Distributed Tracing of Intruders

By

STUART GRESLEY STANIFORD-CHEN
B. Sc. (University of Sussex) 1988
M. S. (University of California, Davis) 1990
Ph. D. (University of California, Davis) 1993

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

_____

_____

Committee in Charge

1995

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1995		**00-00-1995 to 00-00-1995**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Distributed Tracing of Intruders	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California (Davis),Department of Computer Science,1 Shields Avenue /2063 Kemper Hall,Davis,CA,95616**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**84**	

Acknowledgments

Many people helped me as I worked on this thesis, and as I made my transition from Physics to Computer Science. Karl Levitt was my thesis advisor and provided inspirational leadership and guidance throughout. He was also the person who persuaded me to work in Computer Security in the first place. Matt Bishop shared his wide knowledge of Unix Security and the literature of the field with me on many occasions. Biswanath Mukherjee provided insightful and constructive critiques of this work at many stages in its evolution. Norm Matloff was very encouraging and helpful when I was initially considering the move over to Computer Science.

Todd Heberlein had the original idea of thumbprinting, collaborated with me on parts of this project, wrote much of the code used in the experiments, and has been a friend and ally throughout the process - I greatly appreciate his help.

I enjoyed a number of creative brainstorming sessions at the whiteboard with Jeremy Frank. Chris Wee introduced me to Perl (without which language this thesis would have taken several months longer), and answered my many questions about it.

On the personal side, I thank the residents of N St Cohousing Community for providing me with a fascinating place to live. My father Geof proved to me that it is never too late to change fields by taking up a career as an Artificial Intelligence researcher in his mid-forties, after spending the first part of his working life as a local government bureaucrat. As in everything I do, the love and support of my wife Lynnette was critical to success in this project.

Finally, I thank ARPA for funding this research.

Abstract

Unwelcome intrusions into computer systems are being perpetrated by strangers, and the number of such incidents is rising steadily. One of the things that facilitates this malfeasance is that computer networks provide the ability for a user to log into multiple computer systems in sequence, changing identity with each step. This makes it very difficult to trace actions on a network of computers all the way back to their actual origins. We refer to this as the *tracing* problem.

This thesis attempts to address this problem by the development of a technology called thumbprinting. Thumbprinting involves forming a signature of the data in a network connection. This signature is a small quantity which does not allow complete reconstruction of the data, but does allow comparison with signatures of other connections to determine with reasonable confidence whether the data were the same or not. This is a potential basis for a tracing system.

The specific technology developed to perform this task is *local thumbprinting*. This involves forming linear combinations of the frequencies with which different characters occur in the network data sampled. The optimal linear combinations are chosen using a statistical methodology called principal component analysis. The difficulties which this process must overcome are outlined, and an algorithm for comparing the thumbprints which adaptively handles these difficulties is presented.

A number of experiments with a trial implementation of this method are described. The method is shown to work successfully when given at least a minute and a half of reasonably active network connection. This requires presently about 20 bytes per

minute per connection of storage for the thumbprints.

In addition, the existing (very limited) literature on the tracing problem is reviewed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Networked computer systems are under attack, and the number of attacks is growing exponentially. In 1990, 252 incidents were reported to the Computer Emergency Response Team (CERT). In just the first six months of 1994, that number had grown to 1172. In addition to the growth in the number of reported incidents, the number of systems involved per incident is growing - one recent incident involved 65,000 systems.[1]

Furthermore, it seems probable that most incidents are not detected or reported. For example, ASSIST, the Department of Defense incident response team, recently evaluated the security level of one of their sites by launching automated attacks against it continuously for two months. Only one person reported suspicious activity. In a second example, a particularly security conscious DoD site detected 69 attacks in 1992. After installation of an intrusion detection tool, they detected 4100 attacks in just the first quarter of 1993.[2]

Why are so many attacks occurring? Studies reveal computer attacks have similarities with many other crimes: perpetrators have many motives, including greed, revenge, the thrill of the chase, and peer pressure.[3,4] As the Internet continues to grow, and as more and more commercial activity takes place over it, it would seem likely that the problem will continue to worsen.

Studies also suggest that many intruders are deterred by the perceived risks in-

volved. One of the intruder's greatest fears is losing his or her anonymity.[4]

Unfortunately, attackers can take advantage of the architecture of the Internet to hide their point of origin, thus preserving their anonymity. Since many hosts are insecure, intruders assemble a collection of accounts on hosts around the world that they have broken into. When conducting an attack, they log-in through a series of such hosts before assaulting the target. Since the machines in question are in different administrative domains, with personnel who may not know or trust one another in advance, and perhaps do not even have the same legal system, this makes it extraordinarily difficult to trace back the chain of activity to its source. Clifford Stoll's experience is a good example.[5]

Because of these problems, most incident response teams such as CERT make little or no effort to find the intruder. The result: an intruder still has all the potential rewards with almost no risks.

The goal of my research in this area is to develop means by which intruders can be traced efficiently. An important restriction which is imposed on the approaches considered is that they can be retrofitted to the existing Internet. Thus, I do not consider methods which would require changes to the low-level network protocols, or require a verified trusted computing base.

This means that the methods produced will certainly be imperfect. It is impossible to produce a foolproof method to give correct security information when the scheme must be implemented on hosts and networks which have insecure operating systems and insecure protocols. Nonetheless, I feel that some tracing ability, however imperfect, is better than none at all.

1.2 Definitions

We here define some terms which will be used throughout this thesis.

When a person (or a program) logs into one computer, from there logs into another, and another, via network connections or modems, we refer to that as an **extended connection**, or a **connection chain**. We refer to the sections of the chain nearest to the source of the activity as the *upstream* parts of the chain.

An **identification service** is a service which claims to identify which human is responsible for any particular activity on a computer or network. We are mainly interested in network-wide identification services

In general, identification mechanisms fall into two classes. In the first class are methods which attempt to keep track of all individuals on the network and account all activity to network wide user-ids (either fixed user-ids or ones dynamically created at the time users appear on the network).

The second class contains reactive **tracing** mechanisms. In this case, no global accounting of users is attempted until a problem arises. Then the activity is traced back to its source. Thus, the main task of a tracing mechanism is, given some part of a connection chain, to identify the beginning of it.

A **security domain** is the set of machines and networks which are operating the currently discussed security system.

The word **hacker** has several meanings. For the purpose of this thesis, we only use the meaning of a malfeasor attempting to penetrate or misuse a computer facility. We interchangeably use **attacker** or **intruder** to mean the same thing.

1.3 Organization

Chapter 2 of this thesis surveys what work has already been done on the problem of tracing of intruders.

Next, Chapter 3 attempts to characterize the nature of the tracing problem and what solutions are possible. Then thumbprinting is introduced, as a concept, and the desireable properties of thumbprints are explained. Chapter 4 provides a more detailed consideration of possible ways to implement thumbprinting, and the specific way used here. Chapter 5 expands on one aspect of this – the use of principal component analysis to choose the thumbprint function. Chapter 6 describes the experiments performed to date to determine the success of this scheme.

Chapter 7 considers other applications for thumbprinting besides tracing network connections.

Finally, Chapter 8 discusses what conclusions can be drawn from the work to date, and what work remains to be done.

Chapter 2

Related Work

2.1 General Background

The topic which this thesis concerns has received little attention in the literature. Here we review what is known about it.

Firstly, several works describe the exploits of particular hackers and the process of tracking them down. These provide motivation for the work presented here. Clifford Stoll spent months tracking down a particular hacker (who proved to be in the pay of the KGB). A popular account is available,[5] as is a more technical account.[6] Another perspective on this story is a book by Markoff and Hafner,[7] which also details several other incidents.

Several sources written by members of the intruder community describe their view of their practices and various case studies (of uncertain veracity for the most part). Some of the more accessible include the electronic journal Phrack[8] and the magazine 2600.[9] Also worthy of mention is a book intended as a how-to manual for intruders.[10] These sources tend not to give detailed technical information on tradecraft, but rather general advice on tactics (and a fascinating glimpse of the culture). One piece of advice that is frequently re-iterated is for hackers not to use their own accounts when carrying out some misdeed.

From another direction, there are a number of systems which seek to provide distributed identification and authentication by ensuring that all hosts in some administrative domain have access to the same database of user-ids. A simple example is

the NIS (NetworΓ Information System) developed by Sun Microsystems. NIS is used at many sites to maintain centralized records of users.

Another, much more sophisticated, example is Kerberos.[11] Again, the system maintains a central concept of who every user is. Each user has to prove his identity to the system (through cryptographic protocols) before being able to carry out many activities (such as logging in to one of the hosts covered by the system).

However, in many cases, it is impractical to have all the machines in the security domain shara a common user namespace. Thus a number of systems have attempted to provide a distributed identification service, or an ex post facto tracing facility, and we now turn to those. All of these systems have only been deployed on an experimental basis.

2.2 DIDS

The Distributed Intrusion Detection System (DIDS) was initially developed at UC Davis where a prototype was built.[12] Subsequent work on the system has been done by Trident Data Systems. One of the tasks DIDS performs is to keep track of user movements, at least within the domain which the system is monitoring. Specifically it attempts to track all TCP connections and all logins on the network. It maintains a notion of a Network Identifier (NID) at all times for all activities on the system. If a user logs in from one system to another, activity on both systems will be accounted to the same NID. To do this DIDS uses a Distributed Recognition and Accountability (DRA) algorithm.[13]

The way DRA works is as follows. Each monitored machine collects an audit trail which is analyzed by a Host Monitor residing on that particular host. The host

monitor abstracts certain features of this record and ships them to a central DIDS Director for analysis. Events relevant to tracing are the following:

Connection start	CS(saddr, daddr, suid, ts)
Connection accept	CA(saddr, daddr, ts)
Session start	SS(saddr, daddr, duid, ts)
Fail login	FL(saddr, daddr, ts)
Connection end	CE(saddr, daddr, ts)
Session end	SS(saddr, daddr, duid, ts)
Activity record	AR(host, uid, activity, ts)

Here, *saddr* and *daddr* are the source and destination addresses, *suid* and *duid* are source and destination user-ids on the respective systems, and *ts* is a timestamp.

The algorithm involves maintaining a graph where each node corresponds to a particular pair of (host, user-id), and each edge corresponds to a connection between the user-ids on the hosts in question. As events of the kind listed above arrive at the Director, the graph is updated appropriately. The algorithm becomes complex because events may arrive out of order – details and a proof of correctness can be found in the references.[13]

DIDS is capable of reliably keeping track of all users moving around on the network through "normal" means. It has some limitations however – it does not examine User Datagram Protocol (UDP) traffic or asynchronous movement of data through such means as files. Perhaps its biggest limitation is that DIDS only can account activity as long as the activity stays within the DIDS domain. A user logging into an external machine and then logging back into the DIDS domain can achieve anonymity. Users

coming from outside are only traced back to their first incarnation on a machine in the domain.

Finally, as with any such system, if parts of the software infrastructure are replaced with Trojan horses, its effectiveness can be reduced.

2.3 Caller Identification System

A system which is a hybrid between a distributed identification system and a reactive tracing system is CIS (Caller Identification System).[14] This system is invoked in an attempt to authenticate users about to log into a machine at the end of an extended connection. Each machine along the chain keeps a record of what the chain looks like upstream of it. When the user attempts to log into the nth machine from the $n-1$th machine, the nth machine asks its predecessor for information about the chain so far. The nth machine then queries machines 1 through $n-2$ to check that their impression of the connection chain agrees with that of machine $n-1$. Only if all machines along the chain agree (and machine 1 is acceptable to machine n) does the login proceed.

This recursive checking of the chain eliminates some, but not all, of the obvious attacks on this kind of scheme. For example, suppose an attacker logs in to machine $H_1, H_2, \ldots, H_n$, and suppose further that he subverts the system on machine H_k to lie to anyone who asks and say that he came from machine J_{k-1} instead. If he has also subverted J_{k-1} so that it reports that he is logged on to the console, then he has successfully duped the system. It will report that he is coming from J_{k-1} when he is in fact located on H_1. No information that would enable the true location of the hacker to be obtained will then be available, and nothing will indicate to the observer that there is a problem.

We also note, as with DIDS, that the system only provides any authentication service as long as all parts of the connection chain are within the security domain. Finally, the number of queries required by this system is quadratic in the length of the chain which could cause performance problems in some instances.

2.4 Caller-ID

A different approach was actually used by the United States Air Force to track an intruder and arrange for his arrest.[15] This technique, called Caller ID, is controversial and required special permission from the Department of Justice, so it is probably not a technique for general use.

