold value would also allow the reconstruction of the live network topology.

### 6.6.2 Live Network

The similarity metric was also calculated for measurement data obtained from the live network. Figure 6.9 shows an example network graph created by using the data from five different network viewpoints and a threshold value of 0.55. The five viewpoints were selected based on the fact that they generated

## 6.6. Topology Inference Using Signature Similarity

clusters of nodes with five unique signatures. It can be seen that the graph illegally contains a loop [32] and in order to eliminate it the threshold value was increased.

Figure 6.10 shows the same network graph as in Figure 6.9, but with a threshold value of 0.8. The larger threshold value allowed the removal of more edges, but caused some node clusters to become disconnected from the rest of the graph. The disconnectedness is unacceptable since we were able to receive responses from these nodes by probing from all the viewpoints.

Other threshold values were tried, but no single threshold value was found where the generated graph for the live network was without loops and where all the node clusters were connected. A simple heuristic was developed in an attempt to remedy the situation:

Start from a graph without any loops and increase the threshold value steadily while noting where edges between node clusters appear. If an edge appears that causes a loop in the network, remove the edge from the graph; otherwise, leave the edge in the graph.

It was found that the heuristic worked satisfactorily where data from certain numbers and combinations of viewpoints were considered, but that it also failed for some scenarios. In the case of the five viewpoints used to generate Figure 6.9 and Figure 6.10 the heuristic failed in that two edges that create a loop in the network appear at the exact same threshold value. Figure 6.11 shows the two edges (using bold dashed lines) that appear simultaneously at a threshold value of 0.6.

The only way that was found to overcome this limitation, was to use data from more viewpoints to try and eliminate the ambiguity. Figure 6.12 shows the network graph created by employing the heuristic developed earlier and the data from six distinct network viewpoints. All the node clusters were connected without loops. Note how the additional viewpoint data caused a split of the leftmost cluster in Figure 6.11.

The data obtained during the passive node identification phase of our experiments allowed us to convert the node numbers in Figure 6.12 into computer

6.6. Topology Inference Using Signature Similarity

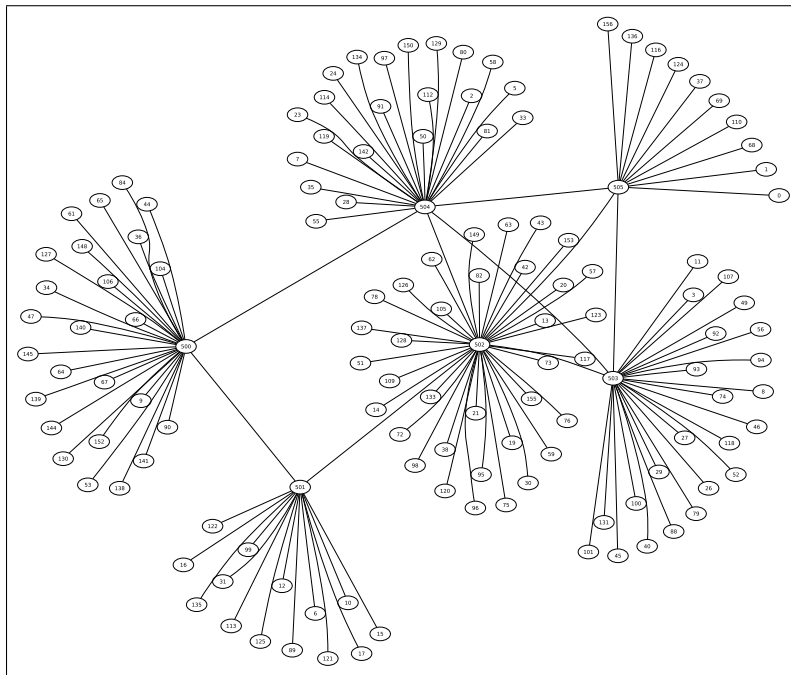


Figure 6.9: Reconstructed Network Graph using Data from Five Viewpoints with Some Edges Removed Based on a Similarity Threshold of 0.55.

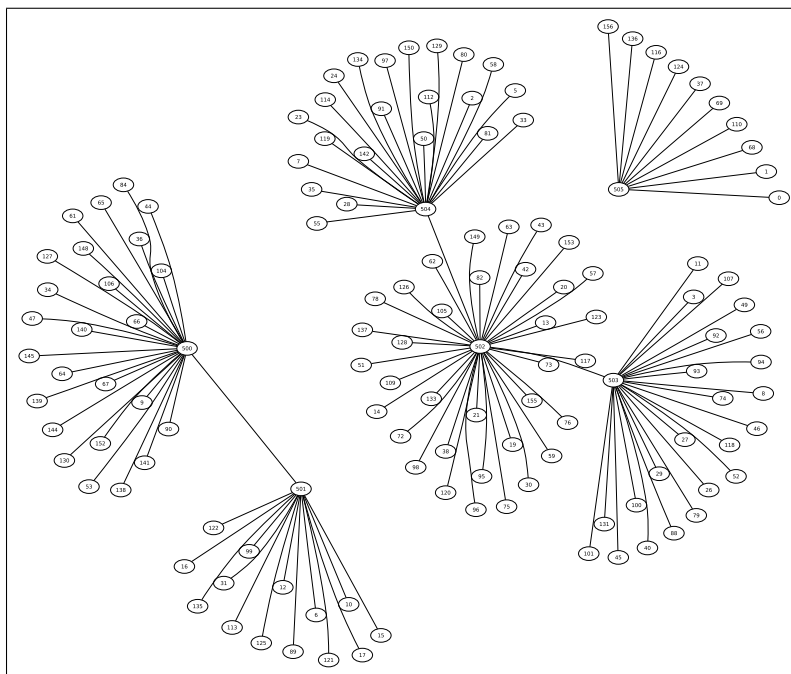


Figure 6.10: Reconstructed Network Graph using Data from Five Viewpoints with Loops Removed Based on a Similarity Threshold of 0.8.

6.6. Topology Inference Using Signature Similarity

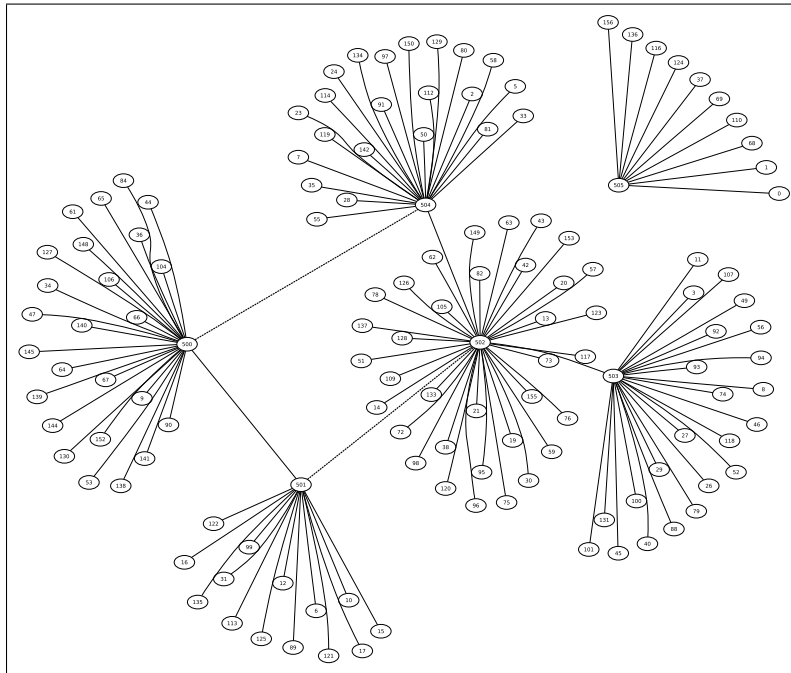


Figure 6.11: Reconstructed Network Graph using Data from Five Viewpoints Showing Edge Appearing Simultaneously at a Similarity Threshold of 0.6.

