REFERENCE MODELS FOR THE CONCEALMENT AND OBSERVATION OF ORIGIN IDENTITY IN STORE-AND-FORWARD NETWORKS

by Thomas E. Daniels

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

CERIAS Tech Report 2002-31

REFERENCE MODELS FOR THE CONCEALMENT AND OBSERVATION OF ORIGIN IDENTITY IN STORE-AND-FORWARD NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Thomas E. Daniels

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2002

REFERENCE MODELS FOR THE CONCEALMENT AND OBSERVATION OF ORIGIN IDENTITY IN STORE-AND-FORWARD NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Thomas E. Daniels

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2002

I dedicate this dissertation to my father, Thomas Arthur Daniels, 1929–2001. His strong and steady yet unassuming guidance made me what I am today. I miss him dearly.

ACKNOWLEDGMENTS

I owe a great deal of my success to the values instilled in me by my family. First, I would like to thank my parents, Grace Marie Jones Daniels and Thomas Arthur Daniels, who always encouraged me to make a better life for myself through advancing my education. Second, I would like to thank my uncle and aunt, Clarence and Betty Jones, for stressing that education was the way to go further in life. Third, I thank my in–laws, the Martin and Martha Walker family, for helping my wife and I numerous times during our many years of post-secondary education. Fourth, I thank my uncle and aunt, Jesse and the late Mary Ellen (Daniels) Masters, for their moral and monetary support to continue my education. Finally, I do not know what I would have done without the constant loving support and friendship of my wife, Dr. Jennifer Lea Walker–Daniels. She has given me two wonderful children and the emotional and practical support to complete this work.

Many people at the Center for Education and Research in Information Assurance and Security have my gratitude. I thank my advisor, Dr. Eugene Spafford, for leading me in directions that I never would have taken otherwise and being instrumental in my growth as a researcher. I also would like to thank my fellow COAST and CERIAS students for their help, camaraderie, and good cheer while putting up with "green horn Tom" and "dissertating Tom," respectively. I especially wish to thank Dr. Diego Zamboni, Benjamin Kuperman, Dr. Wenliang Du, James Early, Sofie Nystrom, and Dr. Ivan Krsul for their advice and influence on my career. I send special thanks to Dr. Judy Oates Lewandowski for commiseration during the dissertation process and also to Dr. Melissa Dark for her advice and guidance in education that reminded me of my love for teaching. Finally, I wish to thank the CERIAS administrative staff for making the work environment joyful and for handling the support functions that made my work possible.

I also wish to thank my committee for their time and effort. Dr. Mikhail Atallah has been a second academic father to me.

Living in Lafayette has been a pleasure thanks to the many friends that I have made. I would like to thank the members of the Tippecanoe Homebrewers' Circle and Anita Johnson for the good times and for sharing the love of brewing good beer. I thank Crystal Stoyke for taking excellent care of my children so that I could complete this dissertation.

I thank Nelson H. F. Beeb for compiling the request for comments BibTeX database and making it available so that I could use it in this dissertation. I also thank Clay Shields for the use of his BibTeX database in this work and for his guidance and advice while he was at Purdue.

I thank Intel for their generous fellowship which supported my early research. I also wish to thank Dr. Sally Goldman for suggesting planar networks as an interesting topology for traceback algorithms. I also thank our sponsors as portions of this work were supported by a contract from the United States Air Force, Grant EIA-9903545 from the National Science Foundation, Contract N00014–02–1–0364 from the Office of Naval Research, and the sponsors of the Center for Education and Research in Information Assurance and Security.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Daniels, Thomas E., Ph.D., Purdue University, December, 2002. Reference Models for the Concealment and Observation of Origin Identity in Store-and-Forward Networks. Major Professor: Eugene H. Spafford.

Past work on determining the origin of network traffic has been done in a case-specific manner. This has resulted in a number of specific works while yielding little general understanding of the mechanisms used for expression, concealment, and observation of origin identity.

This dissertation addresses this state of affairs by presenting a reference model of how the originator identity of network data elements are concealed and observed. The result is a model that is useful for representing origin concealment and identification scenarios and reasoning about their properties. From the model, we have determined several mutually sufficient conditions for passively determining the origin of traffic. Based on these conditions, we have developed two new origin identification algorithms for constrained network topologies.

1. Introduction

1.1 Problem Statement

Store–and–forward networks and the anonymity systems built for them provide mechanisms that a sender can use to conceal the origin of its traffic from other entities in the network. In these networks, a variety of identification techniques have been studied to determine the origin of traffic despite attempts to conceal it. Because there are legitimate and illegitimate uses of origin concealment from the viewpoints of both individuals and governmental entities, origin concealment and identification are both important fields of study.

Past study of origin concealment has not incorporated an underlying theoretical framework. Similarly, past work on determining the origin of network traffic after the fact has addressed specific origin concealment mechanisms instead of looking at the general problem. The result is that there are many proposed origin identification systems without a unifying framework and little knowledge about origin concealment and identification in general.

1.2 Thesis Statement

It is possible to specify a reference model for the concealment and observation of the identity of originators of data elements in store–and–forward networks. The reference model is useful for reasoning about mechanisms of origin concealment. The reference model is useful for reasoning about the placement of monitors, observations needed, and development of new search algorithms used in automated systems for passively determining the identity of the origin of network data elements based on their structural components.

1.3 Motivations

Concealing and determining the origin of network traffic in store–and–forward networks [1] are important interrelated fields of study. Techniques in each field have both legitimate and illegitimate uses. Indeed, the legitimacy of a given use of origin concealment or identification may be dependent on whether the viewpoint is that of an individual, a government, or commercial entity. This subjectivity leads to a tension between those who support methods of network origin concealment and those who wish to identify them. We discuss uses of origin concealment and identification techniques to demonstrate the importance of both fields.

1.3.1 Uses of Origin Concealment

Origin concealment in store–and–forward networks has been researched as a mechanism for protecting the privacy of network users [Cha81, RR97, Moe, LS00]. There are many legitimate uses of origin concealment from the viewpoint of the individual desiring to conceal his identity. Indeed, an article by Gary Marx lists fifteen "socially sanctioned contexts" for concealing the identity of individuals [Mar01]. We summarize several of these contexts from the article in the following list.

- Facilitating open discussion of public issues such as reporting safety problems and political issues.
- Obtaining personal information for research studies such as those regarding sexual or criminal behavior.
- Encouraging readers to consider the content of a message instead of being biased by the identity of the author.
- Avoiding persecution for expressing unpopular beliefs.
- Protecting one's self from unwanted intrusion such as unsolicited contact via email.

Although Marx's article describes these as socially sanctioned, we note that some governments or commercial entities may view these contexts as illegitimate [oS01].

[1]Store–and–forward networks are defined in Section 1.5.

Governments that do not acknowledge rights to freedom of speech do not wish to facilitate open discussion of political issues. Such governments may wish to determine the origins of participants in such discussions when they occur over a network. Similarly, commercial entities may wish to determine the identity of those reporting safety issues or find identifying information about individuals for direct marketing [Sor97] .

As we will discuss in Chapter 2, several origin concealment systems have been studied to make determining the origin of various types of supposed legitimate network use difficult [RR97, LS00, Moe, RSG95, pen96]. These uses include viewing World Wide Web (WWW) pages, sending electronic mail, and posting to electronic newsgroups. Individuals might use these systems to research unpopular subjects or reporting unsafe or illegal practices. Governments and commercial entities might use these systems to covertly collect data on suspected criminals or research competitor's products.

Origin concealment systems may also be used for what others consider illegitimate purposes. Individuals may make libelous statements, send ransom notes, conspire to commit crimes, or commit attacks against network entities. Because these may be against the law, government entities may desire to limit the use of origin concealment or use origin identification techniques to determine the origin of traffic despite attempts to conceal it.

1.3.2 Uses of Origin Identification

Determining the origin of network traffic is an important problem. Malfeasors have used computer networks to commit crimes of many types. In some cases, it has been difficult to determine the identity of the malfeasor or even the computer from which the crime was committed. Many examples of using computer networks for crimes are documented in past work [Par98, Sto89, SM96, FM97, Dit00]. These examples often include using origin concealment techniques. Government and commercial entities desire to determine the origin of these crimes and attacks for civil and criminal litigation.

These crimes have cost victims money and have been difficult to trace. Even the distributed denial of service attacks of February 2000 against Yahoo, Cable News Network, Etrade, Amazon, and EBay which were widely publicized and reputed to have cost millions of dollars [Kab00, And00a, Dit00] were not traced to their origin by analysis of the attacks despite the publicity and monetary loss. An attacker, known as "Mafiaboy," was convicted of attacks against Cable News Network in February 2000 [Cen], but not because of analysis of attack traffic. An informant reported that Mafiaboy bragged about his plans to attack Internet sites using Internet relay chat [Bur00].

According to the 2002 CSI/FBI computer crime survey [Ins02], 40% of the survey respondents detected system penetration from the outside. Forty percent also detected denial of service attacks against them with 74% of respondents citing their Internet connection as a frequent source of attack.

Determining the origin of network data elements that make up crimes and attacks could assist in finding an individual for prosecution, even if not directly admissible as evidence. They also assist victims in identifying a candidate for civil litigation [BDKS00]. On a technical note, determining the origin of an attack may allow a target to filter out the attack nearer to the source thereby conserving bandwidth and protecting the target [Row99]. Determining the origin of an attack may also allow the target to contact administrative personnel at the origin to stop or prevent further attacks. Finally, a mechanism for determining the origin of network traffic could serve to discourage use of networks for criminal acts as the malfeasor would more likely be caught.

Determining the origin of network traffic by governments may not always be legitimate use from the viewpoint of the user wishing to conceal the origin of his actions. As discussed in the previous section, the origin of network traffic may be used to infringe on an individual's freedom of speech. For example, a political dissident may make statements against a government, but that government could use origin identification techniques to find the dissident and pressure him to stop. In extreme cases, this might include imprisonment, torture, or execution.

1.3.3 Tension Between Origin Concealment and Identification

As we have described, the legitimacy of the use of origin concealment and identification techniques is subjective. Both have legitimate and illegitimate uses from the viewpoints of governments and individuals. Hence both areas are important areas of study as the effectiveness of techniques for concealment decreases the effectiveness of techniques for identification and vice versa.

Our approach is to refrain from making judgments about the legitimacy of a technique. We present work in both areas, but it is beyond the scope of this dissertation to determine the legitimacy of origin concealment or origin identification for a given situation or use.

1.3.4 A More General Approach

Existing work in identifying the origin of network traffic has focussed on constrained problem and solution spaces. Specific types of traffic and attacks have motivated much of the work. Many of the approaches take advantage of properties of network floods to reduce storage requirements or use of bandwidth on the network [Bel00, SWKA00a, BC00, CD97, MOT$^+$99]. Some of the work has focused on unintended uses of existing infrastructure and technology for origin identification [Row99, SDS00, Sto99, CNW$^+$99]. Because of these constrained approaches, little work has been done on the general aspects of network origin identification.

The goal of this dissertation is to gain understanding of general approaches taken for network origin concealment and identification. This is important because it is this general understanding that is useful for education and discussion of origin identification [Jen96]. To do this, we develop reference models for origin concealment and identification. As one example of the utility of reference models, consider that few organizations use the OSI protocol suite yet the OSI protocol reference model [Zim80] is still widely used in educating people about networks [Tan88, Sta00, PD96, TJ98].

Chapter 3 describes a reference model of origin concealment in networks. This model may be useful to the national defense community as it is also useful for reasoning about the weaknesses of origin concealment systems. It may also be useful for law enforcement communities when investigating crimes that involve the use of origin concealment techniques.

1.4 Contributions

We make the following contributions to the body of knowledge regarding origin identification and concealment:

- The first taxonomy of systems for identifying the origin of network traffic, and
- the first reference model of origin concealment and observation in networks, and
- a functional model of network monitors for origin identification, and
- sufficient conditions for origin identification using passive observers, and
- new divide and conquer algorithms for passive origin identification in constrained network topologies.

These contributions form a body of work that can be used for reasoning about origin concealment and identification. They also form the basis for new origin identification algorithms that use divide and conquer techniques in tree and planar network topologies that have the potential to increase the acceptable response time for initiation of a trace.

1.5 Terminology

In this section, we discuss the specialized terminology necessary for understanding this dissertation.

Definition 1 *Store–and–forward network – A graph, $G = (V, E)$, of nodes connected by undirected edges (or links) where each node can generate or drop messages. V is a set of nodes $\{v_1, \ldots, v_k\}$, and E is a set of unordered pairs (v_i, v_j) such that $i \neq j$. Nodes forward finite length messages by receiving the entire message on incident edges before transforming the message and sending the result out their incident edges.*

For the rest of this dissertation, we use the term network to refer to store–and–forward networks.

Definition 2 *Network Data Element (NDE) – A bounded length string of characters from a finite alphabet that forms a unit of communication between network nodes.*

Examples of NDEs are IP packets, Ethernet frames, ATM cells, and telegrams.

Definition 3 *Network Node – An entity in a network that may generate, receive, and output NDEs on its edges. It may have non-network inputs and internal state. A network node can apply transformations to NDEs it receives, and output the results.*

Figure 1.1 is a diagram of a network node and its functional units.

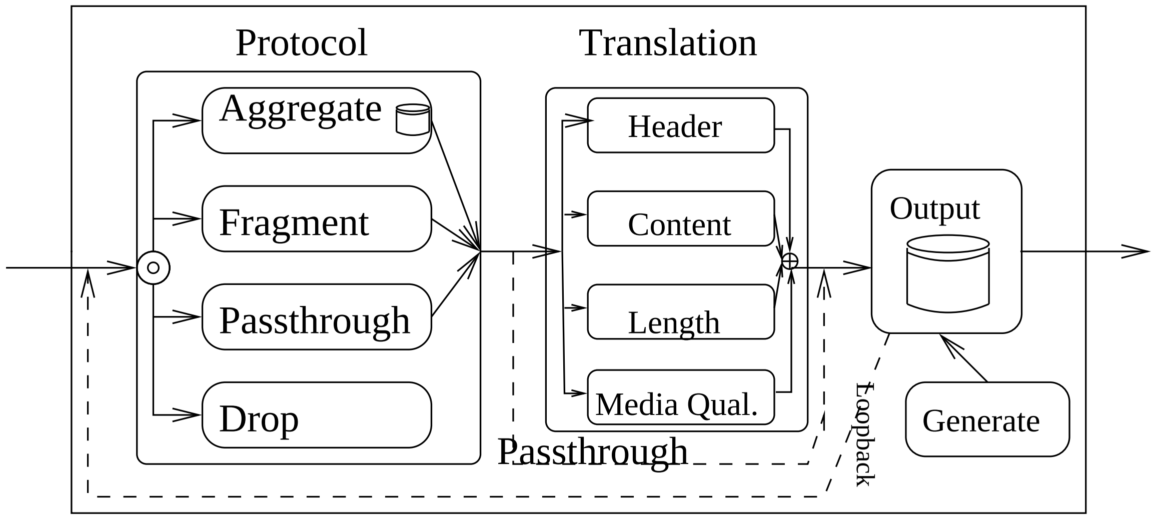

Figure 1.1. A diagram of the functions of a network node.

Definition 4 *Generation – Creation of an NDE that is not a result of a network input. An NDE may be generated in response to non-network input or processing that results from that input.*

Definition 5 *Dropping – A node drops or consumes an NDE if it takes it as input but does not transform it into outputs.*

We use the term drop to refer to NDEs that are received without causing an output NDE regardless of whether the input affects the internal state of the node or not.

Definition 6 *Fragmentation – A transformation that given a single input NDE produces multiple output NDEs.*

Definition 7 *Aggregation – A transformation that given multiple input NDEs produces a single output NDE.*

Definition 8 *Translation – A transformation that outputs a modified version of an input NDE. A translation may modify the content, header, or length of an NDE as well as the characteristics of an NDE specific to its transmission over a medium.*

Definition 9 *Internal Monitor – A software or hardware component of a network node that can observe messages received and generated by the node and report those observations to another node. If an internal monitor is present in a node, we say it is internally monitored.*

Definition 10 *External Monitor – A hardware device dedicated to observing messages traversing a link of a network and capable of reporting those observations to another node. If an edge has an external monitor, we say that it is externally monitored.*

Definition 11 *Monitor – An internal or external monitor.*

Definition 12 *Observer – An abstraction of one or more monitors such that its observations are the union of the observations of those monitors.*

Definition 13 *Observed Network – A store-and-forward network with with one or more monitors present. We define it as a 4-tuple* $G = (V, E, IM, XM)$ *such that* $IM \subset V$ *and* $XM \subset E$*.* IM *is the set of internally monitored nodes.* XM *is the set of externally monitored edges. In an observed network,* $|IM| + |XM| > 0$*.* V *and* E *are sets of nodes and edges as defined for store-and-forward networks above.*

Definition 14 *Origin – The network entity that initially generates an NDE and puts it onto the network.*

Entities in the network may process the NDE and put it back on the network, but the resulting NDE retains its origin. A more formal definition is given in Chapter 3.

Definition 15 *Network Origin Concealment(NOC) – The act of creating or modifying a network data element that is forwarded through one or more nodes of a computer network so that when it is delivered to an arbitrary node, it is difficult to determine the origin of the NDE.*

Definition 16 *Network Origin Concealment System(NOCS) – A NOCS is a computer network that has mechanisms to provide NOC.*

Definition 17 *Network Flow – A set of network data elements generated by some origin to be delivered to the same destination that satisfy a set of pre-defined attributes.*

We usually shorten this term to "flow." We base the definition of network flows on a paper by Mittra and Woo [MW97]. This definition transcends protocol layers and does not require the notion of a connection. This facilitates discussion of past work where the input messages to nodes may employ or be encoded with different protocols than the output caused by them.

Definition 18 *Identifier – A string of bits that uniquely specify a member or set of members from some set of entities.*

Definition 19 *Network Attack – An attempt to exploit a vulnerability in a network node or edge by sending traffic to it.*

Definition 20 *Network Flood – A network attack where an attacker consumes excess processing power in a remote network node or consumes network bandwidth by sending many network flows to a destination.*

Examples of network floods are the smurf attack [CA-98] and SYN floods [CA-96, SKK+97].

Definition 21 *Reference Model – An abstraction of a class of mechanisms that defines the class by describing the members' properties in some structured way. A reference model defines all members of the class and how they interact.*

We base this definition on an intersection of the descriptions of several reference models in networks and security [Sch97, EHM96, Zim80].

1.6 Restatement of the Thesis

It is possible to specify a reference model of network origin concealment and identification systems. The reference model is useful for reasoning about the mechanisms used for origin concealment and origin identification using network monitors. The reference model is useful for reasoning about the placement of monitors, observations needed, and development of new search algorithms used in network origin identification systems.

1.7 Organization of the Dissertation

Chapter 1 introduces the problem and thesis statements along with motivations, contributions of the dissertation, and terminology. Chapter 2 describes past work in origin concealment and origin identification including a taxonomy of origin identification systems we have developed. Chapter 3 describes a reference model for origin concealment of network data elements and demonstrates its utility for reasoning about properties of origin concealment systems. Chapter 4 uses the context of the origin concealment reference model to describe a reference model of passive origin identification. Chapter 5 introduces new algorithms for passive origin identification that have arisen from our reference model of origin identification. Chapter 6 summarizes the dissertation and its conclusions and describes possible future directions for research related to this dissertation.

2. Related Work

In this chapter, we consider the past work in network origin concealment and identification that relates to the thesis statement. In Section 2.1, we consider issues with identity in networks. In Section 2.2, we summarize network origin concealment systems and how they anonymize traffic. In Section 2.3, we present a taxonomy of network origin identification and use it to describe and generalize about past origin identification systems and predict new areas of research. In Section 2.6, we describe other past works that model origin concealment.

2.1 Discussion of Identity in Networks

The designers of computers and computer networks use identifiers to specify a member or set of members from some set of entities. In this section, we choose network nodes and their respective identifiers as having the appropriate level of abstraction for discussing origin identification and concealment in this dissertation. To discuss this choice, we use the ISO OSI [1] protocol reference model [Zim80]. Descriptions of the OSI model, as we shall refer to it, are commonplace in the literature [Tan88, Sta00, Zim80]. The layers of the model are shown in Figure 2.1.

The OSI protocol reference model defines layers of functionality for computer networks. Each layer of the model may contain entities that are bound to their own identifiers. The general approach of the OSI model is that an entity at layer i interacts with entities at layer $i+1$. A layer i entity uses the functionality of layer $i-1$ entities to implement its functionality. An entity at layer $i > 1$ communicates with a layer i entity at a remote network node using entities in the layers below it.

[1] International Standards Organization Open Systems Interconnect

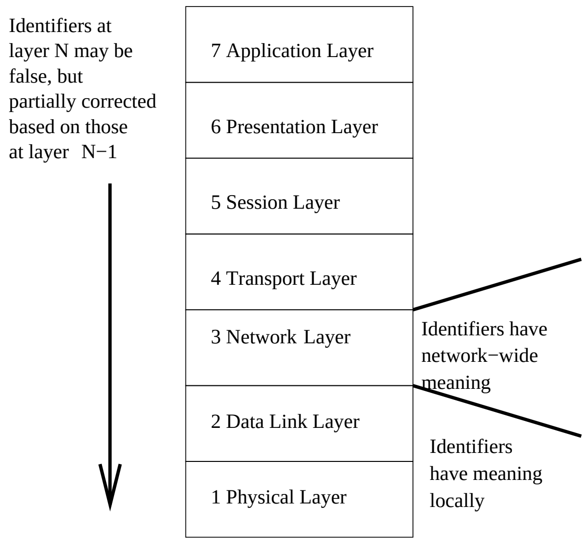
Identifiers at layer N may be false, but partially corrected based on those at layer N−1
7 Application Layer
6 Presentation Layer
5 Session Layer
4 Transport Layer
3 Network Layer
2 Data Link Layer
1 Physical Layer
Identifiers have network−wide meaning
Identifiers have meaning locally

Figure 2.1. The OSI protocol reference model.

2.1.1 Network Layer Identity

The network layer is an appropriate place to discuss network identity. Network layer identifiers refer to network nodes, and in practice, identifiers at a given layer may depend on identifiers at the network layer for context. Examining the OSI model from the lowest layers upward, the network layer is the first where network-wide identifiers are used. Examining the OSI model from the top layers downward, if a layer's identifiers are falsified, then lower-layer identifiers down to the network layer can be used to identify the node of origin, if correct. Hence, the network layer is the lowest layer of the OSI model that deals with identity beyond the local area network.

The network layer in the OSI model is the lowest such that its identifiers are required to be independent of the layer below it. This independence results from the requirement that network layer messages must be able to traverse networks using different data link protocols [Tan88]. Hence, network layer identifiers must not depend on the lower layer identifiers, as they may change every time an NDE is retransmitted.

In practice, the identifiers at level i depend on those in lower levels to identify their entity. An example of this would be an eight character user identifier sent in a remote login protocol to identify the remote user trying to access the computer. The identifier is interpreted as a user identifier at the remote host which is specified by its network layer identifier. If the network layer identifier is incorrect, then the user identifier would be interpreted as a user on a different computer. This user may be the correct person, a different person, or not exist on the node identified by the false identifier.

However, there are examples where an i level origin identifier does not depend on identifiers at layers below. An example is an SMTP [Pos82] email address. The SMTP protocol uses an origin address at the application layer, namely the "from" email address, to be independent from the network address of the sender. For example, a user who travels may send email using the same origin email address from different network nodes. Hence, we can not state that all identifiers above the network layer depend on the network layer identifiers for meaning.

The physical layer deals with network media characteristics, and therefore it is not appropriate for discussing network entities that originate traffic. The entities at this layer are network interface devices and network media. These are entities that may have no network-wide identifiers.

Although the data link layer does deal with identification of network entities that generate traffic, the identifiers present need only have meaning between the nodes connected directly. Media access control (MAC) addresses in Ethernet need only be unique for a local area network not globally. For instance, we have found that a Sun Microsystems workstation can use the same MAC address when connected to multiple distinct local area networks. The network layer does deal with identifiers that have meaning network-wide because it deals with end–to–end communication between nodes of the network, possibly over many hops.

We conclude that the network layer is both the lowest layer where identity beyond that of a local network can be discussed and the lowest layer such that the interpretation of its identifiers require no assumptions about lower layer identifiers. We therefore choose the network layer identity of network nodes as the basis for this dissertation.

2.2 Network Origin Concealment

Network origin concealment systems (NOCS) make the receiver of a network flow uncertain of its origin by modifying or removing its attached origin identifiers so that they are incorrect. In this section, we will discuss two classes of NOCS. The first class of NOCS is "improvised" because it uses the mechanisms available in the network for other purposes to conceal the origin of traffic instead of developing new mechanisms. The second class of NOCS is "prepared" because their designers create or deploy new software or hardware especially devoted to creating uncertainty in the origin of a flow.

2.2.1 Improvised NOCS

Improvised NOCS are sometimes used in network attacks to hide the identity or location of an attacker [Ran99] or criminal. We will discuss known instances of improvised NOC behavior here. We have previously analyzed improvised NOC [BDKS00, DS00a].

IP Spoofing

The spoofing or forging of Internet Protocol(IP) [Pos81a] addresses has been widely described in the literature [Mor85, Bel89, Ran99, Cen96]. *IP spoofing* is when a network node generates an IP packet with an IP source address other than its own.

IP spoofing has been used for numerous types of network denial of service attacks [SKK+97, Dit00]. IP spoofing has also been used to exploit trust relationships based on the IP source address of requests [Mor85, Bel89].

An IP network allows spoofing of IP addresses because delivery of IP datagrams is based only on the destination address. Because the routing infrastructure does not rely on source addresses, the packet is delivered to its destination address, but upon delivery the source address is the forged one and not the actual address of the sender. IP corresponds to the network layer of the OSI model, and therefore, all addresses below it may only have meaning in the local network. Because the identifiers above the IP layer may depend on the IP source address or they may be falsified by the sender, the destination will not be able to trust the source address of an IP datagram unless some form of authentication is used as well.

Reflectors

Reflectors are network nodes that accept an input with a false origin identifier and reply to the false origin [Pax00, Pax01].

An attacker can use a reflector to send network traffic to a target node. By choosing the target of the attack as the origin identifier, the attacker causes the victim to receive the traffic from the reflector. Although the reflected traffic can be sent to any reachable node, the traffic itself is limited to that which the attacker can cause a reflector to generate. Hence, reflectors have been used to conceal the origin of network floods as the content of a flood is not important to its success.

A reflector requires two properties of the network to function. A node must be able to generate traffic with an origin identifier bound to the target. A reflector node must process the traffic by swapping the origin and destination identifiers to create an output. Hence, a reflector is a NOCS. The traffic received by the victim has the origin identifier of the reflector instead of that of the attacker.

There are many protocols that support reflectors [Pax01]. The Internet protocol [Pos81a] itself specifies several including the translation of certain IP packets into ICMP TTL exceeded messages and ICMP echo replies. The TCP state machine is a reflector as it replies to arbitrary SYN packets with SYN-ACK's. Additionally, higher layer protocols such as domain name service (DNS) [Pos94] and file transfer protocol (FTP) [Bhu72] also specify reflector behavior.

An example of a reflector can be found in the United States postal system. An attacker who wished to fill the mail box of a target could send many blank letters with incorrect return addresses. If the letters had insufficient postage or nonexistent destination addresses, the letters would be sent to the target instead of returned to their origin. In this case, the letter would be postmarked from its post office of origin, but there would be no more accurate origin identifier on the letter. Another example of a reflector would involve making telephone calls to answering machines and leaving an incorrect return number. The recipients might then make return calls to an unsuspecting target.

Extended Connections

Network attackers use extended connections to hide their origin [SCH95, SC95]. *Extended connections* are remote terminal sessions created by logging into two or more remote nodes in series [JKS$^+$93]. An extended connection is created by using a remote terminal service to log into a network node and using that node to log into yet another node using a possibly different remote terminal service. This process can be repeated to create extended connections across many nodes. A node in an extended connection can then be used to attack other hosts.

Attackers may use nodes they have compromised, user accounts that have had passwords intercepted, and public accounts. This may complicate the process of analyzing the extended connection as the accounts used may have nothing to do with the attacker. Additionally, the attacker may delete logs from the hosts so as to remove information about the links in the extended connection.

Extended connections can be created when it is possible to use a remote terminal session to log into yet another node. The transformations done to the traffic as it passes through nodes leave the content of the connection mostly unchanged [SC95]. Each link of the connection delays the traffic further. Different remote terminal protocols such as `telnet` [PR83], `SSH` [Sec], and `rlogin` [Kan91] use different control messages and may also encrypt traffic. Hence, the observed content and timing of the extended connection may differ from link to link.

An attack sent using an extended connection appears to originate from the last node in the connection instead of the node the attacker is directly using. As each node may belong to a different owner or even be subject to different legal jurisdictions, it can be difficult to examine the nodes of an extended connection to determine earlier nodes. Owners of the nodes may not trust the person desiring to examine the node.

Even if a node is examined, the attacker may have erased or falsified log entries about the previous link of the extended connection.

Application Gateways

An *Application Gateway* is a network node that enforces security policies by acting on behalf of other nodes. Application gateways accept a request from a client node, and then make that request to a service node on the client's behalf. The application gateway may implement access control, auditing, or authentication policies by choosing its actions based on the value of the request or reply.

A client uses an application gateway by sending a service request to the gateway along with the service node for which the request is intended. The gateway processes the request and then makes the processed request on behalf of the client. The processed request then has network layer origin identifiers of the gateway instead of the client because the gateway expects to receive the response from the service node. The processing of the application gateway may also remove or modify identifiers above the network layer as well. The gateway receives the response from the service node, may process it, and pass it along to the client.