Caller ID is based on the belief that if an intruder hops through intermediate systems prior to making an attack, there is a high probability that these systems have known vulnerabilities which the intruder used to access them. For example, if the intruder hops through $H_0 \rightarrow H_1 \rightarrow \ldots \rightarrow H_n$, where H_n is the target, then H_1 through H_{n-1} contain at least one vulnerability allowing access by an outsider. The Air Force, having knowledge of the same attack methods that their intruder did, simply reversed the attack chain - breaking into H_{n-1}, examining the system tables to see from where the intruder was coming, breaking into H_{n-2}, and so on. Eventually they identified the original point of entry of the perpetrator.

The drawbacks of this tracing technique include the possibility that one cannot break into one of the intermediate systems, one must perform the tracing while the intruder is active, and one runs the risk of accidentally damaging intermediate systems. For many practitioners, the legal situation is likely to cause severe problems also.

Chapter 3

Characterization of the Problem

3.1 Nature of the problem

The problem which this thesis addresses is that of how best to perform reactive tracing on a TCP/IP internet with multiple administrative domains. In general, there are two obvious places to put components of a tracing system - on the hosts in the domain, or attached to the network. Network solutions can either be part of the infrastructure such as routers, or systems passively monitoring the network for security purposes and performing no other function.

Firstly, we consider host-based solutions. These involve one tracing system per network host. Each such system is capable of establishing where a chain that crosses *it* goes next, and tracing is accomplished by the hosts communicating in some way to establish the whole extended connection. The CIS system described in Chapter 2 is an example of this approach. I also developed a system called Foxhound, using similar ideas but with an emphasis on ex post facto tracing rather than authentication. This is described in the next section.

The difficulty with all such host-based tracing systems is that, when an extended connection crosses a host which is not running the system, accountability is altogether lost at that point. This severely limits their usefulness as a general purpose tracing mechanism on an internet. Since many hosts on internets are not secure, the integrity of the tracing system on those hosts cannot be relied upon. Intruders almost reflexively install Trojan horse versions of important system binaries on hosts they

penetrate. Even if most hosts could be secured, the intruder community could easily maintain a set of machines to launder connections, just as they maintain anonymous remailers which allow the origin of email to be disguised.

Another different class of approaches is based on *thumbprinting*. This relies on the fact that the content of an extended connection is invariant at all points of the chain (once protocol details are abstracted out). Thus if the network tracing system can compute summaries (thumbprints) of the content of each connection, these summaries can later be compared to establish whether two connections have the same content. The technical feasibility of this idea will be discussed later in the paper. The main limitation of this approach is that it is still vulnerable to countermeasures. Firstly, the system must still be protected from Trojaning, though this is perhaps easier to do since there are fewer stations involved and they can be special purpose with most parts of the operating system removed. The second weakness is that disguising the content of the extended connection (such as encrypting it differently on each link of the chain) can circumvent the technology.

By far the most compelling advantage of the thumbprinting approach is that it could be useful even when only parts of an internet use it. For example, if only one site and the backbone networks run the system, the one site can already deduce useful information about where an attack is coming from (by comparing the thumbprints of the connection of interest at its site with all thumbprints in the same timeframe on the backbone). We note that since the thumbprints are very small, it is usually impossible to deduce details of the connection content from them. This limits their impact on privacy to traffic analysis.

3.2 Foxhound

In this section, we describe our first prototype system to do host-based tracing. It runs under several variants of the Unix operating system.

The system works by investigating the system tables of the machine in question. The algorithm used is as follows. A server process runs on each of the machines that are part of the tracing system. The tracing process is initiated by an administrator running the client program on one of these machines. The client program is told the process id to investigate.

The client then queries the server on its own host, supplying it with the target process id. The server then traces back the parent of that client, the grandparent of that client, *etc*, all the way back to the *init* process. For each such process, it supplies the client with information about the process, including any open TCP connections which that process has.

The client processes this information, and for every TCP connection supplied to it, it queries the server on the remote end of that connection. It provides that server with the port and address information associated with the connection. The server examines the process table to map that information to a process number. It then repeats the same procedure of finding the parent and grandparent of that process, associating any open connections with them, and supplying the resultant information back to the client.

The client keeps querying servers as long as it can until it has exhausted each particular chain. This can happen in one of two ways - either the client gets information back from a server in which no parent process was associated with a connection, so the tracing can go no further and the client assumes the activity began on that host. Alternatively, the client may query a machine which does not respond, or responds

unintelligibly. The client assumes that this is because the machine in question is outside the tracing system and thus does not have a tracing server operating. It then counts the activity as having originated on that machine (since again, tracing can proceed no further).

The system does not examine the process table for itself. It uses the standard Unix *ps* command and a public domain utility called *lsof* developed by Vic Abell of Purdue University.[16] The latter gives a listing of all open file and socket descriptors associated with each process. Included is enough information to determine the remote end points of any TCP connections which the process has open.

In tests, the system showed itself capable of tracing a straightforward chain of rlogins and telnets back to their source (or the point when the chain entered the set of machines on which Foxhound was being run). The process typically takes a second or two per host in the chain.

Although the first version of Foxhound works well, experience gained with it suggests a variety of possible improvements which we are working on. Specifically, we plan to change the protocol to a recursive one in which the client only contacts the first server which contacts the next one on its behalf and so on. This allows for more administrative flexibility in how much information the client receives. Another improvement is to use a more robust mechanism for connecting processes on a given machine. Presently, we assume that information only flows between parent and child processes. The next version will consider information to be flowing between any processes which share an open file, device, or pipe.

3.3 Thumbprints - idea

The idea of a thumbprint, which was originally proposed by Heberlein et al,[17] is a small quantity which effectively summarizes a certain section of a connection.

In the simplest scenario, suppose an intruder is logged in through a chain of machines, A, B, C, D. Every time the intruder presses a key, that causes a TCP segment to be generated and sent from A to B. B quickly processes the character and sends it on in a new segment to C. C similarly unwraps the character and rewraps it in another segment as part of the connection between C and D. D finally gets the character and acts on it appropriately. Of course, in between each pair of machines shown here, the TCP segment is put into an IP datagram which traverses multiple networks and routers which are transparent to the TCP protocol. Similarly segments may be transmitted several times as part of the error correction protocols. However, allowing for this, the data in the extended connection is the same at all points along it. Thus, there is hope that a suitable summary of the data can later be be used to reconstruct the chain of connections.

The ideal is a function of the connection which uniquely distinguishes a given connection from all other unrelated connections, but has the same value over two connections which are related by being links in the same connection chain. If all components of the system routinely store thumbprints, then in the event of an intrusion being detected, it is possible to trace the connection back by comparing thumbprints from different hosts or networks.

In the short term, there are several applications in which this kind of technology could be deployed almost immediately.

1) In the context of distributed intrusion detection systems such as DIDS,[12] thumbprinting could allow the system to relate activity which went outside the domain but then

re-entered. This might be important when an inside attacker was seeking to disguise himself as an outsider. Indeed Trident Data Systems in conjunction with the Air Force Office of Information Warfare is presently incorporating these ideas into DIDS.

2) Thumbprinting systems could be placed at the places where a network for some site touched other networks. This would allow the administrators of that site to determine whenever their systems were being used as a pass-through site..[18]

3) Sites which were logically a single site, but physically several networks, could use this means to correlate activity between the different sites.

4) Law enforcement in pursuit of particular intruders could use this technology at a variety of places which were under suspicion as the likely source of an intruder.

In the longer term, this technology could become a useful component in a general internet tracing system (akin to the trap-and-trace facility provided by the phone networks). No such facility is presently planned. However, as computer networks become increasingly used for commerce, it may become necessary.

3.4 Thumbprints - desireable properties

A good thumbprint should have the following properties.

1) It should require as little space as possible to minimize storage needs for logs of thumbprints.

2) It should be sensitive; the probability that two unrelated pieces of connection will be close together in thumbprint space should be as small as possible. Of course, if two unrelated pieces of connection happen to have the same content then no thumbprint will distinguish them. The most common case of this is idle connections.

3) It should be robust, *i.e.*, it should change as little as possible when the connection gets distorted by the kinds of errors that are likely in practice. We consider the likely sources of error in the next section.

4) Ideally, thumbprints should be additive. This means that successive ones can be combined into a thumbprint for a longer interval. Thus, when successive thumbprints do not provide a clear comparison, they can be combined to produce a better signal. It also allows thumbprints of intervals of different but congruent lengths to be compared.

5) Finally, it is essential that creating the thumbprints not place an excessive load on the network components. It is useful but less important if they are cheap to compare.

3.5 Sources of error

We have identified the following sources of error. Any thumbprinting scheme must cope with these.

1) Clock skew. Thumbprints on different hosts may not always start at quite the same time, and may not end at quite the same time either. This causes errors in comparing them since characters that in one place are incorporated into the nth thumbprint of a connection may be in the $(n+1)$th thumbprint elsewhere.

2) Propagation delays. Thumbprints may contain slightly different data in different places because the connections they are measuring are delayed due to signal propagation. This has a very similar effect to clock skew in moving some characters from one thumbprinting interval to the next. The worst problems are created by overloaded hosts, rather than by the network itself. Badly overloaded hosts may pause for seconds or tens of seconds before transmitting data they have received.

3) Loss of characters. Since thumbprinting is based on passive monitoring of connections rather than being a party to them, the system cannot have access to the error and flow control features of the transport protocol (TCP). Thus it might lose some characters (*e.g.* due to a buffer overflowing) and not be able to recover them. This is a problem in practice.

4) Packetization variation. Thumbprinting at a low level in the protocol stack is made difficult by the fact that packetization, timing of packet transmission, *etc.* are not invariant at different points in the connection chain.

5) Routing effects. Packets could be missing from a connection at one point in the network because some of them have taken a different route through the network. *At present* this is not a practical problem because on the time-scale of typical connections this does not happen often enough to worry about. It can be compensated for with an additive thumbprinting scheme by combining the thumbprints from several locations - though this could become complicated.

6) Timing variations. If the time between successive characters is used as a basis for thumbprints, there will be noise in the comparisons; different TCP segments may take slightly different amounts of time to traverse a particular host or network. Thus the time interval between two characters on one network may be slightly different from that on another.

7) Countermeasures. Characters could be encrypted and then decrypted at various points in the connection chain. Extra characters could be inserted and then removed to confuse timing based measurements. This issue is not being considered for the time being.

Chapter 4

Implementing Thumbprints

In the last chapter, we characterized what properties a thumbprint should have, and what kinds of errors it might be exposed to. We now turn to how thumbprinting might be done.

4.1 Timing vs. Content

There are two obvious attributes of a connection which could be used in thumbprints. One is the content of the connection – the actual sequence of characters transmitted via TCP. The other is the timings of the characters – the succession of inter-character times which arise from the specific rates at which each user types on a given occasion, complete with rapid typings of some words, long pauses for thought etc.

We first discuss the feasibility of timing as at least a partial basis for a thumbprint. The major concern is that variations in propagation delay will function as noise to mask out the "signal" of a particular sessions inter-character or inter-packet timing.

To explore this timing variation, I used *ping*. This is a Unix utility which sends out an Internet Control Message Protocol (ICMP) packet to a chosen host, which is then obliged to reply. The version I used estimates the round trip travel time and reports the results. The data are summarized below in Table 4.1. I worked mainly from *jaya.cs.ucdavis.edu* (a machine in our laboratory), but also from *jeeves* (a machine not attached to any network) and from *axposf.stanford.edu*. Destinations were *localhost* (the source machine itself), *curie.cs.ucdavis.edu* (another machine in

source	destination	N	μ	σ
jaya	localhost	12288	1.0	0.3
jeeves	localhost	36722	1.7	12.3
jaya	landau.ucdavis.edu	36583	4.9	8.9
jaya	curie.cs.ucdavis.edu	36473	6.6	28.4
jaya	gnu.ai.mit.edu	59829	96.9	368.6
jaya	nova.cc.purdue.edu	18797	135.7	88.6
jaya	mersey.csc.liv.ac.uk	20480	188.3	32.5
jaya	chenas.inria.fr	20479	229.6	46.8
jaya	waikato.ac.nz	43006	277.5	88.1
jaya	spiky.rsinet.tn	42848	462.9	163.4
jaya	sztaki.hu	19943	543.8	406.7
axposf	atina.ar	44672	1538.3	1097.7
jaya	atina.ar	44429	1579.6	1133.6
jaya	shakti.ncst.ernet.in	18475	2872.8	1461.5

Table 4.1: *Ping* timings for various machines around the world. Shown are the machine from which *ping* was run, the target machine, the number of observations and the mean and standard deviation of the round trip time (in ms).

the Computer Science Department), *landau.ucdavis.edu* (a machine in the Physics Department), two other machines across the U.S. and a variety of machines around the world.