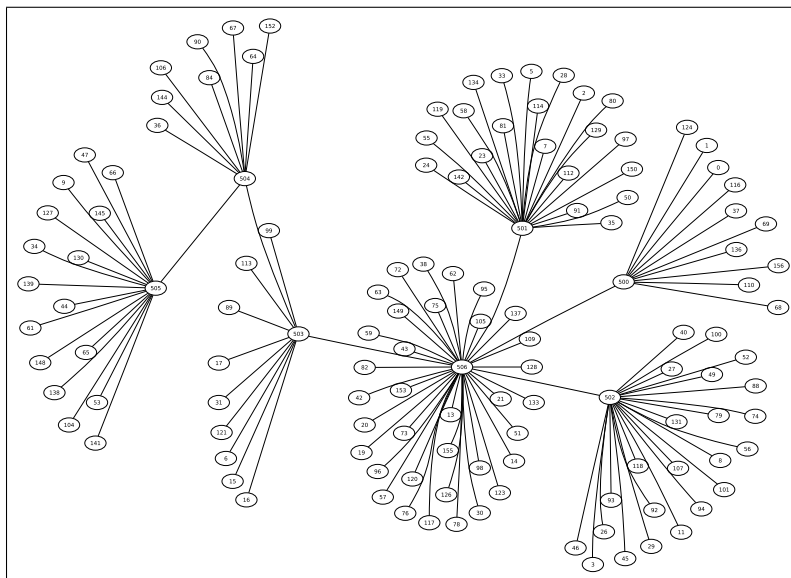


Figure 6.12: Reconstructed Network Graph using Data from Six Viewpoints with All Clusters Connected.

names. The names confirmed that node clusters in close proximity in the graph were also in close physical proximity in the building where the experiments were performed. In other words, the topology of the generated graph conformed to the physical layout of the building. Unfortunately, no link layer map of the network was available for comparison and we can therefore only speculate about the accuracy of the recreated topology.

## **6.7 Conclusion**

The measurement and scanning techniques developed in earlier chapters allowed us to design and execute new experiments in an attempt to infer the layer 2 topology of a live network. It was shown that processing of the data gathered by the live network experiments could identify groups of nodes using the concept of node signatures.

The signature concept of the identified node clusters was further expanded to develop a similarity metric between clusters. The similarity metric combined with a heuristic was then used to infer the connectivity between the clusters.

Even though a topology could be recreated, there exists uncertainty as to its accuracy. The accuracy is definitely influenced by the specific locations and the number of network viewpoints used. It could therefore be useful to in future determine the optimal location of viewpoints or to ascertain whether more viewpoints need to be considered during measurement or processing.

The accuracy of the topology reconstruction technique could not be confirmed due to the unavailability of a network map and this deficiency is the topic of the next chapter, Chapter 7.

# Chapter 7

## Reconstruction Simulations

### 7.1 Introduction and Overview

The previous chapter showed that it is possible to infer the layer 2 topology of a live network, but that there was no measure of how accurate this inferred topology was. Lack of accurate internal network maps makes verification of the reconstruction technique in a live network environment almost impossible. One way to test the algorithm and heuristic developed earlier is to create a known network topology in software, perform simulated measurements on this network, process the results of these virtual measurements and compare the inferred topology with the known starting conditions. These reconstruction simulations are the focus of this chapter.

Three main questions were investigated using the simulations:

1. Are the reconstruction algorithm and heuristic accurate when we consider all possible viewpoints and assume perfect measurements?
2. How does the accuracy degrade if measurement errors are introduced into the round-trip time of packets?
3. How does the number of viewpoints influence the accuracy of the inferred network topology?

As a first step, we needed to create experimental network topologies and this problem is addressed in Section 7.2. The simulation of measurement experiments is discussed in Section 7.3 and Section 7.4 deals with the problem of topology reconstruction and comparison. Once the simulation techniques were refined they were used in Monte Carlo method simulations to determine the accuracy of our reconstruction algorithm and heuristic. The results of the simulations are presented in Section 7.5. Conclusions are presented in Section 7.6.

## 7.2 Topology Generation

Literature concerning topology generation for simulation mainly focus on creating Internet layer 3 topologies [10, 60]. These generated topologies typically contain hundreds to thousands of nodes and need to match real-world networks in terms of statistical node distribution and connectivity. In our case a simpler approach could be followed, as the primary objective was the testing of our topology reconstruction algorithm at layer 2 of the network. The constructed networks in our case contained less than a hundred nodes, but we generated a multitude of different network topologies to test the reconstruction algorithm against.

The following sections describe the procedure that was followed to generate the test network topologies for the simulation.

### 7.2.1 Software Representation and Manipulation

In order to perform simulated measurement of delays from a specific location (or viewpoint) in the network towards a target location, it was required to generate a known and valid abstract network representation in software. “Known” means that we can store the topology of the network in a format that is unambiguous and that the format can be used for easy comparison to a reconstructed network topology stored in the same format. “Valid” means



## 7.2. Topology Generation

that the generated network represents an Ethernet network containing no loops [32]<sup>1</sup>.

It was decided to construct a graph with its vertices representing computer nodes (be it measurement or target nodes), edges representing network links between nodes and edge weights<sup>2</sup> representing packet delivery delays when sending a message from one node to another. By using the graph in this manner, the sum of the edge weight values along a path from one vertex to another represents the total one way delay experienced by a packet travelling between the selected vertices or nodes.

The Boost Graph Library [115] provides a set of C++ classes and algorithms for easy construction and manipulation of graph objects. This library proved very useful in the generation of test network topologies as well as for reconstruction of networks.

### 7.2.2 Creation of a Fully Connected Graph

As a first step in creating an experimental network topology, we created a fully connected (also called complete [116, p. 29]) graph. In this graph every pair of vertices are connected by a single edge. It was decided to randomise the edge weights in the graph in order to create variations in the topology that would emerge during further processing (see Section 7.2.3).

The variables that could be chosen at the time of graph creation are the number of vertices, a random number generator seed value and a maximum random number range. The seed allows for the creation (and possible later recreation) of a specific pseudo-random number sequence, while the range value places a limit on the integer value created by the random number generator.

Programmatically the graph is constructed using the procedure shown in the pseudo-code in Figure 7.1.

---

<sup>1</sup>The fact that a valid Ethernet topology does not contain loops also simplifies the network generation when compared to generation of Internet topologies.

<sup>2</sup>A value that can be associated with every edge.

## 7.2. Topology Generation

87

```
initialise the random number generator using seed
create a single graph vertex
until the requested number of vertices have been added:
  add a vertex (A) to the graph
  for all the previously existing vertices (B):
    generate a random number (R) in the allowed range
    add an edge from (A) to (B) using (R) as the weight
```

Figure 7.1: Pseudo-Code for Creating a Fully Connected Graph.

Figure 7.2 shows a fully connected graph with 7 vertices on the left side of the figure. The edge weights were calculated as pseudo-random integer values between 0 and 100.

### 7.2.3 Pruning of Graph Edges

As the final step in the creation of our test network topology we needed to eliminate all the unnecessary edges from the complete graph as generated in Section 7.2.2. A valid Ethernet network does not contain any loops, but we need a path to exist between any chosen pair of nodes in the network. The problem of finding a subset of graph edges that still connects all the vertices in the graph, while also minimising the total weight (sum of the edge weights) of the graph, has been investigated as early as the 1950's [117]. The problem is commonly referred to as the minimum spanning tree problem [115, p. 89].

The Boost Graph Library contains an implementation of Kruskal's algorithm [117] for finding the minimum spanning tree of an undirected graph. This algorithm was used to reduce the complete graph that was produced earlier to a graph representing a valid Ethernet network. The topology of the resulting graph depended on the random edge weights assigned during the creation of the complete graph and in this way a multitude of topologies could be generated by varying the random number seed used initially.

Figure 7.2 shows the minimum spanning tree (right side of the figure) created

### 7.3. Measurement Simulation

88

from the original complete graph (left side of the figure).