Because the service interacts with the application gateway instead of the client, the request may contain no information linking the client to the request. Instead, the request appears to have come from the application gateway.

Some application gateways are specifically designed for origin concealment purposes. We discuss these as prepared NOCS in the following section.

Network Address Translation

Network address translation (NAT) [EF94] is a mechanism designed to connect a private network to a public network using a single public network address. The private network uses reserved addresses not allowed on the public network. This allows the public network to conserve address space because many nodes on the private network only require a single public address. NAT devices were designed to allow multiple networked entities to share a single public network address, not as an origin concealment system [EF94].

A NAT device maintains internal state about the current network events occurring over it and overwrites the network addresses, ports, and sometimes higher level protocol information of the traffic from behind the device to make it appear that the packets come from the public address. Reply packets are sent to the public address and modified again based on the firewall's internal state to match the reply that a client expects and are then forwarded to the internal network.

NAT devices conceal the origin of outgoing traffic by rewriting the IP source addresses in packets to the public IP address of the NAT device. Additionally, origin identifiers above the IP level may be modified as well if the protocol being translated requires it for proper operation. From the perspective of the destination of the packet, the packet could have originated at any node being serviced by the NAT device.

2.2.2 Prepared NOCS

Prepared NOCS are built with the goal of providing origin concealment. Prepared NOCS include systems such as mixes [Cha81] that attempt to anonymize network transactions in the presence of strong adversaries. We discuss mixes further later in this section. Prepared NOCS also include simple remailers that process email to prevent the recipient from determining the sender [pen96]. A review of several prepared NOCS is presented in Martin's dissertation [Jr.99].

In discussing origin concealment mechanisms so far, we have focused on aspects of using origin concealment for crimes and attacks. Prepared NOCS have a variety of uses such as protecting the privacy of World Wide Web users [RR97], anonymous reporting of dangerous conditions, and as part of electronic voting systems [HMP96]. We refrain from labeling the use of any given NOCS as legitimate or not because it is subjective. The need to determine the origin when a NOCS is in use is relative to who is using the system and for what purpose. For example, a political dissident may use a NOCS to make statements against a totalitarian government. The dissident may believe this a legitimate use of origin concealment whereas the government would probably have a different view.

We discuss past work in prepared NOCS by considering the types of changes made to the traffic whose origin is being concealed. We begin with simple remailers and proxies that modify origin identifiers to achieve origin concealment. We then describe NOCS that modify both identifiers and message content. Finally, we consider NOCS that modify identifiers, message content, and add randomized delay to messages.

Simple Remailers

Simple remailers are application gateways for sending pseudonymous email. One of these systems was `anon.penet.fi` [pen96]. Simple remailers assign a unique identifier to each user of the service. The user sends an email to the service from his normal email address, the service strips off any fields that might identify the user, and sends the email to its intended recipient with the unique identifier as the "from" address. Replies to that identifier go to the remailer and are then sent back to the user.

Remailers are a special purpose application gateway that maintain a mapping between the true email address of a user and its anonymized address to support replies to anonymized addresses. Simple remailers modify the headers of messages to change or remove origin identifiers while leaving the content of the message unmodified. When received at their destination, the email output by the remailer have a "from" email address at the remailer and other identifying information such as previous "Received from" headers have been removed. The result is that the email appears to come from the unique identifier at the remailer instead of the actual originator at the origin node.

World Wide Web Proxies

A number of proxy systems have been developed to remove identifying information from HTTP requests made by users of the network [ws, Pro99]. These systems accept the HTTP request made by the client, and modify or remove identifiers in the request before issuing the request to the HTTP server. The HTTP server responds to the system, which then sends the response on to the client. These systems change the network layer origin identifiers so that the responses return to them instead of the client while possibly removing or falsifying origin identifiers. Identifiers about the network layer may also be removed or falsified by the proxy to prevent the service from collecting them.

As with simple remailers, identifiers on the requests are stripped off or modified by an application gateway, which then handles the request and returns the results to the requester.

DDOS tools

Distributed denial of service tools [And00b] such as Stacheldracht [Dit91], Trinoo [IN-99], and the Tribe Flood Network[2] [Dit99] are prepared NOCS for launching simultaneous network attacks from many network nodes. Once a DDOS tool is installed in a node, it accepts control messages from an attacker via the network. Among other tool-specific commands, the messages instruct the tools to carry out denial of service attacks against certain targets. By using many DDOS tools installed on hosts in a network, a single attacker can flood a target from many parts of the network simultaneously.

DDOS tools are NOCS because they are software written with the goal of modifying the origin identifiers of network traffic as part of their attacks. DDOS tools typically use one or more improvised NOC techniques such as spoofing of packets in a flood. Some DDOS tools also use reflectors [Pax01] for their floods and control messages.

Crowds

Crowds [RR97] is a NOCS that uses repeated encryption and randomly chosen routes through the network to conceal the origin of HTTP requests. Crowds requires every participant to run a service called a *jondoe.* Every jondoe regularly runs a discovery protocol to find other jondoes that are active in the network as well as exchange pairwise secret symmetric encryption keys. When a participant makes an

[2]Some references use the name "Tribal Flood Network." The tool has the name "Tribe Flood Network" embedded in it so that is what we use.

HTTP request, it is filtered through a local application gateway that modifies or removes origin identifiers and then passes it on to the local jondoe.

The local jondoe then sets up a random route through the jondoes. It does so by choosing a large random route identifier and appending it to a route creation request. This route creation request is sent out to a randomly chosen jondoe in the crowd. The jondoe then stores the route identifier associated with the randomly chosen jondoe so further requests on that identifier are routed properly to the next jondoe.

When a jondoe receives a route creation request from another non-initial jondoe, it chooses randomly whether the next hop is to the final destination or to add another jondoe to the route. The random choice is biased to create a configurable expected route length. If the request is to be routed on to another jondoe, then the route identifier is changed to avoid an infinite loop should a cycle occur in the route of jondoes. The new route identifier is stored along with the previous one. If the request instead is sent to its destination, the final jondoe in the route makes the request, and sends the response backwards along the route of jondoes to the submitting jondoe. All messages between the jondoes are link encrypted using symmetric encryption.

Once the route has been established, the originating jondoe uses it for requests by prepending the route identifier to the request, encrypting, and then sending to the next jondoe on the route. Each jondoe on the route decrypts the request, changes the route identifier according to its table, and sends it to the next jondoe on the route.

Shields and Levine authored a paper on a system called Hordes [LS00] that is based on Crowds. Hordes uses the Crowds protocol for origin concealment, but uses IP multicast [Dee89] for concealing the identity of the node servicing a request. The paper states that the average time between making a request and receiving a response is less for Hordes than for Crowds because Crowds returns response messages through the jondoes whereas Hordes uses IP multicast.

Crowds and Hordes achieve origin concealment in several ways. First, the destination receives a request from a jondoe randomly chosen from those participating. This could be the jondoe associated with the origin but probably is not. Second,

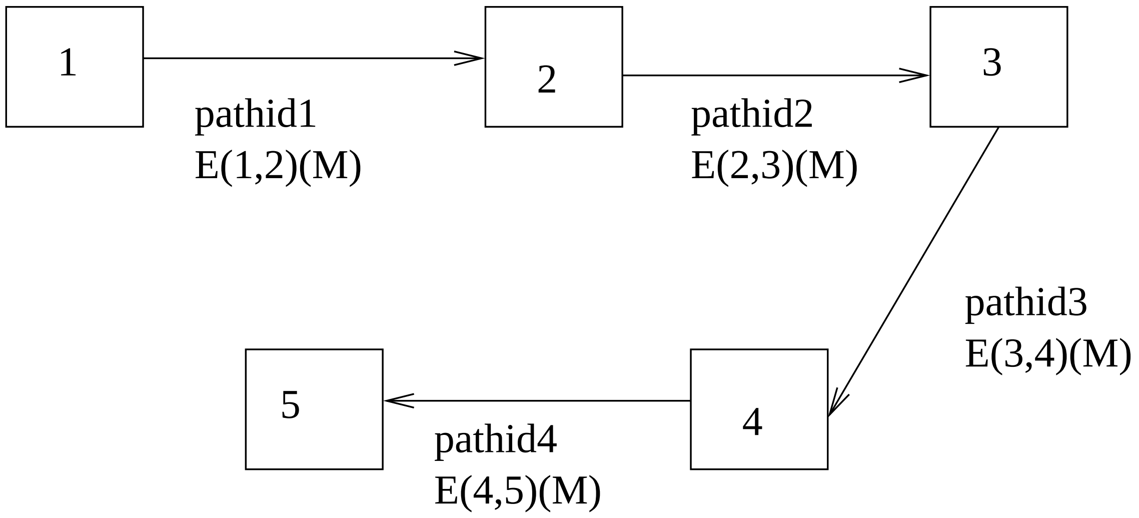

Figure 2.2. A message is sent through a route of *jondoes* in Crowds. At each jondoe, the message is decrypted and re-encrypted

messages take a random route through the jondoes with different encryption between each jondoe. Hence, an attempt to trace messages by observing the links between jondoes could not use the content of the messages to help with the trace. However, timing information could be useful as messages are not delayed beyond that needed to process the messages.

Mix-based NOCS

Chaum originally developed the concept of *mixes* for untraceable electronic mail [Cha81]. Mixes are devices that use nested encryptions to modify origin identifiers and include random reordering and delay to thwart traffic analysis. A number of mixes spread through a network are used to provide resistance to corrupt mixes.

Mixes take as input specially constructed flows called "onions[3]." An onion is constructed by first taking a message of length less than some network-wide constant and padding it out to a fixed length. Messages larger than this can be split into smaller

[3]The term onion as used here comes from work on Onion Routing [RSG95] that used mixes to conceal the origin of IP packets.

chunks for reassembly upon arrival at the destination. The sender then creates the onion by choosing a random sequence of two or more mixes through which the onion will be sent. The sender encrypts the message using the public key of the last mix in the sequence and prepends the address of the final destination for the message. The message is then encrypted with the next to last mix's public key and the key of the last mix is prepended. This process is continued until the onion is addressed to the first mix in the route and the content is encrypted with the first mix's public key.

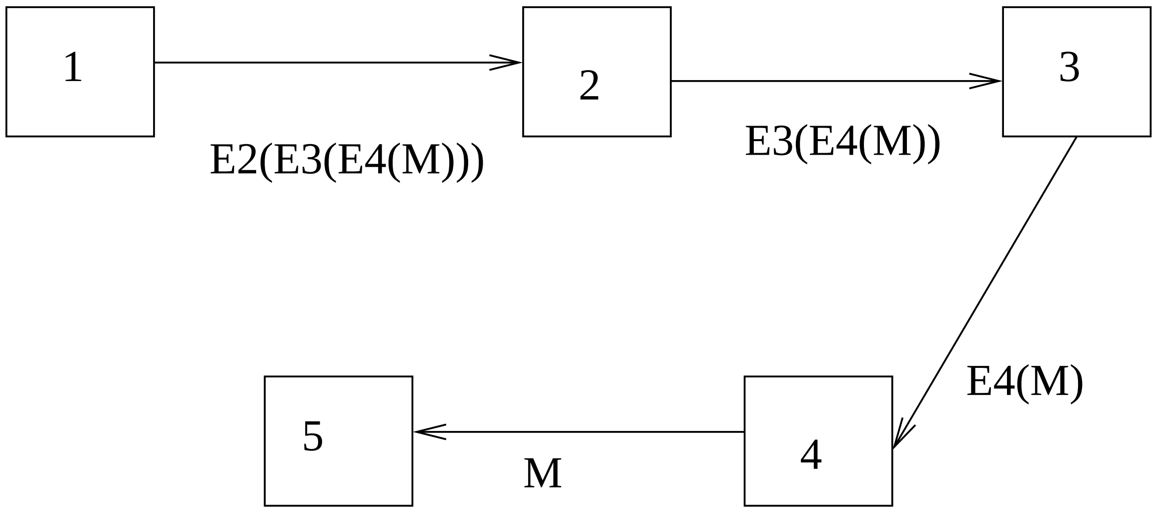

Figure 2.3. A route through a mix network is shown between nodes 1 and 5. Each mix strips off a level of encryption and forwards the result to the next mix in the route. Each step also pads its result to a constant length. Identifiers for each hop are omitted from the diagram.

The mixes process onions in a straightforward manner as shown in Figure 2.3. When a mix receives an onion the mix decrypts it and determines where the onion is to be sent next. It then waits until it has a configurable number of other messages waiting to be output, and sends them out in random order. The final mix in the route delivers the message to the recipient.

Mixes achieve origin concealment because the recipient only sees identifiers related to the last mix in the route. As the originator picks the route of mixes randomly, there should be no relationship between the mixes in the route and the origin. Because the ordering of input and its corresponding output are randomized, timing is not useful for traffic analysis. And finally, because onions are decrypted and padded to a fixed length, linking the content of the onions requires cryptanalysis of assumed strong cryptography.

Mixes have been used to anonymize IP-based application protocols in a system named onion routing [RSG95]. Mixes have also been used to anonymize electronic mail [Moe, GT96] and hide the location of mobile network stations [FKK96]. A commercial privacy service based on a variant of mixes named the Freedom Network [GS99] operated on the Internet. According to a report [Onl01], Zero Knowledge Systems stopped providing the Freedom Network service because it was not profitable.

Mixes modify all content, origin identifiers, and ordering at every hop, and the length is kept constant for all onions by padding or splitting onions. The result is that the characteristics shared between inputs of mixes and their corresponding outputs either apply to all NDEs or require breaking encryption to match inputs to outputs.

2.2.3 Dining Cryptographers

Dining cryptographers [Cha88, WP89, Jr.99] is a protocol for anonymous broadcast communication that has been widely studied. The dining cryptographers protocol provides *unlinkability* between the originator of a message and the message itself. In dining cryptographer protocols, an observer can not determine whether a message is being sent by a node.

Nodes in dining cryptographer networks are arranged in a logical ring. We number the nodes from 0 to $n-1$ such that node i is adjacent to nodes $i-1$ and $i+1$, both modulo n. The protocol proceeds in rounds where each ith node generates a uniformly random bit, b_i, that it shares privately with node $i+1(mod n)$. Each node then computes $x_i = b_i \oplus b_{i-1(modn)}$ where $\oplus$ is the exclusive-or operation. Each node then computes the following:

$$r = \bigoplus_{i=0}^{n-1} x_i = \bigoplus_{i=0}^{n-1} b_i \oplus b_{(i-1)(modn)} \bigoplus_{i=0}^{n-1} b_i \oplus b_i = 0$$

As long as all x's are reported correctly, r is 0 because each b occurs in the formula twice. Because $\oplus$ is commutative and results in 0 when applied to equal quantities, r must be 0.

If a node wishes to broadcast a 1 bit, it broadcasts $\neg x$. If an odd number of nodes sends a 1 in the same round, r will be 1. It will be 0 otherwise. For a zero bit, the correct x is broadcast. Each round can transfer a single bit to all nodes. Because many nodes may try to send at once, dining cryptographer networks use methods from broadcast LAN's to mediate access to this shared media and detect when multiple nodes are sending simultaneously [Jr.99].

Dining cryptographer networks are not NOCS. Every network data element sent in the dining cryptographer network has correct origin identifiers. It is therefore clear which node generated each NDE. Dining cryptographer networks achieve anonymity for senders by using a distributed algorithm where the recipient(s) of the message must cooperate with the sender for the protocol to work. In our definition of network origin concealment, we state that the receiver of the NDEs must be an arbitrary node of the network. Because the dining cryptographer protocol requires that the receiver of the message cooperate with other senders to achieve anonymity, it is not a NOCS by our definition.

2.2.4 Summary of NOCSs

We have described related work in origin concealment in terms of improvised and prepared approaches. NOCSs falsify origin identifiers using intermediaries or by forging them when generating traffic. Some NOCS also modify network traffic as it traverses the network so that it is difficult to trace to its origin. In addition to modifying the traffic, some NOCS delay traffic as well to prevent an observer from inferring the route of traffic based on ordering or relative timing of observations.

2.3 Network Origin Identification

In this section, we describe the past work in network origin identification. We base this survey on a taxonomy that we have developed to study network origin identification systems. By examining the past work, we demonstrate that work in NOI has been done in an ad hoc manner instead of in an organized framework. We also lay the ground work for discussing our origin identification model that we discuss in Chapter 4.

2.3.1 A Taxonomy of NOISs

Our taxonomy has the purpose of helping us name, understand, and reason about origin identification systems based on their mechanisms and where those mechanisms are implemented.

As discussed in Krsul's dissertation [Krs98], taxonomies impose a structure on its members that is explanatory and predictive. A taxonomy is explanatory if it allows one to generalize about the classes it creates. A taxonomy is predictive if it allows us to predict new elements or classes of elements. Below, we describe our taxonomy and discuss how it is both explanatory and predictive.

Our taxonomy is shown in Table 2.1. The left side of the table contains the classes of the taxonomy and the labels along the top of the table are the *taxonomic features* that are used to classify NOISs into their respective classes.

The classes of the taxonomy are defined by the values in the table. An analyst can classify a NOIS by choosing the row of the table with the taxonomic features that describe the NOIS. To make the classification objective, we describe a decision procedure for each of the taxonomic features below.

Taxonomic Features

The features of NOISs used in our taxonomy are based on where network traffic is monitored and how the traffic is affected by NOISs. We define each of the taxonomic features along with questions that allow an analyst to objectively determine the value of the feature for a given NOIS.

The first taxonomic feature is the type of monitors used by the NOIS. The feature has the values internal, external, and both. If the NOIS uses solely internal monitors, then we would choose "internal." Likewise, if the NOIS uses solely external monitors, then we would choose "external." If the NOIS uses both internal and external monitors, then we would choose "both." This feature must have one of these values as a NOIS must monitor the network somewhere to function even if it is only at the destination of the traffic.

The second taxonomic feature is route modification. This and the remainder of the features are boolean. The value of the feature is the answer to "Does the NOIS modify the route of the traffic for which an origin is sought?"

The rest of the features use the notion of marking traffic. A node marks traffic by modifying it, extending it, or creating a parallel flow to deliver information about the origin of the traffic to the destination. A parallel flow is a flow sent to the same destination as the original flow with the expectation that the flow will follow the same route to destination.

The third taxonomic feature is origin marking. The value of the feature is the answer to "Does the NOIS require a cooperative originator to mark the traffic it generates to determine the origin?"

The fourth taxonomic feature is intermediary marking. The value of the feature is the answer to "Does the NOIS require nodes on the route of the traffic between the origin and the destination to mark traffic to determine the origin?"

The fifth taxonomic feature is destination marking. The value of the feature is the answer to "Does the NOIS require the destination of the traffic to mark it to determine the origin?"

Entries with asterisks in the table represent that the value of a taxonomic feature can take any of its values and still be in the class. By limiting the values of the "monitor type used" feature, each active class can be partitioned into three subclasses depending on if it uses internal or external monitoring or both. Past work has not examined the consequences of different monitoring approaches in active systems as the active systems in past work use internal monitoring. Hence, the taxonomy indicates that mixed and external monitoring are areas for further study in active NOISs.

Classes in the Taxonomy

Our taxonomy partitions all NOISs into passive and active classes. Each of these classes is further partitioned into subclasses. The passive class and its three subclasses are shown in upper rows of the table. The active class and its subclasses are shown in the lower rows of the table.

Active NOI techniques modify flows or their routes to reduce uncertainty in their origin. In contrast, passive systems monitor and store traffic without modifying it or changing its route. Some passive systems do modify traffic but this is as a response to information about the origin not as a mechanism to determine the origin. For instance, if a piece of traffic is found to violate some policy based on its true origin, a passive system might drop the traffic [Row99].

In the passive class, we form the subclasses of internal, external, and mixed passive NOISs. These labels refer to where passive systems do their monitoring. Internal passive systems rely on monitoring the effects on the traffic of network devices that are processing the traffic. External passive NOISs monitor the network media directly instead. Mixed passive NOISs are systems that incorporate both internal and external monitoring. In Chapter 4, we develop a model of the data available to external and internal monitors and the requirements for an external passive NOIS to simulate an internal passive NOIS.

The active class is partitioned into subclasses based first on whether it modifies routes and then by where marking is done to flows. As described above, each active subclass shown in the table can be partitioned into three subclasses based on the monitor types used. Because all past work in active NOI uses internal monitors, the external and mixed subclasses may be areas for future work.

Route modifying NOISs dynamically change the route that flows take through the network. As we will describe below, changes in routes can make traffic observable by network monitors that otherwise would not observe the traffic.

Table 2.1
Our taxonomy of NOISs listed with their distinguishing features. Italicized classes have not been observed in the past work. Asterisks indicate that the feature may have any value for this class. Such features may be used to further partition the class into subclasses. Route modification with marking incorporates route modification along with marking of flows using one or more marking types.

	Monitor Types Used	Route Modification	Origin Marking	Intermediary Marking	Destination Marking
Passive					
External	Ext.	no	no	no	no
Internal	Int.	no	no	no	no
Mixed	Both	no	no	no	no
Active Flow Marking					
Origin Marking	*	no	yes	no	no
Intermediary Marking	*	no	no	yes	no
Destination Marking	*	no	no	no	yes
Multitype Marking	*	no	2 or more marking types		
Active Route Modification					
Route Modification	*	yes	no	no	no
Route Modification w/ Marking	*	yes	1 or more marking types		

Marking techniques modify flows or create flows parallel to them to determine their origin. Origin, destination, and intermediary marking refers to the entities that are required by the NOIS to mark the traffic or generate parallel flows. We list three classes where only one of each type of marking is required by NOISs in that class. Multitype marking requires two or more types of marking to function such as by origins and intermediaries. Finally, NOISs in the route modification with marking class modify routes of flows and use one or more type of marking.

2.4 Passive NOIS

As discussed above, passive NOI techniques do not modify the flows being traced, nor do they modify the routes taken by them. The situation is that one or more monitors are located in the network collecting information about the network flows that they observe. When the origin of a flow is desired, some search algorithm is applied to the collected data to trace the flow back to its origin. Table 2.2 shows the values of the taxonomic features that define the passive subclasses in the taxonomy.

We partition the passive class into external, internal, and mixed subclasses. We do so by considering where the passive monitoring is done. If all of the monitoring for the NOIS is done by observing the media directly, then the NOIS is a member of the passive external class. If the monitoring is only done inside components that are forwarding traffic (e.g. routers), the NOIS is a member of the passive internal class. Finally, if the NOIS uses internal as well as external monitors, it is a member of the passive mixed class. We include a discussion of internal and external monitoring behavior and capabilities in Chapter 4.

2.4.1 External

The three notable past works in passive external NOI are traffic thumbprinting [SC95], stepping stone detection [ZP99], and a graph-based approach by Yoda[4] and Etoh [YE00]. Each of these techniques address the problem of network attackers launching attacks using extended connections as discussed in Section 2.2.1.

[4]This Yoda is not the jedi master from *Star Wars: The Empire Strikes Back* [LBK80].

Table 2.2
The section of our taxonomy table defining passive classes.

	Monitor Types Used	Route Modification	Origin Marking	Intermediary Marking	Destination Marking
Passive					
External	Ext.	no	no	no	no
Internal	Int.	no	no	no	no
Mixed	Both	no	no	no	no

Traffic Thumbprinting

Stuart Staniford[5] developed a technique known as thumbprinting [SCH95, SC95] to determine the origin of extended connections. Thumbprinting is based on a formula that can be computed from the content of TCP connections as they pass an external monitor. The thumbprints consist of counts of certain characters in the stream during small time intervals. Staniford used principal component analysis to choose useful characters to be counted for the thumbprint.

By comparing sequences of thumbprints for pairs of TCP streams, a measure of the similarity of the two streams is computed. The assumption is that similar streams are actually different links of an extended connection. The paper describes that the technique could detect links of the same extended connection even when latency was added to the streams by connecting to remote network sites and back.

[5]Stuart Staniford has also published under the name Stuart Staniford-Chen.

A shortcoming of thumbprinting stems from it being specific to unencrypted remote login protocols. Because the method uses the content of TCP streams to compute the thumbprints, even simple link encryption defeats the method [SC95].

Stepping Stone Detection

In an attempt to trace extended connections using link encrypted remote login protocols such as secure shell (SSH) [Sec], a paper by Zhang and Paxson describes a technique to detect and match links in an extended connection despite potential link encryption. To do so, their technique matches links of an extended connection using timing of the ends of "off periods" in the stream. Off periods are those times when there is no apparent activity in the stream for some time period. The paper states that the technique could match streams that constitute links of an extended connection.

Yoda and Etoh's Work

Yoda and Etoh [YE00] approached the problem of matching links in an extended connection by defining the notion of deviation between packet streams in TCP connections. The idea is that a NOIS would use a number of passive monitors throughout a network to collect sufficient information to compute deviations between streams. The assumption is that streams with little deviation will correlate to the same extended connection.

The paper defines deviation informally as the difference between the sequence number versus time graphs of two TCP streams. A formal definition that accounts for differences in initial sequence numbers and latency-based effects on the shape of the graph is used in their experiment. Essentially, their technique creates a monotonic graph of the amount of data in the packets passing through the stream against the timestamps of the observations. To compute the deviations between two graphs, one of the graphs is stepped along the time dimension while summing the relative difference between the graphs over the time dimension. The relative difference is computed by iterating over the range of data quantities in the graph to minimize the average distance between the two graphs on the data quantity dimension. The deviation is then the minimum area between the two graphs when stepped along both of these dimensions.

The deviation technique is evaluated against several large network traces, and the paper finds that low deviations between random pairs of streams are rare. This indicates that false matches are unlikely, but no experimental evidence that the technique properly detects true matches is included. The paper does not include an evaluation of the technique for false negatives.

2.4.2 Internal

Passive internal origin identification techniques monitor network traffic using data sources inside network devices that are forwarding the traffic of interest. Examples of these devices are routers, switches, and hosts. Potentially, internal systems are more powerful than external systems as they may monitor the behavior of the network device as well as the traffic that the device receives. As the network device may connect many networks, an internal monitor can observe the traffic from the media connecting each network to the device.

In this section, we describe the past work in passive internal NOI. For each technique, we discuss the type of traffic it traces, the approach it takes, and any apparent weaknesses. We also discuss how the systems use data that is only available to an internal system. This information will be useful in Chapter 4 when we discuss the data available in the model of NOI.

Hash-based IP Traceback

Hash-based IP traceback [SPS+01] is a passive internal scheme for tracing an arbitrary IP packet based on a universal logging modification to routers in an Internet service provider's network. The paper describes a method for using Bloom filters [Blo70] to create a recognizer for recent traffic passing through the router. They do this by hashing the portions of an IP packet that are not modified during a simple routing transaction. The hash is then used to update the Bloom filter so that the packet can be recognized. Other types of transformations that occur in a router that change the assumed immutable portion of the packets are handled by a separate lookup table.

The network infrastructure for hash-based IP traceback is the source path isolation engine (SPIE). SPIE assumes that the network is broken down into a number of subnetworks each managed by a SPIE collection and reduction agent (SCAR).

When a traceback is requested, the SCAR containing the packet in question requests the Bloom filters and information necessary to interpret the filter from each of its subordinate routers. Using local topology information, the SCAR then performs a directed search of the topology, checking each router's filter for presence of the packet in question. The result of the search is a path of the packet in question through the SCAR's local neighborhood. When the search leaves the SCAR's area, the neighboring SCAR's are asked for a trace. The process repeats until no new router's Bloom filter has a match. This implies that the packet's origin is connected to the last router or the trace has failed because there was insufficient storage to store logs long enough for the trace to complete.

The paper includes simulation and analytic results indicating that the technique would require approximately 0.5% of the IP traffic capacity of the routers per unit time to store its logs. The design of the system allows for straightforward implementation in hardware as well. The paper shows that the speed of the memory for the logs is a critical limiting factor as well as the amount of storage.

SPIE has several shortcomings that result from storing information about all packets. First, the use of the Bloom filter for storage implies that the content and header of packets can not be used for flows that are not IP packets. Also, a user could generate packets that would hash to the same value yet were sent from different origins. This would make the search follow multiple paths possibly causing the search to take enough time so that the data needed to complete the search is lost. Furthermore, SPIE will require custom hardware to calculate hashes for high speed network links.

Intrusion Detection and Isolation Protocol

The intrusion detection and isolation protocol (IDIP) is designed to determine nodes on the path between the origin of a network attack and its target [Row99, SDS00]. IDIP coordinates responses to network attacks in a distributed system of network components. IDIP includes NOI functionality to respond to detected network attacks as close to the source of the attack as possible.

IDIP components log information about network traffic and attacks as the traffic passes through them. IDIP-enabled components include firewalls, border routers, and intrusion detection systems. Because the references give little detail about what is logged and what types of attacks are detected, it is difficult to determine the capabilities of IDIP.

IDIP uses the Common Intrusion Specification Language to describe events to be traced [SDS00]. When an IDIP component detects an intrusion that meets its criteria for tracing, the component generates a description of the attack and sends it to its neighboring IDIP components. Each of these consults its log to determine if the signature matches and if so, sends the description to its neighbors except for the neighbor that made the request.

If the IDIP component transformed the traffic while forwarding it, the IDIP component may amend the signature so that the upstream traffic will still match the signature. The results of these queries to IDIP components are sent to a special component called the discovery coordinator. The discovery coordinator assembles results of the trace requests into paths and directives to IDIP components to block attacks near the origin.

Providing Process Origin Information

Providing process origin information finds the origin of network traffic that has caused a process on a host to be created [BS02]. The paper describes a modification to the FreeBSD [BSD] version of the Unix kernel that keeps track of a TCP or UDP

4-tuple[6] associated with the network session that creates the process if it was started as the result of a remote login. The system also maintains logs of outgoing traffic associating the flows with the process that sent the traffic and the origin stored for that process.