As can be seen from the table, the variations in timings are disappointingly large. While the local machines took only a few milliseconds to return, and had standard deviations in return time on the order of 10 milliseconds, the wider area tests were much worse. The U.S. tests showed variations of order 100 milliseconds, and the international tests showed standard deviations of hundreds of milliseconds. Now, times between successively typed characters are going to be of order a few hundred milliseconds. For example, typing at 30 wpm (a typical rate for an amateur typist) gives an average inter character time of 330ms. Thus, for international connections at least, using inter-character times as the basis for thumbprints will mean dealing

with noise that is comparable in size to the signal.*

Two points bear further examination, and make the situation appear slightly better. Firstly, one might hope that although the time for packet traversal is rather variable, it is slowly variable, so that while widely spaced packets take very different times to traverse the network, successive packets take very similar times. To assess this, I differenced the time-series of successive round-trip travel times. That is, if the original *ping* times are $t_1, t_2, t_3, \ldots$, then I looked at the time-series $t_2 - t_1, t_3 - t_2, \ldots$. If the times were in fact locally stable then these differences should all be small. The average size of the absolute differences for *gnu.ai.mit.edu* was 12.82ms, and for *nova.cc.purdue.edu* it was 31.18ms. The same quantity for the French machine *chenas.inria.fr* was 33.38ms, while for *spiky.rsinet.tn* in Tunisia, it was 71.75ms. *sztaki.hu* in Hungary had 124.4ms. Thus, the differences between successive times are rather smaller than the standard deviations of the time distributions, but are still non-trivial.

The second point (which actually partially explains the first) is that the probability distributions of times are very non-normal. For example, histograms of the round-trip times for the US-wide machines are in Figure 4.1, while the international machines are in Figure 4.2. Although the detailed shape of the distributions varies a lot they all have some common features. They all are bounded below, rise very rapidly in a narrow peak which contains most of the probability, and then have a very long tail out to extremely large times. Even for connections between machines within my department, it is possible to observe *ping* times of several hundred milliseconds. In all cases, the visual width of the peak of the distribution is much less than the

*Of course the one way variation will not be as large as the round trip variation, but it will be between 50% and 70% of it.

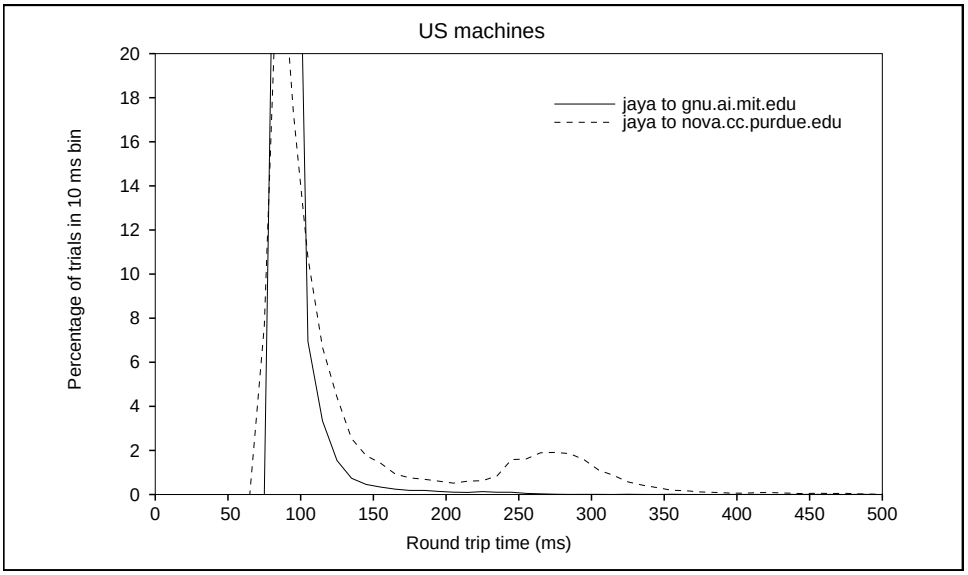

Figure 4.1: Ping distributions for two distant machines in the U.S.

corresponding standard deviation. Thus we see that most of the observed variation in times comes from relatively few very long times, rather than from most of the times being spread wide. This is a somewhat better situation. The situation for international connections might be characterized as typical noise in the inter-character timings of a few tens of milliseconds to a hundred milliseconds or so, together with a few percent of the timings which are rendered worthless by very severe noise. This is still a daunting prospect.

This preliminary analysis is not detailed or realistic enough to categorically rule out the possibility of useful timing-based thumbprints. Nonetheless, it is clear that there is likely to be a considerable problem with noise, and hence I preferred to examine the possibilities of character based thumbprints, which should not suffer from this to nearly such a degree.

Another possibility is to use packetization as a basis for thumbprinting (eg the average size of a packet). Formally, such quantities are not invariant across the extended connection because packetization occurs low in the protocol layering stack.

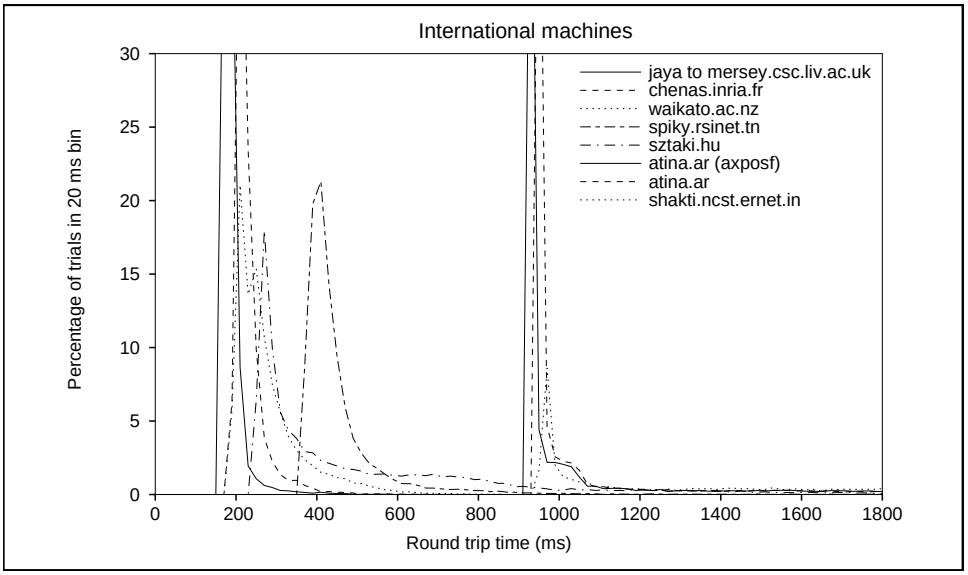

Figure 4.2: Ping distributions for a number of international machines.

Thus I decided to rely solely on the content of the connection after reconstruction up through at least the transport layer. However, it is possible that changes in packetization are sufficiently rare that it is worth *not* reconstructing TCP, but rather relying on the error tolerance of the thumbprinting method.

4.2 Breaking up the character stream

A connection chain may last for a long time. However, not all parts of it typically have equal duration. More upstream portions of the chain are established earlier and end later, and thus contain data which are not present in downstream portions of the chain. Thus it is not wise to attempt to form a single thumbprint for the entire length of some connection – it must be broken up into pieces first.

The only reliable method I currently know of to do this is by time. For example, all the data in the first minute of the connection is thumbprinted, then all the data in the second minute. The disadvantage of this is that geographically separated sites must

be well synchronized in order to be able to compare thumbprints. We refer to the length of time of the interval into which connections are broken as the thumbprinting interval. Failures of synchronization must be small compared to the thumbprinting interval.

I am also studying ways to break up the connection into pieces that do not depend on time, but rather on content based triggers. However, this is not yet achieved. All the work presented in this thesis is based on time divided thumbprints.

A TCP connection supports dataflow in both directions. At present, I lump the data and thumbprint both directions together. This allows for a simpler analysis – it is a topic for future research to determine whether this is an efficient choice or not.

4.3 Thumbprints - rejected candidates

An obvious contender for thumbprints is a checksum such as the Cyclic Redundancy Checksum (CRC). These are very small, they are very sensitive, they are cheap to compute. The big problem is that they are not robust at all – any error in the data used to make the checksum is likely to completely change the value of it. They are also not additive. Message digest algorithms have the same drawbacks.

Other possibilities considered and ruled out due to space considerations were compression techniques, and signature retrieval techniques (as used in the search of large free-text databases).[19]

4.4 Local thumbprints

The scheme I adopted is called *local thumbprints*. The local comes from the fact that the thumbprint is a sum of terms, each of which depends only locally on the character

stream to be thumbprinted.

For the simplest example, suppose the characters that must be thumbprinted are $a_1, a_2, \ldots, a_n$, where each $a_i \in \mathcal{A}$, our alphabet. Further, suppose we have a function $\phi : \mathcal{A} \to \mathcal{R}$ (the thumbprint function - its design is discussed later). Then we can define a thumbprint by

$$T = \frac{1}{n} \sum_i \phi(a_i) \tag{4.1}$$

The advantages of this kind of scheme are as follows. Robustness is good, since if we lose a few characters, only those terms in the sum are affected. Additivity is obviously satisfied in that the thumbprint for a combination of two character sequences is the sum of the thumbprints for the individual sequences (reweighted by the number of characters). The thumbprint is small since it's just a few real numbers (in practice, some quantization of them). It's cheap to compute since the function ϕ can be stored in a lookup table. The remaining question is one of sensitivity - can such quantities effectively distinguish different connections?

Clearly, there exists a possibility that two quite different character sequences could happen to add up to the same thumbprint. A way to improve this situation is to use several independent thumbprints. That is to say, we keep a vector of T_j, where each T_j is defined as

$$T_j = \frac{1}{n} \sum_i \phi_j(a_i) \tag{4.2}$$

The ϕ_j are all chosen independently of each other.

The intuition here is that if two thumbprints are related by the fact that the underlying connection pieces differ by one or two characters, then all of the T_j will differ from their opposite numbers in the other thumbprint by a distance of only one or two units. On the other hand, if some T_j in one thumbprint is close to T_j in the other solely by chance, and not because of any similarity in the underlying connection

pieces, then the chance that the other T_j are also similar is very small.

The question of sensitivity will be addressed empirically later in this thesis.

In essence, equation (4.1) mandates studying linear combinations of the frequencies with which each character occurs in the particular interval of the particular connection being thumbprinted.

4.5 Higher-order local thumbprints

A number of variations on the scheme in the last section are possible. For example, given a function $\psi(a, b)$, we could define a digram thumbprint at some separation k by

$$T_\psi = \frac{1}{n-k} \sum_{i=1}^{n-k} \psi(a_i, a_{i+k}) \tag{4.3}$$

More complex schemes based on trigrams or higher-order combinations are also possible. It might appear that such schemes would be more sensitive than the single character scheme because they capture some information about the order of the characters in (a_i). Ordering information is lost in the single character scheme. I conducted some preliminary experiments which suggested that this makes little difference in practice and so I have focussed on single character schemes. In this section I develop a more systematic description of arbitrary local thumbprints, and then describe the experiments which caused me to cease work on higher-order thumbprints.

We basically separate the problem into two parts. Firstly comes the choice of the *substrate* - whether to use single character based thumbprints, neighboring pairs, pairs at some separation, *etc.* The second part of the problem is, given a substrate, what is the best choice of the thumbprint function ϕ. This second part gets its own chapter (Chapter 5).

Now, we first define the distance between two character sequences $a_1, a_2, \ldots, a_{n_a}$, and $b_1, b_2, \ldots, b_{n_b}$. If we let $n_l = \max(n_a, n_b)$ and $n_s = \min(n_a, n_b)$, then the distance ρ is given by

$$\rho = \sum_{i}^{n_s} (1 - \delta_{a_i, b_i}) + n_l - n_s \tag{4.4}$$

where δ represents the Kronecker-δ. ρ counts the number of characters which are different between the two sequences and thus generalizes the Hamming distance between two bit-vectors. (For simplicity, I assume both sequences start at index 1. If the two sequences are actually shifted relative to one another, it's clear how to generalize the scheme.) Obviously, ρ measures how well we can hope to do with any thumbprinting scheme - if the ρ-distance between two sequences is zero, no thumbprinting scheme will be able to distinguish them. Since we want sequences that really are similar to have similar thumbprints (robustness), we also want the thumbprints of two sequences to be close when the ρ-distance between them is small.