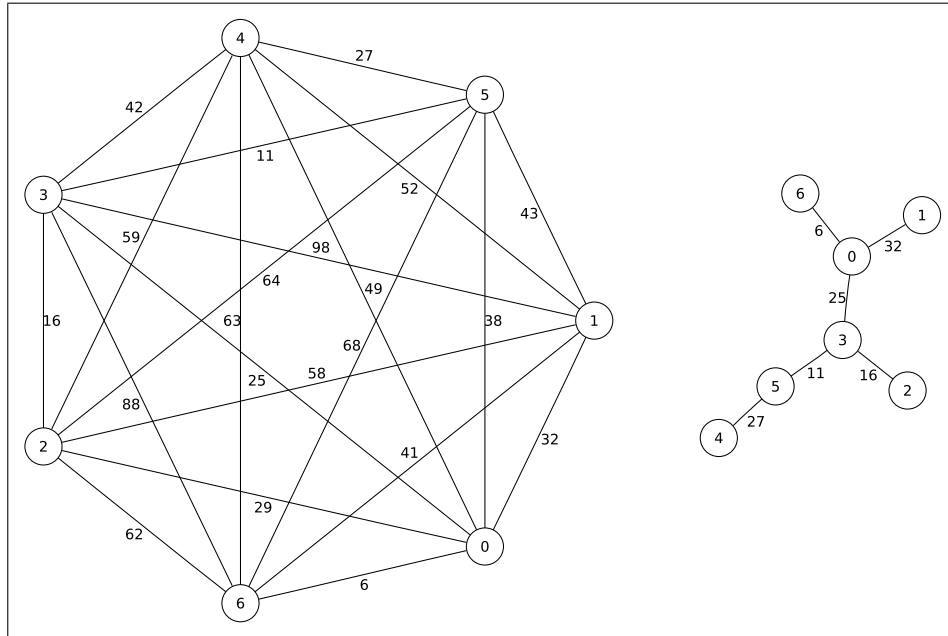


Figure 7.2: Example Generated Test Network.

### 7.3 Measurement Simulation

Once an experimental network topology was created, we needed to simulate network delay measurements similar to the physical measurements described in Chapters 5 and 6. The experimental topology was purposefully set up so that graph edge weights represented packet delivery delays. The problem of finding the total delay experienced by a packet travelling between two nodes therefore reduced to summing the edge weights along the path between the vertices representing the nodes.

The Boost Graph Library was again employed for the task of summing edge weights along paths between vertices. The library contains an implementation of Dijkstra's shortest path algorithm [118]. The algorithm finds the shortest paths and calculates the path lengths between a single source vertex and all other graph vertices. Because our test networks do not contain loops,

the shortest path corresponds to the only path between two nodes.

Depending on our specific experimental need we could now simulate the measurement of delay between a specific network viewpoint and any target nodes by using Dijkstra's algorithm on our experimental network. The path lengths calculated by the algorithm were stored in a text file that was used during further processing and reconstruction.

## **7.4 Reconstruction and Comparison**

This section deals with the problem of programmatically reconstructing a network topology from virtual measurement data as generated by simulation. The focus is now placed on testing the accuracy and robustness of our reconstruction algorithm and heuristic as presented in Section 6.6.

The process consists of three steps, namely:

1. Calculate viewpoint similarity based on the measured delay data.
2. Rebuild the network topology using the similarity data.
3. Compare the resulting topology to the initial test network topology.

In order to ease the explanation of the three steps, we assume for now that perfect network measurements could be performed. Variations on this special case are presented and discussed in the results section (Section 7.5).

### **7.4.1 Viewpoint Similarity**

The simulated measurement results (as described in Section 7.3) were used to calculate a similarity value between network viewpoints. The similarity metric as discussed in Section 6.5 was again used, but with a slight modification: Instead of purely comparing the relative position of two viewpoints in a signature, we compared the differences between the path lengths to the respective viewpoints.

## 7.4. Reconstruction and Comparison

The calculation is best described by a piece of pseudo-code as shown in Figure 7.3. Note that the code is specific to the assumed case of perfect measurement results.

```
load path lengths for all viewpoints
for all pairs of viewpoints ( $v_1, v_2$ ):
  set similarity between  $v_1$  and  $v_2$  to 0
  for all pairs of target nodes ( $t_1, t_2$ ):
    calculate delay differences ( $d_1, d_2$ ):
       $d_1 = \text{path length}(v_1 \text{ to } t_1) - \text{path length}(v_1 \text{ to } t_2)$ 
       $d_2 = \text{path length}(v_2 \text{ to } t_1) - \text{path length}(v_2 \text{ to } t_2)$ 
    if  $d_1$  is equal to  $d_2$ :
      add 1 to the similarity between  $v_1$  and  $v_2$ 
write out the similarity between  $v_1$  and  $v_2$ 
```

Figure 7.3: Calculating Viewpoint Similarity – Perfect Measurements.

The similarity idea can also be visualised in the following way. Imagine that a network (for example, the one on the right of Figure 7.2) consists of marbles connected by pieces of string where the marbles represent vertices, the strings represent edges and the string lengths represent edge weights. To find out how similar two viewpoints (or vertices) are, pick up the *network* in turn by holding each of the two marbles representing the viewpoints and let the other marbles hang down freely. The degree to which the patterns<sup>3</sup> formed by the hanging marbles match between the two views gives an indication of the viewpoint similarity.

The similarity values were written into a text file with each line containing the similarity value (as the first element) and the two node or viewpoint numbers (as the second and third elements). The text file was then sorted so that the highest similarity values were at the top of the file. The sorted file eased the execution of our heuristic for topology reconstruction. As an example, the sorted similarity values for the test network shown in Figure 7.2 are given in Table 7.1.

---

<sup>3</sup>Especially the distance between pairs of marbles.

**7.4. Reconstruction and Comparison**

Table 7.1: Calculated Similarity Values when using Perfect Measurements.

Similarity	Node 1	Node 2
15	4	5
15	2	3
15	0	6
15	0	1
11	3	5
10	3	4
10	1	6
9	0	3
7	3	6
7	2	5
7	1	3
6	2	4
6	0	2
5	0	5
4	2	6
4	1	2
4	0	4
3	5	6
3	1	5
2	4	6
2	1	4

## 7.4. Reconstruction and Comparison

92

### 7.4.2 Reconstruction Heuristic

Once the similarity values between viewpoints were calculated and sorted, we employed our heuristic developed in Section 6.6 to reconstruct the network's topology.

The heuristic proved easy to code once a function was developed for detecting loops in the network topology. The Boost Graph Library was again employed as it contained a depth-first search algorithm [115, p. 67] ready for use. The depth-first search algorithm calls a class method once it detects a *back edge*<sup>4</sup> and this immediately indicates that a loop exists in the current network topology being searched.

The procedure is shown in the pseudo-code in Figure 7.4. Since the file containing the similarity values was sorted previously, we could easily process one line at a time and be sure that the highest similarity values were processed first.

```
create an empty graph with no edges
while lines exist in the similarity file:
  read the two viewpoint ( $v_1$  and  $v_2$ ) numbers
  add an edge between  $v_1$  and  $v_2$ 
  if a loop exists in the network:
    remove the edge between  $v_1$  and  $v_2$ 
write out all existing edges
```

Figure 7.4: Pseudo-Code for Reconstructing a Network Topology.

### 7.4.3 Comparison

The final processing step during both the initial test network construction phase (see Section 7.2) and the network reconstruction phase writes out graph

---

<sup>4</sup>A back edge represents a connection from a vertex to one of its ancestors in a search tree [115, p. 67].

## 7.5. Monte Carlo Method Results

93

edges in a textual format. Every edge is written out onto a single text line as two numbers representing the edge endpoints. The smaller of the two numbers is written first.

The initial test network and the reconstructed network could now be compared by first sorting the lines of their two textual representations<sup>5</sup> and then textually comparing<sup>6</sup> the two files.

Vector graphics<sup>7</sup> representations of the test and reconstructed networks were also saved into files that could be used for visual inspection. Visual inspection of the topologies helped to identify differences when the textual comparison failed.

## 7.5 Monte Carlo Method Results

The whole process, as described in the previous sections, from test network construction to reconstruction and comparison was automated so that a multitude of simulations could be performed. Simulation parameters were varied prior to individual simulations in order to create a variety of scenarios.

Three groups of simulations were considered in order to test the accuracy and robustness of our reconstruction algorithm and heuristic:

1. Assume perfect measurements from all network viewpoints.
2. Introduce measurement errors when calculating path lengths (network delays).
3. Use only measurement data from a subset of the network viewpoints.

The execution and results of these simulations are presented in the following sections.

---

<sup>5</sup>The Unix `sort --general-numeric-sort` command was used to sort the files using general numeric value.

<sup>6</sup>The Unix `diff` command was used.

<sup>7</sup>The Scalable Vector Graphics (SVG) file format was used. Refer to <http://www.w3.org/Graphics/SVG/>.