A host's administrator can use the logs to determine the host's precursor in an extended connection after the fact by examining the logs. Another application of the log is finding the controller of a Trinoo [IN-99] distributed denial of service (DDOS) client.

The providing process origin information technique relies on modifications to FreeBSD system calls to associate origin information with processes. The system uses the heuristic that when `setlogin` is called, the origin information from the last completed `accept` call by the process's nearest ancestor is stored as the origin of the process. This information is then inherited by the process's descendents during the `fork` call. This technique works as long as there is one open network connection for the process calling `setlogin`. This heuristic works for Unix-style login services [Ste98] that fork a new process for each request satisfy. It is possible to create remote login services that do not operate this way, though. Also, it is possible for a remote user to use system mechanisms such as `cron` to run processes that will appear in the logs to have been run locally although a remote user inserted the cron entry.

Recursive TCP Session Token Protocol

Carrier describes a similar technique to the providing process origin information approach discussed above, but Carrier's technique is less invasive as it does not require modifications to the kernel. Instead, a session token protocol (STOP) daemon runs outside the kernel to traverse a system's process tree and network data structures. The result is an augmentation of the `ident` protocol [Joh93].

A STOP daemon listens for `ident` requests, walks the process tree for the requested TCP connection while storing process data to a log, and returns a user token

[6]The 4-tuple consists of the IP source and destination address as well as the TCP or UDP source and destination ports.

that is a SHA [Sch96] hash of the stored information. An administrator of a remote site can turn in a token to the local administrator for access to the stored process tree. The information stored is a representation of the process tree with process names, open sockets, and other processes that are connected to processes in the tree via pipes.

A STOP daemon can also issue recursive queries to other host's `ident` or STOP daemons. The local logs would then include the tokens from those remote daemons whose hosts have connections to the local process subtree.

STOP has similar shortcomings to the process origin work of Buchholz [BS02]. Although no kernel modifications are required, the STOP daemon requires a component to walk the process tree of the operating system. As the interface to this information differs among Unix variants, different operating systems may require substantially different versions of the STOP daemon. Also, processes run from `cron` will be reported as run by a local user although a remote user may have scheduled the process. Finally, an attacker could disable the STOP daemon or subsitute a compromised version of the STOP daemon to feed incorrect information to the next host in an extended connection.

Caller Identification System in the Internet Environment

The Caller Identification System in the Internet Environment(CISIE) is a passive internal NOIS [JKS$^+$93]. We present a further analysis of CISIE in [BDKS00]. As in the recursive TCP Session Token Protocol, CISIE was designed to augment the `ident` protocol to report the origin of extended connections. In CISIE, all hosts in some administrative domain run a CISIE daemon. When a login is attempted from host A to host B, B contacts A to receive further identification information for the user logging in from host A. This information consists of a list of user names and hosts that make up the extended connection (if any) that the user has used to log into A. The result is that a host in the chain will have a trace of the last consecutive CISIE enabled hosts of the extended connection.

Our past analysis of CISIE [BDKS00] indicates that a CISIE-enabled host could be compromised allowing an attacker to substitute a false path for the path prior to the compromised host. This is the result of a faulty authentication mechanism specified in the paper. Furthermore, the paper does not indicate the mechanism for linking the identity of an outgoing connection with the incoming connection.

DoSTracker

DoSTracker [CD97] is a Perl script that was written for MCI and released to the public. It finds the source of a packet flow that makes up a flooding denial of service attack by logging into the administration systems of Cisco routers. DoSTracker requires administrative access to all routers on the path of the attack.

Given a subnet mask for the victim of an attack and the address of the edge router that is routing the flow, DoSTracker logs into the edge router. It then instructs the router to display debugging information for all packets destined for the victim subnet and tries to determine if the source address of each packet is forged. If the user provided a forged source address as input, any packet with that source address would be considered suspect, otherwise the router is queried to determine if the source address exists in its routing table and if not, the packet is suspect. Once a suspect packet is detected, DoSTracker determines all routers on the incoming interface for the packet, logs off of the current router, and then begins the process again for each of the determined routers. If more than one router is directly accessible by that interface, as possible in FDDI networks, the process occurs in parallel until one of the routers observe a suspect packet. Manual passive internal techniques similar to that used by DoSTracker have been described in Cisco technical documents [CIS].

DoSTracker has several shortcomings. It can only trace floods that are occuring during the trace. It also makes the assumption that the flood traffic is similar enough that the signature used at each step of the trace will match. A savvy attacker could construct the flood from different packets, including using different forged source addresses on each packet. In this case, the trace would fail. DoSTracker also relies on features specific to one brand of router that may not exist in other router models or brands. Hence, it is only useful in environments where relatively constrained type of router is used.

MBIT

Ohta et. al. give a high-level description of a technique for tracing denial of service network attacks [OMT$^+$00, MOT$^+$99]. The technique relies on what they describe as "RMON-like" probes for collecting data from routers. The paper suggests that by keeping packet counts of certain types of flows through routers that they propose to trace various types of network floods including ping and SYN floods [SKK$^+$97, CA-96].

Neither of the papers describing the work include details of the data collected, but one indicates that the correlation coefficient can be used to correlate flows of packets between links using the data from the probes mentioned earlier [MOT$^+$99]. The approach is to maintain time series data for different types of packets observed by the probes. The paper includes an example showing that the correlation coefficient on these time series can be used to correlate ICMP echo requests and echo replies resulting from a smurf [CA-98] attack between two links.

Because of the high-level nature of the papers, it is not possible to present a thorough description of the approach. However, some shortcomings of the technique can be considered. Because the technique relies on correlation coefficients, it is not useful for correlating single packet flows. Additionally, we do not know the number of different types of data collection probes needed. It is possible that as new attacks are detected, new probes need to be added, possibly requiring modifications to all routers in the network.

Distributed Recognition and Accountability

The distributed recognition and accountability (DRA) algorithm is an intrusion detection approach to tracing extended connections [KFTG$^+$93, KFH$^+$93]. The authors propose a system where hosts in the network send audit records detailing connection start, connection accept, session start, failed login, connection end, session end, and unusual local host activity. When applicable, these records include timestamps, source and destination identifiers, user identifiers, and user activity. This data is then stored at a central location for analysis.

From this data, the paper includes a proof that the DRA can assemble the reported data into the path of a user's extended connection under strong assumptions about the reported data. One shortcoming of the DRA approach is its requirement that all ways that users can change their user id are logged by the system. Attacks such as buffer overflows against hosts [KST98] allow users to change their user identifiers in ways that are not designed into systems and so these transitions are not necessarily logged. Furthermore, the DRA approach relies on each simultaneous user on a host having a unique user identifier. It may not be possible for DRA to distinguish between a legitimate user's connections and an attacker's connections if they are both using the same user identifier.

2.4.3 Mixed

None of the past work in passive NOI has addressed the issues of mixing internal and external observers in one NOIS. Few of the current proposals for Passive NOISs have been implemented in actual networks, and therefore, cases where internal monitoring are not possible have not been encountered.

The existence of the mixed class in the taxonomy indicates that the taxonomy is predictive. Because the taxonomy motivated us to consider the case where internal and external monitors are used together, we have discovered a new problem area in NOISs. As a result, we discuss a way to model a NOIS with combined passive internal and external monitoring devices in Chapter 4.

2.5 Active NOIS

Active NOISs modify flows or their paths through the network to determine their origin. Instead of directly marking flows, some past work has suggested that intermediaries create flows "parallel" to those being traced to provide origin identification information to the receiver [Bel00]. Parallel flows are flows that are output by the same node forwarding the flow with the same destination as the flow being monitored. It is assumed that given stable routing, the two flows will follow the same path to the destination. NOISs that generate parallel flows are similar enough to NOISs that mark flows directly that we consider parallel flows used in the context of origin identification to be a marking technique.

2.5.1 Flow Marking

Flow marking techniques change flows or create flows parallel to them to reduce uncertainty in their origin to some monitoring agent. For brevity, we will refer to the changes or extensions made to flows as *marks* and use the word *mark* as a verb to refer to modifying or extending the flows. In the past work, the monitoring agent has been the receiver of the flow, but it is not necessarily so. We partition the flow marking class into four subclasses: *origin marking, intermediary marking, destination marking, and multitype marking.* The flow marking section of the taxonomy table is shown in Table 2.3.

Table 2.3
The section of our taxonomy table related to flow marking NOISs.

	Monitor Types Used	Route Modification	Origin Marking	Intermediary Marking	Destination Marking
Active Flow Marking					
Origin Marking	*	no	yes	no	no
Intermediary Marking	*	no	no	yes	no
Destination Marking	*	no	no	no	yes
Multitype Marking	*	no	2 or more marking types		

Origin Marking

Origin marking systems for origin identification are those that require a cooperative originator to mark the flow such that the mark decreases the monitoring agent's uncertainty in the origin of the flow. If the NOIS also marks flows at entities other than the origin, then the NOIS belongs to the multitype marking class. As can be seen in Table 2.1, the origin marking class includes those systems using internal, external, or both types of network monitors to detect the marks.

Origin marking schemes are the most studied form of NOI [DH98, Sch96, Sta99, Opp96]. Numerous techniques with marks based on cryptography and shared secrets have been suggested and analyzed in the past.

A shortcoming of these techniques is that they require a cooperative, accurate originator to be effective. An uncooperative originator may choose to omit or use incorrect marks. In this case, the traffic is delivered, but the marks may increase the receiver's uncertainty in the origin. Furthermore, the delivered traffic can be used to flood the destination thereby constituting a denial of service attack.

Accurate origin marking by cooperative originators is the field of origin authentication in cryptography. Origin authentication is a well understood and solved problem in networks [DH98, Sch97, Sch96].

A cooperative yet inaccurate originator may erroneously mark flows. This could result from a misconfiguration or error in the origin system. Erroneous marks may still be useful for determining information about the origin, but this would depend on the types of errors made. For example, if the flows are marked independently of the origin, then they would be of no use for directly determining the origin. In contrast, if the flows are marked as coming from nodes in the same subnetwork as the origin, then the marks are not completely accurate, but they still give us information about the origin.

2.5.2 Intermediary Marking

Intermediary marking schemes mark flows on the path between the origin and destination of the flow. The monitoring agent can analyze the marks to determine the path that the flow took through the network.

Probabilistic Packet Marking

Researchers have proposed a number of active intermediary schemes to trace IP flooding attacks to their origin. Savage et. al. [SWKA00a, SWKA00b] introduced techniques we call probabilistic packet marking (PPM). PPM techniques randomly select packets as they are forwarded by routers and modify portions of the IP packet headers with information about the current edge in the path. The receiver of a number of marked packets analyzes the marks on the packets in the flood to assemble a path back to the origin of the flood.

Savage et. al.'s approach to PPM modifies the 16 bit IP identification field with a portion of the router's address interleaved with its 32 bit hash, an offset representing which portion is being embedded, and a hop counter that is incremented by the PPM processing at each hop[7]. When a PPM router sees a 0 in the hop counter, it XOR's a portion of its address-hash combination into the current mark. Each router in the path then increments the hop counter until delivery.

The Savage scheme is optimized for implementation in hardware and therefore a significant proportion of the packets forwarded could be marked. Hence, the probability that a packet will be marked is recommended to be 4%. Given this probability, the paper presents a simulation showing that a 4000 packet flood from 30 hops away can be traced to the first PPM router in the path greater than 95% of the time.

Others have modified the Savage scheme by changing how the packet is marked. Song and Perrig [SP01, SP00] proposed several marking schemes for IP packets that rely on being able to determine a network map when tracing an attack. Because their marking scheme, which uses the IP identification field as well, relies on being able to determine a map of the network, routers in the scheme can store hashes of their

[7]Savage's scheme uses a separate hop counter from the IP TTL field.

addresses instead of full edge information. The paper states that the computational cost of reconstructing the path is asymptotically the same as in the Savage scheme. The paper also describes a simulation indicating that the technique is able to determine the sources of a distributed denial of service attack from 1500 different sites. Additionally, the papers introduce an authenticated marking scheme that relies on hash-based message authentication codes (HMAC's) [Sch96] and hash chain commitments to keys. By replacing their uniform hash functions with HMAC's, the marks can be authenticated

Dean et. al. present another PPM technique that uses algebraic coding theory to generate the marks [DFS01]. Routers generate the marks by computing points on a polynomial modulo a prime p over p's Galois field. The points are evaluations of the polynomial where the IP addresses of the router are the coefficients. More specifically, Dean's technique codes router IP addresses by having each router multiply and add to the current mark. The mark is embedded in the header in fields similar to that of Savage et. al. The marks are interpreted using algorithms for reconstructing polynomials of degree d given $(d+1)$ points to reconstruct the path.

The initial paper [DFS01] states that the algebraic coding technique should resist false paths injected by the attacker so it was found unnecessary to use a distance field as in previous PPM schemes [SP01, SP00, SWKA00a, SWKA00b]. In later work [DFS02], it was reported that they could not find an algebraic approach that "is more efficient than Savage et. al." Additionally, the technique was shown to have unacceptable false positive rates.

Park and Lee [PL01a, PL01b] introduced a minimax analysis applicable to all PPM schemes. The paper's analysis showed that PPM could reduce a victim's uncertainty in the source of a single source flood to about five different edge networks. In a DDOS scenario, their analysis indicates that a coordinated attack could amplify the uncertainty in the edge networks participating in the attack beyond a measure of how distributed the attack is. The general technique they suggest for confusing a PPM system is to determine a weak node in the path of the attack and set the identification fields of all sent packets to appear to be marked by routers upstream from this weak point [PL01a]. The weak point is a router with a marking probability less than some constant determined in their paper. The papers report that PPM is unlikely to be effective for tracing widely distributed DDOS attacks that are inserting false marks on the packets in a coordinated way.

Itrace and Related Schemes

Researchers have suggested a scheme similar to PPM named Itrace[8] that uses parallel packets for identification information [Bel00]. Itrace-enabled routers would randomly choose packets as they forward them, but instead of modifying them, the router generates an ICMP traceback message to be sent on to the destination of the packet. ICMP traceback messages contain identifiers for the link from which the router received the packet, the router doing the marking, and the link to which the packet is forwarded. ICMP traceback messages also contain a timestamp, as much of the traced packet as possible, and an authentication field.

Because Itrace generates additional traffic, a configurable marking probability between 0.1% and 0.005% is suggested by the document. As noted in the document, a marking probability of 0.005% results in less than 0.1% increase in traffic to the receiver.

Several options for authentication mechanisms for Itrace are considered, but it is unknown if any are necessary. The document requires all Itrace messages to be sent with their time to live (TTL) fields set to 255. Because each router decrements the TTL, the receiver of an Itrace message will know the number of hops to the router that generated the message assuming the Itrace message took the same path as the packet in question. This makes it difficult for an attacker to generate convincing Itrace messages for routers closer to the receiver than the attacker because it is assumed that the attacker does not control other nodes along the path of the attack. Note

[8]As of July 2002, Itrace is a draft standard and may yet change as it is refined by the Internet Engineering Task Force.

that this does not prevent the attacker from generating false paths that come before the first Itrace-enabled router on the packet's path.

During a flood, the destination would begin the assembly of itrace messages into a path by examining the content of each itrace message to determine if the message corresponds to traffic in the flood. This is possible because the itrace message contains as much of the packet that caused it to be generated as possible.

The relevant itrace messages are then sorted by their TTL values. By comparing itrace messages with TTL's differing by one hop, the destination can assemble messages with equal edge identifiers to build a route back towards the origin of the flood as far as the first itrace router on the route. This may be more difficult in the case of simultaneous floods from distinct origins as messages with equal TTL's may result from unrelated floods.

Mankin et. al. [MMW$^+$01, WZaM00] suggests a refinement to Itrace that allows autonomous systems to declare their desire for Itrace messages through border gateway protocol (BGP) routing updates. The approach is that an autonomous system issues a BGP update containing a code that indicates that an address range desires to receive Itrace messages. Routers will only generate Itrace messages for those destinations that have requested them. Routers still generate Itrace messages with the same probability, but only towards those address ranges that have requested them. In the simulations, the use of the intention bit increased the number of Itrace messages received by a victim of an attack that were useful for tracing the attack by at least 100 times.

Another addition to Itrace that the paper suggests is that routers should more frequently mark traffic that has a destination further from the current router. Using a value metric for Itrace messages of her own devising, simulations in the paper indicate that using the intention mechanism and more frequently generating messages for those packets further from their destinations leads to about 40% more Itrace messages considered "valuable." The paper points out that the value metric is arbitrary, and does not indicate exactly how marking selections are skewed towards greater distances.

Link Testing

An intermediary marking scheme proposed by Burch and Cheswick [BC00] marks an ongoing packet flood by causing intermediaries to drop packets of the flood. The mark in this case is a decrease in the data rate of the flood at the receiver. As the paper does not name this technique, we give it the name, *link testing*.

Link testing is proposed as a computer-assisted technique instead of a completely automated approach because the technique floods the network infrastructure with traffic, and therefore humans maintain direct control over these attacks. In this way, the operator can attempt to prevent harmful floods.

Link testing starts out by previously assembling a network map from the target of the attack to all reachable edge networks. Additionally, the link testing system must find a set of amplifying reflectors such that at least one amplifying reflector has a forward route through each link to the target to be tested. An amplifying reflector is a reflector that for every byte of input traffic outputs $k > 1$ bytes of output traffic. The authors suggest sending a broadcast UDP chargen request to generate the traffic–essentially, a variant of the smurf [CA-98] attack.

The system then presents a user at the target with the local network map and the amount of traffic being received in the flood. The user may pick a link from the map to test for carrying the flood. The system uses amplifying reflectors to flood the router nearest the target on the link to be tested. They do this by spoofing the reflector input with the router's inbound link address so that the resulting floods increase traffic on the link. While doing this, the user looks for a decrease in the amount of flood traffic at the target. If one is observed, the link is assumed to carry the flood and the user discontinues the reflector flood and continues his search upstream in the map.

The experiments in the paper demonstrate that link testing works on Lucent's intranet. They were usually able to trace floods to the building containing the origin host and often back to the origin's Ethernet network. No quantitative analysis or controlled experiments were reported. A shortcoming of link testing is that the utility of the approach is diminished if an attacker knows the technique is in use. A savvy attacker can use many hosts as in a DDOS attack or can vary the flood's data rate in time to make it more difficult to discern when the appropriate link is being tested. Additionally, link testing causes spikes in load that might interfere with other network services.

2.5.3 Destination Marking

Destination marking schemes mark flows at the destination so that the marks follow a return path to the origin of the flow. This requires the flow to use two way communication. One or more monitoring agents then look for the marks in an attempt to determine a portion of the path back to the origin.

We have developed a destination marking scheme that we called subliminal traceroute [DS00b]. Subliminal traceroute embeds a traceroute-like protocol [KS94] into an active TCP session to determine the path that the return packets are taking. Using subliminal traceroute, we could trace a a straightforward man-in-the-middle[9] TCP spoofing attack. This attack relies on a compromised intermediary on the connection's route. The intermediary is used to redirect packets from those destined for the claimed source to the spoofer. The technique has the shortcoming that a savvy attacker could detect marked packets and emulate the network so that the trace appeared to lead to the claimed source instead of the spoofer. Most of our paper on subliminal traceroute is included in the appendix on page 145.

Subliminal traceroute was implemented by modifying a TCP implementation to change the IP time to live (TTL) values on packets immediately after TCP three-way handshake. Governed by a finite state machine, the first packet modified is set to a TTL of 1 and successive packets are marked with the next greater TTL value until the system's maximum TTL is reached. The result is that the target collects ICMP time exceeded messages for each packet with a TTL of less than the distance to the

[9]Man-in-the-middle TCP spoofing is done by monitoring a TCP connection on the route between the source and destination to determine the TCP sequence numbers necessary to impersonate one of the endpoints in the connection.

source of the flow assuming no filtering of the ICMP messages and route stability. These can then be assembled into a trace to the source of the TCP session.

An active destination marking scheme has been developed for tracing an extended connection while it is in progress [WRWY01]. The technique is called *sleepy watermarking* because it injects a watermark into the return path of the extended connection at the destination. The sleepy component is that the monitoring components in the network are inactive until activated by observing the watermark.

Sleepy watermarking relies on an active network and an intrusion detection system that detects intrusions as they occur. When an intrusion via a remote login protocol is detected, the target injects a watermark into the return path of the extended connection. The watermarks suggested are what the authors call null strings. Null strings are those that will pass through links of the extended connection over the used login protocol but will not be observed by the user at the end of the extended connection. An example would be a number of spaces followed by an equal number of backspaces. The watermarks are randomly chosen with the goal of reducing the probability of colliding with traffic not containing the watermark.

When the target begins watermarking a connection, it sends a notice to its active network gateway to begin looking for its watermark. The gateway activates and begins examining TCP connections for the watermark. The gateway can then activate the upstream gateway from which the connection appears to be coming for more trace information. The process repeats until a gateway only detects the watermark entering the network and not exiting it. After a configurable amount of time, the gateway stops monitoring and deactivates.

A shortcoming of sleepy watermarking is that an attacker could detect the watermark. If the attacker can determine the watermark before it is embedded, he could embed it in other traffic to subvert the analysis. As with other techniques, sleepy watermarking is vulnerable to encrypted remote login protocols as the watermarks will not be observable. It also uses modifications to the remote login services to insert watermarks into connections. Furthermore, it requires trust relationships between the active network gateways to allow trace request to execute.

Multitype Marking

We have discussed three marking types: origin, intermediary, and destination marking. Multitype marking NOISs require two or more of these marking types to operate. We know of no past work that requires more than one of these types of marking. Some intermediary marking schemes have allowed for the origin to mark as well, but as the origin may be compromised and hence make incorrect markings, origin marking has not been required by these systems. The possibilities, origin-intermediary, origin-destination, intermediary-destination, and origin-intermediary-destination marking have not appeared in past work perhaps because only two instances of past work have examined destination marking alone.

2.5.4 Route Modification

In this section, we discuss two active route modification NOISs. Both of them create tunnels to redirect traffic over a trusted path to an observation point. The two classes that do route modification are shown in Table 2.4.

Centertrack [Sto99] is a proposal for a route-modifying NOIS for tracing ongoing attacks against an edge network by an Internet service provider(ISP). The scenario is that a customer at an edge network would report to an ISP that an attack is in progress against it. The ISP would then instruct all of its edge routers to send all traffic bound for the customer network to a central router where it would be sent to the customer. The central router can then be queried to determine from which edge router the attack is coming. If that edge router attaches to a customer's network, then it is assumed to be the origin. If the edge router attaches to another ISP, the neighboring ISP could repeat the process if it implements Centertrack. If the attack traffic is not detected at the central router, it indicates that the attack is from inside the ISP itself as it does not cross an edge router to enter the network.

Centertrack uses IP–over–IP tunneling [KA98] that is already implemented by the edge routers to send traffic from the edges to the central router and back to the edges.

Table 2.4
The portion of our taxonomy table related to route modifying NOISs. We include route modification with marking in the active route modification class because we consider route modification to be a more unusual feature than the marking features.

	Monitor Types Used	Route Modification	Origin Marking	Intermediary Marking	Destination Marking
Active Route Modification					
Route Modification	*	yes	no	no	no
Route Modification w/ Marking	*	yes	1 or more marking types		

IP–over–IP tunneling creates a new IP datagram for every datagram to be tunneled. The datagram to be tunneled is put into the content of the new datagram which is addressed to the end point of the tunnel.

Centertrack does not store information on traffic before a trace request is received. It therefore can not trace flows that have completed. Additionally, Centertrack can not trace attacks that originate in or use extended connections through non-edge networks. Hence, if an attacker compromises an internal router in an ISP from which he launches an attack, Centertrack would report that the attack is originating inside the ISP.

Deciduous [CNW^{+}99] is an active, route modifying NOIS for ongoing network attacks. Deciduous is based on binary search of a network represented as a linear graph with the target at one end of the graph. In a linear network, Deciduous chooses the midpoint router, R, and sets up an IPSEC security association on it so that traffic bound for the target is carried over an IPSEC tunnel. The target then determines whether the attack is now arriving over the tunnel or not. If the attack is arriving over the tunnel, then the attack is known to originate in entities connected to R or routers further away from the target (by hops). If the attack does not arrive over the tunnel, then the attack is known to come from routers nearer the target than R. In either case, a new midpoint router is chosen and a binary search commences. Deciduous makes the assumption that traffic takes the shortest path (in hops) through the network.

Based on the linear network approach and the assumption that traffic follows the shortest path, the authors present a method to convert any network topology into a virtually linear topology for the purposes of applying the Deciduous approach. The authors do a breadth-first search of the network to create sets of nodes i hops along their shortest path from the target. When an attack is detected, Deciduous creates tunnels with routers $d/2$ distance from the target where d is the diameter of the network found in the breadth first search. Depending on whether the attack comes from the tunnels or not, the next round of tunnels would be set at $d/4$ or $3d/4$ and so on.

Deciduous makes a strong assumption about routing behavior that may be violated by asymmetric and policy-based routing [Pax97]. The paper states Deciduous can isolate the origin of an attack under their assumptions in $\log d$ rounds. The paper does not present a bound on the number of tunnels necessary during any one round of the search, so it is possible that many tunnels may need to be created during a round to proceed.

Route Modification with Marking

The combination of route modification and marking techniques have not been studied in the past work. By looking at Table 2.4, we can see that there are seven possible combinations of monitoring types. Each combination would specify a subclass of route modification with marking. Furthermore, each of these subclasses could be partitioned according to the monitor type used into three subclasses. This results in 21 subclasses of route modification with marking that can be defined using our taxonomy.

One scenario where route modification and marking could be combined would be the deployment of two coexisting NOCSs. An intermediary flow marking NOCS could be implemented in some subset of the network nodes while some subgraph of the network implements Centertrack. The marked traffic would traverse the virtual star of Centertrack and be output with the marks in place. The marks would have no effect on the routing behavior or correlation done by Centertrack. Hence, Centertrack would report the entry point of the flow into the subgraph, and marks from nodes upstream from the subgraph would be visible at the destination. Furthermore, the Centertrack system could observe the marks and give the administrative unit on the path upstream the information from the marks to assist in their response to the traffic.

2.5.5 Summary of NOI Work

We have presented our taxonomy of network origin identification systems and shown how the past work in NOI fits into it. We have described a decision process for classifying members of the taxonomy and described elements in the classes. The taxonomy is both explanatory and predictive.

Explanatory

The taxonomy is explanatory as we can make some general statements about the classes and their members. First, active NOISs can be considered an extension to passive systems. Second, NOISs can be studied based on what components of the network implement NOI functionality.

Active NOISs can be viewed as passive systems augmented to modify traffic. An active system marks the traffic in some way, but this does not assist in origin identification unless the marks are observed. Hence, as a first step to modeling origin identification techniques, we model passive techniques in Chapter 4, as some results may apply to active NOISs as well.

By partitioning both active and passive classes based on the nodes of network that implement origin identification functions, we can make general statements about classes of NOISs. Passive internal NOISs require monitoring systems placed in network nodes. Active origin marking systems require a cooperative originator to mark traffic properly. Destination marking NOISs require a return flow to the origin so that the marks can be observed as they return to the origin. Further general statements about passive NOISs will be made in Chapter 4 when we describe a reference model for passive origin identification.

Predictive

The taxonomy is predictive as it has assisted us in motivating new work in this dissertation. Comparing the active class with the passive class has motivated our work in Chapter 5. The lack of past work in passive mixed NOI motivates us to investigate edge observed networks in Chapter 4. When the taxonomy was originally developed, it motivated our work on subliminal traceroute as no destination marking techniques had been described at that time. Finally, there are many ways that the classes of the taxonomy can be further partitioned. These new subclasses can be treated as predictions of future work in NOI.

The active class contains NOISs that change the routes of the traffic it is tracing. One question is: does route modification have any applications in passive systems? We answer this question in Chapter 5 where we develop algorithms that use monitors in constrained network topologies. Because a passive system cannot dynamically modify routes, we constrain the network in advance to limit the types of routes available.

Another prediction made from the taxonomy is based on the lack of NOISs in the mixed passive class. This motivates some of our work in modeling passive origin identification. We develop a partitioning of the network based on the edges observed by both internal and external monitors to develop an approach to using both types of monitors.

As shown in Table 2.1, the taxonomy has three passive classes. One of these does not contain past work. This may motivate future work in passive mixed approaches.

The active flow marking class has four subclasses shown, and one of these does not contain past work. By considering the types of monitor used and two or more marking types, the flow marking class could be broken down further into 18 classes only three of which contain past work. The result is 15 new areas of possible further work in active flow marking NOI. If we disregard the monitor type for these classes, we still have three new classes of flow marking to consider all in the multitype marking class.

The active route modification class has two classes shown with only one of those containing past work. The route modification with marking class can be partitioned further into seven classes based on multiple marking types. All of these subclasses could be further partitioned based on monitor type. Again, we have many areas for future work in these new subclasses.

2.6 Other Related Work

Schneider and Sidiropoulos create a formal definition of anonymity using CSP, a formal language for defining entities, processes, and events [SS96]. The paper applies their definition via CSP to several origin concealment systems as well as dining cryptographers to show that the protocols can be proven anonymous or not.

This work concentrates on strong anonymity from the perspective of an all-powerful eavesdropper and is therefore too specific for our purposes. While the definition does yield several analysis techniques, it relies on the analyst to translate the behavior of the systems into formal logical statements to apply the model. One of the goals of our reference model of origin concealment is to provide a description of techniques used in origin concealment scenarios, and how they affect network traffic.