Now let us consider how to represent our thumbprints. We refer to the *substrate* of a thumbprint as a short sequence $I = [i_1, i_2, \ldots, i_k]$. Then our thumbprints will be of the form

$$\sum_{i}^{n-i_k} \phi(a_{i+i_1}, a_{i+i_2}, \ldots, a_{i+i_k}) \tag{4.5}$$

For example, for single character based thumbprints, the substrate will be the short sequence $[0]$. For next-neighbor thumbprints, the substrate will be $[0, 1]$. For pairs separated by four intervening characters, it will be $[0, 5]$.

We now define the degree κ of a thumbprint to be the cardinality of the substrate. Thus single character thumbprints have degree 1, while pair based thumbprints have degree 2. Now any given term in the sum that forms the thumbprint depends on κ characters. Therefore, if the space of characters is $\mathcal{A}$, the domain of a particular thumbprint function is $\mathcal{A}^\kappa$. Hence, given a substrate, we can completely characterize

a sequence of characters to be thumbprinted by the number of times each element in the domain occurs when the substrate is applied over that sequence.

For example, if $\mathcal{A}$ is the set of lower case letters of the alphabet, and the substrate is $[0, 1]$, then we can characterize the sequence, as far as any possible thumbprint function is concerned, by counting the number of times aa occurs in the sequence, the number of times ab occurs, and so on. The important point is that if we know these counts, then we can forget about the original sequence – we have all the information in it which thumbprints based on this substrate can discern. In particular, if two sequences have the same counts of character pairs then no such thumbprint can distinguish them, even though they may be different.

We can formalize this notion. Suppose x is a variable which runs over all the possible points in A^κ. Then we can use x to index the basis vectors in a vector space that will help us to characterize the problem. For example, consider the vector of character counts $\bar{N} = [N_x]$, where each component N_x represents the number of times the character combination x occurs in the sequence to be thumbprinted. For example, if the substrate was $[0, 2]$, then N_{aa} would be the number of times that an a in the input was followed after one arbitrary intervening character by another a.

Now, if we represent by ϕ_x the value that a thumbprint function ϕ takes at position x in its domain, then a thumbprint becomes

$$\bar{\phi}.\bar{N} = \sum_x \phi_x N_x \tag{4.6}$$

Thus a given thumbprint function for this substrate can be represented by a vector in the vector space, and the thumbprint for some particular sequence becomes just an inner product between the thumbprint function vector and the $\bar{N}$ vector for that sequence.

With that in mind, we can define the distance between two sequences with respect

to a particular substrate I as something like

$$\rho_I = \sqrt{(\bar{N} - \bar{N}').(\bar{N} - \bar{N}')} \tag{4.7}$$

The role of ρ_I is to measure the best that any thumbprint over substrate I can do. It is independent of any particular choice of ϕ. Thus we can attempt to use it to compare different substrates, without worrying about the complication of what particular thumbprint functions we use. The results should probably be interpreted with caution though - in some sense it measures the best case performance of the substrate, not the average case.

How does ρ_I relate to ρ? Suppose the two sequences have no characters in common (*e.g.* one is all a_s and the other is all b s). Then $\rho = n_l$, the length of the longer sequence. However, $\rho_I = n_a + n_b$.[†] Thus, for sequences of approximately equal length, $\rho_I \lesssim 2\rho$. However, ρ_I may not nearly saturate this bound.

As an alternative way to think about the problem, consider the effect of small changes to a sequence. Suppose we lose just one character from the sequence, but know that it is lost.[‡] The distance between the old sequence and the new in terms of ρ is just 1. However, the distance in terms of ρ_I is κ_I, the degree of the substrate, since that is how many terms in the thumbprint sum will be affected by the lost character. Thus, other things being equal, shorter substrates are better as they give less response to error.

But are other things equal? To see what happens with these metrics in practice, I used a sample of about one megabyte of my computing activity in a terminal session (using the Unix *script* utility). I took a series of pairs of random samples from this

[†]Actually, for substrates of order greater than one, this is only a large n approximation.

[‡]This would typically be the case because TCP provides sequence numbers for all bytes, so a missing packet shows up as a gap in the sequence numbers.

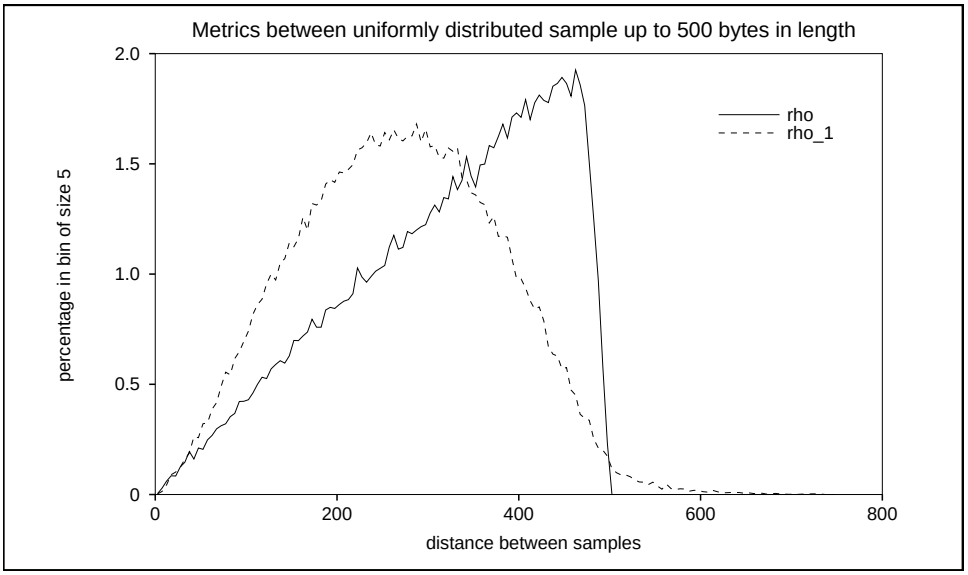

Figure 4.3: Histograms of metrics between random samples.

dataset. The pairs were non-overlapping, and the length of the samples was random with a uniform distribution from 0 characters up to 500 characters. For each pair, I computed the distance between them in the various metrics introduced above. I then made histograms of those distances.

The results of this begin in Figure 4.3. It shows the metrics ρ (solid), and ρ_1 (dotted). The important thing to get from the graphs is the behavior for small distances. It is noticeable that the ρ_1 curve goes to zero in exactly the same manner as the ρ curve does. My interpretation of this is that, by and large, only those pairs of samples which were very similar (*i.e.* close together in ρ) end up close together in ρ_1. This is encouraging as it suggests that, although in principle two completely different sequences could end up with the same counts for various characters, this is improbable in practice.[§] What would have been alarming would have been if the histogram for ρ_1 had tended to a finite probability density at zero distance.

[§]In other words the set of permutations of a sequence is a very rare subset of the set of all the sequences. This is obvious for completely random sequences, but not quite so obvious for human or computer generated ones.

Next come a pair of figures which are the analog of Figure 4.3 for other substrates. In Figures 4.4 and 4.5, $\rho_{2,1}$ the distance for a substrate of $[0,1]$ (pairs of successive characters), is contrasted with $\rho_{2,i}$, (pair of characters at separation i), for various i.

The first, perhaps rather surprising, inference that can be drawn from these figures is that, when comparing by counting character pairs, the separation of the pairs makes almost no difference. Pairs at separation one are slightly poorer than pairs at larger separations, but the difference is too small to be of any practical importance. Once the separation is more than one, no difference in the results can be discerned.

The second thing that can be discerned from these graphs comes from looking at them against Figure 4.3. It is clear that at small distances, $\rho_{2,i}$ is better at distinguishing dissimilar sequences than ρ_1. However, the difference is not huge, and remembering that a change in an isolated character produces twice as much change in $\rho_{2,i}$ as it does in ρ_1, we surmise that this is the cause of $\rho_{2,i}$'s better performance. Since this is an empty gain (because it proportionately affects errors just as much as it affects genuine differences), I conclude that single character counts are just as good as character pair counts at distinguishing different sequences.

I want to emphasize at this point that this whole scheme of looking at different substrates and different thumbprint functions is more general than it might appear. Suggestions for potential thumbprints which I have heard include the number of characters in the sequence, or the number of times particular strings occur in the sequence, *etc.* All of these are special cases of this general scheme. For example, the number of characters in the sequence is obtained by taking substrate $\{0\}$, and a thumbprint function which has value 1 over its entire domain. The number of times "telnet" occurs in the sequence is a thumbprint in which the substrate is $\{0,1,2,3,4,5\}$ and the thumbprint function takes the value 1 on the sequence $\{`t', `e', `l', `n', `e', `t'\}$, and

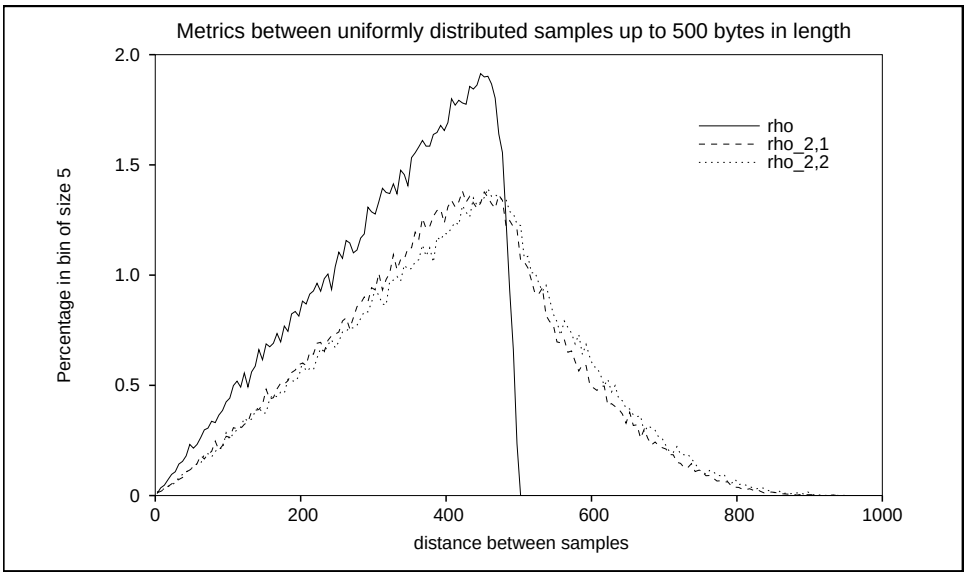

Figure 4.4: Metric at separation 2.

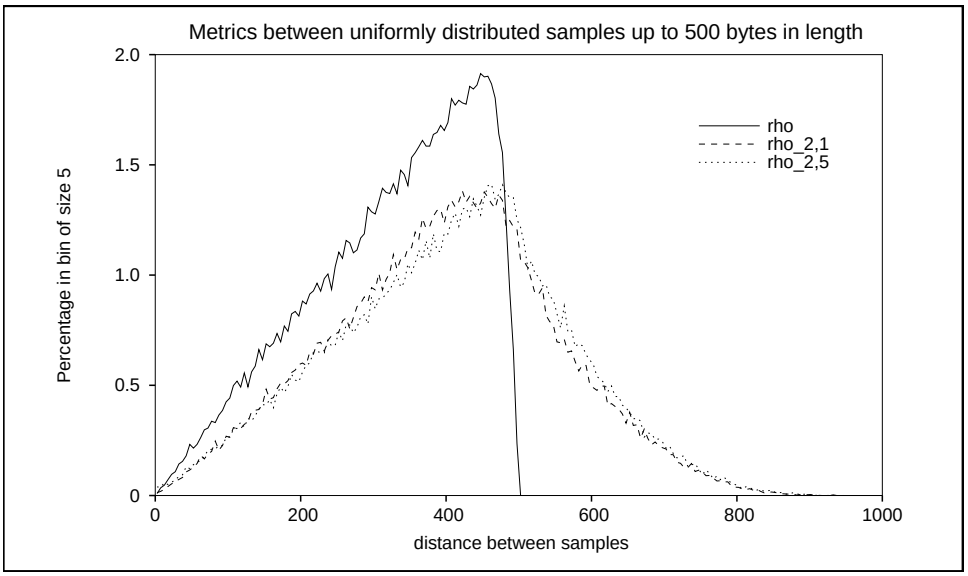

Figure 4.5: Metric at separation 5.

1	2	3	4	5	6
38.2	6741.1	6975.7	2587.2	3446.5	2451.2
11.8	13505.5	92.8	2569.7	3388.7	2446.0

Table 4.2: Thumbprints in concept experiment.

zero on every other sequence.