## 7.5. Monte Carlo Method Results

94

### 7.5.1 Perfect Measurements

We started out by assuming that perfect network measurements can be performed. A failure of the algorithm to generate an exact replica of the test network topology under these conditions would have indicated a serious flaw and would have forced revision of the algorithm.

#### Network Generation and Measurement

The procedure for creating an experimental network as described in Section 7.2 was used with an additional step added: After the network without loops was constructed, the edge weights of this network was again randomised. The extra randomisation step ensured that we did not only test the algorithm against the minimal spanning tree constructed network.

The parameter ranges used during the network generation phase were:

- Number of nodes: 5, 10, 20 and 50
- Random number generator seed: 0 to 499
- Random number (used as edge weight) range: 0 to 1000

Simulated measurements were now performed from every node in the network to every other node (refer to Section 7.3). The path lengths from a specific viewpoint to all nodes in the network were stored in a text file indexed by the viewpoint number.

#### Reconstruction and Comparison

The reconstruction of network topologies and comparison with the generated test networks proceeded exactly as described in Section 7.4.

## 7.5. Monte Carlo Method Results

95

### Results and Observations

A total of 4000 simulation runs were performed (500 for every network size) and the reconstructed network topology matched that of the generated topology in every single case.

Figure 7.5 shows example topologies for a generated test network (on the left) and the reconstructed version of the network (on the right). The network has 20 nodes and a seed value of 53 was used during generation.

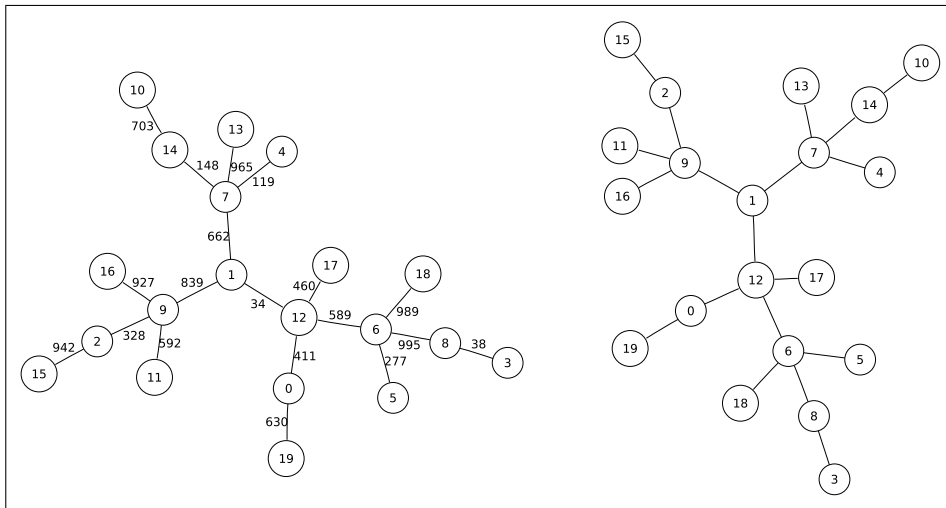


Figure 7.5: Generated (left) and Reconstructed (right) Topology.

The simulation results showed that our similarity metric and reconstruction heuristic performed without error when perfect measurements are used. It provided a good starting point for further simulations and questions.

### 7.5.2 Measurement Errors

The positive results obtained during reconstruction simulations using perfect measurements (see Section 7.5.1) created confidence in our reconstruction algorithm and heuristic; however, in a real network it is impossible to generate these perfect measurement results. The algorithm was now tested in a simulation environment where measurement errors were introduced.

### Network Generation and Measurement

Experimental network topologies were again constructed as described in Section 7.2. The edge weights were however changed to a fixed value of 100 units after the network was constructed. The fixed value simulated a network where a predetermined and constant packet delivery delay is introduced by every network element. This situation would typically occur in a network where all the elements operate at the same bit rate. The fixed value also allowed for easier evaluation and quantification of the effects produced by measurement errors of various sizes.

The parameter ranges used during the network generation phase were:

- Number of nodes: 5, 10, 20 and 50
- Random number generator seed: 0 to 1999
- Random number (used as edge weight in initial network) range: 0 to 1000

The shortest path algorithm as described in Section 7.3 was again used to calculate the path length from every network viewpoint to all others, but an error was added to the path length before it was written to the output file. The error was calculated as a random value between 0 and upper limits of 50, 55, 60, 65 and 70. The choice of the error range values would become clear in the following sections.

### Reconstruction and Comparison

The viewpoint similarity code as described in Section 7.4.1 could not be used as it assumed perfect measurements and used a direct comparison of path length differences. The similarity metric code was adjusted to not only increase similarity between nodes when path length differences matched exactly, but also when they were within the fixed edge length value chosen in the network creation phase. The adjusted pseudo-code is shown in Figure 7.6.

```
load path lengths between viewpoints
for all pairs of viewpoints  $(v_1, v_2)$ :
  set similarity between  $v_1$  and  $v_2$  to 0
  for all pairs of target nodes  $(t_1, t_2)$ :
    calculate delay differences  $(d_1, d_2)$ :
       $d_1 = \text{path length}(v_1 \text{ to } t_1) - \text{path length}(v_1 \text{ to } t_2)$ 
       $d_2 = \text{path length}(v_2 \text{ to } t_1) - \text{path length}(v_2 \text{ to } t_2)$ 
    if the absolute difference between  $d_1$  and  $d_2$  is less than 100:
      add 1 to the similarity between  $v_1$  and  $v_2$ 
  write out the similarity between  $v_1$  and  $v_2$ 
```

Figure 7.6: Calculating Viewpoint Similarity – Measurement Errors.

It is tempting to calculate  $d_1$  and  $d_2$  by using the absolute value of the difference between the path lengths, but taking the absolute value discards information about the temporal order of the viewpoints in the node signatures and results in erroneous similarity values.

Application of the reconstruction heuristic and comparison of topologies were performed as described in Section 7.4.

## Results

A total of 40000 simulation runs were completed covering the range of input parameters. The results of the simulations are shown in Table 7.2.

**Perfect Reconstruction** Perfect topology reconstruction occurred when the error range for the measured path lengths were set to 50 units or below. This specific value where perfect reconstruction occurs can be explained by the setup of our initial test topology and the way we compare path length differences to calculate node similarity. In our similarity metric discussed earlier (see Figure 7.6), we compared path length differences to pairs of target nodes and increased similarity if the absolute difference was less than 100 units.

**7.5. Monte Carlo Method Results**

Table 7.2: Reconstruction Results when Measurement Errors are Introduced.

<b>Nodes</b>	<b>Error Range</b>	<b>Topologies Matched</b>	<b>Topologies Did Not Match</b>
5	50	2000	0
10	50	2000	0
20	50	2000	0
50	50	2000	0
5	55	1999	1
10	55	1995	5
20	55	1936	64
50	55	1237	763
5	60	1989	11
10	60	1934	66
20	60	1497	503
50	60	48	1952
5	65	1956	44
10	65	1786	214
20	65	893	1107
50	65	1	1999
5	70	1924	76
10	70	1568	432
20	70	431	1569
50	70	0	2000

## 7.5. Monte Carlo Method Results

99

When the error range is set to 50, the maximum path length error is 49 units and the minimum error is 0 units. The maximum difference between two paths is therefore  $49 - 0 = 49$  units and the minimum difference is  $0 - 49 = -49$  units. If we now assume the worst case when comparing path length difference between two viewpoints, the maximum difference that can be attained is  $49 - (-49) = 98$  units. The error is therefore always small enough to pass our similarity test and the network topology can be reconstructed perfectly.

**Reconstruction Errors** As soon as the path error range was increased above 50 units, the reconstruction algorithm started to fail in specific cases. It was also clear from the results that the chance of reconstructing an erroneous topology increased as the error range was increased and also as the number of nodes in the network was increased.

An increase in the error range caused our similarity metric to fail more often, as the chance of encountering path differences that did not satisfy our condition for similarity increased. Similarly, an increase in the number of nodes in the network caused a lot more paths to be evaluated and also increased the chance of encountering an error.