In [SS99], Syverson and Stubblebine present a formalization of anonymity based on group principals. The group principal concept is used to formalize knowledge by certain subgroups of principals and apply this to reasoning about anonymity systems. The paper shows part of their approach applies to a simple anonymizing web proxy. The paper states that their approach is best suited for model checking systems. As with the CSP approach the paper provides a model for proving formal truths about systems, but one that does not describe the mechanisms involved in the systems.

Both of these approaches are formal models for discussing anonymity systems based on logic suitable for theorem proving environments. As both papers suggest, it is a substantial effort to encode the behavior of the system into logic statements suitable for the frameworks. While capable of proving certain facts about these systems, they are not useful as a reference model, as they do not describe the mechanisms at work. Instead, they rely on the analyst to describe the mechanisms of systems in a formal language so that reasoning can proceed. Our reference model of origin concealment captures the functionality of NOCS and how they affect the observable features of traffic.

2.7 Summary of Related Work

This section discusses origin identity in network protocols, the techniques used to conceal that origin identity, and systems that determine that origin identity.

We choose to discuss origin identity in terms of network layer identifiers. We make this choice because it is the lowest layer where network-wide communication is done. Furthermore, the network layer is the highest layer that is required to be independent of the identifiers used at layers below it.

We have described past work in prepared and improvised origin concealment. These systems function by modifying traffic so that there is uncertainty in its origin. To do this, origin identifiers are modified or removed. Additionally, NOCS may modify content and timing of traffic to thwart traffic analysis.

We have developed the first taxonomy of NOISs based on an objective decision procedure for classifying its members. The taxonomy is explanatory and predictive as it motivates some of the work in the rest of this dissertation. We have also described past work in NOI in terms of the taxonomy. The taxonomy can be partitioned into 48 subclasses with 42 of the classes currently containing no past work.

Finally, we have presented two past works in modeling anonymity systems. Both works are useful for reasoning about origin concealment, but do not form reference models as they require the analyst to encode the behavior of the functions in logic statements instead of directly analyzing transformations on traffic.

3. A Reference Model of Network Origin Concealment

In this Chapter, we develop a reference model of network origin concealment. We describe the features of the reference model and then build the model in stages. We discuss the transformations network entities apply to NDEs and how these achieve origin concealment goals to demonstrate that the model is useful for reasoning about NOC mechanisms.

3.0.1 Reference Models

A reference model for a class of mechanisms is a construct that defines the class by describing the members' properties in some structured way. A reference model defines the components of the class's members and how the components interact. The model should be useful for reasoning about members of the class and the class as a whole. We base this definition on an intersection of the descriptions of several reference models [Sch97, EHM96, Zim80].

Reference models have been developed to better understand a number of areas of networking and security including firewalls [Sch97], general computer network communication [Zim80], and programming language security issues [EHM96]. Jensen [Jen96] notes that one of modeling's most useful features is to aid in education as a reference model allows one to ignore details of individual systems for understanding and analysis.

The main benefits of a reference model of NOC are that it assists understanding of

- the components that make up a NOCS,
- the interactions among those components, and
- how those interactions lead to increased uncertainty in the origin of traffic.

Network functionality in nodes has been modeled as layers [Zim80] and graphs [PD96] of protocols. Both of these approaches focus on how different functionalities inter-operate to implement protocols in network devices. We take a different approach by modeling a number of different transforms on the network traffic and how these transforms can be composed to model how these transforms reduce a receiver's confidence in the origin of traffic.

3.1 Network Data Elements

A network data element (NDE) is a bounded length string of characters generated by a network node and communicated from node-to-node over the edges of a network. We use the term NDE so that the model remains generic enough to cover various networking approaches such as IP packets, ATM cells, and onions [RSG95].

NDEs have four types of directly observable characteristics:

- Content,
- Identifiers and metadata,
- Observation time, and
- Media qualities.

A network's purpose is to transfer content between nodes. We can observe the content of NDEs and use this content for some analysis of NOCSs. However, because we are interested in analysis of NOCSs that can be automated, we only consider the structural nature of the content.

Identifiers and metadata could be modeled as one entity, but we separate them from content so that we can more easily model transformations that only modify one or the other. For example, a node may modify the header but not the content of an NDE as it forwards it. Similarly, a node may modify the content without changing the header.

NDEs include identifiers that indicate origin and destination nodes, membership in a set of NDEs, or routing information. In some cases, and as we assume for the rest of this work, the identifiers for the origin may be false or nonexistent.

Observation timing refers to when observations of NDEs are made. If the observation is made by a node, then the observation time may be the time of receipt or output of the NDE by a node. If the observation of the NDE is made on a network link, then the observation times may be the beginning or ending times of the observation.

Media qualities are measurements of NDEs that are specific to the media over which they are transmitted. Examples of media qualities in computer networks are signal strength, voltage, and direction of propagation. Media qualities have been used in the past to determine the origin of radio-telegraph and cellular telephone transmissions [Kah67, Gla99, oA48] so it is important that these are kept in mind for origin concealment and discovery purposes.

Although possibly useful for analysis when the originator is on a shared media environment such as broadcast wireless or bus-based networks, we do not consider media qualities in any depth as digital network components obscure media qualities of an originator when a flow is retransmitted.

We have reduced our model of the observable characteristics of an NDE, m, to a 3-tuple: $(m.h, m.c, m.t)$ where $m.h = h_1, h_2, \ldots, h_k$ and $m.c = c_1, c_2, \ldots, c_n$ are bit strings representing the header and content of the NDE. $m.h$ contains metadata and identifiers in some known format. $m.t$ is the time when the NDE is observed.

3.2 Relays

We base our functional model of NOC on a *relay* abstraction. As shown in Figure 3.1, we define a relay as an input/output box that models a subgraph of the network. A transform relation $N.T()$ indicates pairs of NDEs that are causally related by processing of N. $N.in$ is the set of all possible inputs to N, and $N.out$ is the set of all possible outputs from N. Given an $m \in N.in$ and $m' \in N.out$, $N.T(m, m')$ holds true if N transforms m to output m'.

A relay may also generate and drop NDEs. Generation can be modeled for a relay N as a function of time, $N.G(t, S)$, which outputs NDEs in $N.out$. An NDE that is received by a relay but never transformed into an output is dropped. We can model the dropping of an input as a function of the NDE, $N.T(m, \emptyset)$.

The relay also has some internal state, S, that we represent as a string of bits that we assume cannot be directly observed from outside the relay. S may be different for every input processed by N.

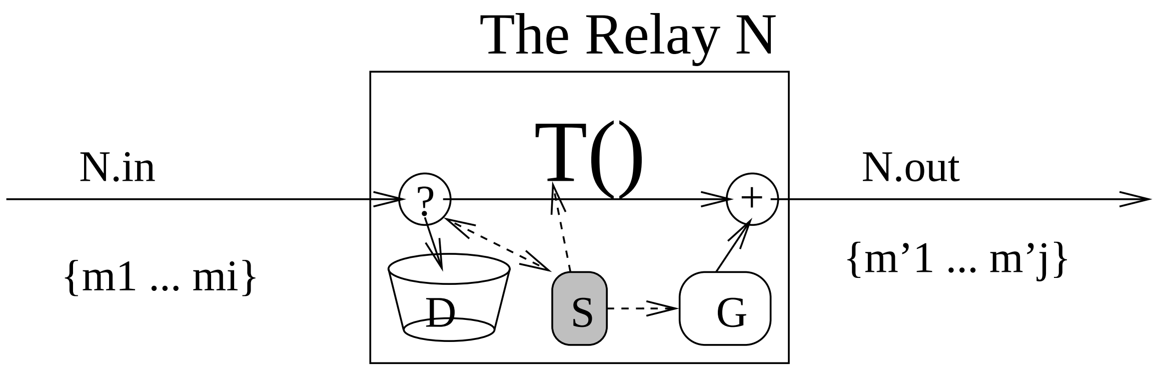

Figure 3.1. A network relay has a set of possible inputs and output NDEs and a transform relation, $T()$. The relay may also drop, D, or generate NDEs, G.

Although the figure shows only one input and one output point, this is solely for convenience. There may be many input and output edges but they may be aggregated into inputs and outputs as shown by labeling the NDEs with their corresponding edges and times. The edge data can be modeled by including it in the NDEs $m.h$ while realizing that it will be modified to account for the output edge.

3.2.1 The Origin of NDEs

An NDE is generated by a node. The generation process may be a result of a local user's input, state internal to the node, or an event scheduled by a user. To reason about the origin of an NDE after it has been transformed by one or more relays, we define the origin set of m as $origin(m)$.

$origin(m)$ is a set of nodes in the network that have contributed to the creation of m. We can define the origin set in terms of relays as follows. If a relay, N, generates m, $origin(m) = \{N\}$. If an NDE, m, is output by a relay, N, yet not generated by

N, then the $origin(m)$ is the union of the origin sets of the input NDEs that were transformed into m. If NDEs $m_1, \ldots, m_i$ are transformed into an NDE, m', output by N, then $origin(m') = \bigcup_{j=1}^{i} origin(m_j)$. If an output NDE, m', is the result of a transformation of a single input, m, then $origin(m') = origin(m)$.

We now introduce two terms related to NDEs in a NOCS: ancestor and descendent. An NDEs ancestors are those NDEs that have been transformed to yield the NDE. More formally, an NDE, m_0, input to a node n_1 is an ancestor of m_k output from n_k iff there exists some path through $k > 0$ nodes $(n_1, ..., n_k)$ such that $\bigwedge_{i=0}^{k-1}(n_{i+1}.T(m_i, m_{i+1}))$. Similarly, the descendents of m are those that have m as an ancestor.

3.3 Goals of Origin Concealment

We wish to build a functional reference model of network origin concealment systems. Functional models are based on goals of the system and the functionality that achieves those goals [Gan]. We therefore start by informally describing two goals of origin concealment. We then state these goals in terms of relays. We then describe network functionality in terms of its effect on the observables of the data elements and consequently how the goals are achieved.

3.3.1 Origin Decorrelation

The first goal of NOC is *origin decorrelation* which we will denote G_{ori}. We derive G_{ori} from the definition of NOC. G_{ori} is achieved if the network has facilities that allow NDEs to be delivered to a node such that the NDEs origin identifiers are untrusted. An untrusted origin identifier is one that may not correctly identify the origin of the NDE. Because it follows from our definition of NOC, a NOCS requires a mechanism to achieve G_{ori}.

There are two basic ways that the past work has achieved G_{ori}. One way is through spoofing of origin identifiers. Another is through intermediaries that modify NDEs on behalf of the origin. In either of these cases, the origin identifiers are modified so that they are incorrect or less specific when the NDE is received.

We note that the content of an NDE may have semantic identifiers and information that may be correlated with a specific origin. An example could be textual directions to meet at the originating user's terminal. We do not consider this type of information as it is unlikely to be useful for automated analysis.

As we discuss mechanisms for translation of headers below, we will cover G_{ori} in more detail.

3.3.2 NDE Decorrelation

The second goal of NOC is *NDE decorrelation*, or G_{nde}. G_{nde} is achieved when a data element is received from a relay such that it is difficult to determine the input element(s) that were transformed to create it, if any. G_{nde} is not required for origin concealment as G_{ori} is.

Below, we will discuss functional units of origin concealment systems and describe how each functional unit contributes to NDE Decorrelation. We base this discussion on how the functions affect the observable features of NDEs. G_{nde} is not required for origin concealment as G_{ori} is. Rather, mechanisms for G_{nde} thwart attempts to analyze NDEs to trace them to their origin.

3.4 Functional Model of Origin Concealment Systems

For each functional unit, we describe what it does and how well it achieves either of the goals of origin concealment. Furthermore, we develop a "toolbox" of techniques for origin identification systems by discussing what features remain that can be used to correlate inputs with outputs for each mechanism.

In Figure 3.2, we show a functional model of a network relay. The solid lines show the paths of NDEs passing through the relay. The rounded boxes indicated functional units that operate on one or more NDEs and the disk symbols indicate that storage may be required for this mechanism. The dashed line illustrates that certain functional units may be be applied repeatedly to an NDE.

NDEs traverse the functional units of the model by following the directed line segments. NDEs are input on the left and begin in the protocol unit. The protocol unit applies one of four possible transformations to the NDE: aggregation, fragmentation,

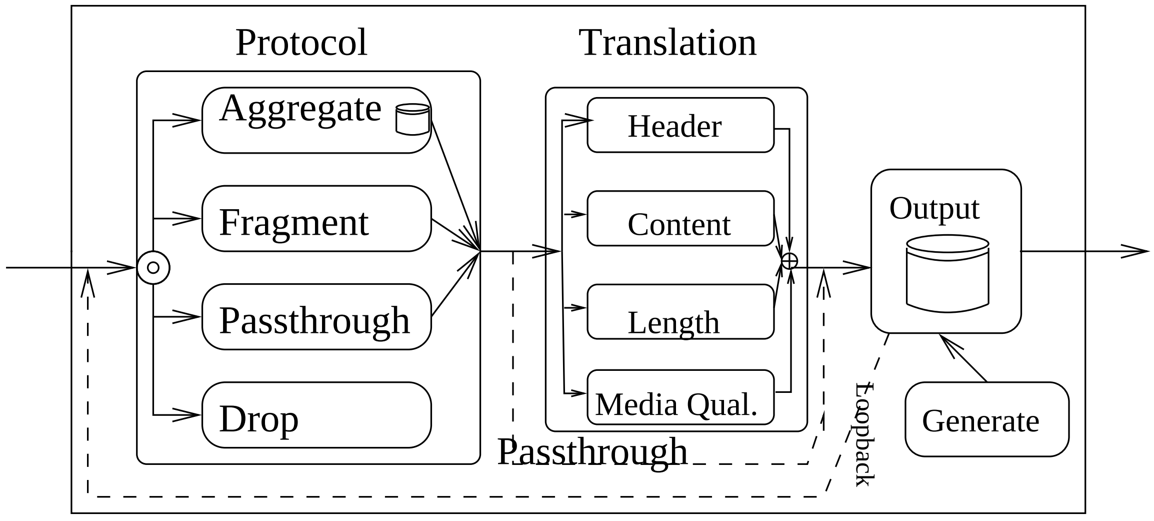

Figure 3.2. Functional model of a network relay useful for discussing origin concealment mechanisms.

passthrough, or drop. The NDEs output by the protocol unit are then processed by the translation unit. The translation unit modifies an NDE by changing its header, content, length, or media qualities. The NDEs may also bypass the translation unit by following the passthrough line. The resulting NDE may then either go to the output unit or back to the protocol unit. The output unit merges generated NDEs with those that pass through the system. Also, the output unit delays each NDE, possibly changing the ordering of the outputs.

3.4.1 Generality of the Functional Model

The functional model is general. We argue that the functional model is able to model transformations on all of the features of the NDEs that can be modified. The model covers transformations from zero or more inputs to zero or more outputs. Furthermore, the protocol and translation transformations can occur in any sequence by following the loopback edge and passthrough unit.

In Section 3.2, we described relays as entities having a relation, $T()$, that models the transformation of inputs into outputs by the relay. A relay may also drop and generate NDEs. A relation such as $T()$ may relate one input to only one output, one input to many[1] outputs, many inputs to one output, or many inputs to many outputs. Furthermore, generation creates an output without an observable input while dropping takes an input but has no output. Table 3.1 shows the number of inputs to a functional unit needed to produce output. It also describes the number of outputs that result.

Table 3.1
Functional units of our functional model of origin concealment and the type of input/output behavior that they implement.

Functional Unit	Input Count	Output Count
Generation	0	1
Drop	1	0
Passthrough	1	1
Translation	1	1
Output	1	1
Aggregation	Many	1
Fragmentation	1	Many
Aggregation, Translation, Fragmentation	Many	Many

As can be seen in the table, the functional model covers all two item permutations of 0, 1, and many, except for zero–to–zero which is inactivity, zero–to–many which is repeated generation of NDEs, and many–to–zero which is repeated drops of NDEs. It is therefore complete from the perspective of n–to–m transformations. Note that many–to–many transformations are handled in the model by aggregating many NDEs into one, translating it, and then fragmenting the result.

[1]Many in this context means greater than 1.

Another aspect of completeness is that the model can describe translations of all observable features of NDEs. The translation unit can modify the header, content, length, and media qualities of an NDE. The output unit, as will be described further, may delay NDEs thereby modifying timing and ordering.

A final aspect of completeness is that the model can describe the application of its functional units in an arbitrary order. The passthrough unit does not modify an NDE. An NDE can pass through the protocol unit to be translated and then return to the protocol unit for aggregation or fragmentation. Also, the translation unit may apply an identity transformation to an input that allows for the relay to only perform protocol unit behavior. Because protocol and translation functions can be applied repeatedly and each may not modify its input, a single relay can account for an arbitrary path through a network. More generally, a relay can model a network of multiple nodes. In this case, the NDE may follow the loopback edge multiple times with each pass through the model representing the processing of a node.

The functional model can be used to describe conversion between protocols such as from ATM cells to IP packets. The data in the first protocol can be fragmented or aggregated and the header format could then be translated to the new format.

3.4.2 Analysis of the Functional Model

For each of the functional units, we describe how the unit contributes to G_{ori} and G_{nde}. To assess the unit's role in achieving G_{ori}, we describe how and if the unit modifies features of NDEs that are useful for directly determining the origin. To assess the unit's role in achieving G_{nde}, we describe how the unit may modify an input so that its corresponding output differs from it. We also describe the remaining unmodified features of the traffic that may be useful for correlating inputs and outputs after processing by the unit. These can be used to find shortcomings in NOCS either for determining the origin or improving origin concealment systems.

3.4.3 Protocol Unit

The protocol unit's (PU) function is to determine the path that an input NDE will take through the relay. Essentially, the PU determines whether the NDE will be dropped, fragmented, aggregated, or not before being translated. The PU takes an NDE as input, and outputs it on one of its edges thereby deciding which function will be applied to the NDE.

The PU assists us in reasoning about an origin concealment system because we can make statements about dropping an NDE and whether an observer can determine the type of transformation done to the input. For instance, if a certain type of NDE will always be dropped by the relay, this can then be taken into account by saying the PU drops packets of these types. This is useful because we do not need to take these known dropped inputs into account when considering the possible ancestors of an NDE.

The behavior of the PU dictates how many outputs result from the transformation of an input. For instance, if the PU selects to drop the NDE, one input results in zero outputs. Similarly for fragmentation and aggregation we have one input transformed into many outputs and many inputs transformed into one output.

In this document, we assume that we know the functioning of the components that implement it well enough to know the transformations that may be applied to the inputs. However, in some systems [Cha81], we do not know which operations will be applied to any given NDE. Therefore, the PU's behavior may not be predictable.

NOC Goals Achieved by Protocol Unit

The behavior of the PU indirectly affects G_{nde}. Let us assume that the functioning of the PU can be simulated by observing the inputs to the relay so that the action taken by the PU for every input is known. We can then determine which NDEs will be dropped by the relay along with the paths that the other inputs will take through the relay. This would assist a NOIS designer because the NOIS could ignore the inputs to be dropped when correlating inputs to an output.

Also, if the type of transformation to be applied to an input is known, then a NOIS can subdivide the problem into a number of separate correlation subproblems according to the type of transformation that is to be applied. For instance, a NOIS would try to correlate those that are to be fragmented only with the outputs that are fragmented. Hence, G_{nde} may be better achieved if a relay has a PU that a monitor can not simulate. Mixes [Cha81] do this by encrypting the control information used by the PU so that external monitors can not simulate its behavior.

3.4.4 Dropping and Reception

If the PU decides to drop an NDE, then it will not have any descendants. For the purpose of the model, we do not differentiate between dropping and receiving NDEs.

As we have already described, it is advantageous for a NOCS to make unpredictable which inputs are dropped. If some inputs are predictably dropped, then a NOIS can exclude these from consideration.

In some cases, it may be clear which inputs are to be received by the relay as they are addressed to it, but additionally, there may be a nonzero probability that any input is dropped. For some of the inputs, we can then determine that they will be dropped, but others will be dropped unexpectedly.

3.4.5 Generation

A relay generates NDEs based on its internal state and non-network inputs. The generation unit creates an NDE and sends it to the output unit. The output unit buffers and then outputs NDEs onto the relay's adjacent edges.

NOC Goals Achieved by Generation

Some relays can create NDEs with incorrect origin identifiers. If this is possible, then the generation function can achieve G_{ori} by doing so. An example of such behavior is IP spoofing as discussed in Chapter 2. Generally, an NDE is created with incorrect identifiers and then output onto an edge. If there are no security mechanisms in place to detect and drop NDEs with incorrect identifiers, the NDE will arrive at its destination with an incorrect origin identifier.

The generation unit can also take part in achieving G_{nde}. In some NOCS [Cha81, Moe, RR97], some observable attributes of NDEs are generated with constant values. The result is that all NDEs generated in the NOCS have the same value for the attribute. If this attribute is preserved by the relays in the NOCS, then it cannot be used by an NOIS to correlate inputs and outputs because every NDE has the same value.

An example of this is the length of onions in mix networks. When an onion is generated, it is padded to some constant size. Also, when mixes process onions, extra data is added to pad the onion back out to the constant size, and therefore, every onion in the mix network has the same size. Trying to correlate onions based on size would be pointless as every onion would correlate with every other based on size.

3.4.6 Output Delay and Ordering

The output unit determines the output time of NDEs that it receives from the translation unit and the generation unit. As shown in Figure 3.2, the output unit may have storage to buffer outputs so that the outputs need not emerge in the same order as the inputs arrived. The generated NDEs are inserted into the sequence of outputs in an order determined by the output unit.

NOC Goals Achieved by Delay and Reordering

Delay is useful for achieving G_{nde} in two ways. The first effect of delay is to increase the amount of storage that a monitor needs to collect observations to correlate an input with an output. The greater the possible delay between input and output, the more traffic that can pass the monitor. Therefore, the monitor must collect more observations to ensure that it still has a record of the matching input when the output NDE emerges from the relay. Hence a longer delay is useful for achieving G_{nde}, but is not appropriate for some protocols because a protocol may need low latency to function acceptably.

In general, predictable output ordering is disadvantageous for achieving G_{nde}. If a relay queues outputs in a FIFO manner with only one–to–one transformations, every output can be correlated with its input by counting inputs and outputs separately and matching the NDEs with equal counts. This assumes that one can get an accurate initial match between an input and output so that the count can begin. If one can predict the output ordering of a relay given the assumptions from above, then one can correlate the inputs to the outputs.

Mixes address this by collecting a number of messages and then randomly reordering them before outputting them. Hence, the order of an output only indicates that it resulted from some set of equally likely inputs.

3.4.7 Fragmentation Unit

We define fragmentation to be a functional unit that takes as input a single NDE and outputs multiple NDEs identical to the input. Note that this is more general than IP fragmentation [Pos81a] as it encompasses any transformation that produces many output NDEs given one input NDE. Fragmentation does not modify the length of the NDE. The fragmentation unit enables modeling many–to–one transformations by outputting more than one copy of its inputs. The translation unit can then modify these outputs before they leave the relay.

IP fragmentation can be modeled this way. The fragmentation unit of a router would create a copy of the packet for each fragment. The translation unit would then modify the headers to set fragmentation flags and set the offset fields and lengths. The translation unit would also modify the content by removing all but the portion that should be in the fragment. The fragmentation unit can also model multicast processing. The input NDEs are copied n times and the translation unit modifies each copy so that it is output on different edges.

NOC Goals Achieved by Fragmentation

Fragmentation alone does not modify origin identifiers, so it does not directly achieve G_{ori}, but it can take a role in achieving G_{nde}. A relay can use fragmentation (along with translations) to create outputs of equal size while ensuring that no content is thrown away. For instance, if all outputs are of size n, then the relay can fragment all inputs with size greater than n. The fragments can then be length and content translated to ensure that each output is of size n and that the fragments can be later aggregated to form the original NDE. The size of the resulting outputs have no relation to the size of their ancestors. This is done in Crowds [RR97] to ensure that the sizes of input messages can not be matched to the sizes of output messages.

Correlation Techniques for Fragmentation

If the relay applies fragmentation with the identity translation[2] to an NDE, a NOIS can use string matching techniques [Apo99] to match an output fragment to its respective input Because the header and content of the NDEs are not translated, the outputs are exact copies of the input.

When fragmentation is combined with translation, the matching approach needed to correlate an input to an output will depend on the translation applied.

3.4.8 Aggregation

The aggregation unit takes as input multiple NDEs and combines them into one NDE. This allows us to model many–to–one transformations by relays. Given inputs $(m_1, \ldots, m_n)$, aggregation outputs an NDE m' such that $m'.h = conc(m_1.h, \ldots, m_n.h)$ and $m'.c = conc(m_1.c, \ldots, m_n.c)$ where $conc()$ concatenates its arguments. m' is then passed to the translation unit.

In practice, we have rarely observed aggregation in store-and-forward networks and the past work in origin concealment is devoid of aggregation. We include it to permit modeling of many–to–one and many–to–many transformations that future origin concealment systems may use.

Origin Concealment Goals and Aggregation

All other functional units in the model take either one input or no input. The result is that an output is the result of one or fewer inputs and so it has been clear that an output's origin set is the same as the input transformed into it. By aggregation, the origin set of an input may increase in size. It does so when inputs aggregated have unequal origin sets. In practice, it is likely that aggregated NDEs share the same origin set and therefore so does the output NDE.

Increasing the size of the origin set may be useful for achieving G_{ori}. In a NOIS that does traces of the traffic, aggregation would make the NOIS trace multiple inputs

[2]The identity translation refers to a translation that does not modify the observable features of NDEs other than media qualities.

to determine all members of the origin set. Hence, more work may be done to find the origin set than if aggregation were not present.

Aggregation, as we have described it, is very specific. The model relies on translation of the aggregated NDE to model more general types of many–to–one behavior. A defragmenting IP firewall as implemented in the Linux kernel [MM00] is an example of a relay that does aggregation. Fragments that make up an IP datagram are aggregated into a single datagram. The content of the inputs are concatenated to form the new packet's content, but the header is translated into the appropriate header for the assembled datagram based on the values of the aggregated headers.

The ability of a NOIS to match inputs with their aggregated output depends on the translation done to the output. If the translation is the identity translation on either header or content, then the NOIS could match inputs to an output by searching the output for substrings equal to the input NDEs content or header. Aggregation is therefore not sufficient for G_{nde}.

3.4.9 Translation

The translation unit takes a single input NDE and outputs a single transformed version of the input NDE. The translation unit may transform one or more of the header, content, length, and media qualities, or not translate them at all if the NDE follows the passthrough. Figure 3.2 shows four attributes representing the non-timing observable features of NDEs that may be modified by the relay. The figure indicates that the header, content, length, and media qualities may be modified and these changes are combined to create the new NDE.

Length refers to the length of the content. We separate content from its length to facilitate discussion cases where the translations are length preserving. The length and content are dependent on one another and we assume they must remain consistent. The length translation must accurately represent a change in the content and vice versa.

An understanding of the translations involved may allow us to decide whether a given input correlates to given output and consequently whether G_{nde} will be achieved or not. In some cases, the behavior of a translation unit can be simulated, and therefore the simulation can be applied to the input to determine the value of the output. In other cases, the translation could involve enough uncertainty to the analyst such that any input could be transformed to any output. Such an example is a relay that applies a one-time pad [Sch96] to its input. The resulting output could have come from any of the inputs. In other cases, the translation may be such that it is computationally difficult to determine the matching input. An example is the application of the Data Encryption Standard (DES) [Sch96] to an input using a secret key. For a given output, determining the input that encrypts to it is a computationally expensive problem.

By considering the difficulty of correlating an input and output based on the features translated (or not), we can discuss how effectively the relay achieves G_{nde} and discover weaknesses. For instance, if a given translation only modifies content and there is some variability in the headers, then we may be able to correlate traffic inputs and outputs based on the values of their headers. However, if the headers are all the same, then any input will match any output and G_{nde} may be achieved to some extent.

Media Quality Translation

A relay translates an NDEs media qualities if it modifies the characteristics of the NDE specific to the media carrying it. We assume that translation of media qualities by relays in digital computer networks is such that the output media qualities yield no information about the input NDE beyond that implicit in knowing the output edge. This is because the output NDEs from a node on a given edge are put on media by the same hardware. Hence, no matter what the input NDEs values are, the media qualities of the output are dependent on the output edge and not the input NDE. The translation of media qualities is noisy as the media qualities of an input are translated to the same output values regardless of the input values.

Media quality measurements for NDEs can make it difficult to achieve G_{ori} in broadcast radio transmission networks [Kah67, Gla99, oA48]. Because of their broadcast nature, any receiver in range may observe an NDE. Additionally, multiple receivers may measure the same NDEs media qualities resulting in the ability to determine the location of the origin by comparing signal strengths or direction of propagation.

In other types of message passing systems, media qualities of outputs might be useful for determining an NDEs ancestor, but because they are not present in typical digital computer networks, we ignore the translation of media qualities for the rest of this document.

Header Only

Header-only translations modify the header of NDEs without modifying their content or length. Header-only translations occur in IP routers [Pos81a], electronic mail processing [Pos82], and some reflectors [Pax00]. G_{ori} can be achieved by header only translations. Because origin identifiers are in the headers of NDEs, header only translations may modify them. Other fields of the header may be modified as well and this may be useful for achieving origin concealment.

There are four types of header translations that may be applied to origin identifiers. The header may be translated so that the origin identifiers that are output refer to the same origin as when input. Alternatively, the translation may remove the origin identifier, change the origin identifier to one bound to the relay itself, or change the identifier arbitrarily. We now consider the effects of each of these translations and their effect on achieving G_{ori}.