When seen in this light, my intuition is that such schemes are not particularly advantageous.

4.6 An example

To give the reader some feel for the kind of data I analyze, I present Table 4.2. This was an early proof of concept experiment, and it differs from the current setup in that it is based on total counts of characters, not frequencies, and it used a single (*i.e.* $K = 1$) randomly chosen thumbprint function. However, it illustrates several important points.

The top row labels time in minutes. The other two rows are the thumbprints obtained in two different places on an extended connection chain during each of those minutes. Notice that in minutes 4 through 6, the thumbprints agree quite well, but not exactly. Errors of one or two percent like this are quite common due to missed packets or synchronization errors. The thumbprints in minute 1 do not agree well. This too is very common at the beginning of a connection chain. As each successive link in the chain is set up, its thumbprint is initially based on no data, while thumbprints of earlier links of the chain are based on some text (*e.g.*, the command to log into the next machine). The most interesting point is that minutes 2 and 3 also match very poorly. However, if these are added together, then the combined thumbprint for the

top row is 13716.8, while that in the bottom row is 13598.3. This represents quite good agreement. We inspected the datastreams here and determined that the cause was several large packets of characters which in the lower row fell in minute 1, but were delayed due to an overloaded host so that by the time they were recorded in the upper row of the table, they fell into minute 2. I believe this kind of scenario is not uncommon, and it illustrates the importance of having thumbprints which are additive and tolerant of noise.

4.7 Comparing thumbprints

Given that we can create thumbprints of connection intervals, we need a procedure to compare them. This has to distinguish when two connections were the same, and when they were different. It is complicated firstly by the need to cope with displacements of some characters across interval boundaries, and secondly by the existence of noise in the data due to dropped packets. I developed a procedure which seems to handle both of these difficulties well.

Since the noise distribution is very difficult to characterize (because missed packets, by definition, are missed), we work with the known distribution of unrelated thumbprints and attempt to establish that related ones are atypical if they are considered to be drawn from a distribution of unrelated ones.

Specifically, we start with the K thumbprint components $T_k(C,t)$ for a particular connection C and time interval t. To compare this with some other set of thumbprints $T_k(C',t)$, we form the quantity

$$\delta_t(C,C') = \log\left(\prod_{k=1}^{K} |T_k(C',t) - T_k(C,t)|\right) \tag{4.8}$$

The idea is that the product of differences between the $T_k(C',t)$ and the $T_k(C,t)$

will be much smaller if C and C' are related than if they are not. This will make $\delta_t(C, C')$ larger than expected.

However, if successive thumbprints match over time, that further increases our confidence that the connections are the same. We wish to incorporate this fact into our procedure.

We can consider the $T_k(C, t)$ to be drawn from some probability distribution $P_k(T)$ of thumbprints of all intervals of all rlogin and telnet connections on the Internet. This in turn induces a probability distribution for the δ_t, viz:

$$\delta_t(C, C') \sim P(\delta) \tag{4.9}$$

under the assumption that C and C' are independent.

Of course, we cannot know this distribution $P(\delta)$, but we approximate it by the following procedure. We take our list of connection-intervals observed in our data (excluding injections) and randomly draw two of them. Then we compute δ from them using the above procedure as if they had actually been taken at the same time. Doing this many times gives us a histogram $P'(\delta)$. Ours is based on a Monte Carlo sampling of 10^7 differences. We take this as an approximation to the true P. Now given this, we define the statistic $p_t(\delta_t)$ by

$$p_t(\delta_t) = \int_0^{\delta_t} P'(x)dx \tag{4.10}$$

Intuitively, p_t is (an approximation to) the probability of observing a δ as small as δ_t or smaller by comparing independent connection intervals. We refer to this as the significance of the comparison at time t. A very small p_t implies a significant result.

Now, to agglomerate a comparison over time the most naive procedure is to take the product of the p_t for all the intervals in which we can compare C and C'.

$$p_{\text{naive}}(C, C') = \prod_{t=1}^{s} p_t(C, C') \tag{4.11}$$

where we assume that t runs from 1 to s. It is natural to think of $p_{\text{naive}}(C, C')$ as the probability that all the thumbprints would be as close as they are if C and C' were unrelated connections. This is not correct for several reasons.

Firstly, in taking the product of the probabilities for successive intervals, we are assuming independence of successive thumbprint comparisons over time, which is unlikely to be exactly the case even for unrelated connections. It is not feasible presently for us to quantitatively assess the lack of independence, and so our approach is to make the approximation that successive intervals of unrelated connections are independent, and then study how badly this fails when we apply the whole comparison analysis to control data. We find that although the assumption is not perfect, nonetheless we are able to distinguish control data from connections which really should match.

More importantly, under the null hypothesis that C and C' are independently and randomly chosen connections, the p_t are random variables drawn from a uniform distribution on $[0, 1]$. Thus when we take their product, the result is drawn from the distribution of the product of s $U(0, 1)$ distributions. This distribution can be calculated analytically (see Appendix A), and the result is

$$U^s(x) = \frac{(-\log x)^{s-1}}{(s-1)!} \tag{4.12}$$

Thus we define

$$p_{\text{basic}} = \int_0^{p_{\text{naive}}} U^s(x)dx \tag{4.13}$$

So p_{basic} is the probability of p_{naive} being as small as the observed value or smaller, under the hypotheses of unrelated connections and independence over time.

This statistic still takes no account of the need in some cases to add together successive thumbprints because of leakage of characters from one interval into a neighboring one. Our algorithm is as follows. We compute p_t for each t. If $p_t < \tau$, where the

tolerance τ is some small value (10^{-3} in this study), then we immediately count this value of t as a good match. After we have done this for all t, we go back through the data and look for situations in which consecutive values of t do not constitute good matches. We then combine those thumbprints in pairs, and produce a combined value of δ as

$$\delta_t^{(2)}(C, C') = \log\Big(\prod_{k=1}^{K} |T_k(C', t) + T_k(C', t+1) - T_k(C, t) + T_k(C, t+1)|\Big) \tag{4.14}$$

Now, the $\delta_t^{(2)}$s are not drawn from the same distribution as the δ_ts. However, we can again produce an estimate of this distribution by Monte Carlo sampling of summed differences of independent thumbprints drawn from our data. This allows us, in a similar way to before, to compute $p_t^{(2)}$ as the percentile point of $\delta_t^{(2)}$ in its distribution. Thus $p_t^{(2)}$ is the significance of the comparison of C and C' for the combined intervals t and $t+1$.

The question that then arises is: is the comparison of the combined intervals t and $t+1$ more significant than the comparison of the two intervals taken separately? To answer this, the natural thing to do is to compare $p_t^{(2)}$, with

$$p_t^{(1,2)} = \int_0^{p_t p_{t+1}} U^2(x)dx \tag{4.15}$$

which is our measure of how significant the comparison is in the two intervals taken separately.

We then adopt either $p_t^{(2)}$ or $p_t^{(1,2)}$, whichever is the smallest. It is important to note that both of these numbers are drawn from $U(0,1)$ under the hypotheses of unrelated connections and independence in time. The fact that they have the same distribution is the justification for comparing them. We do this wherever it is advantageous and the individual p_t s failed to meet the tolerance. Suppose we perform this comparison

r times. We then have $s - 2\Gamma$ numbers p. Some of these may be p_t s, some $p_t^{(2)}$ s, and some $p_t^{(1,2)}$s. We take the product of all of these and compute

$$p_{\text{pair}} = \int_0^{\prod p} U^{s-2r}(x)dx \tag{4.16}$$

This we then take as the significance of the full comparison of C and C' over the s time intervals. For convenience we look at

$$l_{\text{pair}} = -\log(p_{\text{pair}}) \tag{4.17}$$

This number is always positive, large when the comparison is very significant, and small when it is not.

We note that it would be straightforward to extend these ideas to adding more than two consecutive thumbprints together. This analysis has not yet been carried out.

We make a last point; in practice when comparing thumbprints of related connections, there is a significant chance that the thumbprints will be *exactly* the same. This causes the analysis above to produce an infinite answer. Alternatively, it is possible for δ_t to be so small that it was smaller than any of the values used in constructing the Monte Carlo approximate histogram δ_t. This again gives an infinite answer. In both of these cases, we refer to this as a dead hit at time t. Thus the analysis produces two values: the number of dead hits, and the significance l_{pair} of the observations which were not dead hits.

Generally, any dead hits are very strong grounds for suspecting the two connections have identical content. l_{pair} comes into play when the data are too noisy to allow of this.

Chapter 5

Principal Components

5.1 Theory

This thesis makes use of a methodology from statistics called Principal Component Analysis. Although this is adequately discussed elsewhere,[20,21] I include a brief explanation here since the technique is not widely known amongst computer scientists.

Suppose we have N data points x_i each of which takes its value in some L dimensional space $x_i = (x_{i1}, x_{i2}, \ldots, x_{iL})$. We might visualize the data as in Figure 5.1 (the reader must imagine the remaining $L - 2$ dimensions). The basic aim of principal component analysis is to reduce the number of dimensions we have to consider, but in such a way as to preserve as much of the information in the original data points as possible.

We can see that, for the particular data set in Figure 5.1, the data very approximately lie along a line at about 45^0 above the horizontal. Thus, if we wanted to approximate the data set by projecting it into just one dimension, this direction would naturally be the best one to choose. The reason for this is that this direction captures more of the variation in the data than any other. Intuitively, this direction is the first principal component direction.

If we wish to reduce the problem less drastically however, we will need to choose additional directions. Intuitively, the second principal component is the direction which captures most of the variation in the data, *amongst all those directions perpendicular to the first principal component.* The third principal component is the direction with

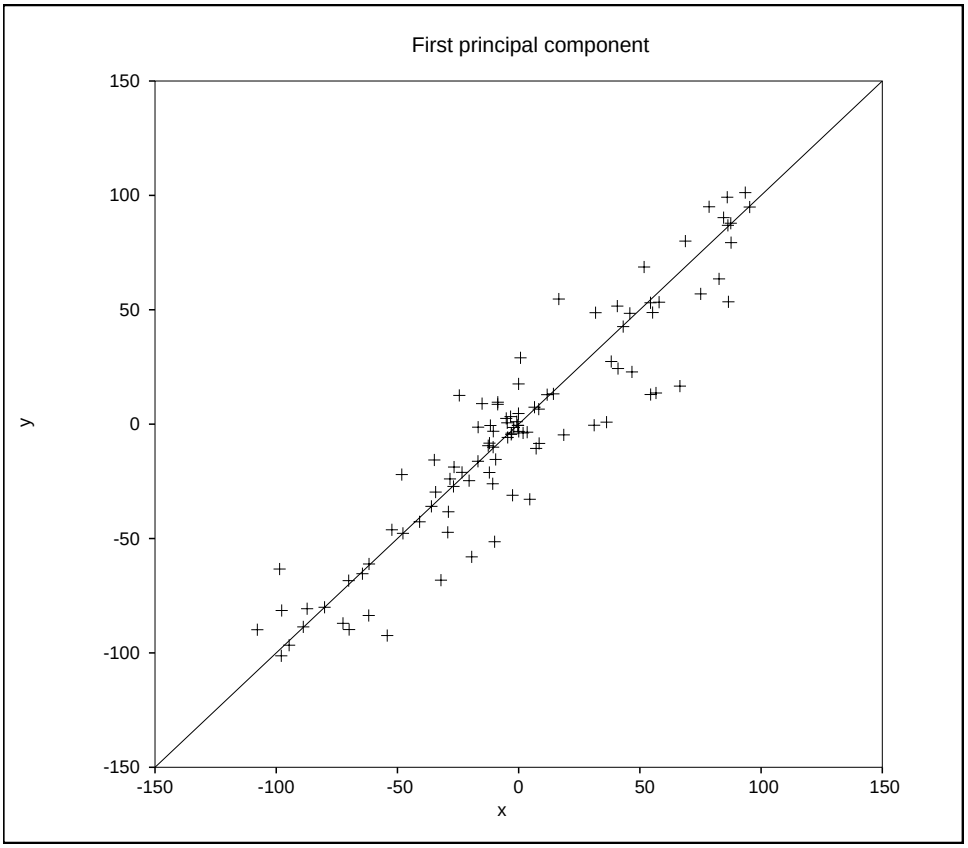

Figure 5.1: First Principal component of synthesized data.

the largest variation that is at right angles to the first two. And so on.