It is interesting to note how the algorithm actually fails. If we consider the results of a network with 10 nodes and an error range of 55, we see that only 5 out of the 2000 reconstructions were incorrect. In all 5 cases two nodes at the edge of the reconstructed network were swapped. Figure 7.7 shows an example of one of the failed reconstructions. The generated test network is on the left and the reconstructed network on the right of the figure. It can be seen that nodes 6 and 8 are swapped.

Table 7.3 shows the distribution of errors made during reconstruction for an error range of 55 units. It can be seen that the majority of reconstruction errors are due to the difference in a single network edge between the reconstructed and test networks.

Figure 7.8 shows the worst reconstruction result for a 50 node network and an error range of 55. The generated test network is at the top and the

## 7.5. Monte Carlo Method Results

100

Table 7.3: Error Distribution for an Error Range of 55 Units.

Nodes	Reconstruction Errors				
	1	2	3	4	Total
5	1	0	0	0	1
10	5	0	0	0	5
20	63	1	0	0	64
50	604	136	21	2	763

reconstructed network at the bottom of the figure. It can be seen that 4 errors occurred during reconstruction, but that the errors are still towards the outer edge of the network.

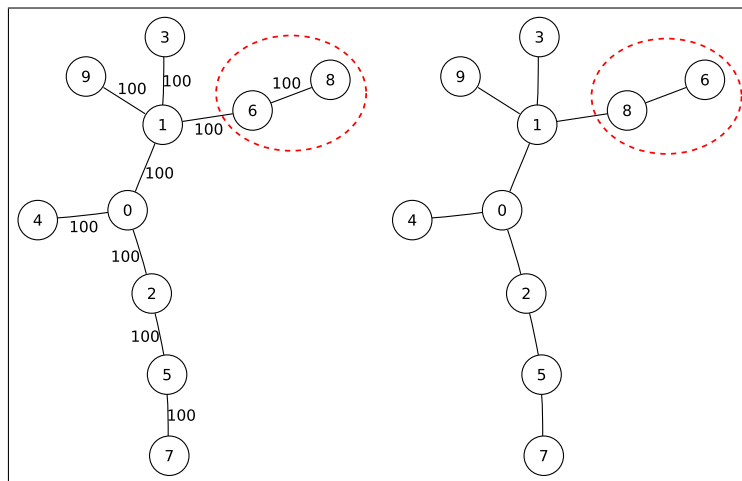


Figure 7.7: Reconstruction Error at the Edge of a 10 Node Network.

For error ranges larger than 55 the number of errors in the reconstructed networks increased at an alarming rate. The errors still occurred mostly toward the outer edges of the networks, but whole branches of the reconstructed networks differed from those of the test networks.

### Conclusive Observations

The reconstruction algorithm delivered accurate results provided that the measurement error on path lengths were less than half of the minimum edge





## 7.5. Monte Carlo Method Results

102

length. In physical terms it means that network delay measurements to any specific target node must be accurate to less than half of the minimum delay experienced by a packet passing through any of the network elements en route.

It was also satisfying that when the reconstruction algorithm failed due to measurement errors, the topology errors were towards the edge of the network and not at the network core.

### 7.5.3 Limited Viewpoints

All the simulations considered thus far assumed that we could obtain measurement data from all the nodes or viewpoints in the network. This section deals with the final set of simulations where the effects of measuring network delays from only a limited number of network viewpoints were investigated.

#### Network Generation and Measurement

Network topologies were constructed using the same procedure as described in Section 7.5.1.

Instead of performing simulated measurements from every node in the network to all the other nodes (as in Section 7.5.1), the simulation allowed for specifying a list of nodes to be excluded from simulated measurements.

#### Reconstruction and Comparison

The reconstruction of network topologies and comparison with the generated test networks proceeded as described in Section 7.4. The only difference was that similarities were only calculated between viewpoints for which simulated measurements were performed earlier.

### Results and Observations

As soon as a viewpoint is excluded from the measurement simulations it could not be evaluated during reconstruction and was therefore missing from the reconstructed network. Comparison between a reconstructed network, with one or more nodes missing, and the test network would therefore always fail. Statistics of comparison results could not produce any insight into the effects of excluding certain viewpoints from the reconstruction process. It was therefore decided to focus on a few topologies and visually examine the effect of increasing the number of excluded nodes.

Figure 7.9 shows how an initial network topology changed as nodes or viewpoints were excluded during the reconstructions process. The initial topology is shown in the top left corner of the figure and the node about to be excluded is drawn using a stippled edge. The next topology in the sequence then shows the reconstructed network with the node excluded.

The figure shows the importance of using data from enough viewpoints to identify the true network topology. Viewpoints close to the core of the network, especially when they have multiple branches, have to be used for measurement to minimise ambiguity in calculating the final topology. The figure also shows how the algorithm tries to preserve the number of *hops* between all nodes in the network. When a viewpoint or node close to the core of the network is ignored during reconstruction, it is replaced by another node.

## 7.6 Conclusion

The main aim of the simulations presented in this chapter was to gauge how accurate the topology inference technique developed in Chapter 6 was. The simulations showed that very accurate topologies can be reconstructed provided two conditions were met:

1. The measurement error on the round-trip delay measured to a target node has to be less than half of the minimum delay experienced by the packet inside a network element en route to the target.



2. Measurements have to be performed from a large set of unique network viewpoints or locations.

The measurement error condition is met by our measurement technique introduced in Chapter 5, but the second condition, of enough unique viewpoints, is not easily met in an ad hoc measurement experiment as the one conducted in Chapter 6. Identifying all unique network viewpoints requires a network topology map and this is what we are trying to create in the first place. The technique for estimating the number of switches between a probe and a target node, as developed in Section 5.4, might be useful to identify viewpoints that were missed during initial measurements.

The simulation results also provided insight into how the topology generated by the reconstruction algorithm degrades as errors, either measurement errors or lack of viewpoint measurements, are introduced.

# Chapter 8

## Conclusion

### 8.1 Completed Work

At the outset of our investigation we set ourselves the ambitious goal of discovering the physical topology of an Ethernet network. The research problem was approached from various angles and the completed work is summarised in the following paragraphs.

The literature study, that was performed as part of our research, was summarised in Chapters 2 and 3. Chapter 3 discussed related research in the area of network topology discovery and also identified a specific subproblem that became the focus of our research, namely the problem of discovering the physical topology of an Ethernet network without cooperation from the network's elements. It was also realised that the research problem required investigation into the workings of Ethernet hardware and protocols. The results of this specific investigation was the focus of Chapter 2 and established the context for the rest of our work and experiments.

Chapter 4 concentrated on the development of techniques in order to identify as many nodes of the Ethernet network in a passive manner as possible. It was shown that a list containing the hardware and IP addresses of the nodes present in the network could be created and that the list could be combined with additional identifying information such as the network device vendor

and the host's browser name to create an inventory of devices on the network.

The discovery of the elements internal to the network was the focus of Chapter 5. It was shown that network elements could be identified by exploiting the effects they have on the delivery time of ARP packets. The half-duplex nature of Ethernet hubs reveals these elements, while the store-and-forward delay that network switches introduce reveals their existence at the data link network layer.

The techniques developed in Chapter 4 and 5 were combined in Chapter 6 in an attempt to infer the topology of a live Ethernet network. The experiments led to the development of an algorithm and a heuristic that could be used to infer the topology of a network based on round-trip time measurements towards target nodes made from various positions in the network.

The accuracy and robustness of the algorithm developed in Chapter 6 could not be determined due to the unavailability of an accurate network map for comparison. Chapter 7 employed computer simulations in order to support, refine and test the topology inference algorithm. The simulations also provided insights into the limitations of the algorithm.

We have shown that it is possible to infer the topology of an Ethernet network provided certain constraints are met. These constraints and limitation are the topic of the following section. Reflection during the work (and also after the work had been completed) also identified opportunities for possible future work and these are the topic of Section 8.3.

## **8.2 Limitations**

Our experiments, performed to identify the internal hubs and switches of an Ethernet network (the focus of Chapter 4), were performed in a very controlled test environment where the network elements all performed at the same speed. The assumptions and reasoning about the behaviour of network elements might prove invalid in the presence of, for example, mobile

or wireless nodes. The effectiveness of the technique can therefore not be assured for all possible network environments.