If an NDEs header is translated so that the output's origin identifiers are bound to the same node as the input's, then the translation is *origin preserving.* If an NDE is generated with a correct origin identifier and all relays on the route use origin preserving header translations, then G_{ori} is not achieved for the NDE. However, if the origin identifier is made incorrect at some relay in the NDEs path, the origin preserving translations will not correct the modified origin identifier, either. Hence, an originator of traffic must be able to generate traffic with incorrect origin identifiers or one of the relays on the route of the NDE must not be origin preserving to achieve G_{ori}.

The remaining three types of header translations applied to origin identifiers are *non-origin preserving*. Non-origin preserving translations remove or change the origin identifiers of NDEs. One type of these translations is removal of origin identifier(s). Because the NDE will have no indication of its origin when received by its destination, removal of the origin identifier achieves G_{ori}. A receiver would require knowledge of the origin in addition to that observable in the NDE to reply to it.

Another type of non-origin preserving header translation is replacing the origin identifier with one bound to the relay. Application gateways, NAT, and reflectors do this to achieve G_{ori}. The NDE appears to come from the relay that modified the origin identifier instead of the true origin and so we can not trust the origin identifier to be correct.

The final type of non-origin preserving header translation is done when a relay replaces the origin identifier of an NDE with an arbitrary origin identifier. A hypothetical DDOS client can be modeled as a fragmentation unit and a translation unit. The fragmentation unit receives an NDE with a command to flood a given destination thereby generating many output NDEs. The NDEs are then translated to randomly chosen origin identifiers and output by the relay. Because the origin identifiers are random, a receiver of the NDEs can not trust them to be accurate. Again, such a case does not allow for a reply by a recipient unless the recipient has information in addition to the NDE.

Fields other than origin identifiers may be modified by header translations. These fields may contain information that indicate a set of possible origins for an NDE. The translations done by the relay may contribute to this information. An example of this behavior is the IP time–to–live (TTL) field. Every router that inputs an IP packet decrements the TTL field before outputting it. As the initial value of the TTL field is no more than 255 [Pos81a], $255 - TTL$ is an upper bound on the number of hops away the origin is from the monitor. Hence, the set of possible origins is all nodes within the bounded hop count from the monitor.

A NOCS may modify the non-identifier fields so long as it does not prevent an NDE from arriving at its destination. In the TTL example above, a NOC relay could set the TTL to the number of routers on the route from the relay to the destination instead of decrementing it. The TTL then gives the receiver no information about the origin as any originator could have chosen the TTL with such a value.

If a header-only translation is applied by a relay, a NOIS may correlate inputs to outputs by matching the value and length of contents and timing as well as unmodified portions of the header. SPIE [SPS+01] uses this approach by storing hashes of portions of the header unmodified by IP manipulations and the content of IP packets. Hence, a relay that only modifies the headers of NDEs is not sufficient to achieve G_{nde}.

Content Only

Content-only translation modifies the content of an input NDE without modifying the header beyond what is necessary to maintain consistency between header and content. Some network protocols require that fields in the header depend on the value or length of the content for relays to transform NDEs. Checksum and length fields in headers are examples of this dependency as changes to content require changes to these fields for the NDE to be delivered to its destination.

Translations of content are useful for achieving G_{nde} but not sufficient. Because the header information independent of the content is not changed, by content only translations, inputs NDEs may be matched to their corresponding outputs by comparing headers except when inputs potentially matching the output have equal content–independent header sections.

If the input NDEs may have different lengths, then their length may be useful for matching an input to an output. Some NOCSs [Cha81, Moe] only generate uniform length NDEs and use length preserving transformation so that length can not be used to match inputs to outputs. In NOCSs that allow different length NDEs, length preserving translations may allow a NOIS to correlate inputs to outputs based on their lengths. Hence, length modifying translations may help achieve G_{nde} by making the length of an NDE's predecessor unpredictable.

Header and Content Translation with Delay

Header and content translation modifies both the header and the content. The header may be modified arbitrarily—not only the content–dependent fields. As with header-only translation, the NOCS can achieve G_{ori} as origin identifiers in the header may be modified.

If the content translation is length preserving and NDEs may have varying lengths, then it may be possible to use the length of NDEs to correlate inputs with outputs. Unless input NDEs have uniform length, it is therefore advantageous for NOCSs to modify the length of NDEs using a noisy translation. For example, mixes pad input onions to uniform length at each translation.

As with other translations, correlating an input with a given output translation will depend on the type of translation applied to the header and content. If the translation is noiseless, then the correlation is straightforward.

If the translation is noisy, then there is a nonzero probability that two or more NDEs in a given output's candidate set could be translated into it. We define this as the *probability of confusion* or $P_C(m')$. For a noiseless translation, $P_C(m') = 0$. As a relay's $P_C(m)$ approaches one, then G_{nde} is more likely achieved for m. This is because $P_C(m)$ indicates the probability that one or more inputs may be mistaken for the corresponding input of m.

$P_C(m)$ may be increased by incurring delay in the output unit. The greater the possible delay between an input and its corresponding output, the larger the output's candidate set can become. This provides more opportunities for confusing outputs to be input.

To understand this, consider a router with no internal buffering so that it must follow a FIFO queueing discipline and drop traffic that it can not forward immediately. The candidate set for any output packet can be no larger than the number of input edges to the router because the output must be the result of a translation of a single input and because there is no buffering, there could be no more input candidates than the number of edges. $P_C(m')$ is then the probability that two or more of the input edges received packets that could be translated into the output immediately prior to the output.

In contrast, mixes use the strategy of buffering $n > 1$ input NDEs before any are output. This combined with a translation such that an output is a decrypted version of the input makes $P_C(m') = 1$. The noisy translation in this case is public key decryption using the relay's secret key, followed by removing an address from the front of the message, and then sending the remaining content to the removed address. The message is also padded to a fixed length with random strings. $P_C(m')$ is one because mixes use a strong encryption algorithm [Sch02].

3.5 Analysis of Origin Concealment Systems Using the Model

In this section, we apply the functional reference model to a number of NOCSs. We do this to show that the model is useful for reasoning about origin concealment. The resulting analyses indicate the data that is useful for correlating inputs and outputs in an origin identification system. Further, it helps us decompose the NOCSs into functional units. This allows us to analyze each functional unit for its effect towards achieving origin concealment goals.

3.5.1 IP Spoofing and Routing

As described in Section 2.2.1, IP networks allow a node to generate an IP packet with an incorrect IP source address. If no filtering based on the source address is in place on the route between the origin and the destination, then the packet will be delivered with the incorrect IP address. If there is no other type of origin identifier or authentication mechanism available, the destination cannot trust the IP source address.

Below, we cover each functional unit of the model and how each applies to IP spoofing and routing. We consider how the functional units affect G_{ori} and G_{nde} along with the consequences for origin concealment and identification. We will see that generation along with one translation achieves G_{ori}, but that G_{nde} is not achieved by IP spoofing and routing.

Generation and Dropping

As has been stated numerous times, the generation unit in IP spoofing can create IP packets with arbitrary source addresses. If the packet has no other information directly identifying the origin, then G_{ori} is achieved. Often, layers below IP have information identifying the origin such as MAC addresses. After the packet has traversed one router, these addresses will be modified to that of the router. Hence, G_{ori} is achieved in one hop unless the router only routes outgoing traffic for the origin.

As they are forwarded, IP packets may be dropped either predictably or unpredictably. A packet is predictably dropped if there is a procedure to determine that a packet will be dropped based upon knowledge of the functioning of the relay and the value of the packet itself. Packets that are dropped for which there is no decision procedure we call unpredictably dropped.

In a NOIS, predictably dropped packets can be eliminated from consideration for correlations with outputs. Examples of such cases are incorrect checksums, when the time-to-live field is zero, packets that do not conform to protocol specifications, and packets that are addressed to a node in the current relay. Other packets may be unpredictably dropped for reasons such as hardware and software failure and buffer overruns caused by excess traffic crossing the node. Hence, G_{nde} is not achieved by predictable dropping of IP packets.

Unpredictably dropped packets are candidates for correlation with an output packet As we shall see in the translation section below, IP routing does not translate packets such that an unpredictably dropped packet is likely to be incorrectly correlated with an output.

Translation

IP routing translates packets in a straightforward manner that is easily correlated. In this section, we describe the translations used in IP routing and why they do not achieve either of the goals of origin concealment. IP routing consist of three basic types of translations: forwarding of packets and translation of packets to error and control messages [Pos81a].

Forwarding receives a packet on an interface, determines the output interface based on a routing table, and outputs the packet on the output interface. Output to an interface involves a header translation to fields below the network layer to set the addresses and possibly change the protocol for the next IP router. At the IP layer, forwarding involves header translations that decrement the time-to-live field and recomputing the checksum. Forwarding translations in IP routing do not modify the content or length of the packets.

The translation of IP packets into control and error messages involves the relay transforming an IP packet into an Internet control message protocol (ICMP) packet. We will call the translation to error or control messages an ICMP translation.

ICMP translations can occur by request as with ping and traceroute. They can also occur when a packet is discarded as when a path to the destination is unknown [Pos81a] When an ICMP translation is applied to an input packet, header, content, and length translations occur. The IP header is translated to address the ICMP message to the source address of the original packet. The header is also extended to include an ICMP header indicating the type of ICMP message. The ICMP message's content is the IP header and as much of the content from the input packet as will fit in a maximum size packet.

Because the input packet is included in the output's content, neither G_{nde} nor G_{ori} are achieved by ICMP translations. The input and corresponding output can be correlated by comparing the input packet to the content of the output. Hence, G_{nde} is not achieved any more than before translation. Although the header of the output is translated so that its source address is that of the relay issuing the ICMP message, G_{ori} is not further achieved as all of the input's header is available in the content of the ICMP message.

Fragmentation

The IP fragmentation mechanism is used to send large packets over links that only support transmission units smaller than the packet. The result is that a single packet can be split into several packets. The translations that accompany this fragmentation do not affect the goals of origin concealment though.

The header translations only modify a fragment offset, a fragment bit, and the checksum field. The content translation translates the input packet's content into substrings. The result is that any origin identifiers in the header that may be correct are preserved and G_{ori} is not further achieved. Additionally, by matching the unmodified header fields and matching output content with substrings of inputs, a NOIS can correlate the packets. Hence, G_{nde} is not achieved by fragmentation either.

Aggregation

The IP protocol does not include features for aggregation except for reconstructing fragmented packets at the destination. As these fragments all have the same origin, the origin set cannot grow as a result of aggregation. Hence, aggregation has no effect on origin concealment goals.

Delay and Output

IP packets are forwarded according to a policy set by the user or designer of the router. In practice, delay may be managed according to some quality of service policy [Com95] in the relay. Delay is not explicitly added in IP routing except to achieve some quality of service policy. Hence, the amount of delay is dependent on the traffic levels and the policy in place. Because packets can be correlated based on content and headers, delay has the effect of increasing the amount of storage needed by a NOIS to store inputs while waiting to correlate a corresponding output.

Summary of IP Spoofing and Routing

The model has facilitated our analysis showing that G_{ori} is achieved by IP spoofing because nodes can generate packets with arbitrary origin identifiers and lower layer identifiers are removed by one forwarding translation.

G_{nde} is not achieved by the transformations in IP routing. Forwarding modifies only small portions of the packet leaving the content and most of the header unmodified. ICMP translations include a possibly truncated copy of the input packet in the content of the output. Furthermore, fragmentation transformations modify little of the header and include substrings of the input packet in the output. Aggregation does not occur except when assembling fragments at the destination.

3.5.2 TCP SYN Reflector

We discussed reflectors in Chapter 2. A TCP SYN reflector is based on the three-way handshake mechanism in TCP [Pos81b]. A TCP/IP packet with a set synchronize (SYN) bit is received by a relay running a TCP service, the relay responds to the source address on the packet with both the SYN and ACK bits set (SYN-ACK). An originator uses a TCP SYN reflector by creating a SYN packet with the spoofed source address of the eventual target and addresses the packet to the relay. The relay then replies to target with a SYN-ACK packet with source address of the relay. This can be used for flooding a target with many SYN-ACK packets thereby using all of the target's network bandwidth.

Details of TCP headers as discussed below can be found in networking textbooks [Com95, Ste94].

Generation and Dropping

As above, IP spoofing is used here as well. Hence, the client must be able to generate TCP/IP packets with the source address of the target to use the SYN reflector. Because the packet is generated with a spoofed source address, G_{ori} is achieved by the generation unit exactly as in IP spoofing.

SYN and SYN-ACK packets may be dropped as they are simply TCP header and content carried by IP packets. Hence, the analysis of the dropping is the same as that for IP routing above.

Translation

Because SYN and SYN-ACK packets are IP packets, they are translated as with IP packets as described above. Additionally, we have the TCP SYN reflector translation that translates the headers of the packets.

The reflector translation swaps the destination and source IP addresses of the input packet while choosing a new value for the IP packet identifier field. Additionally, the time-to-live is reset to the default initial value. The translation also swaps TCP source and destination port fields and sets the acknowledgement sequence number to the value of the input's sequence number. The output's sequence number is translated to a new value from the state of the relay. Additionally, the ACK bit is translated from unset to set. The reflector translation does not affect content as the content of SYN and SYN-ACK packets is required to be empty.

The result of the TCP SYN reflector translation is that an output packet has sufficient information to reconstruct the input packet with some exceptions. One exception is the IP packet identifier field which is modified to a value unrelated to the input value. Other exceptions would be IP option fields, header length, and potentially the service type field which would be reset to the default of the relay. Other exceptions are the time-to-live field and checksum field. Hence, given an output from the reflector, we could correlate an output with input based on the IP addresses, TCP ports, the TCP SYN and ACK bits, lack of data payload, and one of the sequence numbers. Hence, G_{nde} is not achieved unless there are more than one observed inputs from different origins that could correlate with a given observed output based on these fields. In general, multiple inputs from distinct origins that could be confused would be highly unlikely to occur by chance. If they were constructed intentionally, then determining the origin of either or both would be useful. Therefore, correlation to either of the inputs would probably be acceptable in a NOIS.

Fragmentation

A node can generate a TCP SYN based on IP fragments. However, the Internet Protocol will not fragment existing TCP SYN (or SYN-ACK) packets as they are smaller than the minimum transmission unit required by the protocol. Hence, fragmentation of a SYN or ACK by a properly functioning forwarding node is against the protocol specifications [Pos81a].

Aggregation

If the originator of a TCP SYN chose to fragment a TCP SYN when it was generated then the TCP SYN reflector would aggregate the fragments and reply with a SYN ACK packet. The origin set of the resulting SYN ACK packet is the union of those of the fragments. If the fragments could be generated by a number of origins, then the resulting SYN ACK could have an origin set of size greater than one. Hence, each fragment might need to be traced by a NOIS to determine the entire origin set.

This type of aggregation is possible in any IP-based translation that assembles fragments and then outputs a new IP packet based on the assembled fragment. In practice, it is likely that all fragments come from the same origin node and so the output's origin set is size one.

3.5.3 Crowds

Crowds is a NOCS discussed in Chapter 2. Every pair of nodes in Crowds share a symmetric encryption key so that it is known only to the pair. This key is used to encrypt all communications between the nodes.

A communication session between nodes in Crowds proceeds in two stages: set up and transmission. A session begins when a node creates a random route through a sequence of jondoes (Crowds nodes) over an underlying network. Traffic is sent from an originator by passing NDEs through this sequence of jondoes. Each jondoe decrypts, determines the next jondoe on the path, re-encrypts the NDE, and sends it to the next jondoe in the sequence.

Generation and Dropping

All jondoes generate NDEs of some fixed size. Every NDE is always encrypted using a key known only to the current jondoe and the next jondoe in the path when transmitted.

A generated NDE is either a set up request or a message to be sent. These NDE types are indistinguishable to external monitors as each is simply an encrypted packet of the same size and both are encrypted. If the message to be sent is smaller than the fixed size, it may be padded. If larger, the message may be fragmented by the origin node.

Crowds generates message NDEs by removing identifiers directly correlatable to the origin. This contributes to Crowds achieving G_{ori}.

Every possible originating node is required to have a local jondoe that is trusted not to reveal the origin of a request. The local jondoe must receive Crowds NDEs from remote jondoes. This local jondoe forwards Crowds NDEs after decrypting and re-encrypting with different keys. Hence, even if one can observe all input and output NDEs to and from the local jondoe, this can create confusion as to which is a local input and which is received from other jondoes. This contributes to G_{ori} as an NDE from a remote site could be mistaken for a local NDE. It also contributes to G_{nde} as a remote NDE being forwarded could be mistaken for a locally generated NDE.

Translation

If a message NDE is on its last jondoe in the sequence, the message is decrypted and addressed to be sent to its final recipient as indicated in the message. This is a header, content, and length translation. The headers are translated from a Crowds message to another protocol such as HTTP. The content is decrypted and the length will change if the Crowds message was padded to achieve some set size.

If the message NDE is to be sent to another jondoe, it is decrypted and then encrypted and addressed for the next jondoe. This is a header and content translation. The headers are modified to address it to the next jondoe in the sequence and the content is encrypted differently. The length is not changed though.

If we consider only header, content, and length as being observable, Crowds achieves G_{nde}. Under these conditions, correlating an input with a given output would require breaking the encryption used in the translation. When we consider delay below, we will see that inputs and outputs may be correlated using timing.

Fragmentation

Fragmentation in Crowds NDEs can occur at the origin to achieve uniform size. This is important so that correlation can not be done based on length of NDEs.

Aggregation

If fragmentation is done by an origin in crowds, then the fragments are aggregated by the last jondoe in the path. Each of these fragments has the same origin. The resulting NDE is an HTTP request. Because of different encryption mechanisms and integrity checks, each input NDE must share the same origin to be assembled successfully. Therefore, the origin set of the HTTP request must be size one.

Output and Delay

As noted in the Crowds paper [RR97], delay is not intentionally added to NDEs traversing the network. Input NDEs are processed and output as quickly as possible. If only one NDE arrives during a time period sufficient to process an NDE, it is likely that the next output NDE correlates with it. A passive NOIS in a crowds network may correlate a jondoe's input to its corresponding output using timing if there are no simultaneous input NDEs.

3.5.4 Analysis of Anonymity Systems in General

We can make some general statements about anonymity systems that are also origin concealment systems using the reference model. We base this on an analysis of mixes. We choose mixes because they have been shown to provide sender anonymity under passive attacks [Cha81]. An analysis of mixes using the origin concealment reference model indicates that the header, content, and media qualities are obscured at each step. Delay is added so that ordering can not be used to correlate NDE's as well. Paddingi combined with fragmentation is used to ensure that every NDE in a mix network is of equal size.

If we consider a modified mix that does not obscure one of the varying observable features, we introduce a shortcoming into the system. This can be seen by comparing mix networks with Crowds. Crowds uses transformations similar to those of mixes yet does not delay the traffic traversing the network. Because of this, timing of traffic can be used to correlate NDE's across a Crowds network. Crowds addresses this problem by defining anonymity measures in terms of how many nodes be compromised or monitored while retaining anonymity for the sender. Additionally, every jondoe in the network may generate traffic indistinguishable from the traffic being transformed by it. If the length of NDEs was allowed to vary in a mix network, we would also introduce a shortcoming. The lengths of NDEs would be useful for correlating inputs to an output.

From the above analysis, we can see that providing anonymity using origin concealment systems requires transforms to a variety of NDE features. Specifically, anonymity systems where arbitrary edges may be observed must transform every observable varying feature such that input NDEs cannot be correlated with outputs. If a varying observable feature is not modified by a relay, then the feature introduces a basis for correlating NDE's.

3.5.5 Summary of Analyses

We have applied our model to a number of NOCSs. By doing so, we have determined weaknesses useful for correlating inputs and outputs. This is useful for designing NOISs as it allows us to choose attributes of traffic to use for correlation. The model has also helped us identify the methods used in each NOCS to achieve origin decorrelation. This could be useful for determining where to apply conventional authentication techniques so that incorrect origin identifiers could be detected. Finally, we have introduced an analysis of general anonymity systems based on mixes. It shows that every varying feature of NDE's must be transformed by relays for an anonymity system where arbitrary edges may be monitored.

3.6 Summary of Chapter

We have presented a functional reference model of network origin concealment systems. We have discussed how each functional unit of the model contributes to origin concealment through two goals: origin decorrelation and NDE decorrelation.

We conclude that the model is useful for reasoning about origin concealment systems and their weaknesses.

We have used the model to analyze three NOCS and discussed the mechanisms used and their shortcomings that allow us to design passive NOISs. IP Spoofing achieves G_{ori} by using an incorrect origin identifier when the packet is generated. A NOIS can use the content and the header fields that are preserved by routing to correlate IP packets. TCP SYN reflectors use IP spoofing to achieve G_{ori}. The packets are then processed by a TCP implementation to create replies that arrive at the ultimate destination. The result is that a NOIS can attempt to match inputs to the TCP implementation to its outputs based on source and destination addresses and one of the TCP sequence numbers. All of these are in the header of the packets. The content is of little use to a NOIS as it is empty. Furthermore, aggregation could occur at the TCP reflector if the sender uses fragmented TCP SYN's. To our knowledge, this is a new variant of the reflector attack that further complicates the task of passive origin identification because the analysis must take into account multiple inputs that combine to generate an output SYN-ACK. Crowds modifies traffic at every hop by encrypting it and using a randomly generated path through the network. The headers, length, and content are not useful for correlation as they are encrypted differently at each step, but timing could be used as jondoes do not delay traffic more than necessary for functioning. Finally, we have used the model to state that a general fact about anonymity systems under certain monitoring assumptions.

4. Extending the Model to Passive Origin Identification

In this Chapter, we extend the reference model from Chapter 3 to apply to Network Origin Identification Systems (NOISs). We will show that these extensions to the model are useful for reasoning about the placement of observers, the observations needed, and functionality sufficient for origin identification.

4.0.1 Review of the Network Assumptions

We model a network as an observed network as defined in Section 1.5. A node, $v_0 \in G.V$ may generate an NDE, m, which traverses a sequence of nodes, $[v_0, ..., v_k]$ where $(v_i, v_{i+1}) \in G.E$ for $0 \leq i \leq k-1$. An input m traverses a node v_i if it causes v_i to output an m'. This is modeled in the relation $v_i.T(m, m')$.

The overall goal of passive origin identification is given an observation of m_j at v_j, determine the origin set of m_j with the evidence available from the monitors. The goal of superset origin identification is given an observation of m_j at v_j, determine the smallest set of nodes that could contain $origin(v_j)$ given the evidence available. We call it superset origin identification because we are looking for a set of nodes that is a superset of $origin(v_j)$. The goal of approximate origin identification is to find some arbitrary v_i where $0 \leq i < j$ such that an ancestor of m_j traversed.

4.1 Review of Passive Network Origin Identification

In this section, we remind the reader of our taxonomy of NOISs that we discussed in Chapter 2. Table 2.1 on page 31 shows the classes and the attributes used to form them.

Passive NOISs monitor the network in one or more locations but do not modify the NDEs observed. A passive NOIS might monitor the network at several points and gather the results at one place for analysis, or it might monitor at a number of nodes in the network, storing observations for later queries. A number of passive systems (or proposals for them) were discussed in Section 2.4.

Passive NOISs may monitor internally or externally. Internal monitoring is done inside a network component that processes an NDE. An example would be a system of routers that can be made to log packets to trace a denial of service attack such as DoSTracker [CD97]. External monitoring is done using access to network media by a device that does not forward traffic. A NOIS that relies on external monitoring is described by Staniford [SC95].

In Figure 4.1, we have seven network components with four monitors in place. M1, M3, and M4 are external monitors, and M2 is an internal monitor in component N2. The figure depicts the actual network topology and layout of monitors in the network. A monitor is an actual device that records or makes observations of network traffic available for querying. It is important that we define the monitor here as we will later use an abstraction called "the observer" to reason about and aggregate observations of traffic.

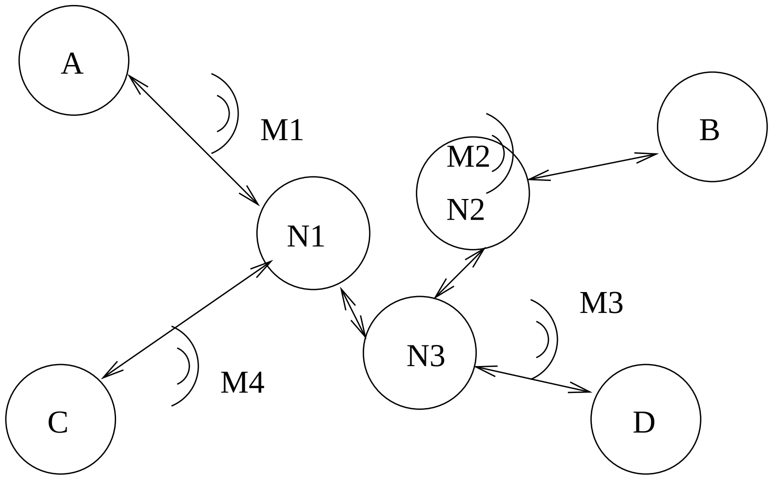

Figure 4.1. A network with three monitors installed. M1, M3, and M4 are external monitors. M2 is an internal monitor in the network component N2.

An external monitor directly observes the network media. The external observer can observe units of traffic crossing the network media to record their value, location, and time of detection. Depending on the media type, the external monitor may also be able to determine directionality and signal characteristics such as line voltage. For instance, in a broadcast network, it may be difficult for the external observer to determine the component that put the NDE onto the media and likewise which component(s) will process the NDE. Alternatively, if the media is strictly point–to–point, this information is available assuming the directionality of the NDE can be detected. The ease with which media may be externally monitored depends on the type of media. For instance, wireless networks such as 802.11b can be monitored with a receiver that is in range of transmitters, whereas monitoring a fiber optic cable will require physical access [Tan88].

The features of an NDE that an external monitor can observe are the same as the ones we described in Chapter 3. They are contents, headers, times of observation, and signal qualities. We can represent an observation of an NDE, m, as $(m.h, m.c, m.t, m.s)$. We typically neglect the $m.s$ as signal qualities are unlikely to be available or useful.

An internal monitor can record observations inside a network component transferring NDEs. An internal observer can observe the NDEs on all edges adjacent to it. Furthermore, an internal monitor may also detect NDEs created, received, or dropped by the component. An internal monitor is potentially more powerful than an external one because the internal monitor may determine the corresponding output NDEs caused by the inputs. This may require significant modification of the network component, including modifications to the operating system or hardware. If an internal monitor cannot detect transformations, generation, or dropping of NDEs, it is equivalent to external monitors on all edges adjacent to the internally monitored node.

4.2 Relays Revisited

In Chapter 3, we introduced the notion of the relay as a mechanism for discussing origin concealment systems and their components. As the goal of an origin identification system is to determine the identity of the origin of network traffic despite efforts at origin concealment, we will use the relay abstraction here as well.

A relay can model a single node from the network or a connected component of the network. The relay, N, has a set of possible inputs, $N.in$, and outputs $N.out$, and some transform relation $N.T()$ that pairs input NDEs with output NDEs. It also can generate, $N.G(t, S)$, or drop traffic.

For origin identification purposes, we usually discuss observed relays. An observed relay is one with all edges connecting it to the rest of the network monitored. Given an observed network, $G = (V, E, IM, XM)$, we can then define an observed relay. N is an observed relay if it is a subgraph of G and $\forall (u, v) \in G.E, (u \in N.V \wedge v \in (G.V - N.V) \rightarrow ((u, v) \in XM \vee u \in IM \vee v \in IM))$. In other words, every edge between the subgraph and the rest of the graph is covered either by an external monitor or an internally monitored node incident with the edge.

We can then define a contraction of the original graph such that every node in the new graph is an observed relay. We call this the *edge observed graph* as the edges corresponding to an edge in the edge observed graph are observed by a monitor. Simple versions of these graphs are when $IM = V$, all nodes internally monitored, and $XM = E$, all edges externally monitored.

An edge observed graph, $G' = (V', E', IM', XM')$, can be computed for a connected network, G, by finding connected components. First, let $V' = IM' = IM$ as all internally monitored nodes are observed relays. Next, form a graph $G_{tmp} = (V, E - XM - \{(u, v) \in E : u \in IM\})$. G_{tmp} has all edges monitored by internal and external monitors removed. Find the connected components in G_{tmp}. For each connected component in G_{tmp}, add a node to V'. If two connected components in G_{tmp} shared an edge in G, add an edge to E' between the corresponding nodes. Finally, because G is connected, the edges between the connected components were all monitored. Hence, the externally monitored edges in the edge observed graph are those in E' not adjacent to an internal monitor so that $XM' = E' - \{(u, v) \in E' : u \in IM'\}$.

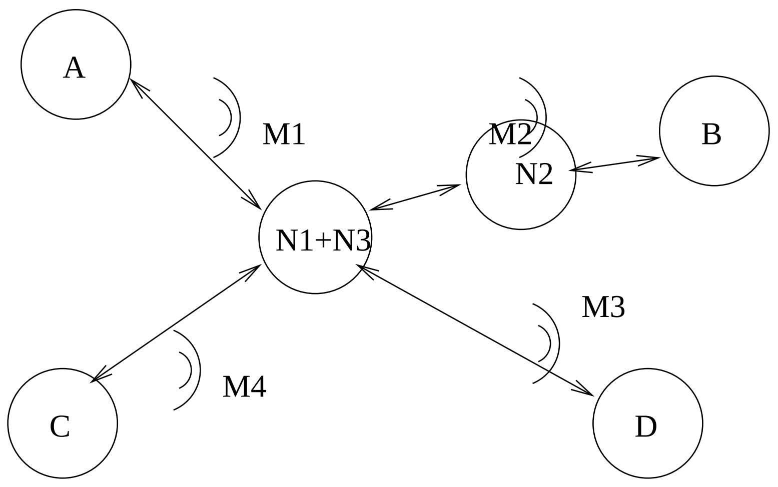

Figure 4.2. An edge-observed contraction of the network shown in Figure 4.1. $N1$ and $N3$ were contracted into a single relay. All other nodes in the original network were observed relays already.