More formally, the total variance in the data set is

$$\sigma^2 = \frac{1}{N}\sum_{j=1}^{L}\sum_{i=1}^{N}(x_{ij} - \overline{x_{-j}})^2 \tag{5.1}$$

where the $\overline{x_{-j}}$ are the means of the individual components,

$$\overline{x_{-j}} = \frac{1}{N}\sum_{i=1}^{N} x_{ij} \tag{5.2}$$

The variance due to each of the original co-ordinate directions is just the corresponding term from the sum over j in (5.1):

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \overline{x_{-j}})^2 \tag{5.3}$$

The variance is essentially the mean square distance of the data points from their average value. Thus each σ_j^2 measures how far the data are spread in the jth direction.

However, it may be that none of the original co-ordinate axes capture the variation of the data particularly well (as in Figure 5.1 where most of the variation is not in the x or the y direction, but rather at 45^O to these (corresponding to the linear combination $x + y$).

If we apply any orthogonal transformation to the data set (a change of co-ordinates corresponding to a rigid rotation of the space), we get new co-ordinates. In the new co-ordinates, we can again compute the variance in each co-ordinate direction, and the total variance. Now a very important fact is that the total variance turns out to be the same. (We do not prove this fact here but essentially it is because the total variance is the trace of the covariance matrix and traces are invariant under orthogonal transformations.)

Thus it is reasonable to compare the proportion of the total variance explained by some co-ordinate in the new system with the proportion explained by each of the co-ordinates in the old system. In particular, we can ask which co-ordinate direction in which rotated co-ordinate system has the highest possible proportion of the variance. This is the first principal component direction. Specifically, it is represented by a linear combination of the original x_{ij} which explains the greatest amount of variance.

Now, given that we have a quantity we wish to maximize, we can certainly apply standard iterative function maximization techniques to find the direction which maximizes it. However, in high dimensionality with a lot of data, this is likely to be very expensive in computer time. The reason principal components are nice is that there is a cheap algorithm to derive them.

Specifically, it can be shown that the principal components are the eigenvectors of

the sample covariance matrix.

$$C_{jk} = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \overline{x_{\_j}})(x_{ik} - \overline{x_{\_k}}) \tag{5.4}$$

Not only that, but the eigenvalues of this matrix represent the variance explained by each principal component. Thus the largest eigenvalue is the variance explained by the first principal component, the second largest corresponds to the second principal component and so forth.

The covariance matrix can be computed in $O(NL^2)$ and it can be diagonalized in $O(L^3)$. Both of these are quite manageable provided the dimensionality L is not too excessive.

We make some final observations on the drawbacks of principal component analysis. The main difficulty comes because of the definition of what is to be maximized – the variance (5.3). Since this is a sum of *squared* distances, the effect of outlying points (ones which are on the edge of the distribution) is considerably amplified. Thus the method is not robust (in the statistical sense).

There are obvious ways to change (5.3) to make it less sensitive to outliers (for example by changing the exponent from 2 to 1). The difficulty with all such schemes is that no efficient algorithm is then known for finding the principal components - one has to resort to iterative methods of uncertain convergence properties. I explored one such possibility at some length but found that I could not obtain reliable answers in a reasonable length of time.

5.2 Application to thumbprints

Given that we are using single character local thumbprints, the question still arises as to which such thumbprint function is best. If the vector of character frequencies

for a particular period of some connection is $\bar{f} = (f_1, f_2, \ldots, f_L)$, the thumbprint can be written as a linear combination

$$T = \sum_{a=1}^{L} \phi(a) f_a \tag{5.5}$$

or, component-wise

$$T_j = \sum_{a=1}^{L} \phi_j(a) f_a \tag{5.6}$$

Thus we condense the vector of L character counts into a vector of K thumbprint components. The question that must be addressed is which linear combinations of the f_i should be used?

The aim of principal component analysis is to take a series of vectors and find a set of linear combinations of the components which explains the maximal proportion of the variance of the vectors. This exactly answers our need. We wish to use the linear combinations with the greatest variance, since character frequencies, or combinations of frequencies, which vary very little are unlikely to be useful in distinguishing amongst different connections, while highly variable frequencies are the most likely to be different in unrelated connections.

5.3 Sample Results

Although I don't describe my experiments fully until the next Chapter, I give some results here for illustration of the way in which principal component analysis works out in practice. I obtained from my data sets a total of 28677 distinct samples. Each of these represented one minute of some, unknown, rlogin or telnet connection on our Local Area Network. For each of these I formed the character frequency vector (which has 128 components). I then applied principal component analysis to these vectors. The largest eigenvalues are shown in Figure 5.2.

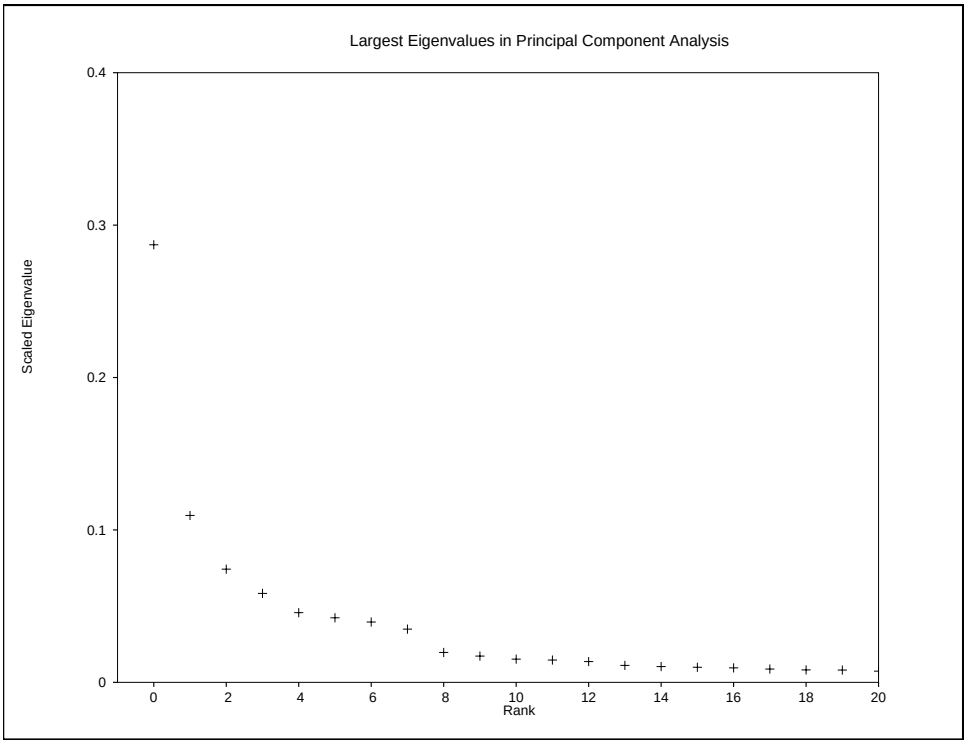

Figure 5.2: The largest 20 eigenvalues of the covariance matrix of our samples of character frequency in network connections.

Ideally, we could look at this picture and there would be some obvious place to stop - the first N principal components would explain almost all the variance, and we could ignore the ones after that. This is not the case; the graph becomes very flat after the first few components. Rather arbitrarily, we decided to use the first $K = 6$ components in this study. We hope to study more carefully the impact of this decision in future work.

The corresponding coefficients are shown as a function of ASCII character in Figure 5.3 and Figure 5.4.

The first vector (which explains 28% of the variance in our character frequencies) is clearly measuring how many spaces (ASCII 32) there are in the traffic versus other characters. Succeeding vectors make little use of the space frequency. It is striking that the statistical procedure picks very different things to emphasize than humans

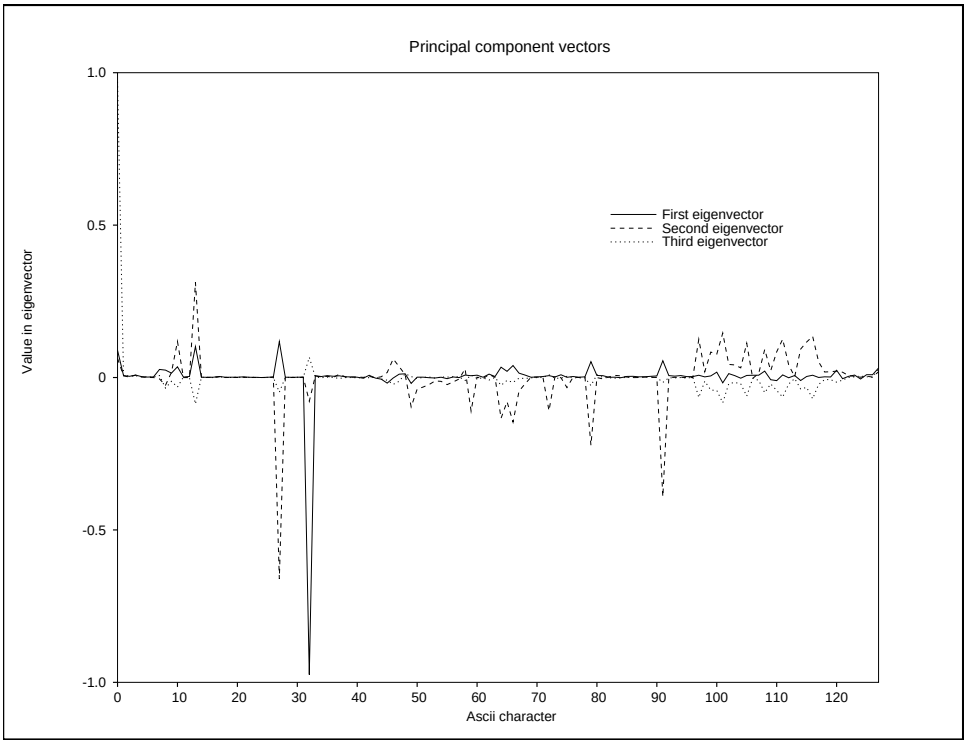

Figure 5.3: The first three principal components. The value is graphed for each ASCII character.

might expect. Our expectation was that most of the meaningful information was in the relative frequency of letters of the alphabet. However, it seems in fact to be more useful to work with punctuation characters, terminal codes, and white space. Letters of the alphabet are mainly treated as a block (the lower case letters occur from ASCII 97 to ASCII 122).

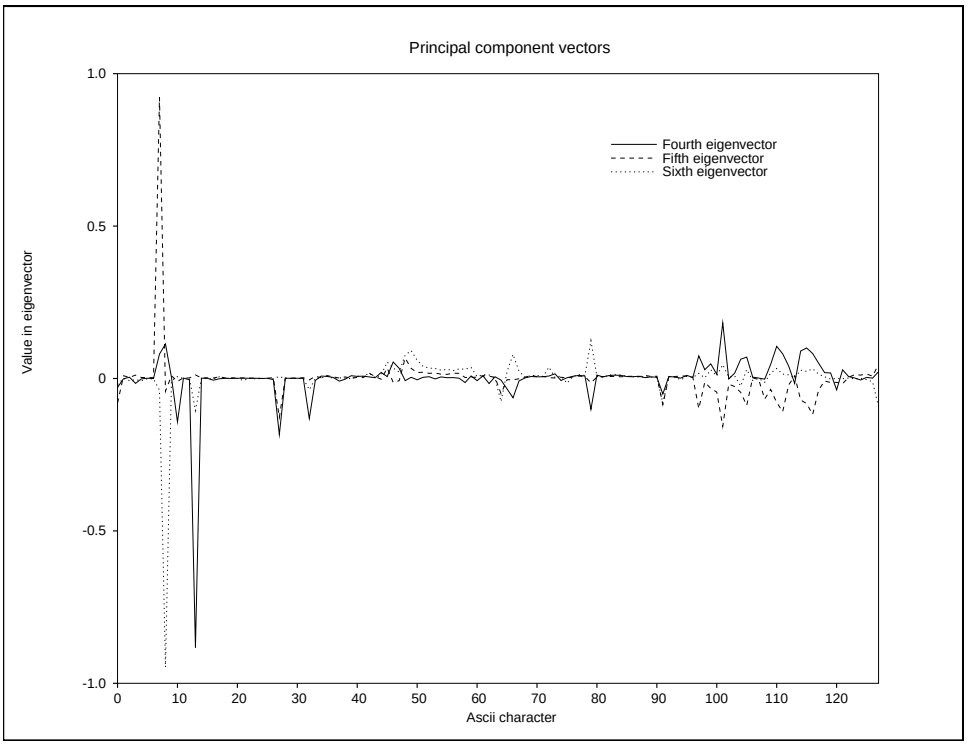

Figure 5.4: The next three principal components. The value is graphed for each ASCII character.