The simulations conducted in Chapter 7 showed that placement of the measurement points in the network is of utmost importance in order to generate an accurate network topology. In a large network it might be impractical to physically move a single measurement station to all the relevant positions in the network. One solution might be to place measurement software into nodes, but this immediately requires cooperation from the nodes. It is also difficult to determine where relevant measurement positions are, because the very artifact that would help, a network map, is what we are trying to create.

### 8.3 Future Work

In Chapter 6 the concept of node clusters was created and we then connected these clusters in order to form a larger network topology. Instead of merely connecting these clusters, it might be possible to use the signature similarity between the clusters as an attractive force in the network graph. Clusters with similar signatures would then automatically move closer together if a suitable spring- or force-based network layout tool can be constructed or identified.

During our reconstruction simulations described in Chapter 7 it was shown that a network element disappears from our reconstructed topology when measurements from that specific network position are discarded during processing. It might be possible to reconstruct the appearance of the network **from** a specific location by using measurements **towards** that location. This could especially be useful where an internal network element, such as a managed switch, has a layer 2 network address that can be used as the target address for measurements.

The errors made by our reconstruction algorithm, when a limited number of viewpoints in the network were considered (refer to Section 7.5.3), were only illustrated visually. A more stringent mathematical approach might provide

improved quantification of the errors.

The round-trip delays experienced by single ARP packets of various sizes showed that an estimate could be made about the number of store-and-forward elements present between the measurement node and a target node (refer to Section 5.4). Information about the number of elements present between a measurement point and target node could be combined with the results of the reconstruction algorithm to identify nodes missing from the reconstructed graph. These nodes could then either be manually added to the network graph, or an attempt could be made to perform measurements from these nodes. In the same way, the technique used in Chapter 5 to identify hub elements internal to the network could be used to augment the map created by our reconstruction algorithm.

Network simulations, while proving extremely helpful, are not a substitute for empirical real-world experiments. Experiments using real networks would therefore provide the ultimate test for the reconstruction techniques and algorithm that was developed.

Despite the limitations and possible improvements mentioned in this and the previous section, the reconstruction techniques were developed in an area where shortcomings in related techniques were identified. It is hoped that the strengths of the techniques developed during the course of this research effort could be combined with existing and related techniques in order to create improved topology discovery tools.



# Glossary of Abbreviations

<b>AFT</b>	Address Forwarding Table
<b>ANSI</b>	American National Standards Institute
<b>ARP</b>	Address Resolution Protocol
<b>ARPA</b>	Advanced Research Projects Agency
<b>CAIDA</b>	Cooperative Association for Internet Data Analysis
<b>CDP</b>	Cisco Discovery Protocol or Cabletron Discovery Protocol
<b>CRC</b>	Cyclic Redundancy Check
<b>CSMA/CD</b>	Carrier Sense Multiple Access with Collision Detection
<b>DDoS</b>	Distributed Denial of Service
<b>DHCP</b>	Dynamic Host Configuration Protocol
<b>EDP</b>	Extreme Discovery Protocol
<b>FCS</b>	Frame Check Sequence
<b>FIFO</b>	First-In-First-Out
<b>GNU</b>	GNU's Not Unix

<b>GPL</b>	GNU General Public License
<b>ICMP</b>	Internet Control Message Protocol
<b>IDS</b>	Intrusion Detection System
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IETF</b>	Internet Engineering Task Force
<b>IPv4</b>	Internet Protocol version 4
<b>ISO</b>	International Organisation for Standardisation
<b>LAN</b>	Local Area Network
<b>LLC</b>	Logical Link Control
<b>LLDP</b>	Link Layer Discovery Protocol
<b>MAC</b>	Medium Access Control
<b>MIB</b>	Management Information Base
<b>NAC</b>	Network Access Control or Network Admission Control
<b>NDP</b>	Nortel Discovery Protocol
<b>NMS</b>	Network Management Systems
<b>OAM</b>	Operational, Administration and Maintenance
<b>OSI</b>	Open Systems Interconnection
<b>OUI</b>	Organisationally Unique Identifier
<b>PC</b>	Personal Computer
<b>PCI</b>	Peripheral Component Interconnect

<b>PDML</b>	Packet Details Markup Language
<b>QoS</b>	Quality of Service
<b>REDD</b>	RTLinux Ethernet Device Drivers
<b>RFC</b>	Request For Comments
<b>RTLinux</b>	Real-Time Linux
<b>SATNAC</b>	Southern African Telecommunication Networks and Applications Conference
<b>SMB</b>	Server Message Block
<b>SNMP</b>	Simple Network Management Protocol
<b>SQL</b>	Structured Query Language
<b>SVG</b>	Scalable Vector Graphics
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>UDP</b>	User Datagram Protocol
<b>UTP</b>	Unshielded Twisted Pair
<b>VOIP</b>	Voice over IP
<b>VP</b>	Viewpoint
<b>XML</b>	eXtensible Markup Language

# Bibliography

- [1] K. Ueno, K. Shiga, and S. Morita. *A Mathematical Gift, 1: The Interplay Between Topology, Functions, Geometry, and Algebra (Mathematical World)*. American Mathematical Society, October 2003. [1](#)
- [2] V. Runde. *A Taste of Topology (Universitext)*. Springer, July 2005. [1](#)
- [3] M. Dodge and R. Kitchin. *Mapping Cyberspace*. Routledge, 2001. [1](#), [19](#), [20](#)
- [4] R. Black, A. Donnelly, and C. Fournet. Ethernet topology discovery without network assistance. In *ICNP '04: Proceedings of the Network Protocols, 12th IEEE International Conference on (ICNP'04)*, pages 328–339, Washington, DC, USA, 2004. IEEE Computer Society. [1](#), [2](#), [20](#), [26](#)
- [5] H.-C. Lin, Y.-F. Wang, C.-H. Wang, and C.-L. Chen. Web-based distributed topology discovery of IP networks. In *ICOIN '01: Proceedings of the The 15th International Conference on Information Networking*, page 857, Washington, DC, USA, 2001. IEEE Computer Society. [1](#), [2](#), [20](#)
- [6] H.-C. Lin, H.-L. Lai, and S.-C. Lai. Automatic link layer topology discovery of IP networks. In *Communications, 1999 IEEE International Conference on*, volume 2, pages 1034–1038, 1999. [1](#), [25](#)
- [7] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz. Topology discovery in heterogeneous IP networks: the Net-

- Inventory system. *Networking, IEEE/ACM Transactions on*, 12:401–414, June 2004. [1](#), [25](#)
- [8] K. C. Claffy. CAIDA: Visualizing the Internet. *IEEE Internet Computing*, 5(1):88, 2001. [1](#), [21](#), [23](#)
- [9] B. Lowekamp, D. O’Hallaron, and T. Gross. Topology discovery for large Ethernet networks. In *ACM SIGCOMM Computer Communication Review, Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 237–248, 2001. [1](#), [25](#)
- [10] O. Heckmann, M. Piringler, J. Schmitt, and R. Steinmetz. On realistic network topologies for simulation. In *MoMeTools ’03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 28–32, New York, NY, USA, 2003. ACM Press. [1](#), [21](#), [85](#)
- [11] D. Apostol, T. Foote-Lennox, T. Markham, A. Down, R. Lu, and D. O’Brien. Checkmate network security modeling. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX ’01.*, volume 1, pages 214–226, 2001. [2](#), [23](#)
- [12] G. Vigna, F. Valeur, J. Zhou, and R. A. Kemmerer. Composable tools for network discovery and security analysis. In *ACSAC ’02: Proceedings of the 18th Annual Computer Security Applications Conference*, page 14, Washington, DC, USA, 2002. IEEE Computer Society. [2](#), [23](#)
- [13] Cisco Network Admission Control (NAC) solution addresses today’s security challenges. White paper, Cisco Systems, 2006. [2](#), [22](#), [23](#)
- [14] W. Ren and H. Jin. Distributed agent-based real time network intrusion forensics system architecture design. In *AINA ’05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, pages 177–182, Washington, DC, USA, 2005. IEEE Computer Society. [2](#), [23](#)