In forming an edge observed network, we may logically merge multiple external monitors or have ignored external monitors. The merging of monitors occurs when two connected components from G_{tmp} share monitored edges in the original graph. The result is that the two edges are merged into one in G'. We discuss observers below that allow us to model the merger of monitors. An edge observed graph may not incorporate a monitored edge from the original graph if the edge's removal does not take part in partitioning the graph into components. An example of this would be a single monitored edge in a biconnected graph. Any edge's removal does not partition the graph.

4.3 Observers

An *observer* is a abstraction of one or more monitors combining subsets of the data captured by the monitors. In general, an observer is formed by selecting some subset of observations from one or more monitors while appending each observation with the an identifier of the monitor that made the observation.

Given a relay modeling an edge observed relay, N, we can define an input observer by selecting all inputs observed incoming on edges to N by each monitor on edges incident with N. Similarly, we can form an output observer for N. The result is that we can form a relay with observed inputs, $O.in$, and observed outputs, $O.out$.

An internal observer for a single node is simply the observations from the node's internal monitor with an identifier for the node appended. Figure 4.4 shows an internal observer and the data it can collect.

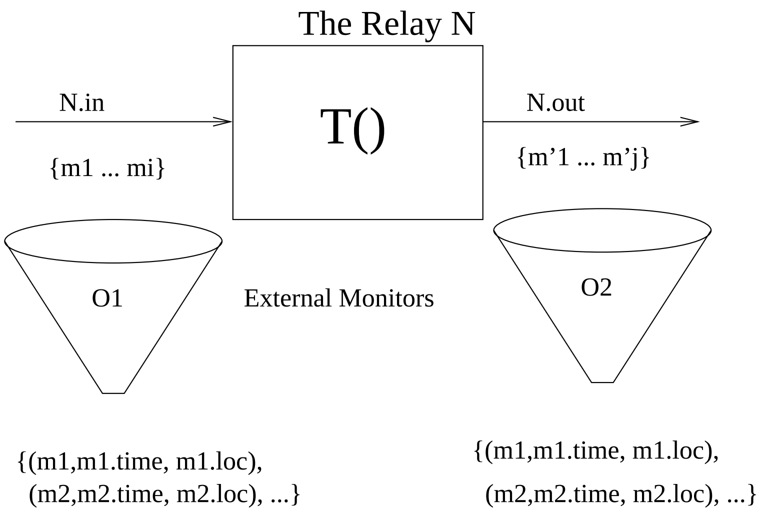

Figure 4.3. Functional diagram of a network component with external observers at its inputs and outputs.

Figure 4.3 shows a model of an external origin identification system around a relay, N, with two observers, O_1 and O_2, and an NDE correlation function. O_1 is an input observer representing the inputs of one or more actual monitors. They could be implemented as many distinct monitoring devices on many network media connected to nodes represented by N or only one device monitoring a single full duplex connection to N.

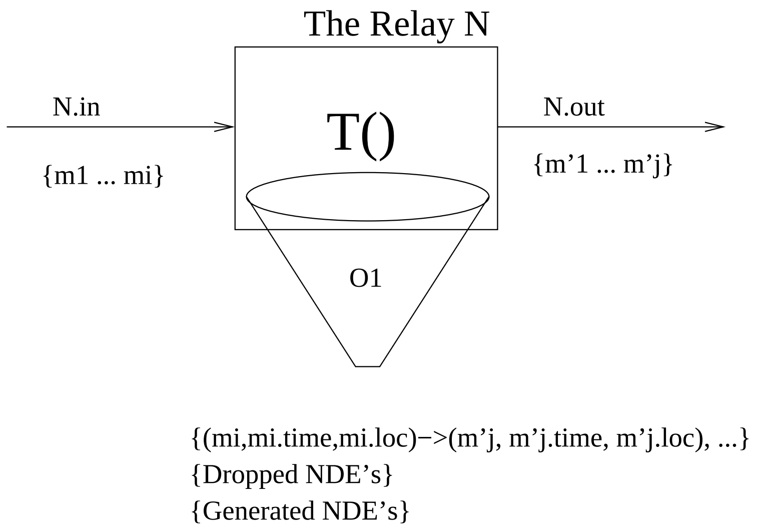

Figure 4.4. Functional diagram of a network relay with an internal observer. O1 can observe the input NDE, the time of input, and the receiving interface. O1 can also observe output NDE, the output time, and interface, but more importantly, O1 may correlate inputs and output NDEs. The observer can also detect NDEs dropped and generated by N.

Input/output edge refers to the connection where the NDE was observed. This could be denoted by one of N's interface names or an incoming point to point media name. We also assume that the NDE can be observed as being incoming or outgoing with respect to N. These can be represented as hardware layer identifiers in the NDEs identifier vector.

Figure 4.4 shows a network relay N with inputs and output NDEs as in Figure 4.3. In this case, $O1$ monitors N internally and produces data in addition to the external case. The relationship between NDEs input and output may be determined directly. In the figure, the NDEs generated by N are reported as well as those dropped.

The availability and quality of this relationship information depends on a number of factors. If the observer can be tightly integrated into N's software or hardware, then the accuracy of the relation could be high. In DoSTracker [CD97], built-in debugging facilities in routers are used to provide accurate relationship information. If the behavior of N can be simulated, an internal observer could also provide information about the state of N and its inputs so that this dependence information could be created by simulation. This might require a less tightly integrated observer than above and would require further processing of another entity to simulate N.

When N is an arbitrary connected component of the network, the concept of external monitoring is clear, but it is not yet defined for internal monitoring. External observers only require access to the media connecting N to the remainder of the network, so it does not matter whether N is a network or a single node for making external observations.

We define internal monitoring of a relay based upon the possible observations for internal monitoring of a single node. First, assume the network is N as described in Figure 4.3. Our requirements are that we can observe incoming and outgoing NDEs, their dependency relationships, and which of them were dropped or generated by N. We note that this says nothing of the NDEs that never leave the boundary of N. We will justify this in an upcoming section.

To achieve internal monitoring for a relay of more than one component, a necessary condition is that N is an observed relay. A second necessary condition is the ability to determine the dependence between incoming and outgoing NDEs. We must also be able to infer that NDEs are dropped or generated in N. We call the problem of determining the dependencies between NDEs *NDE correlation*, and we will describe it in greater detail later in this chapter.

There are situations where a monitor may not be necessary on every edge to internally monitor a relay. Knowledge of a network might give information that certain edges could not carry the type of NDE in which we are interested nor carry their ancestors. In this case, we could simply eliminate the edges from the graph for that type of NDE. An example of the situation might be a well-secured email gateway between a classified and unclassified network. Even though there is a connection, traffic other than email may not pass through the gateway, so if the NDEs of interest to the NOIS are neither email nor related to email then the gateway need not be monitored.

4.4 Functions of a Network Monitor

A network monitor collects observations and may store them for later analysis or retrieval. The internal processing of a network monitor can be represented by functional components for control, selection, data reduction, storage, and reporting. Figure 4.5 illustrates these components and how they interact.

The control unit of a monitor allows a remote entity to send it commands. In some cases, such as DoSTracker [CD97], the process of making observations may interfere with the functioning of the device by consuming processing power, I/O bandwidth, or delaying NDEs further. This is in addition to possible storage used to store the observations. Because of this, a monitor may have a mechanism for beginning, ending, and configuring the observation process so that resources can be devoted to the normal functioning of the device. The control functions are also responsible for authenticating commands to the monitor to prevent unauthorized

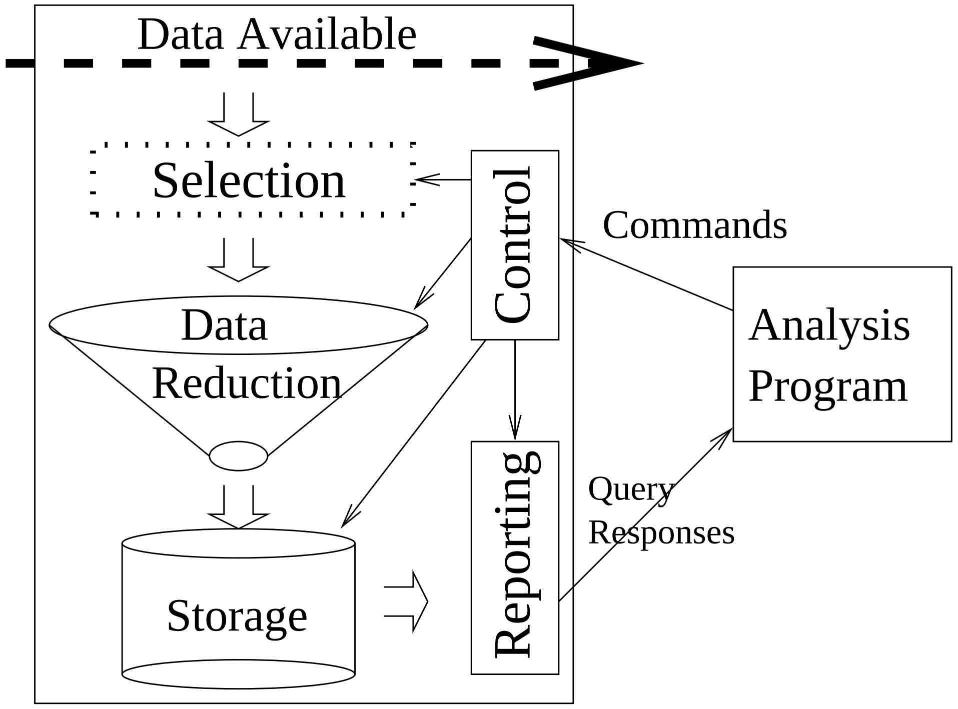

Figure 4.5. The diagram shows the functional components of a passive network origin identification system. The box to the left contains a network monitor with its functional units.The box to the right is an analysis system that sends commands to the monitor, and receives data from the monitor to trace an NDE.

access. Other commands for retrieving observations are passed on to the reporting unit. Alternatively, the control unit may not be configurable.

The selection unit in a monitor takes commands from the control function. The selection functionality extracts NDEs from all the NDEs observable by the monitor. Commands from the control unit can instruct the selection unit to use either event-based or random sampling or both to select NDEs from those observable.

Event-based sampling is the method used in the past work. In event-based sampling, a monitor awaits an event to trigger the taking of an observation. Events include reception of an NDE matching some criteria, a change in state of the monitor, or specific commands from the control unit. Random sampling would be the alternative to event-based sampling. In random sampling, the monitor would select an NDE for observation based on some probability distribution. We know of no instances of this among the past work in passive NOISs, but random sampling for marking has been used in active flow marking schemes discussed in Chapter 2.

DoSTracker's [CD97] selection unit is the router's own filtering mechanism set to raise an alarm when a packet matching a signature is input. SPIE's [SPS$^+$01] selection unit is basically the identity function as it monitors every IP packet input. Some types of monitors in IDIP [Row99] use a network intrusion detection system's pattern matching system as a selection unit.

Data reduction is a function of monitors that has been the focus of past work [SCH95, SC95, SPS$^+$01]. Because the amount of storage available to a monitor is limited, it will eventually fill. Data reduction techniques attempt to store reduced yet still characteristic data about the NDEs selected by the selection unit. SPIE [SPS$^+$01] stores hashes of IP packets using a Bloom filter, which later allows recognition of the hash without actually being able to recreate it. Traffic thumbprinting [SCH95, SC95] reduces time slices of TCP streams to counts of selected characters from the streams. Compression techniques could also be applied in the data reduction unit.

The data reduction unit may not be present in all NOISs,as in some cases all selected NDEs may be stored in total. A case where this could be appropriate is in a low bandwidth network where in-depth manual analysis is to be performed on the traffic. In this case, it could be difficult to choose portions of the NDEs to discard because it may not be apparent which portions of them will be useful for analysis.

Depending on its use, a monitor can have a great deal of storage for observations or little. As indicated before, there is a relation between the history length and the amount of storage available to the monitor. The history length, h, is the amount of time that a monitor can continuously take and store observations without discarding any observations. We can compute $history \geq \frac{storage}{obsfreq \times obssize}$ where $storage$ is the amount of storage on the monitor, $obsfreq$ is the maximum frequency with which observations are generated, and $obssize$ is the maximum size of an observation. We observe that the history length is inversely proportional to the size of observations, and therefore aggressive data reduction can extend the history length of a monitor.

The storage unit in a monitor receives data from the data reduction unit and stores it on some media. This storage is not necessarily local to the monitor as it could be a repository at some remote node. The storage unit may choose to overwrite or update previously stored observations according to some policy. One policy is first in-first out, where the oldest data is overwritten as new data becomes available. Another policy could prioritize the observations by type of traffic, so that traffic more likely to contain an intrusion attempt would be held longer in storage than benign traffic. IDIP [SDS00] includes a priority mechanism for storing observations so that more threatening traffic may be tracked longer than more benign traffic.

A monitor's reporting unit informs remote entities of observations either by answering queries or raising alarms. The reporting unit receives commands through the control unit from an analysis program to report on observations from storage. In the case of a query, the analysis program makes a request for observations matching certain criteria and the resulting observations are returned from storage. IDIP [SDS00] includes intrusion detection systems in its monitoring components. The intrusion detection systems can configure monitors so that when observations match certain patterns, the observations are reported to an analysis system.

The types of queries that a reporting unit can answer are dependent on the data reductions made before storage. Thumbprinting and SPIE use data reduction techniques similar to hash functions. Because of this, the data collected in these systems is only useful for reporting whether an observation of a piece of traffic with a certain hash value was observed.

The types of queries that a reporting unit can answer are also dependent on the amount of storage available. Some monitors such as the routers in DoSTracker [CD97] have no or little persistent storage for observations and are therefore limited to reporting observations that are currently being made by the monitor.

4.5 Analysis Programs

Passive NOISs have analysis programs that analyze observations from monitors to determine the origin of NDEs. These analysis programs run on one or more network components to determine information about the path or origin of an NDE by analyzing the observations reported by network monitors.

Analysis programs interact with network monitors by sending them commands via the network or other channels. Monitors may then respond to these commands using the same or alternate channels. The past work in passive NOI use the network to communicate with its monitors and therefore make a critical assumption. The assumption is that the network will not drop commands or responses between network monitors and the analysis program.

In the past work in which analysis programs have been described, the analysis program runs on nodes distinct from the destination of the NDE. This is a good choice, as it is likely the job of a network administrator to begin the analysis program. In some cases, a network intrusion detection system initiates an analysis program and therefore the analysis program may begin before the NDE has been received by its destination. Additionally, the observer could be administered by a third party such as an Internet service provider or law enforcement.

4.5.1 Types of Correlations

Analysis programs in passive NOISs use two types of correlation to process observations from monitors. The first is *node correlation* of which *origin correlation* is a special case. The second type of correlation attempts to determine relationships between observations of NDEs and is called *NDE correlation.*

Correlation is a long-studied subject in statistics [Hac75]. In this reference model, we use correlation as a term for the process of relating certain types of entities in the model based on our observations. Mansfield et.al. [MOT$^+$99] used traditional statistical correlation techniques to trace denial of service attacks, but others [BS02, MOT$^+$99, BDKS00] have used the term correlation more loosely to describe determining relationships between NDEs and their origins and ancestors. We will use the term correlation with the understanding that we are not referring strictly to conventional statistical methods.

Node Correlation

In some cases, an analysis program may be able to determine a node in the path of an observed NDE directly from the observation. An example of node correlation is used in conjunction with Staniford's thumbprinting work [SC95]. When two extended connections are matched, the NDEs in the connection can be correlated with their upstream host in the path by their source addresses. The source address is probably correct for this link in an extended connection, as it is difficult to spoof the source address in an ongoing TCP stream such as those for which thumbprinting is effective.

Origin correlation is a special case of node correlation that determines identifying information about the origin directly from an observation of an NDE. Origin correlation has been used in traffic analysis of radio signals [oA48, Kah67] to determine the origin of radio transmissions.

None of the past work in passive origin identification in computer networks has focused on directly correlating NDEs to their origin, but have done so implicitly by making assumptions about the behavior of the network. For example, passive systems for tracing packets [CD97, MOT$^+$99] have assumed that packets being traced were from hosts at the edge of the network and therefore, when a packet was found to have come from an edge router for an autonomous system, it was assumed that the packet originated in that autonomous system.

Other related work that could be useful for origin correlation is passive operating system fingerprinting [Pro02]. This work analyzes network traffic for behavior indicating that it was generated by a particular operating system. One example of this is the default value of the time to live field in IP packets. Some operating systems set it to 255 while others use lesser values. Fingerprinting of operating systems is not likely to be of great use, as it can only tell us which operating system and version thereof was running. Furthermore, the characteristics used by an operating system could be faked by a clever attacker.

A method that we have considered is based on trusting the network to properly decrement the IP time to live field. As its initial value can be no larger than 255, we know that the originator of an IP packet can be no more than $255 - ttl$ hops away from the observer. Given an accurate routing map for that time period, the set of possible originators could be reduced considerably. In response, an originator could choose an initial *ttl* so that it is 0 on arrival, thereby yielding no information about the origin. Furthermore, broken or compromised nodes on the path could modify the *ttl* arbitrarily.

NDE Correlation

Analysis programs in passive origin identification for computer networks have used a variety of NDE correlation techniques. NDE correlation refers to techniques that determine relationships between observations of NDEs so that the analysis system can form a chain of evidence that leads to a set of possible origins for an NDE.

Three types of NDE correlation have been used for passive origin identification. We call them shared origin correlation, cause correlation, and ancestor correlation.

Shared origin correlation (SOC) is the most general type of these correlations. Given an observation of an NDE output from a relay, $\omega' = obs(m')$, each correlation type attempts to find observation(s) of NDEs input to the same relay, $\omega = obs(m)$, such that $origin(m) = origin(m')$.

The SOC approach makes assumptions about the protocol in use or type of attack being traced to make this correlation. In DoSTracker, a signature is created by the analysis agent of the NDE to be traced. Each router along the path in turn is queried for inputs matching the signature. When a match is found, it is not an ancestor of the original. The assumption is that because the packets match the signature, they are part of a group of NDEs that are generated by the same origin.

SOC is useful when monitors have small histories. Because the correlation is based on NDEs matching a criterion that indicates that they are likely to share the same origin, the monitor can wait to observe another NDE that matches the criterion instead of storing previously observed traffic.

The use of SOC can be a weakness as an attacker could send each NDE from a different origin. If this is the case, then a trace may fail as the next monitor may never see another NDE matching the signature.

For a relay, N, cause correlation determines one or more input observations, $\omega = obs(m)$, given an output observation, $\omega' = obs(m')$ such that $N.T(m, m')$. This is what an internal monitor may do directly, but to do so for an externally observed relay is more difficult. There may be many different input NDEs to N that produce a given output. Hence, there may be many NDEs observed as input that could have produced the output that we wish to cause-correlate.

A relay may also have state that is unknown to the analysis program, and transformations based on this state may make cause correlation difficult. An example is a relay such as a mix that encrypts its inputs with a strong secret key encryption system. There may be only one input that can be encrypted to form a given output, but it is computationally expensive to determine the matching input.

Ancestor correlation is a process that given two observations, determines if one is an ancestor of the other. This is a generalization of cause correlation because the cause of an NDE is an ancestor of the NDE as well. In some cases, ancestor correlation may actually be easier than cause correlation. If the route of an NDE crosses a relay multiple times, then determining all of these may be easier than picking one out as the exact cause of the NDE.

4.6 Sufficient Functionality For Passive Origin Identification

In this section, we introduce five mutually sufficient conditions for passive origin identification. They are network separation by trusted monitors, enough storage per monitor to accommodate the analysis, an analysis program to collect and process observations from the monitors, a trusted communication path between analysis programs and the monitors, and correlation of an input to a given output across every relay. Together, they form sufficient functionality for passive origin identification.

4.6.1 Network Separation by Trusted Monitors

If we can make no assumptions about the route taken by a piece of traffic other than best effort delivery based on destination address, we can make assertions about the origin identity based on observations of the network. One way is to separate the network into two or more components such that all NDEs flowing between the components are monitored. This is what we mean by network separation. It could also be the case that some network monitors are compromised so that they return false data or refuse to return data altogether. A trusted monitor will report data that reflects the data it observes when requested.

We have already seen network separation in "edge observed" networks and observed relays. Network separation is actually necessary for origin identification using passive techniques. To see why, consider a network with monitors inside such that no separation occurs. Hence, if the monitored edges were removed, the network would remain connected. No nodes can be internally monitored in this scenario, as an internally monitored node would be separated from the network into a component of size one. This means that only edges are observed and that NDEs observed on an edge could come from any node in the network because there is always another path not observed. If there were not, the monitors would form a separation.

Additionally, we argue that separation is a sufficient placement for some level of passive origin identification. If we have monitors such that they separate the network into two components, then the route of the NDE either crosses the separator or it does not. If it does not, then the origin must be in the same component as the destination. If it does cross the separator multiple times, then the origin must be in the component from which the first observation was seen to arrive. If it was not, then there would be an observation of it crossing the separator at an earlier time.

From this argument, we can see that separation of the graph by monitored edges is a necessary and sufficient placement to achieve some level of origin identification, but this is deceptive. For instance, the separation might be a single internally monitored node. We then have an edge-monitored graph with one node representing the monitored node and its neighbor(s) representing the rest of the network. Certainly, observations at this node may tell us which of the neighboring subgraphs contains the origin, but the worst case is that the node has one neighbor representing all other nodes in the network. Hence, the number of candidates can be reduced to one in the unlikely event that the monitored node is the origin or $|V| - 1$ otherwise. It is worst case in the sense that if the NDE originates in the remainder of the network, we can only determine that the flow originated somewhere in that remainder.

If a graph is separated by monitored edges, but one of the monitors is compromised, then an attacker could (among other things), falsify observations to cause a traceback algorithm to go towards the wrong component or fail to report observations to cause the search to stop earlier than if the monitor was uncompromised.

4.6.2 Storage

A monitor must have at least enough storage so that its history length exceeds the interval between observing an NDE and an eventual query for the observation. As discussed in Section 4.4, the needed history length is dependent on the time between observation of an NDE and its arrival at its destination. Additionally, the needed history length as discussed in Chapter 5 can be dependent on the search algorithm used by the analysis program. This dependence makes it difficult to state anything definitive about the history length without knowledge of the approach of the analysis program and the maximum latency of the network, as these may be of arbitrary or unbounded length.

4.6.3 The Rest

We now describe how the rest of the conditions can be combined to determine the origin of NDEs in a network based on the conditions already discussed. Because we can do so, these conditions are sufficient for passive origin identification. Furthermore, because these conditions are based on the components of the reference model, the reference model is useful for reasoning about origin concealment, and it is also sufficient to describe a scenario for origin identification.

Internal Monitors

Assume that we have a network, $N = (V, E, IM, XM)$ where $XM = \emptyset$ and $IM = V$. N is an "edge observed" network, and because nodes are internally monitored, the relation between inputs and outputs may be observed. We now describe the internal directed search algorithm (commonly called traceback) in terms of this network based on the conditions described so far.

The NDE, m, arrives at v_k, and the administrator desires to trace it. We describe the steps of the algorithm for internal observers below.

1. An observation, $obs(m_k)$, is given to the analysis program as an argument.
2. In our analysis, we have assumed that a neighbor node, v_{k-1}, is apparent from the observation, and therefore the monitor at v_{k-1} is queried for an input that correlates with $obs(m_k)$. If the neighbor node is unknown from the observation, all neighbors can be queried this first time.
3. The result of the query is either an $obs(m_{k-1})$ such that $v_{k-1}.T(m_{k-1}, m_k)$ or v_{k-1} generated m_k. If it was generated there, then the algorithm halts reporting v_{k-1}.
4. The analysis program repeats the above with $obs(m_{k-1})$ as its argument.

In steps 1 and 5 we see that the analysis program coordinates the directed search. In step 2, the analysis program may query the monitor for either a predetermined pattern allowing the monitor to do the correlation or query the monitor for its entire storage, which allows the analysis program to do the query itself. Accurate NDE correlation allows the analysis program to make the correct choice for the next node to be queried.

As the analysis program repeats, it may query monitors that are increasingly distant. This process cannot proceed unless there is a communication path that is trustworthy between the analysis program and the monitors. If a monitor is untrustworthy, it could claim no correlations exist and the analysis would halt. If the monitors are trustworthy and the communication path does not modify or drop queries or responses, then the analysis program will get the correct next step in the path in each round of these steps. The result is that there will be only one node of the graph returned at the end of the run of the analysis program. Hence, we achieve origin identification.

External Monitors

A network with external monitors, $N = (V, E, IM = \emptyset, XM)$, can be reduced to an "edge observed" graph, $N' = (V', E', IM' = \emptyset, XM' = E')$ by the algorithm described earlier. As described earlier, if the observers do not form a separator for the network, N' will be a single node.

We can form an analysis program as we did for internal directed search. The difference is that this time we treat each node in N' as an observed relay with input and output observers composed from the actual monitors in XM. We begin in the node of the observed graph that contains v_k. The algorithm queries the input observer of the current node to find $v'_{k-1} \in V'$. This is done by cause-correlating the received observation with the flows observed input to the node. The prime indicates that the actual v_{k-1} may be grouped with other actual nodes in the "edge observed" graph. If the correlation step succeeds, a new node in V' is searched. If not, then then the algorithm replies with the nodes in V corresponding to the current node in V'.

Note that the difference between the internal and external algorithm is in the query step. In the internal case, one query needs to be made to an internal observer and the correlation is made. In the external case, the input observer is an abstraction of one or more actual monitors. The result is that many actual monitors may need to be queried to accomplish this step. Furthermore, if the correlation relies on timing or ordering of observations from different monitors, we have the problem of synchronizing distributed clocks [Lam78].

4.7 Conclusions

We have presented a reference model of passive network origin identification. It is composed of a functional model of a network monitor that communicates with an analysis program. The network monitor has a control unit for interpreting commands from an analysis program and a reporting unit that provides observation information to the analysis program. The monitor also has functional units for selecting, reducing in size, and storing observations of NDEs and the behavior of relays. The analysis program can query the monitors for observations and correlate observations to determine the origin of NDEs.

We have described how several conditions are sufficient for determining the origin of NDEs. Because these conditions are derived from the reference model discussed in this chapter, the model has therefore been useful for reasoning about the observation of origin identity. Additionally, we have reasoned about the relation between storage and history length of monitors.

5. Topologies for Efficient NOI Algorithms

Past work in passive network origin identification (NOI) has made few assumptions about the network topology. The result is that these NOISs have used a directed search algorithm that uses $O(n)$ queries to network observers. In this chapter, we introduce two new search algorithms that we have developed for NOI in trees and planar graphs. By choosing somewhat restrictive topologies, we have developed divide and conquer tracing algorithms that use sub-linear numbers of queries at the expense of increased computational cost. We argue that *divide and trace* algorithms can be implemented so that the acceptable reaction time to begin a trace is potentially increased. The result is a new approach in passive network origin identification based on designing a network topology for monitoring instead of installing observers in existing networks.

We assume for this chapter that network data elements (NDEs) follow cycle-less paths through a network of undirected edges. This is acceptable because in this chapter we are concerned with NDEs such as packets, frames, and cells that we expect to be routed to their destination over a cycle-less path, if not the shortest path. We also note that network protocols often have time-to-live mechanisms designed to prevent packets from surviving routing loops.

We also assume trusted internal observers and that the network infrastructure will not block or modify tracing queries. In other words, we assume that routers will forward our tracing requests and that the network components implement internal observers that will answer them according to their observations.

5.0.1 The Problem

We have an observed network $N = (V, E, IM = V, \emptyset)$. An NDE, m, follows a simple path $[v_0, v_1, \ldots, v_k]$ through the network. The problem is to determine v_0 by querying the members of IM.

We assume that the monitors in use are trusted internal monitors that can report a matching input given an output.

As described in Chapter 4, we described how a network could be contracted into an "edge observed" network, N'. A virtually observed network is a network whose nodes and edges have been contracted so that each node in the new network represents a connected component in the old graph with all edges observed. An observed edge in the new network replaces one or more monitored edges in the original network as we merge observations from multiple monitors to form an observer.

5.1 The Directed Search Algorithm

The directed search algorithm(DSA) which is generally referred to as traceback is used in much of the past work in NOI [Row99, CD97, MOT$^+$99]. In this section, we describe the DSA using internal observers. As described in Chapter 4, if we can decompose the network into subgraphs so that all edges between subgraphs are covered, and we can achieve NDE correlation between those edges, then we can contract the subgraphs into internally monitored relays and treat them as nodes of the graph. Because of this contraction, we can then convert this internal algorithm to one that uses external observers.

The DSA begins with a node v_k receiving an NDE, m, on edge, (v_{k-1}, v_k) and hence making an observation, o_f. v_k queries the node, v_{k-1} for an observation that correlates with o_f. If none exists, the NDE must have come from the queried node, or the observation is no longer available, so the algorithm terminates. If o_f correlates with some $o_{f'}$ observed at the adjacent node, then repeat the above process again at v_{k-1} with the NDE, f'', such that $v_{k-1}.T(f'', f')$ and the incoming edge of f''. Each iteration of this algorithm we will call a round. Note that because we are using internal observers, the input edge of the correlated NDE is available therefore indicating the node for the following round. Hence, each round of the algorithm requires only 1 correlation operation.