Chapter 6

Proof of Concept Experiments

In order to test my ideas about thumbprinting in a realistic setting, I developed C++ code to thumbprint actual network traffic. This code presently runs on a Sun 4/280 workstation on one of our departmental ethernet LANs. The code uses the network interface in a promiscuous mode (through the */dev/nit* device provided in SunOS). The software analyzes each packet and associates it with the particular pair of machines and ports it is traveling between. It reconstructs the data flowing on each such connection up through the transport layer (TCP). It divides that data up into consecutive intervals, and saves the frequencies with which each character occurs in that interval for that connection. At present I restrict my attention to rlogin and telnet connections (as determined by the internet port number used).

This section describes the code I used, my experimental procedures, and the results of experiments to date.

6.1 Thumbprinting code

An outline of the organization of the code is shown in Figure 6.1. The code is organized in layers which (for the most part) match the layering of the TCP/IP protocol suite. The code is written in C++, and each of the boxes in the picture corresponds to a C++ object.

At the lowest level is a tap. This object either reads directly from a network interface (such as Sun's */dev/nit* for example), or reads from a file of saved network

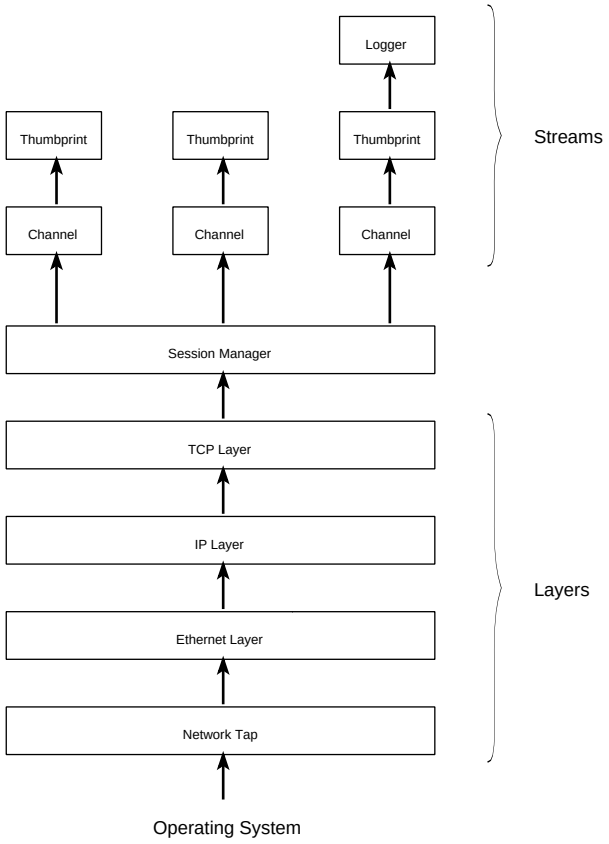

Figure 6.1: Architecture of the network sniffing code.

packets depending on which type of tap is in use. It strips off any OS specific information, and passes up the raw packet to the next layer. In our case, the next layer is an ethernet layer. It strips off and processes the ethernet header from each packet, and passes the data up to the next protocol layer (determined since ethernet packets are self-identifying). Next there is an IP layer, which removes and checks the datagram header, and passes the remainder of the packet upwards. There could be several layers atop IP, but for our purposes only the TCP layer is shown. The TCP layer is responsible for stripping off the TCP header, checking the TCP checksum, and passing the data up to the next layer.

Up to this point, all the layers function alike (indeed they all inherit from an abstract layer superclass). All are stateless, and a given program is only likely to

have one of each kind, which passes packets up to the next one. However, at this point in the hierarchy, this changes. The TCP layer passes data up to the session manager. The session manager then assigns this packet to one of many possible channels - these correspond to connections. At any given time, there is a channel associated with each open connection (with its own unique source and destination address and port combination). The channels are responsible for reconstructing the error free data stream guaranteed by TCP from the packets they receive (as far as is possible given that sniffing may occasionally miss packets which the destination received). Above the channel, there is a thumbprint sequence object which receives the reconstructed character stream from the channel. It in turn assigns the data to one of a number of thumbprint objects which it maintains. This assignment is based on time - successive intervals have separate thumbprints.

In the future, additional layers may handle removing telnet or other protocol related information before thumbprinting is done. All layers above the channel should be subclassed from an abstract stream type - several such layers are available already, but here we concentrate on thumbprinting.

6.2 Data-taking practices

Two points should be mentioned here. Firstly, I mask all characters down to 7 bits since I have found eight bit characters to be comparatively rare in the interactive connections on our network and it is convenient in the rest of the analysis to work with only 128 characters rather than 256. Secondly, I have found that ASCII character 24 plays a peculiar role in the data. This character is used by the telnet protocol as part of its negotiation over which terminal type is in use.[22] Normally, this character is very

infrequent. However, in a very few of the connections monitored, massive numbers of these appear (tens or hundreds of thousands per minute). We do not presently understand the cause of this, though I suspect an implementation bug in some version of telnet. The resulting variability in the frequency of this character means that it receives significant weight in the analysis which is undesirable. Therefore I set the frequency of this character to zero, regardless of its actual value.

A typical experimental protocol is as follows. While the thumbprinting software is running, I execute a script which sets up an extended connection across several machines and then causes data to flow back and forth across the connection in a way described in the next section.

I arrange for the extended connection to cross the LAN segment I am monitoring several times. This allows me to compare the thumbprints at those points in the chain. The only difference between this set up and a more realistic one where two geographically separated pieces of the extended connection were being compared is that I do not have to arrange for synchronization between the monitors.

The reason I injected my own simulated connections into the network traffic was to make it easy to find them again when I came to analyze the data, and to allow me to control variables such as how many machines the extended connection crossed before returning to our monitored LAN.

6.3 Content generation

When I use a script to create a connections to later study, some data must be issued into that connection. What should that data be like? I used two separate content generators, labelled CG1 and CG2.

CG1 took a file of previously saved computer activity, and re-issued it in a random manner designed to simulate a human. Specifically the program chose a random line in the file and issued the characters of that line slowly (as though a human were typing them). Then it issued a random number of lines following that at high speed.

The major drawback with CG1 is that, because it draws its lines in sequence from a file, not all the injected connections will have data that is independent of the data in other injections. To get around this, I developed CG2.

CG2 takes a file of computer activity of some kind, and loads the whole thing into memory. It then issues randomly drawn characters from the file. It loops over three steps

1. Pause randomly for up to τ seconds.

2. Issue randomly up to c characters with a 200 ms spacing.

3. Issue randomly up to m characters as fast as possible.

In each case the randomness is a uniform distribution. Thus the total data rate (in bytes per second) is

$$D = \frac{\tau + 0.2c}{m + c} \tag{6.1}$$

I chose $c = 50$, and varied τ and m to study the dependence of the thumbprinting on the content generation parameters.

6.4 Overview of experiments

There are two main sequences of experiments. In the first, a relatively small amount of data was taken with a one minute thumbprint interval. This was analyzed simply with a view to establishing that the method worked at all.

The second set of data was taken with a ten second thumbprinting interval, and involved considerably higher statistics. The idea here was to start to understand more quantitatively what the limits of the method are, and what the priorities are for further work.

6.5 Results with a one minute interval

In total, this dataset involved about a week's worth of collection. Some of this represented our injected connections, but much of it was unrelated activity by other users. All of the injected connections used CG1 as the content generator.

I begin by describing our control data-set. I scanned through all the connections we had recorded thumbprints for. I excluded any which were deliberately created by us as experiments, and any which had less than five minutes worth of data. I then paired the connections randomly. Any pairs which involved the same set of machines, or which were closer than an hour together in time were excluded in an attempt to reduce the chance of accidentally comparing connections which had the same content. I used a total of 40000 pairings in the control. For each of these, I applied our comparison methodology to four minutes. (I excluded the first minute of the connections.) I observed exactly one dead hit in one minute of these comparisons. I checked and found that the character totals were identical, and some detective work with these suggests that this was the last minute of two unrelated connections which happened in both cases to contain little more than a prompt, and the word 'logout'. This kind of thing is bound to happen occasionally.

The histogram of the obtained values of l_{pair} is shown in Figure 6.2 as the dotted line. The solid line is the curve that would apply if successive values of p_t were inde-

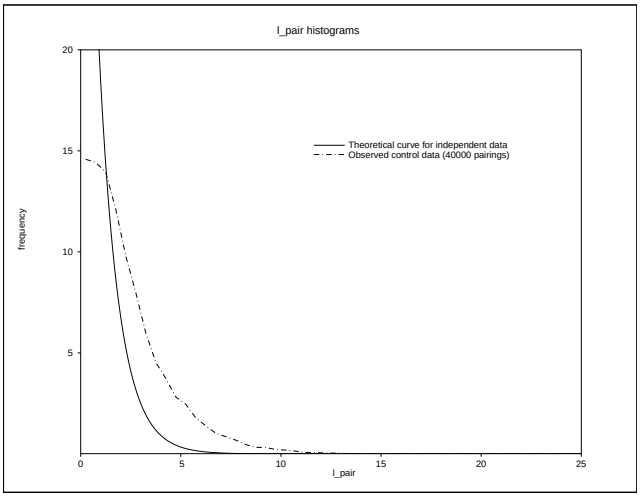

Figure 6.2: Histogram of l_{pair} for control data, together with the theoretical distribution assuming independence in time.

pendent so that p_{pair} was distributed uniformly on $[0, 1]$. Clearly (as expected) this assumption is violated and thus comparisons between unrelated connections tend to be more significant than this assumption would allow. However, it is not so grossly wrong as to make us abandon the natural comparison suggested in and immediately after equation (4.15). We also speculate that the extreme right tail of the control histogram contains comparisons between connections which chance to have some related data (a risk when all data is taken on the same network).

We applied the same comparison procedure to four sets of injected data. In Run I, our extended connection began on *toadflax*, went to *k2*, and then went back to *toadflax* where both *toadflax* and *k2* are within our department. In Run II, the extended connection went from *k2* to *toadflax* to *k2* and back to *toadflax*. This gave three legs of the extended connection that could be compared. Thus there are two independent sets of thumbprint differences for each injected connection. As for the control data, I looked at four minutes worth of data in each case, after dropping the first minute (which usually gives an unreliable comparison).

These two runs gave similar results, so I combined them. There were a total of

Run	N	4 hits	3 hits	2 hits	1 hit	0 hits
I & II	302	60	17	14	7	2
III	54	20	26	30	19	6
IV	28	4	32	29	32	4

Table 6.1: Number of trials and percentage of each number of hits for the experimental runs described in the text.

302 comparisons. The percentage of trials with various numbers of hits is given in Table 6.1. In all, 98.3% of the comparisons gave at least one dead hit. Five comparisons were sufficiently disturbed by noise as to give no dead hits. The values for l_{pair} in these cases were 36.49, 37.46, 37.76, 39.70, and 42.34. Comparison of these values with the control histogram in Figure 6.2 makes it clear that they are very large, indicating that the method clearly can identify these connections despite the noise.

In our next experiments, we tested the method on extended connections over long-haul networks. These are harsh conditions, (but ones that are perhaps typical of intrusions). The delays between typing a character and seeing the echo were typically several seconds over these chains. In Run III, the connection chain went

$$\text{toadflax} \rightarrow \text{k2} \rightarrow \text{helvellyn}$$

$$\rightarrow \text{alps.cc.gatech.edu} \rightarrow \text{k2} \rightarrow \text{toadflax}$$

Here, *alps.cc.gatech.edu* is in Georgia while the rest of the hosts are in Davis. We compared the chain as it left and re-entered *toadflax*. All but three of the 54 comparisons gave some dead hits. On those that did not, the values of l_{pair} were 22.74, 41.29, and 44.31. Again, these numbers are very far out into the tail of the control histogram, although the smallest of these does cross with the most significant of the control comparisons.

The schema in Run IV was

$$\text{toadflax} \rightarrow \text{k2} \rightarrow \text{helvellyn} \rightarrow \text{po.csc.liv.ac.uk}$$
$$\rightarrow \text{alps.cc.gatech.edu} \rightarrow \text{k2} \rightarrow \text{toadflax}$$

po.csc.liv.ac.uk is in Liverpool, England. Only one of the 28 experimental connections gave 0 dead hits, and it had an l_{pair} value of 28.09. Thus, even in this long chain, we can successfully match up the endpoints of the connection in all cases.

I also studied whether I could reliably pick our injected connections out from our control connections. For each pair of samples from an injected connection, I chose one of the pair and compared it to all the connections in our control set. We then assessed whether it was more similar to its actual partner than to any of the unrelated data.