- 
- [15] V. Corey, C. Peterman, S. Shearin, M. S. Greenberg, and J. V. Bokken. Network forensics analysis. *IEEE Internet Computing*, 6(6):60–66, 2002. [2](#), [23](#)
- [16] H. Burch and B. Cheswick. Mapping the Internet. *Computer*, 32(4):97–98,102, 1999. [2](#), [19](#), [23](#)
- [17] D. G. Waddington, F. Chang, R. Viswanathan, and B. Yao. Topology discovery for public IPv6 networks. *SIGCOMM Computer Communication Review*, 33(3):59–68, 2003. [2](#), [21](#), [24](#), [26](#)
- [18] A. Adams, T. Bu, T. Friedman, J. Horowitz, D. Towsley, R. Caceres, N. Duffield, F. Presti, S. Moon, and V. Paxson. The use of end-to-end multicast measurements for characterizing internal network behavior. *Communications Magazine, IEEE*, 38(5):152–159, May 2000. [2](#), [24](#)
- [19] W. R. Stevens. *The Protocols (TCP/IP Illustrated, Volume 1)*. Addison-Wesley Professional, January 1995. [5](#), [11](#)
- [20] L. Parziale, D. W. Liu, C. Matthews, N. Rosselot, C. Davis, J. Forrester, and D. T. Britt. *TCP/IP Tutorial and Technical Overview*. IBM Redbooks, 2006. [5](#)
- [21] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley Professional, 1st edition, January 1997. [5](#), [7](#), [12](#), [15](#), [52](#), [61](#)
- [22] D. Gollmann. *Computer Security*. John Wiley & Sons, 1st edition, February 1999. [6](#)
- [23] C. Kozirook. *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. No Starch Press, October 2005. [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [14](#), [51](#)
- [24] L. Senecal. Understanding and preventing attacks at layer 2 of the OSI reference model. In *Proceedings of the 4th Annual Communication Networks and Services Research Conference (CNSR'06)*, pages 6–8. IEEE Computer Society, 2006. [6](#)

- [25] D. Bruschi, A. Ornaghi, and E. Rosti. S-ARP: a secure address resolution protocol. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 66, Washington, DC, USA, 2003. IEEE Computer Society. [6](#), [11](#)
- [26] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley, January 2004. [6](#)
- [27] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 3rd international edition, May 1996. [6](#), [7](#), [11](#), [15](#), [50](#)
- [28] D. C. Plummer. RFC826 – An Ethernet Address Resolution Protocol. Internet Request For Comments, November 1982. [9](#), [10](#)
- [29] IEEE Std 802-2001 (Revision of IEEE Std 802-1990) – standard for local and metropolitan area networks: overview and architecture. IEEE Standard, 2002. [10](#), [12](#), [13](#)
- [30] S. Convery. *Network Security Architectures*. Cisco Press, 2nd edition, April 2004. [11](#), [22](#)
- [31] IEEE Std 802.1X-2004 – standard for local and metropolitan area networks: Port-Based Network Access Control. IEEE Standard, 2004. [11](#), [22](#)
- [32] R. M. Metcalfe and D. R. Boggs. Ethernet: distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, 1976. [11](#), [79](#), [80](#), [86](#)
- [33] B. Taylor. Interview: Robert Metcalfe, recipient, National Medal of Technology, March 2005. ITworld.com Voices. [11](#)
- [34] B. Metcalfe. What is this thing they call Ethernet?, December 2002. Electronic Design Technology Report. [11](#)
- [35] P. Hochmuth. Ethernet inventor makes the Inventors Hall of Fame, February 2007. Network World. [12](#)

- 
- [36] Cisco Systems Inc. *Internetworking Technologies Handbook*. Cisco Press, 3rd edition, December 2000. [12](#), [14](#), [16](#)
- [37] IEEE Std 802.3-2002 – standard for local and metropolitan area networks, part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. IEEE Standard, 2002. [12](#), [13](#), [14](#), [16](#)
- [38] M. Zalewski. *Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks*. No Starch Press, April 2005. [16](#), [17](#), [51](#)
- [39] A. V. Chiarella. *Cisco CCNA Self Study Guide: Routing and Switching Exam 640-607*. OnWord Press (Acquired Titles), 1st edition, June 2002. [17](#), [50](#), [52](#)
- [40] O. Arkin. Next generation infrastructure discovery and monitoring systems. White paper, Insightix Ltd., January 2005. [17](#), [18](#)
- [41] O. Arkin. On the deficiencies of active network discovery systems. White paper, Insightix Ltd., June 2005. [17](#)
- [42] O. Arkin. Demystifying the myth of passive network discovery and monitoring systems. White paper, Insightix Ltd., June 2005. [17](#), [18](#)
- [43] S. Webster, R. Lippmann, and M. Zissman. Experience using active and passive mapping for network situational awareness. In *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*, pages 19–26, July 2006. [18](#)
- [44] K. Hafner and M. Lyon. *Where Wizards Stay Up Late: The Origins Of The Internet*. Simon & Schuster, reprinted edition, January 1998. [19](#)
- [45] J. Yu, F. B. X. H. M. Zeng, Z. H. Li, and Y. X. Chun. A distributed architecture for Internet router level topology discovering systems. In *Parallel and Distributed Computing, Applications and Technologies*,



2003. *PDCAT'2003. Proceedings of the Fourth International Conference on*, pages 47–51. IEEE, August 2003. 20, 24
- [46] A. Westerinen and W. Bumpus. The continuing evolution of distributed systems management. *IEICE Transactions on Information and Systems*, E86-D:2256–2261, November 2003. 20
- [47] D. C. M. Wood, S. S. Coleman, and M. F. Schwartz. Fremont: A system for discovering network characteristics and problems. In *USENIX Winter*, pages 335–348, 1993. 20, 26, 46
- [48] B. Huffaker, D. Plummer, D. Moore, and K. Claffy. Topology discovery by active probing. In *Applications and the Internet (SAINT) Workshops, 2002. Symposium on*, pages 90–96, 2002. 20
- [49] B. Cheswick, H. Burch, and S. Branigan. Mapping and Visualizing the Internet. In *USENIX Annual Technical Conference*, pages 1–12, June 2000. 20, 23
- [50] R. Siamwalla, R. Sharma, and S. Keshav. Discovering Internet topology. Submitted to Infocom'99. Cornell University. 20, 21, 24, 26
- [51] V. Bahl, P. Barham, R. Black, R. Chandra, M. Goldszmidt, R. Isaacs, S. Kandula, L. Li, J. MacCormick, D. A. Maltz, R. Mortier, M. Wawrzoniak, and M. Zhang. Discovering dependencies for network management. In *Proceedings of the ACM SIGCOMM Hot Topics in Networks (HotNets) 2006*, pages 97–102, November 2006. 21, 26
- [52] SearchNetworking.com Product Leadership Awards – Network and IT management platforms, 2007. Last accessed March 2007 at [http://searchnetworking.techtarget.com/productsOfTheYearCategory/0,294802,sid7\\_tax306257\\_ayr2007,00.html](http://searchnetworking.techtarget.com/productsOfTheYearCategory/0,294802,sid7_tax306257_ayr2007,00.html). 21
- [53] J. W. Byers. Inference and labeling of metric-induced network topologies. *IEEE Transactions on Parallel and Distributed Systems*, 16(11):1053–1065, 2005. 21

- [54] G. Shao, F. Berman, and R. Wolski. Using effective network views to promote distributed application performance. In *Parallel and Distributed Processing Techniques and Applications*, pages 2649–2656, 1999. [21](#)
- [55] B. B. Lowekamp. Combining active and passive network measurements to build scalable monitoring systems on the grid. *SIGMETRICS Performance Evaluation Review*, 30(4):19–26, 2003. [21](#)
- [56] D. T. Stott. Layer-2 path discovery using spanning tree MIBs. Technical report, Avaya Laboratories, March 2002. [21](#), [25](#)
- [57] B. Karacali, L. Denby, and J. Meloche. Scalable network assessment for IP telephony. In *Communications, 2004 IEEE International Conference on*, volume 3, pages 1505–1511. IEEE, 2004. [21](#)
- [58] M. Bearden, L. Denby, B. Karacali, J. Meloche, and D. Stott. Assessing network readiness for IP telephony. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 4, pages 2568–2572. IEEE, 2002. [21](#), [26](#)
- [59] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004. [21](#)
- [60] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: degree-based vs. structural. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 147–159, New York, NY, USA, 2002. ACM Press. [21](#), [22](#), [85](#)
- [61] S. C. Au, C. Leckie, A. Parhar, and G. Wong. Efficient visualization of large routing topologies. *International Journal of Network Management*, 14(2):105–118, 2004. [21](#)