We choose to express the cost of running the algorithms in terms of the number of correlations or queries done. This is reasonable as each query sends a request over the network to an observer. Minimally, this may only require the observer to consult an indexing mechanism such as the Bloom filters in SPIE [SPS+01], but it could also require a significant time-consuming correlation computation where a traffic shape pattern is compared to many observed patterns [YE00]. It is therefore reasonable to consider the number of correlations as a measure of the time cost if the remainder of the algorithm is significantly less costly.

DSA using external observers differs from the internal version in that each round consists of 1 or more correlations with the data from observers on the other edges incident to the currently queried node. The cost in number of correlations per round is then $O(maxdegree(N))$. It is bounded by the maximum degree of the network, as the sender might send the NDE via a path consisting of nodes with maximum degree.

The number of correlations and rounds in the internal DSA is $O(k)$. Additionally, the number of rounds in the external DSA is $O(k)$ so that a trace requires $O(k \times maxdegree(N))$ in the external case.

We now consider the role of network topology in the performance of the DSA. The number of rounds needed for the DSA to complete a trace in balanced trees is $O(\log n)$. To see this, consider that the worst case is to trace from a leaf to another leaf through the root. The number of nodes on such a path is $O(\log n)$, as the tree is balanced and the path has no cycles. An even better case is the star topology which is simply a balanced $(n-1)$ary tree. The worst case is 1 correlation when the trace is initiated from a leaf node. The central node can determine the origin of an NDE by detecting the input edge. In both of these cases, the balanced nature of the trees makes the DSA effectively a divide and conquer algorithm.

We now ask if there are sub-linear correlation algorithms for graphs with fewer assumptions than balanced trees. We first describe a new algorithm for arbitrary undirected tree topologies that requires $O(n \log n)$ computation, but $O(\log |V|)$ correlations for a trace in $O(\log |V|)$ rounds. We then expand this work further by describing an algorithm for planar graph topologies that requires $O(n \log n)$ computation, but $O(\sqrt{n})$ correlations in $O(\log n)$ rounds.

5.2 Divide and Trace in Trees

The divide and trace algorithm in trees (DTT) is a divide and conquer algorithm for determining the origin of a NDE in an undirected tree. It requires $O(\log n)$ rounds with one correlation operation per round for internal observers, while requiring $O(n \log n)$ computation total.

We describe DTT using internal observers, DTT_{int}, by breaking it into rounds. As shown in Figure 5.1, an invocation of DTT_{int} takes as input the graph, G, the current instance of the NDE to be traced, m, and the node where m was observed, v. Each invocation of DTT_{int} as the algorithm runs is "a round."

The algorithm works by partitioning the graph into two or more subgraphs by finding the tree's centroid. A centroid is a node in the tree, such that if removed, the largest resulting subgraph will have at most half as many nodes as the original graph. If two centroids exist in the tree then we choose the centroid, which if removed will split the tree into the most subtrees and arbitrarily otherwise. In an undirected tree, it is well known that there must exist one or two centroids [Knu73].

As there are no cycles in a tree, if the path originated in one of the subgraphs other than the one currently under consideration, the path must include the centroid as there is no other node with edges connecting those subgraphs. Similarly, if a path includes the centroid then v_0 must either be the centroid or be in another subgraph as there are no cycles in the tree. The algorithm then asks the observer at the centroid if an NDE correlating with m has been observed there. If not, v_0 must then be in current subgraph. If so, the centroid indicates from which subgraph the NDE came or that it originated at the centroid. We can then disregard all but the indicated subgraph for the search as the centroid separates the subgraph from the others.

```
$DTT_{int}(G, m, v)$
  if (number of nodes in $G$) == 1 then return $G$ else
    $c = centroid(G)$
    Query $c$ to correlate $m$ with its observations
    Let the result, $R = (v', m')$,
      where $m$ caused $m'$ at $c$ and $v'$ is next node in the path
    If $R$ was observed at $c$ and $m' = nil$ then  return $DTT_{int}(\{c\}, m, nil)$
  remove $c$ from $G$ yielding subtrees $G_1, G_2, ..., G_n$
  If match, $m$, exists at $c$,
    return $DTT_{int}$($G_i$ containing $R.v'$ , $R.f'$, $R.v'$)
  else
    return $DTT_{int}$($G_i$ containing $v, f, v$)
```

Figure 5.1. DTT_{int} Algorithm

From the algorithm, we can see that each round except the last uses one correlation, hence it suffices to bound the number of rounds to get a bound on the number of correlations required for a trace. As each round except the last decomposes G at the centroid, the graph passed to the next round is at most $\frac{|G|}{2}$. Therefore, the number of rounds and correlations used by the algorithm is $O(\log n)$. Because the graph is split into at least half at each round, after i rounds the working graph has at most $\frac{1}{2}^{i}n$ nodes. From this expression, we see that a $n(i) = 2^i$ node network can be traced in at most i rounds.

We still have not considered the cost to find a centroid. A centroid of an undirected tree is a node such that if it is removed, the resulting graph has 2 or more disconnected subgraphs, each of which has no more than half the nodes of the original graph. A tree has either one or two centroids. If it has two centroids, then they must share an edge.

It is well known [Knu73, p. 387] that finding the centroid of a tree can be done in $O(n)$ time by assigning each node a child count. To do this, choose a root node and use depth first search on the graph to count the number of nodes in each node's subtree. The centroid is then a node where all of its subtrees and $|V|$ minus the sum of the subtrees are at most $\frac{|V|}{2}$. Finding all centroids in the tree can be done in advance so that a round of DTT_{int} does not have to compute the centroid during execution. Repeated applications of the above approach can precompute all centroids in $O(|V| \log(|V|)$ time.

5.3 Divide and Trace in Planar Graphs

We use the linear separator theorem from Tarjan and Lipton [LT79] to create an $O(\log n)$ round tracing algorithm using $O(\sqrt{n})$ total correlations in planar graphs. The algorithm which we name DTP is very similar to DTT_{int} as it still uses a $O(|V|)$ running time algorithm to partition the network, but each round uses multiple correlations done in parallel. DTP_{int} is shown in Figure 5.2.

```
$DTP(G, m, v)$
  if $|G| == 1$ then return $v$ else
    Find a separator graph, $G_0$, shattering $G$ into $G_1, ..., G_n$
    In parallel, query each $\{c_i \in G_0.V : (c_i, u) \in G.E \wedge u \not\in G_0.V\}$
      for correlations with $m$
    Let $R = \{(m_{(u,v')}, d) : u \in G_0 \wedge v' \not\in G_0$
      $\wedge m_{(u,v')}$ correlates with $m$ on $(u, v') \wedge d =$ node receiving $m_{(u,v')}\}$
    If $|M| = 0$, then return $DTP(G_i, m, v)$ where $v \in G_i$
    else
      Select $m_{(u,v')}$ with earliest time of observation from $M$
      return $DTP(G_i, m, d)$ where $d \in G_i$
```

Figure 5.2. *DTP* Algorithm

5.3.1 Analysis

In every round but the last, the algorithm shatters the graph by finding a separator graph. The algorithm then queries every node in the separator that is adjacent to the resultant subgraphs. By following the earliest observation that correlates with m from these queries, we know that the NDE must have originated from the graph it was observed leaving earliest. If it did not originate in that graph, it would have passed the separator boundary even earlier and would have been observed.

Tarjan and Lipton [LT79] developed an algorithm for determining a balanced separator graph for planar graphs in $O(|V|)$ time . The separator is guaranteed to have at most $\sqrt{8|V|}$ nodes. If the separator is removed from G, G is broken into 2 or more disconnected graphs of at most $\frac{2}{3}|G|$ nodes. If we consider every call of DTP a round, we can see that every round works on a graph no larger than two thirds the size of its predecessor. Therefore, the DTP terminates in $O(\log n)$ rounds. We can form a recurrence for the total number of queries, $T(n) = \sqrt{8n} + T(\frac{2}{3}n) = O(\sqrt{n})$.

Finding the balanced separators in the algorithm takes $O(|V|)$ time for a graph as $T(n) < kn + T(\frac{2}{3}n) = O(|V|)$ for a constant $k > 0$. As the separators are not dependent on the behavior of the DTP algorithm, we can precompute them. We could compute all balanced separators in $O(|V| \log |V|)$ by recursively applying the Tarjan and Lipton algorithm to each of the separated subgraphs and the separator. Because no nodes of the graph are gained or lost in each level of the recursion tree and the recursion tree has depth $O(\log |V|)$, computing all of the separators can indeed be done naively in $O(|V| \log |V|)$.

Although we may precompute the separators and make the queries in parallel, there is still a $O(\sqrt{|V|})$ time expense per round to compare the replies to determine the earliest observation. So, we have an algorithm that is $T(n) = \sqrt{8n}+T(\frac{2}{3}n) = O(\sqrt{|V|})$ running time even if we precompute the separators. If we can assume that the time to query the observers and wait for the reply is bounded by a constant time and large compared to the time to receive and choose the earliest result, then the running time reduces to $O(\log |V|)$.

5.4 Comparison of Algorithms

In this section, we compare the strengths and weaknesses of our divide and conquer algorithms with the DSA. We find that the divide and trace approach is likely to be of more use in a centrally administered network where the structure of the network can be controlled and network components are perhaps more likely to be trustworthy. Furthermore, divide and trace only works where NDEs can be correlated with their ancestors an arbitrary distance along their path.

5.4.1 Advantages of Divide and Trace

The divide and trace algorithms have a number of advantages over DSA. Because the DSA's time cost is $O(k)$, DSA's worst case cost is actually the maximal simple path length in the topology. In trees and planar graphs, we observe that we can construct such graphs where the maximum acyclic path length is $|V| - 1$. Therefore, DSA can take $O(|V|)$ rounds, even in these simplified topologies.

Divide and trace algorithms have another advantage over the DSA that is not immediately obvious. In some situations, divide and trace algorithms may allow searches to proceed that would fail if DSA were used because observers have limited memory.

So far, we have assumed that the observers that we are querying have unbounded memory and can therefore store observations for an unlimited time into the past. If we change this assumption, we see that for a fixed amount of memory at the observers, we can store a fixed number of observations and therefore given a continuous stream of traffic, we can observe only so far back in time at each observer. Furthermore, an observer in a heavily loaded part of the network will not be able to store observations as far into the past as an observer in a less busy part. It appears that the amount of storage needed at network observers is dependent on the time to do origin identification for an NDE,

An example of this behavior can be evaluated in a linear network. Consider a linear network, $G = (V, E)$, such that $V = \{v_1, v_2, \ldots, v_n\}$ and $E = (v_i, v_{i+1}) : 1 \leq i \leq n - 1$. If we assume each v_i is equally likely to be the origin, we see that the expected number of rounds for DSA is $\frac{n-1}{2}$ and the worst case is $n-1$. t_{round} is the time to complete a round of DSS, and we assume it is constant here. The most storage needed by any node of G is $obsrate \times ((n-1) \times t_{round} + transit(v_1, v_n))$ bytes for v_1 when v_1 is the origin where $transit$ is a function giving us the time for an NDE to cross the network between two nodes. $obsrate$ is the rate at which observers in the network fill their memory with observations.

If we use DTT on this network, we need $obsrate \times (\log n \times t_{round} + transit(v_1, v_n))$ bytes for storage at any node in the network as $\log(n)$ is the maximum number of rounds to reach any node and the longest transit time between nodes. If we have precomputed the centroids, DTT should have a similar time cost per round neglecting network latency, and so t_{round} should be similar for both DSS and DTT.

As can be seen in the example, the amount of storage needed by the observers to answer queries by the algorithms is dependent on the time needed to complete a trace. Examining the model we see that the maximum storage needed for DTT is always less than for DSA assuming equal constants. Evaluating this for real networks would be difficult as each observer may have a different $obsrate$, and t_{round} may vary by round.

Informally, consider that nodes in the “middle” of the network may carry more traffic than those near the “edges.” If this is the case, the nodes near the “edge” will be able to report on events further in the past than the “middle” ones assuming equal memory. Given this, it appears advantageous to correlate near the middle of the network first so that the less persistent observations are evaluated first. The divide and trace methods do so, but it is unclear how important this effect is.

5.4.2 Disadvantages of Divide and Trace

Divide and trace algorithms have several disadvantages compared to DSA. Most importantly, these divide and trace algorithms put constraints on the network topology to achieve their running times. We do not know how many real world networks are trees or planar. They also require up to date topology information and additional work to determine a separator whereas DSA can proceed by discovering topology information it needs from each observer in turn. Others have developed technology for discovering network maps [BC01, CBB00], but this information may have errors or omissions as the methods are not perfect. For this reason, these divide and trace approaches are probably best suited for large private networks where topology can be planned by an authority, and topology information can be kept current.

Divide and trace also assumes that an NDE can be correlated with one of its ancestors at any observer on the path whereas DSA only requires that a queried NDE be recognized by an observer at one hop back and that its immediate ancestor can be identified. Because of this, these divide and trace approaches would not likely be useful for tracing NDEs in networks where correlation across multiple relays is difficult or error prone. We do however believe that DTP could be used to trace IP packets in planar IP networks with trusted router observers.

DTP may use significantly more network capacity per round, as it may query $O(\sqrt{|V|})$ observers per round, whereas DSS and DTT make one query per round. We expect that each query and result to be slightly larger than the NDE that is being traced. We therefore do not expect the queries to consume too much bandwidth.

Because DTP uses timestamps to determine the earliest crossing of the separator, the observers must maintain some level of synchronization in their clocks. Because we are concerned with ordering the observations of events where NDEs cross the boundary edges between the separator and other subgraphs, synchronization to within half the minimum time between boundary crossings is sufficient for the ordering. All conditions being equal, the least time between events occurs when a node has two edges that cross the boundary of the separator graph. The minimum time between events is the time for the node to receive on one edge and re-send on the other. The observers on those edges would need clocks synchronized to within half of this minimum time to order the events [Lam78]. Network protocols exist for distributed time synchronization [Mil89]. A clever way of forgoing time synchronization for DTP is to use the NDE's own hop count to order observations, if it exists. In IP networks, the time to live field could be part of the observations. Because the field is decremented at each hop, it could be used to order correlated observations of the same IP packet.

5.4.3 Relation of the Algorithms to the Reference Model

In Chapter 4, we describe mutually sufficient conditions for passive network origin identification. They are network separation by trusted monitors, enough storage per monitor to accommodate the analysis, an analysis program to collect and process observations from the monitors, a trusted communication path between analysis programs and the monitors, and correlation of an input to a given output across every relay. We can see that each of these issues is pertinent to the algorithms discussed in this chapter.

By comparing these algorithms to the sufficient conditions for NOI, we see that all have the issues of storage and a trusted communication path. In DSA, DTT, and DTP, the storage needed is dependent on the behavior of the network and the behavior of the algorithm. If we can not bound the time between observation of the NDE by monitors in the network and its delivery, then we can not bound the storage needed by the monitor. Similarly, a trusted communication path is needed by all of the algorithms so that queries to or reports from the monitors can pass.

In DSA, each round separates the network into the queried node and the remainder of the network. This occurs in both the internal and external forms of the algorithm. We discussed the issue of the storage needed for DSA above as a function of several factors in the network. DSA describes the behavior of the analysis program and we have described the need for correlation across the relays either by using internal monitors or by analyzing external observations.

In divide and trace techniques, the analysis program separates the network into roughly equal portions. This is what yields the logarithmic number of rounds used by these algorithms. The need for network separation has not been previously discussed in the past work. This may be because past work has relied on the DSA where correlation is done across relatively simple relays. Divide and trace techniques are based on relays consisting of sets of nodes based on the separators themselves.

5.4.4 Potential Applications of DTP

An example of a planar network is an irregular grid of wireless network access points in a campus setting such as a university. Clients access the network via the nearest access point. The access points only connect to each other via land lines laid out in the plane of the earth. By acting as routers on these links, clients using the access points exchange NDEs. Planarity can be maintained by ensuring that the land links do not cross over one another. Alternatively, the network could be made planar by adding monitors at the points where links crosss.

DTP could then be used to determine the wireless access point where a client is attaching to the network. From that and nearby access points, techniques such as triangulation by signal strengths might be applied [Gla99, Kah67] to find the physical location of the transmitter.

5.5 Conclusions

In this chapter, we have described two new algorithms for origin identification in constrained network topologies. The algorithms improve in terms of number of rounds over the traditional traceback approach by constraining network topology and precomputing network separators. These algorithms could be used to motivate development of policy for layout of an organization's network topology to ensure high performance tracing of network data elements.

6. Summary and Conclusions

This dissertation deals with origin concealment and identification in store-and-forward networks. We introduced past work in origin concealment and identification. To assist in this, we developed a taxonomy of origin identification systems. We developed a reference model of origin concealment that is useful for reasoning about NOCS and their shortcomings. Based on this, we developed a reference model for passive observation of origin identity that is useful for reasoning about sufficient conditions for passive origin identification. Furthermore, we have used the model to develop and reason about new search algorithms for origin identification systems.

In the past, work in origin concealment has occurred in either anonymity systems or in analyzing methods used to conceal network attacks. We bring these areas together in the field of origin concealment. Additionally, past work in origin identification has focused on authentication mechanisms for cooperative originators and some initial work on determining the origin of network attacks. Our work brings these areas together to discuss origin concealment and identification in general store-and-forward networks.

In the following sections, we discuss our contributions to the field and their importance.

6.1 Analysis of Origin Concealment and Identification

We have presented a functional model of origin concealment that is useful for reasoning about NOCS and their shortcomings. The model assists the analyst by splitting the behavior of the NOCS into a number of functional units that can then be analyzed for weaknesses and strengths.

As part of our introduction to past work in origin identification, we introduced a taxonomy of origin identification systems. The taxonomy is based on the structure and monitoring behavior of the origin identification systems. It classifies all past work and contains an objective decision procedure for classification. Furthermore, the model is predictive as there are 42 unoccupied classes out of 48 total in the taxonomy. Each of these classes is a possibility for future study. Finally, the taxonomy allows us to break the problem of origin identification into smaller parts. We do so in Chapter 4 to produce a reference model of passive origin identification.

6.2 A Reference Model of Passive Origin Identification

The reference model of passive origin identification identifies the components of a passive origin identification system and how those components interoperate. Network monitors may be either external or internal and the data that can be collected by them depends on this placement. Internal monitors are potentially more powerful as they may determine the mapping between inputs and outputs of the relay. Furthermore, we define the functional units of network monitors in terms of data selection, data reduction, storage, control, and reporting units. All of these may be configured or commanded by a control program. The result is that there are five mutually sufficient conditions for passive origin identification. These combine to allow us to determine the origin of NDEs using passive monitors.

We conclude that the reference model is useful for reasoning about passive origin identification as we have been able to develop a set of mutually sufficient conditions for passive origin identification. These sufficient conditions are useful in analyzing and developing the divide and trace algorithms for origin identification.

6.3 Divide and Trace Algorithms for Passive Origin Identification

We chose the analysis program from the reference model of passive origin identification to study in greater depth. Past analysis programs in passive systems have taken a directed search approach that traces NDEs backwards along their path, one node at a time. The result is that past analysis programs have a $O(n)$ running time, where n is the size of the network.

We developed a different type of analysis program by constraining the network to tree and planar topologies. This allows us to quickly determine a balanced network separator so that we can apply divide and conquer techniques to the origin identification problem. The result is a $O(\log(n))$ analysis program for arbitrary trees and a $O(\sqrt{n})$ analysis program for planar networks. Both of these may require $O(n \log n)$ when the topology changes

The results are two algorithms that are useful for analysis programs in passive NOISs when the network's topology is limited or can be controlled by policy such as in a campus environment. Because the algorithms query the interior of the network first, less storage may be required at the monitors. Furthermore, we use the reference model and the mutually sufficient conditions to analyze divide and trace, thereby demonstrating the reference model's utility.

7. Future Work

We suggest five general areas of future work suggested by this dissertation. Below, we consider each of these areas as it relates to the dissertation. We also discuss the possible contributions that this future work could make.

Our taxonomy of origin identification suggests many areas of future work. Each class in the taxonomy that is unpopulated is a possible area for future work. All of the unpopulated classes except one are active and result from merging various marking types and route modification. We therefore expect productive future work in determining how the combination of a number of marking types and route modification can be used in NOISs. Such work could result in new methods of marking that improve accuracy of the active NOISs or reduce the computational resources necessary in the network.

An area that is not well studied in computer networks is flow-origin correlation. Future work should study how media qualities facilitate origin identification in shared media networks such as fiber optics and wireless networks. Additionally, the behavior of various hosts could be studied to determine characteristics that can be used to link an origin host with its traffic. These techniques would allow network devices to either detect spoofing of traffic or store information for later correlation of traffic directly to its origin.

The functional model of network monitors for origin identification describes a variety of areas for future work. Certainly, analysis programs will continue to be an area of future work both in new methods for correlating network traffic, but also in search algorithms for processing the data collected. We expect future work in selection of traffic to further examine specialized hardware for selecting traffic for analysis. Data reduction and compression are also of interest as they extend the history length of the monitor. One area in particular would be to study the compressibility of features of network traffic for storage and later correlation. Perhaps traditional compression models could be used to extend the history length of monitors. In the areas of storage and reporting, future work may consider the feasibility of using remote storage for monitors either over dedicated communication links or the network itself.

Another future work related to the reference model for passive origin identification is the extension of the model to further study active systems. Perhaps a better understanding of the capabilities and mechanisms needed by active systems could be discovered in this way.

An area of possible work in active origin identification is the modification of network protocols to facilitate origin identification using network monitors at places in the network in addition to the destination. We would initially investigate the requirements of such a system and propose adding fields to NDEs and adding cryptographic operations to network protocols to facilitate NDE correlation and space-efficient storage of observations.

The divide and trace techniques from Chapter 5 are areas for further study. An initial approach is to study existing networks to determine if they are planar or tree graphs. Applying these algorithms to real networks may involve finding methods for handling cases where networks are nearly planar or nearly trees. Future work could look for methods of simplifying such networks while maintaining the performance characteristics of the algorithms. Implementation and stochastic modeling could be used to investigate the implications of divide and trace techniques on the history length needed by monitors in realistic networks. Furthermore, an implementation of divide and trace in a planar network could help us evaluate its performance in realistic situations.

LIST OF REFERENCES

LIST OF REFERENCES

[And00a] Dave Andersen. Distributed denial of service attacks (DDOS). http://wind.lcs.mit.edu/~dga/ddos.txt, February 2000.

[And00b] David Anderson. Distributed denial of service attacks (DDOS). Available at http://wind.lcs.mit.edu/~dga/ddos.txt, November 2000.

[Apo99] Alberto Apostolico. *Algorithms and Theory of Computation Handbook*, chapter Chapter 13: General Pattern Matching. CRC Press LLC, 1999.

[BC00] Hal Burch and Bill Cheswick. Tracing anonymous packets to their approximate source. In *Proceedings of LISA 2000*, pages 313–321, New Orleans, Louisiana, December 2000.

[BC01] A. Broido and K.C. Claffy. Internet topology: Properties of IP graphs. In *Proceedings of IEEE SPIE Conference*, Denver, Colorado, August 2001.

[BDKS00] Florian Buchholz, Thomas E. Daniels, Benjamin A. Kuperman, and Clay Shields. Packet tracker final report. Technical Report CERIAS TR 2000-23, Center for Education and Research in Information Security and Assurance (CERIAS), Purdue University, West Lafayette, Indiana, 2000. Available at https://www.cerias.purdue.edu/papers/archive/2000-24.pdf.

[Bel89] S.M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communication Review*, 19(2):32–48, APR 1989.

[Bel00] Steven M. Bellovin. Internet draft: ICMP traceback messages. Available at http://www.ietf.org/internet-drafts/draft-bellovin-itrace-00.txt, March 2000.

[Bhu72] A. K. Bhushan. RFC 354: File transfer protocol, July 1972.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[BS02] Florian P. Buchholz and Clay Shields. Providing process origin information to aid in network traceback. In *Proceedings of the Annual USENIX Technical Conference*, Monterey, California, 2002.

[BSD] FreeBSD operating system. http://www.freebsd.org.

[Bur00] Lynn Burke. Hot on the trail of "mafiaboy". http://www.wired.com/news/politics/0,1283,34354,00.html, February 2000.

[CA-96] CERT Advisory CA-96.21. TCP SYN Flooding and IP Spoofing Attacks. http://www.cert.org/advisories/CA-96.21.tcp_syn_flooding.html, September 1996.

[CA-98] CERT Advisory CA-98.01. 'Smurf' IP Denial-of-Service Attacks. http://www.cert.org/advisories/CA-98.01.smurf.html, January 1998.

[CBB00] Bill Cheswick, Hal Burch, and Steve Branigan. Mapping and visualizing the internet. In *Proceedings of USENIX 200 Conference*, San Diego, California, 2000.

[CD97] H. Chang and D.Drew. DoSTracker. This was a publically available PERL script that attempted to trace a denial-of-service attack through a series of Cisco routers. It was released into the public domain, but later withdrawn. Copies are still available on some websites., June 1997.

[Cen] National Infrastructure Protection Center. Major investigations: Mafiaboy. `http://www.nipc.gov/investigations/mafiaboy.htm`.

[Cen96] CERT Coordination Center. Cert advisory ca-1996-21 tcp syn flooding and ip spoofing attacks. `http://www.cert.org/advisories/CA-1996-21.html`, September 1996.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[Cha88] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[CIS] CISCO. Characterizing and tracing packet floods using Cisco routers. Available at `http://www.cisco.com/warp/public/707/22.html`.

[CNW$^+$99] H.C. Chang, R. Narayan, S. F. Wu, B. Vetter, M. Brown, X. Wang, J. Yuill, C. Sargor, F. Gong, and F. Jou. Deciduous: Decentralized source identification for network-based intrusions. In *Proceedings of 6th IFIP/IEEE International Symposium on Integrated Network Management*, Boston, Massachusetts, May 1999.

[Com95] D. E. Comer. *Internetworking with TCP/IP*. Prentice-Hall, third edition, 1995.

[Dee89] S. E. Deering. RFC 1112: Host extensions for IP multicasting, August 1989.

[DFS01] Drew Dean, Matt Franklin, and Adam Stubblefield. An algebraic approach to ip traceback. In *Proceedings 2001 Network and Distributed Systems Security Symposium*, pages 3–12, San Diego, California, February 2001.

[DFS02] Drew Dean, Matt Franklin, and Adam Stubblefield. An algebraic approach to ip traceback. *ACM Transactions on Information and Systems Security*, 5(2):119–137, May 2002.

[DH98] S. Deering and R. Hinden. RFC 2460: Internet Protocol, Version 6 (IPv6) specification, December 1998.

[Dit91] David Dittrich. The "stacheldracht" distributed denial of service attack tool. Available at `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt`, December 1991.

[Dit99] David Dittrich. The ”tribe flood network” distributed denial of service attack tool. `http://packetstorm.securify.com/distributed/tfn.analysis.txt`, October 1999.

[Dit00] Dave Dittrich. DDoS — Is there really a threat? Invited talk at Usenix Security Symposium 2000: Slides available at `http://staff.washington.edu/dittrich/talks/sec2000/`, 2000.

[DS00a] Thomas E. Daniels and Eugene H. Spafford. Network traffic tracking systems: Folly in the large? In *Proceedings of the New Security Paradigms Workshop 2000*, Ballycotton, Ireland, 2000.

[DS00b] Thomas E. Daniels and Eugene H. Spafford. Subliminal traceroute in TCP/IP. In *Proceedings of the National Information Systems Security Conference*, Baltimore, Maryland, September 2000.

[EF94] K. Egevang and P. Francis. RFC 1631: The IP network address translator (NAT), May 1994.

[EHM96] Marlena Erdos, Bret Hartman, and Marianne Mueller. Security reference model for the java developer's kit 1.0.2. Available at `http://java.sun.com/security/SRM.html`, November 1996.

[FKK96] Andreas Fasbender, Dogan Kesdogan, and Olaf Kubitz. Variable and scalable security: Protection of location information in mobile IP. In *Mobile IP, 46th IEEE Vehicular Technology Society Conference*, Atlanta, Georgia, March 1996.

[FM97] David H. Freedman and Charles H. Mann. *At Large: The Strange Case of the World's Biggest Internet Invasion.* Simon and Schuster, 1997.

[Gan] Jagdish S. Gangolly. The functional model. `http://www.albany.edu/acc/gangolly/ssa3.html`.

[Gla99] Joanna Glasner. Feds ok cell phone tracking. *Wired News*, September 1999.

[GS99] Ian Goldberg and Adam Shostack. Freedom network 1.0 architecture and protocols. Available at `http://www.freedom.net/info/freedompapers/Freedom-Architecture-Protocol%s.pdf`, November 1999.

[GT96] Ceki Gulcu and Gene Tsudik. Mixing email with Babel. In *Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security*, San Diego, California, February 1996.

[Hac75] Ian Hacking. *The emergence of probability: a philosophical study of early ideas about probability, induction and statistical inference.* Cambridge University Press, London, United Kingdom, 1975.

[HMP96] P. Horster, M. Michels, and H. Petersen. Some remarks on a receipt free and universally verifiable mix-type voting scheme. In *Proceedings of ASIACRYPT '96*, pages 125–132, Kyongju, Korea, 1996.

[IN-99] CERT Incident Note IN-99-07. Distributed denial of service tools. `http://www.cert.org/incident_notes/IN-99-07.html`, November 1999.

[Ins02] Computer Security Institute. *2002 Computer Crime and Security Survey.* Computer Security Institute, San Francisco, California, 2002.

[Jen96] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, volume 1. Springer-Verlag, New York, second edition, 1996.