- [62] M. van den Nieuwelaar and R. Hunt. Real-time carrier network traffic measurement, visualisation and topology modelling. *Computer Communications*, 27(1):128–140, 2004. [21](#)
- [63] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings of*, volume 3, pages 1371–1380. IEEE, 2000. [22](#)
- [64] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001. [22](#)
- [65] D. Cappelli, A. Moore, T. J. Shimeall, and R. Trzeciak. Common sense guide to prevention and detection of insider threats. Technical report, Carnegie Mellon University CyLab, July 2006. [22](#)
- [66] B. Cheswick. The design of a secure Internet gateway. In *USENIX Summer*, 1990. [22](#)
- [67] M. Keeney, E. Kowalski, D. Cappelli, A. Moore, T. Shimeall, and S. Rogers. Insider threat study: Computer system sabotage in critical infrastructure sectors. Technical report, National Threat Assessment Center of the United States Secret Service and CERT Coordination Center, May 2005. [22](#)
- [68] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson. CSI/FBI computer crime and security survey. Technical report, Computer Security Institute, 2006. [22](#)
- [69] T. Markham and C. Payne. Security at the network edge: a distributed firewall architecture. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings*, volume 1, pages 279–286, 2001. [22](#)
- [70] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A. Surendran, and D. Martin. Automatic management of network security policy. *DISCEX*, 02:1012, 2001. [22](#)

- [71] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc., 1998. 22
- [72] O. Arkin. Bypassing network access control systems. White paper, Insightix Ltd., September 2006. 22
- [73] *Building a Network Access Control Solution With IBM Tivoli and Cisco Systems*. IBM Redbooks, January 2007. 22
- [74] J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *IWIA '04: Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04)*, page 48, Washington, DC, USA, 2004. IEEE Computer Society. 23
- [75] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malware*, pages 24–33, New York, NY, USA, 2003. ACM Press. 23
- [76] Internet mapping project. Last accessed March 2007 at <http://www.cheswick.com/ches/map/index.html>. 23
- [77] R. H. Anderson, R. Brackney, and T. Bozek. Advanced network defense research. In *RAND Workshop Proceedings*. RAND Corporation, August 2000. 23
- [78] B. D'Ambrosio, M. Takikawa, D. Upper, J. Fitzgerald, and S. Mahoney. Security situation assessment and response evaluation (SSARE). *DIS-CEX*, 01:0387, 2001. 24
- [79] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang. Maximum likelihood network topology identification from edge-based unicast measurements. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and*

- modeling of computer systems*, pages 11–20, New York, NY, USA, 2002. ACM Press. [24](#), [51](#)
- [80] N. G. Duffield and F. L. Presti. Network tomography from measured end-to-end delay covariance. *IEEE/ACM Transactions on Networking*, 12(6):978–992, 2004. [24](#), [52](#)
- [81] M. Coates, A. Hero, R. Nowak, and B. Yu. Internet tomography. *IEEE Signal Processing Magazine*, 19(3):47–65, May 2002. [24](#), [26](#)
- [82] M.-F. Shih and A. Hero. Network topology discovery using finite mixture models. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, volume 2, 2004. [24](#)
- [83] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz. Topology discovery in heterogeneous IP networks. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 1, pages 265–274, 2000. [24](#), [25](#)
- [84] Y. Bejerano, Y. Breitbart, M. Garofalakis, and R. Rastogi. Physical Topology Discovery for Large Multi-Subnet Networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 342–352, April 2003. [25](#)
- [85] P. Narayan, Y. Brietbart, M. Garofalakis, S. Iyer, C. Martin, G. Prabhakar, R. Rastogi, and A. Silberschatz. Physical and service topology discovery in heterogeneous networks: the NetInventory system. In *Telecommunications Network Strategy and Planning Symposium. NETWORKS 2004, 11th International*, pages 279–284, June 2004. [25](#)
- [86] Y. Sun, Z. Wu, and Z. Shi. The physical topology discovery for switched Ethernet based on connections reasoning technique. In *Communications and Information Technology, 2005. ISCIT 2005. IEEE International Symposium on*, volume 1, pages 44–47, 2005. [25](#), [26](#)

- [87] Y. Bejerano. Taking the skeletons out of the closets: A simple and efficient topology discovery scheme for large Ethernet LANs. In *25th Conference on Computer Communications, IEEE INFOCOM 2006*, April 2006. 25, 26
- [88] Y. Li, C. Pei, C. Zhu, and J. Li. An algorithm for discovering physical topology in single subnet IP networks. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, volume 2, pages 351–354, March 2005. 25
- [89] E. Decker, P. Langille, A. Rijssinghani, and K. McCloghrie. RFC1493 – Definitions of Managed Objects for Bridges. Internet Request For Comments, 1993. 25
- [90] Discovering and mapping level 2 devices. White paper, Hewlett-Packard Co., March 2001. 25
- [91] A. Bierman and K. Jones. RFC2922 – Physical Topology MIB. Internet Request For Comments, 2000. 26
- [92] IEEE Std 802.1AB-2005 – standard for local and metropolitan area networks: Station and Media Access Control Connectivity Discovery. IEEE Standard, 2005. 26
- [93] Link Layer Discovery Protocol (LLDP) A New Standard for Discovering and Managing Converged Network Devices. Technical brief, Extreme Networks, 2006. 26, 27
- [94] B. Boardman. Layer 2 layout: Layer 2 discovery digs deep, November 2003. Network & Systems Management Workshop. 27
- [95] M. McFarland, S. Salam, and R. Checker. Ethernet OAM: key enabler for carrier class metro Ethernet services. *Communications Magazine, IEEE*, 43(11):152–157, November 2005. 27
- [96] Online manual page for tcpdump. [http://www.tcpdump.org/tcpdump\\_man.html](http://www.tcpdump.org/tcpdump_man.html). 30

- 
- [97] SQLite home page. Last accessed March 2007 at <http://www.sqlite.org/>. 31
- [98] PDML specification. <http://analyzer.polito.it/30alpha/docs/dissectors/PDMLSpec.htm>. 34, 37
- [99] Ethereal display filter reference. <http://www.ethereal.com/docs/dfref/>. 35
- [100] K. Kline and D. Kline. *SQL in a Nutshell: A Desktop Quick Reference*. O'Reilly, January 2001. 40
- [101] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991. 46
- [102] K. Anagnostakis, M. Greenwald, and R. Ryger. cing: measuring network-internal delays using only existing infrastructure. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 2112–2121, 2003. 52
- [103] RTLinux/GPL web site. <http://www.rtlinux-gpl.org/>. 53
- [104] M. Barabanov and V. Yodaiken. Real-Time Linux. *Linux Journal*, 1996. 53, 54
- [105] RTLinux/GPL Ethernet Device Drivers project page. Last accessed February 2007 at <http://redd.sourceforge.net/>. 53
- [106] F. Bruckner. RTLinux Ethernet Device Drivers. In *Eighth Real-Time Linux Workshop*, pages 125–127, 2006. 53
- [107] RTL8139(A/B) Programming guide: (V0.1), 1999. Realtek Chip design & System design. 53
- [108] S. Perez, J. Vila, and I. Ripoll. Building Ethernet Drivers on RTLinux–GPL. In *Real-Time Linux Workshop*, 2003. 53

- [109] C. Dougan and Z. Mwaikambo. Lies, misdirection and realtime measurements. Technical report, Finite State Machine Labs, 2004. [53](#)
- [110] J. P. Delport and M. S. Olivier. Detecting uncooperative Ethernet elements using accurate round-trip time measurements. In D. Browne, editor, *Southern African Telecommunication Networks and Applications Conference (SATNAC)*, volume 1, pages 153–156, 2005. [66](#)
- [111] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 618–627, 2002. [68](#)
- [112] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. [74](#)
- [113] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software – Practice and Experience*, 30(11):1203–1233, 2000. [74](#)
- [114] N. Hu and P. Steenkiste. Quantifying Internet end-to-end route similarity. In *Passive and Active Measurement Conference 2006*, 2006. [76](#)
- [115] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002. [86](#), [87](#), [92](#)
- [116] G. Chartrand. *Introductory Graph Theory*. Dover Publications Inc., 1985. [86](#)
- [117] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, February 1956. [87](#)
- [118] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. [88](#)