[JKS$^+$93] Hyun Tae Jung, Hae Lyong Kim, Yang Min Seo, Ghun Choe, Sang Lyul Min, Chong Sang Kim, and Kern Koh. Caller id system in the Internet environment. In *UNIX Security Symposium IV Proceedings*, pages 69–78, Santa Clara, California, 1993.

[Joh93] M. St. Johns. RFC 1413: Identification protocol, January 1993.

[Jr.99] David Michael Martin Jr. *Local Anonymity in the Internet.* PhD thesis, Boston University, 1999.

[KA98] S. Kent and R. Atkinson. RFC 2401: Security architecture for the Internet Protocol, November 1998.

[Kab00] M. E. Kabay. Distributed denial-of-service attacks, contributory negligence and downstream liability. ACM Ubiquity: Available at `http://www.acm.org/ubiquity/views/m_kabay_1.html`, 2000.

[Kah67] David Kahn. *The Codebreakers: The Story of Secret Writing.* Scribner, 1967.

[Kan91] B. Kantor. RFC 1258: BSD rlogin, September 1991.

[KFH$^+$93] Calvin Ko, Deborah Frincke, Todd Heberlein, Karl Levitt, and Biswanath Mukherjee. An algorithm for distributed recognition and accountability. Technical Report CSE-93-7, University of California–Davis, November 1993.

[KFTG$^+$93] Calvin Ko, Deborah A. Frincke, Jr. Terrence Goan, L. Todd Heberlein, Karl Levitt, Biswanath Mukherjee, and Christopher Wee. Analysis of and algorithm for distributed recognition and accountability. In *Proceedings of First ACM Conference on Computer and Communications Security*, Fairfax, Virginia, 1993.

[Knu73] Donald Knuth. *The Art of Computer Programming*, volume 1. Addison Wesley, second edition, 1973.

[Krs98] Ivan Krsul. *Software Vulnerability Analysis.* PhD thesis, Purdue University, 1998.

[KS94] G. Kessler and S. Shepard. RFC 1739: A primer on Internet and TCP/IP tools, December 1994.

[KST98] Ivan Krsul, Eugene H. Spafford, and Mahesh Tripunitara. An analysis of some software vulnerabilities. In *Proceedings of the National Information Systems Security Conference*, Crystal City, Virginia, 1998.

[Lam78] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[LBK80] George Lucas, Leigh Brackett, and Lawrence Kasdan. Star Wars Episode V: The Empire Strikes Back. Motion Picture from 20th Century Fox, 1980.

[LS00] Brian N. Levine and Clay Shields. Hordes – a protocol for anonymous communication over the Internet. In *Proceedings of the 7th ACM Conference on Computer and Communication Security*, Athens, Greece, November 2000.

[LT79] R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Mathematics*, 36:177–189, 1979.

[Mal93] G. Malkin. RFC 1393: Traceroute using an IP option, January 1993.

[Mar01] Gary T. Marx. *Documenting Individual Identity*, chapter Identity and Anonymity: Some Conceptual Distinctions and Issues for Research. Princeton University Press, 2001.

[Mil89] D. L. Mills. Internet time synchronization: The network time protocol. Network Working Group Request for Comments: 1129, October 1989.

[MM00] Scott Mann and Ellen L. Mitchell. *Linux System Security: The Administrator's Guide to Open Source Security Tools.* Prentice Hall PTR, 2000.

[MMW$^+$01] Allison Mankin, Dan Massey, Chien-Long Wu, S. Felix Wu, and Lixia Zhang. On design and evaluation of intention-driven ICMP traceback. In *Proceedings of IEEE International Conference on Computer Communications and Networks*, Miami, Florida, 2001.

[Moe] Ulf Moeller. Mixmaster protocol version 3. Available at `http://www.eskimo.com/\~rowdenw/crypt/Mix/draft-moeller-v3-01.txt`.

[Mor85] Robert T. Morris. A weakness in the 4.2BSD Unix TCP/IP software. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey, February 1985.

[MOT$^+$99] G. Mansfield, K. Ohta, Y. Takei, N. Kato, and Y. Nemoto. Towards Trapping Wily Intruders in the Large. In *Proceedings of the Second Annual Workshop in Recent Advances in Intrusion Detection(RAID)*, West Lafayette, Indiana, September 1999.

[MW97] Suvo Mittra and Thomas Y.C. Woo. A flow-based approach to datagram security. In *Proceedings of the ACM SIGCOMM '97*, Cannes, France, September 1997.

[oA48] Department of Army. *Fundamentals of Traffic Analysis (Radio Telegraph).* Aegean Park Press, Laguna Hills, California, 1948. Also known as Department of the Army Technical Manual TM 32-250 and Department of the Air Force Manual AFM 100-80.

[OMT$^+$00] Kohei Ohta, Glen Mansfield, Yohsuke Takei, Nei Kato, and Yoshiaki Nemoto. Detection, defense, and tracking of Internet-wide illegal access in a distributed manner. In *Proceedings of INET'2000*, Yokohama, Japan, 2000.

[Onl01] USA Today Online. Online anonymity tool to shut down. Available at `http://www.usatoday.com/life/cyber/tech/2001/10/5/{anonymous-emailer-sh%utdown}.htm`, October 2001.

[Opp96] Rolf Oppliger. *Authentication Systems for Secure Networks.* Artech House Publishers, 1996.

[oS01] United States Department of State. 2001 country reports on human rights practices. Available at `http://www.state.gov/g/drl/rls/hrrpt/2001/`, 2001.

[Par98] Donn B. Parker. *Fighting Computer Crime.* Wiley, 1998.

[Pax97] Vern Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions On Networking*, 5(5):601–615, 1997.

[Pax00] Vern Paxson. Reflectors talk. Talk at the CERIAS Attack Traceback Workshop, October 2000.

[Pax01] Vern Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communication Review*, 31(3), July 2001.

[PD96] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[pen96] Temporary injunction in the anonymous remailer case. `http://www.penet.fi/injuncl.html`, August 1996.

[PL01a] Kihong Park and HeeJo Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, 2001.

[PL01b] Kihong Park and HeeJo Lee. On the effectiveness of route-based packet filtering for distributed for DoS attack prevention in power-law internets. In *Proceedings of ACM SIGCOMM'01*, San Diego, California, 2001.

[Pos81a] J. Postel. RFC 791: Internet Protocol, September 1981.

[Pos81b] J. Postel. RFC 793: Transmission control protocol, September 1981.

[Pos82] J. Postel. RFC 821: Simple mail transfer protocol, August 1982.

[Pos94] J. Postel. RFC 1591: Domain name system structure and delegation, March 1994.

[PR83] J. Postel and J. K. Reynolds. RFC 854: Telnet Protocol specification, May 1983.

[Pro99] Lucent Personalized Web Assistant Project. Available at `http://www.bell-labs.com/projects/lpwa`, 1999.

[Pro02] The Honeynet Project. Know your enemy: Passive fingerprinting. Technical Report Available at `http://project.honeynet.org/papers/finger/`, The Honeynet Project, March 2002.

[Ran99] Marcus Ranum. Some tips on network forensics. Computer Security Alert, September 1999.

[Row99] Jeff Rowe. Intrusion detection and isolation protocol: Automated response to attacks. Presentation at RAID'99, September 1999.

[RR97] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. Technical Report 97–15, DIMACS, April 1997.

[RSG95] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Proxies for anonymous routing. In *12th Annual Computer Security Applications Conference*, pages 95–104. IEEE, December 1995.

[SC95] S. Staniford-Chen. Distributed tracing of intruders. Master's thesis, University of California at Davis, 1995.

[SCH95] S. Staniford-Chen and L.T. Heberlein. Holding intruders accountable on the Internet. In *Proc. of the 1995 IEEE Symposium on Security and Privacy*, pages 39–49, Oakland, California, May 1995.

[Sch96] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, second edition, 1996.

[Sch97] Christoph L. Schuba. *On the Modeling, Design and Implementation of Firewall Technology.* PhD thesis, Purdue University, West Lafayette, Indiana, May 1997.

[Sch02] Bruce Schneier. Why cryptography is harder than it looks. Technical Report Available at `http://www.counterpane.com/whycrypto.html`, Counterpane Labs, 2002.

[SDS00] Dan Schnackenberg, Kelly Djahandari, and Dan Sterne. Infrastructure for intrusion detection and response. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX) 2000*, Hilton Head, South Carolina, January 2000. IEEE.

[Sec] SSH Communications Security. Secure shell software. `http://www.ssh.com/`.

[SKK$^+$97] Christoph L. Schuba, Ivan Krsul, Markus Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on TCP. In *IEEE Symposium on Security and Privacy*, Oakland, California, May 1997. IEEE.

[SM96] Tsutomu Shimomura and John Markoff. *Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw – By the Man Who Did It.* Hyperion, New York, New York, 1996.

[Sor97] David E. Sorkin. Unsolicited commercial e-mail and the telephone consumer protection act of 1991. In *Buffalo Law Review*, volume 45, 1997.

[SP00] Dawn X. Song and Adrian Perrig. Advanced and authenticated marking schemes for ip traceback. Technical Report UCB/CSD-00-1107, University of California-Berkeley, June 2000.

[SP01] Dawn X. Song and Adrian Perrig. Advanced and authenticated marking schemes for ip traceback. In *Proceedings of InfoCom 2001*, Anchorage, Alaska, 2001.

[SPS+01] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Ken, and W. Timothy Strayer. Hash-based IP traceback. In *Proceeding of the ACM SIGCOMM'01*, San Diego, California, August 2001.

[SS96] Steve Schneider and Abraham Sdiropoulos. CSP and anonymity. In *Proceedings of Fourth European Symposium on Research in Computer Security (ESORICS 96)*, pages 198–218, 1996.

[SS99] Paul F. Syverson and Stuart G. Stubblebine. Group principals and the formalization of anonymity. *Formal Methods '99, Vol. I, LNCS 1708*, 1:814–833, 1999.

[Sta99] William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 1999.

[Sta00] William Stallings. *Data and Computer Communications*. Prentice Hall, 2000.

[Ste94] W. R. Stevens. *TCP/IP Illustrated Volume 1*. Addison-Wesley Publishing Company, 1994.

[Ste98] W. R. Stevens. *Unix Network Programming*, volume 1. Prentice Hall PTR, second edition, 1998.

[Sto89] Clifford Stoll. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Doubleday, 1989.

[Sto99] Robert Stone. Centertrack: An IP overlay network for tracking denial-of-service floods. Presentation at The North American Network Operators' Group (NANOG Meeting 17, October 1999. `http://www.nanog.org/mtg-9910/ppt/robert/sld001.htm`.

[SWKA00a] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 295–306, Stockholm, Sweden, August 2000.

[SWKA00b] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. Technical Report UW-CSE-2000-02-01, University of Washington, 2000.

[Tan88] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, Inc., second edition, 1988.

[TCD] tcpdump network sniffer. Available at `http://www.tcpdump.org/`.

[TJ98] Ed Tittel and David Johnson. *A Guide to Networking Essentials*. Course Technology, Cambridge, Massachusetts, 1998.

[WP89] Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Eurocrypt '89*, Houthalen, Belgium, 1989.

[WRWY01] X. Wang, D. Reeves, S.F. Wu, and J. Yuil. Sleepy watermark tracing: an active network-based intrusion response framework. In *Proceedings Of IFIP Conference on Security*, Paris, France, 2001.

[ws] Anonymizer web site. Available at `http://www.anonymizer.com`.

[WZaM00] S. Felix Wu, Lixia Zhang, and Dan Massey andAllison Mankin. Intention-driven ICMP trace-back. Internet Draft: draft-wu-itrace-intention-00.txt, March 2000.

[YE00] Kunikazu Yoda and Hiroaki Etoh. Finding a connection chain for tracing intruders. In *Proceedings of 6th European Symposium on Research in Computer Security (ESORICS 2000)*, 2000.

[Zim80] H. Zimmerman. OSI reference model–the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, COM-28(4):425–432, April 1980.

[ZP99] Y. Zhang and V. Paxson. Stepping Stone Detection. Presentation at SIGCOMM'99, New Areas of Research, August 1999.

A. APPENDIX

This appendix is based on an excerpt from our paper “Subliminal Traceroute in TCP/IP” [DS00b]. We omit the past work from the paper as it has already been discussed in Chapter 2.

In this appendix, we discuss the problem known as TCP spoofing. We also attempt to motivate our approach to tracing spoofed connections as opposed to preventing them as in the past work.

A.0.1 The Problem

TCP spoofing attacks have been widely discussed by others in the literature [Mor85, Bel89]. Most of these discuss attacks that hinge upon the guessability of initial TCP sequence numbers (ISN) so that an arbitrary host can exploit an address-based trust relationship to establish a client write-only TCP session. The session is client write-only because the server will respond to the claimed IP address of the client which will not be routed to the attacking client, and therefore the client will not receive any of the server’s responses unless it is on the route from server to client. Many operating systems have made this attack much more difficult by using randomly generated initial sequence numbers thereby requiring the attacker to receive at least one packet (SYN-ACK) from the server to carry out the attack.

We consider a less general TCP spoofing attack where guessability of ISN’s is not an issue. The attacker, Mallory, sits between the client, Alice, who he wishes to impersonate, and the target server, Bob. We assume that Mallory is able to read packets from Bob bound to Alice and also cause either the network to drop the packet or Alice to ignore the packet. As shown in Figure A.1, Mallory can then create a TCP connection to Bob while masquerading as Alice. In this case, Mallory may not be exploiting trust relationships as in the past work [Mor85], but instead Mallory may simply be trying to hide his true identity (IP address) from Bob. Physically, Mallory may accomplish this attack by controlling a host on the route between Bob and Alice or by using one of the routing elements between Bob and Alice to redirect TCP packets from a spoofed stream to him.

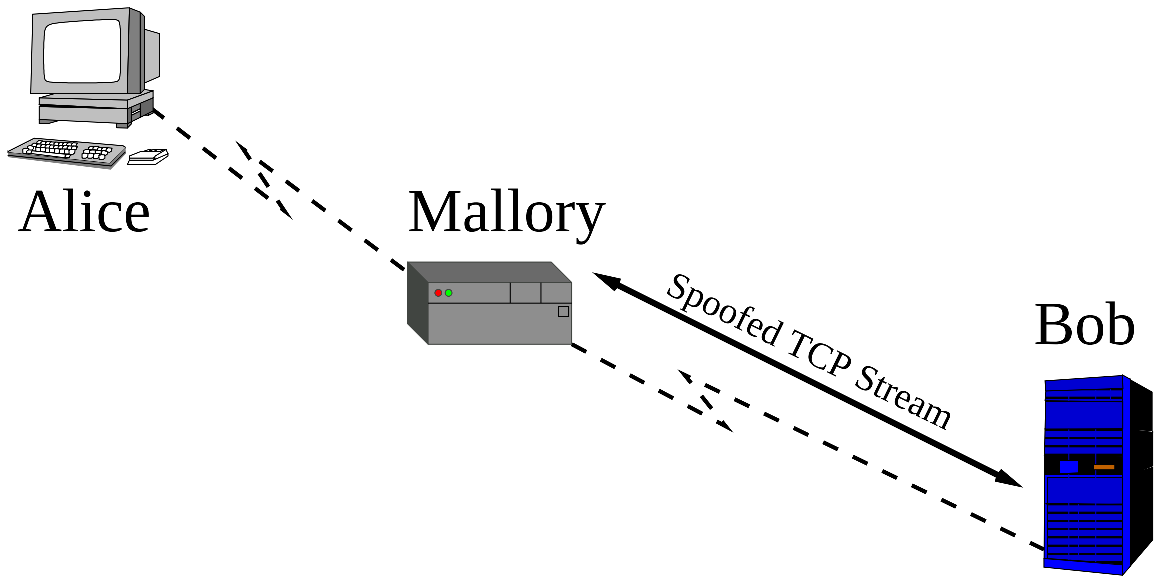

Figure A.1. A TCP spoofing man-in-the-middle attack allows Mallory to create a bidirectional TCP stream with Bob while masquerading as Alice.

A.0.2 Why Trace?

In most networks, an address is usually considered to be the one identifier that uniquely refers to an entity. Unfortunately, in IP addressing this is not necessarily true. For instance, there are private address spaces such as 10.0.0.0 where many disparate networks use those addresses to refer to their hosts and then use network address translation (NAT) to connect to other networks. In this case, even an authenticated address of 10.0.0.1 would be of little use because we would not know the network the host was on. A trace may take us only back to the NAT gateway, but this is still much better than the internal address. Furthermore, NAT may be used in several layers thereby requiring several hops of a trace to uniquely identify a host. We understand that conventional tracing does not penetrate NAT borders, so we use this as an example of how a trace may be preferable to a single address. In our work, we do not claim to address the problem of tracing through NAT gateways, but we include this as an example of the insufficiency of the IP address to uniquely identify a host.

There is a better reason for considering a trace useful. A trace can be wrong but not completely so. For instance, the last few hops of a trace may be wrong or unobtainable, but we may be able to trust the trace only as far back as the attacker's Internet service provider, A trace therefore provides levels of accuracy whereas a wrong address is likely to be completely wrong. The reason for this is if the attacker can falsify his IP address, he may use an arbitrary address instead of one near him. On the other hand, it may be very difficult to subvert enough routers to make them falsify substantial portions of the trace.

A.1 Our Approach

Our approach is to build a traceroute-like [Mal93] mechanism into an active TCP stream so that a server can collect traces by modifying an active TCP stream. This approach requires no modifications to the network infrastructure but has weaknesses of its own as we shall see.

Traceroute works as shown in Figure A.2. By sending UDP packets to a host with successively increasing time to live (TTL) values, we cause the routers along the path to reply with ICMP "time exceeded" messages when the TTL expires on a packet. When the TTL becomes great enough that the host receives the UDP packet, the assumption is that the host is not listening on the packet's destination port and therefore the system replies with an ICMP port unreachable message. This signifies the end of the trace.

An obvious approach to tracing a man-in-the-middle spoofed connection is to simply use traceroute to attempt a trace. Unfortunately, this is easily defeated as the attacker can choose to allow the UDP packets to continue on to their original

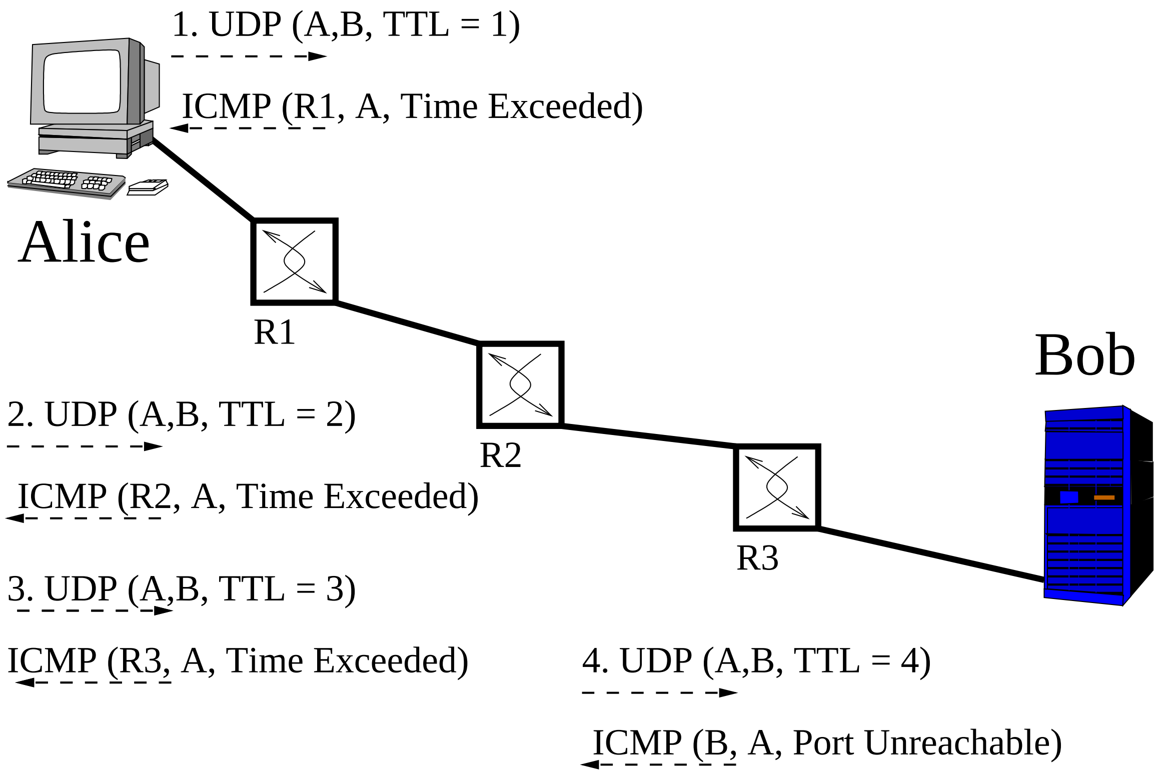
1. UDP (A,B, TTL = 1)
ICMP (R1, A, Time Exceeded)
Alice
R1
R2
R3
Bob
2. UDP (A,B, TTL = 2)
ICMP (R2, A, Time Exceeded)
3. UDP (A,B, TTL = 3)
ICMP (R3, A, Time Exceeded)
4. UDP (A,B, TTL = 4)
ICMP (B, A, Port Unreachable)

Figure A.2. The traceroute protocol works by sending UDP packets with successively greater TTL's to the target host.

destination. In this case, the route will lead to the host that is being imitated, not our attacker. For instance, an attacker may defeat traceroute by filtering the traffic based on whether it is UDP or TCP.

Our approach is to perform a traceroute inside a TCP stream so that if the attacker is redirecting TCP packets headed for a spoofed client or consuming the TCP packets prematurely, the trace will indicate it and presumably lead us to the attacker. We do this by setting the TTL low on every other ACK sent by the TCP state machine. When a low TTL ACK times out, the remote router responds with a ICMP time exceeded packet that is then captured by the host. We allow TCP to compensate for and re-send the lost ACK as part of its reliable service.

Note that it is more difficult for our attacker to simply ignore the packets in our mechanism because they are actually part of the TCP stream in which we want to participate. The only distinguishing characteristic of the packets used in our scheme is an occasional TCP acknowledgment with a low TTL. If the attacker chooses to ignore low TTL TCP packets that are part of the stream, he may ignore the stream itself. Of course, this is dependent on the operating system of the remote host and the length of the route or routes taken by the traffic.

A.1.1 Design and Implementation

The ST system was designed around the socket API in the Linux 2.2.10 kernel. Our additions to the socket API allow user mode processes to configure which TCP sockets to trace, and the kernel socket structures provide a convenient abstraction for storing state about each trace in progress.

To allow processes to specify which TCP sockets to trace, we added a new socket option, SOCK_SUBLIM,to the `setsockopt` system call. By setting SOCK_SUBLIM to 0 (the default), no trace is done on the socket. Setting SOCK_SUBLIM to 1 will enable the tracing functionality, which we will soon describe. Other values are allowed so that different trace strategies may be added. For instance, if the TCP stream is expected to be short-lived, a more aggressive trace style may be activated. We have modified a version of SSH [Sec] to set this socket option.

To implement ST, we modified the portion of the TCP implementation that sends acknowledgments. To this part of the code we added a simple state machine (shown in Figure A.3) that dictates the TTL set on each acknowledgment sent by the system. Each edge has two lines associated with it—the top line is the firing condition for the edge and the bottom line is the action taken when the edge is traversed. Note that each edge has an implicit firing condition: a TCP ACK is ready to be sent.

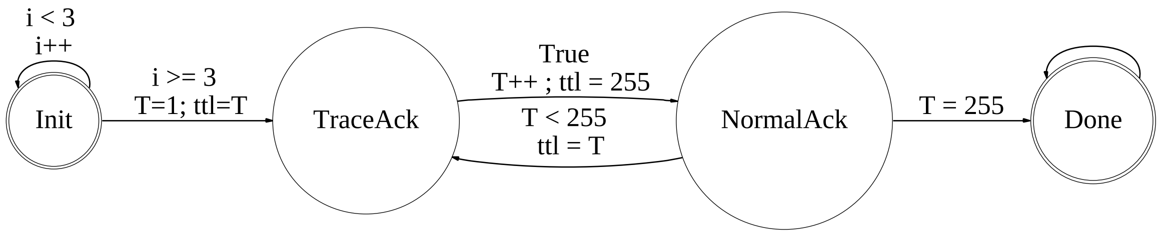

Figure A.3. The finite state machine that determines the behavior of the subliminal traceroute. The top line of the edge label represents the firing condition while the bottom line represents the action taken during the transition. *i* and *T* are per socket variables and initialized to 0 at time of socket creation.

The values of most of the constants in the state machine were chosen somewhat arbitrarily. We let 3 ACK's go by unmodified so that there is sufficient traffic for the TCP handshake to proceed as normal, and the TCP stream will get established. After this initial stage, the first ACK is sent with a TTL of 1, and every other ACK is then sent with the next higher TTL value. It is important to note that the variables referenced in the state machine are kept separate for each socket in its own data structures. The value of 255 was used as it is the maximum value for a TTL in IP.

The resulting ICMP time exceeded messages are then collected with Tcpdump [TCD]. The source addresses of the ICMP messages constitute the trace between the server and the client. Packets from multiple simultaneous streams can be distinguished by looking at the message's contents to find the triggering TCP packet's addresses and ports.

A.1.2 Tradeoffs

There are many tradeoffs in the design and configuration of ST. The state machine controlling the trace can be modified to do a more aggressive trace by reducing the initial waiting period and sending multiple low-TTL packets at once instead of for every other ACK. Also, the state machine could be modified to do many low-TTL ACK's at first so that the trace is performed aggressively.

We chose not to send ACK's in parallel with the TCP stream because it might have interfered with the TCP state machine by causing side effects in the kernel code. However, doing so would allow for a fast trace instead of the current implementation which requires $numberofhops + 3$ acknowledgment cycles to occur to complete the trace. This might come into play if one was tracing HTTP requests where there may be few acknowledgments sent back and forth, and hence too few ACK's may be exchanged for the trace to complete. Our system works fine in most interactive login sessions because each character sent by the client calls for an acknowledgment.

Other design tradeoffs involve implementation details such as our choice to modify the kernel itself. It should be possible but more difficult to create a kernel module that links between the driver module and the IP stack that performs a subliminal trace. In this case, it might be more difficult to implement control over which TCP streams are traced.

Another problem with ST is that there is no guarantee of correctness of the trace. For each TTL value, we only send one ACK, and therefore if any ICMP time exceeded packet is lost, the trace will be missing a host. Also, time exceeded messages may return out of order causing the trace to have certain hops listed in the incorrect order.

A.2 Results

We have implemented the ST system and demonstrated that it works in an SSH daemon modified to enable tracing. After a login, each packet sent by the client causes the next hop of the trace to be captured by a script running Tcpdump. By sorting the ICMP time exceeded messages by the destination address found in the packet payloads, we can reconstruct a trace for a given peer even if multiple traces occurred simultaneously.

A.2.1 Defeating the Trace

Unfortunately, we also discovered a mechanism by which the attacker can defeat the ST mechanism. To do this, we capture all IP packets with TTL equal to one, drop the packet, and reply with an ICMP time exceeded message with source address spoofed to appear to come from some other router. Similarly, we can do this for all packets with TTL $\leq n$ to simulate the last n hops of a faked path.

As a proof of concept, we modified the same kernel used to implement ST to simulate fake routers that immediately precede the host. We changed the IP layer of the kernel so that IP packets with TTL below some threshold were dropped and an ICMP time exceeded message was sent in response. One instance of this technique replied to $TTL = 1$ packets with ICMP messages apparently from a host at the National Security Agency. Similarly, $TTL = 2$ packets appeared to time out at a Central Intelligence Agency host. This made the last two traceroute hops preceding the modified host to appear rather comical. Of course, a real hijacker could use this technique to fake the trace between himself and the host he is impersonating.

If implemented naively, it is possible to detect this faked route. As the same host is simulating a number of routers to send false timeouts, the delay times observed in a traceroute command should be nearly the same. To overcome this, a robust implementation would simulate the increasing latency of successive routers along the path. This could be done by adding variable delays based on path measurements, but it would greatly complicate a kernel implementation as it would require scheduling the sending of ICMP time-exceeded messages.

A.3 Conclusions and Future Work

We have demonstrated a subliminal traceback technique in TCP streams for tracing “man in the middle” attacks to their source. Although we have also found a way for the attacker to defeat the trace, we still believe it raises the bar for a would–be attacker as he would have to either drastically modify his host by modifying the kernel or drivers to reject low-TTL'd TCP packets or by somehow intercepting low TTL'd packets before they arrive at the host.

For future work, we believe that by measuring hop times on a route using conventional traceroute and then comparing them with times for a subliminal traceroute, we can make it more difficult to fake a trace as described above. Another possibility for future work involves continually doing a subliminal traceroute in a TCP stream with the goal of detecting TCP hijacking attacks.

VITA

VITA

Thomas E. Daniels was born in Cameron, Missouri in 1972. He grew to adulthood on a farm near Cameron and graduated from Cameron High School in 1990. Thomas continued his education at Southwest Missouri State University in Springfield, Missouri completing his B.S. degree with special distinction in computer science in 1995.

Thomas was a software and systems engineer for one year at a United States government contractor in Washington, D.C. before returning to academia to pursue the doctorate. Thomas received his M.S. in computer science in 1999 while an advisee of Eugene H. Spafford. Upon completion of his Ph.D. at Purdue University, Thomas joined the faculty of Iowa State University in the Department of Electrical and Computer Engineering.

Thomas has a wonderful, supportive wife, Dr. Jennifer L. Walker-Daniels, and two wonderful children, Zoe Elizabeth and Thomas William.