To compare the value of two matches, it is convenient to have a method to combine the number of dead-hits with the significance level where there is not a dead-hit. Thus, we must give a significance level to a dead-hit. To do this, I looked at the significance level of all of our comparisons on an individual, minute-by-minute basis. We found that the highest significance level achieved for a minute of comparison which was not a dead-hit to be 13.82. We therefore set the significance of a dead hit at 14. We then combined all significances into a single number which incorporated the dead-hits.

For each of our injected connections, I then computed its total significance in this manner, and the total significance of comparing it with all the unrelated control connections. We formed the ratio $\mathcal{R}$ between the total significance of the correctly matched comparison, and the best of the unrelated comparisons. Thus I get one value of $\mathcal{R}$ for each injected connection. If things are working correctly $\mathcal{R}$ should be more

Run	N	med $\mathcal{R}$	worst $\mathcal{R}$
I & II	302	7.51	3.89
III	54	6.59	2.44
IV	28	7.23	3.49

Table 6.2: Cross control: ratio of significance of true comparison with best of control comparisons.

than one. Preferably quite a bit more than 1.

Table 6.2 tells the story. For each group of runs, I present the median value of $\mathcal{R}$ and the worst case value of $\mathcal{R}$. The essential point is that in every case, the comparison involving the two samples from the same connection had a significance level at least twice as great as the best comparison of an injected connection to a control connection. The reader is reminded that the significance here is on a logarithmic scale.

6.6 Results with a ten second interval

The data reported in the previous section convinced me that the method was basically viable. It remained to try and establish what parameters controlled its success. The experiments reported here are a first attempt at this, though more remains to be done.

I first describe the histograms required for the comparison algorithm. I sampled a total of 159335 ten second intervals of ambient rlogin and telnet activity on our target LAN. To this was applied the principal component analysis, and eigenvectors were derived in the same manner described earlier.

The first three are shown for comparison with those obtained for one minute intervals (Figure 5.3). The ten second eigenvectors are shown in Figure 6.3.

These differ in detail from the earlier ones; however, many of the same broad

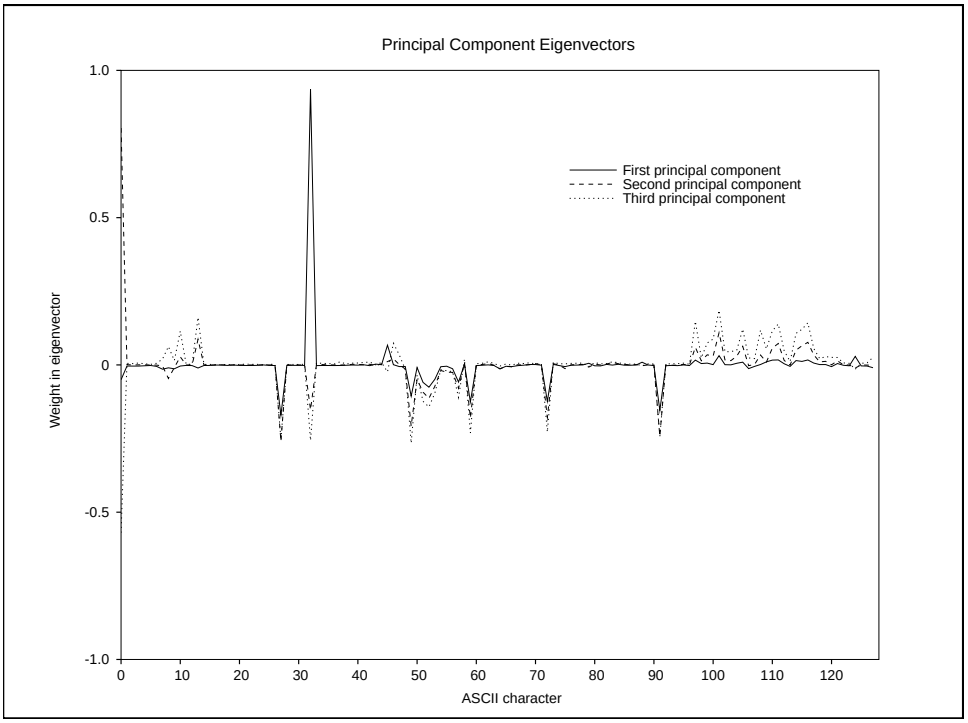

Figure 6.3: The first three principal components for the ten second data. The value is graphed for each ASCII character.

features persist. As discussed earlier, the overall sign of the principal components is irrelevant.

Next, all these samples were thumbprinted. Then 10^7 pairings were randomly chosen, and the thumbprint differences were computed. This gave the histogram of $P(\delta)$ referred to in equation (4.9). Similarly, 10^7 differences of sums were computed.

The injected connections themselves consisted of hours-long connections rather than the four minute ones used earlier. I artificially broke them into pieces as necessary. The beginning and end of each connection were not used as this was during the time the chain was being set up, and so data in that period was not constant throughout the chain.

All connections contained data generated by CG2. An important difference this implies is that the connections contain periods longer than the thumbprinting interval

Run #	N	X	τ	m	start	stop
1	332	po	30	2000	18:11	18:39
2	1588	po	30	2000	18:55	21:06
3	1506	po	30	2000	21:26	23:29
4	638	po	30	2000	12:15	13:12
5	3458	po	30	2000	21:23	02:02
6	478	alps	30	2000	09:00	09:47
7	3706	alps	30	2000	09:50	15:00
8	11870	alps	30	2000	15:15	07:46
9	4122	alps	30	2000	08:43	14:30
10	1980	alps	30	2000	15:11	17:59
11	8774	alps	30	500	20:23	08:37
12	7580	alps	30	200	21:32	08:05
13	7296	alps	15	200	21:46	07:55
14	6226	alps	15	500	23:20	08:01
15		alps	60	2000	20:54	23:59
16		alps				

Table 6.3: Overview of ten second injected connections. Here, N is the number of consecutive ten-second intervals used (including idle ones), X is the target machine, τ is the maximum length of "thinking" pauses in the content generator (in seconds), m is the number of characters generated per burst by the content generator. Approximate start and stop times of the runs are also given.

which are idle. In contrast to the one minute data, where I only performed comparisons on non-idle connections, here I explicitly allow idleness (in an attempt to be more realistic). Naively, if two connnections being compared are both idle during the same time, it would count as a dead hit (since the thumbprints are the same (zero) in both cases). Clearly this is a bad idea since idleness is very common. Thus I treated the case of idleness specially - when one or both of two intervals being compared were idle, that comparison got counted as having probability 0.5 (*i.e.* of having only average significance).

In this analysis, I wished to have a single measure of how well the thumbprinting worked in order to compare different settings. In order to do this, I firstly broke each of my injected connection up into pieces, with each piece containing q consecutive intervals. Then, I applied the thumbprint comparison algorithm detailed in Section 4.7 to compare each of these pieces between the two versions of the injected connection that I had thumbprints for. I arrived at a total significance level using the procedure described in the last section (of assigning each dead hit a significance level).

Next I compared all of the pieces of my injected connections with various pieces of ambient connection traffic. (These pieces were also of length q.) I observed the proportion of these comparisons in which the unrelated connection actually did *better* than the true partner. This proportion, which I refer to as the failure rate, is an estimate of the probability that an unrelated connection will do better than the true comparison. In the best case, this proportion will be zero – true connections are identified unerringly. In the worst (reasonable) case, the proportion is 0.5 – the comparison of two connections which do match is no better than an average comparison of unrelated connections.

It is convenient to use the inverse of the failure rate, which I refer to as the mean

time to failure. The interpretation of this is as follows. If I compare two pieces of my connections, C_1 and C_2 say, which are supposed to match, and then begin comparing C_1 to arbitrary unrelated connections, on average how many such comparisons will it take till I find an unrelated connection which is more like C_1 than C_2 is? This average time is the mean time to failure.

The mean time to failure (MTF) is the principal tool used to study the success of thumbprinting in this section. Clearly we would like the mean time to failure to be large - perhaps in the region of $10,000$ and upwards. With this in mind, I used $100,000$ comparisons to estimate the failure rate in each case. Thus, I cannot accurately measure appropriately low failure rates, but can measure unacceptably high failure rates quite well. However, this is fine as the intent of this study is to determine the domain of applicability, and thus we focus on the edges of this domain.

The basic data here is summarized in Table 6.4 which had $K = 2$ (two independent thumbprints), Table 6.5 with $K = 4$, and Table 6.6 with $K = 6$. Each entry in the table is the mean time to failure for a particular run analyzed in a particular way (with an estimate of the standard error where the statistics allowed a sensible estimate). The rows of the tables represent the different experimental runs, and the columns represent the size of the pieces that each connection was broken into. Thus a column labelled "4" was analyzed in chunks of four consecutive ten second intervals.

What can be inferred from this data? The most obvious dependence, regardless of the number of thumbprints, is on q, the length of interval considered. When thumbprinting is performed with only two ten-second intervals to work with, the MTF is almost always unacceptably low - one has only to try a few tens or hundreds of random connections to come across one as good as the real partner. With four consecutive intervals to work with, the MTF becomes marginal, but finally with

	2	4	8
1	56 ± 22	339 ± 331	191 ± 191
2	42 ± 9	5555	> 100000
3	67 ± 19	1176 ± 860	> 100000
4	21 ± 4	66 ± 34	1063 ± 720
5	37 ± 6	735 ± 375	12500
6	203 ± 121	1000 ± 921	> 100000
7	43 ± 7	524 ± 255	> 100000
8	59 ± 7	3571 ± 2093	> 100000
9	56 ± 10	621 ± 561	> 100000
10	93 ± 24	1667 ± 1667	> 100000
11	89 ± 13	5556	> 100000
12	97 ± 16	2174 ± 613	> 100000
avg	72 ± 14	1915 ± 562	

Table 6.4: Failure rate for two independent thumbprints applied to varying length of connection (columns) and experimental runs (rows).

	2	4	8
1	69 ± 34	10000	25000
2	51 ± 13	100000	> 100000
3	101 ± 32	6250	> 100000
4	25 ± 5	130 ± 111	8333
5	52 ± 9	4166	> 100000
6	284 ± 273	3125	> 100000
7	60 ± 11	16666	> 100000
8	82 ± 11	20000	> 100000
9	82 ± 19	680 ± 653	> 100000
10	111 ± 35	25000	> 100000
11	138 ± 25	16666	> 100000
12	149 ± 32	6666	> 100000
avg	100 ± 20	17446 ± 7852	

Table 6.5: Failure rate for four independent thumbprints applied to varying length of connection (columns) and experimental runs (rows).

	2	4
1	123 ± 40	1515 ± 1351
2	49 ± 12	16667 ± 6722
3	95 ± 31	9091 ± 8298
4	21 ± 5	99 ± 93
5	48 ± 8	2083 ± 1393
6	326 ± 312	5882 ± 4885
7	60 ± 11	33333
8	80 ± 11	33333
9	85 ± 19	787 ± 775
10	109 ± 33	50000
11	160 ± 32	100000
12	169 ± 37	4761
avg	110 ± 23	21462 ± 8500

Table 6.6: Failure rate for six independent thumbprints applied to varying length of connection (columns) and experimental runs (rows).

eight, MTFs become satisfactory in almost all cases (except when $q < 4$). Thus this algorithm, on this network, with this thumbprinting interval, will work adequately with a minute and a half of a moderately active connection provided that there are at least 4 independent thumbprint components. It becomes marginal with much less than that.

Other dependencies are much less clear. We first consider dependence on K the number of independent thumbprint components. Firstly, the average MTFs are not significantly different between the cases of 2, 4, or 6 independent thumbprints. However, we can discern some differences by being more subtle and using the fact that it is the same underlying data in each table, so that we can pair up the observations. For example, if we look at the $q = 2$ column only, and compare the case of $K = 2$ with $K = 4$, we see that in every single row, the MTF for $K = 2$ is smaller than that for $K = 4$. If we made the null hypothesis that $K = 2$ and $K = 4$ were equally good as thumbprinting strategies then it would be equally likely for either to have

the larger MTF. Hence the probability of all of the larger MTFs being in the $K = 4$ table would be $2^{-12} = 2.4 \times 10^{-4}$. Thus, this provides strong evidence that $K = 4$ is in fact more effective than $K = 2$ for our particular experimental setup. A similar pattern emerges by looking at the $q = 4$ column.

However, no such pattern shows up to distinguish the $K = 4$ and $K = 6$ cases. Thus, as far as this analysis can determine, taking six independent thumbprints per ten second interval has no benefits relative to taking four. Taking four is perceptibly better than taking two, but the differences are not massive (and may not justify the increased storage required).

Disappointingly, not much else stands out from the data set. It is clear that some individual connections were less successfully thumbprinted than others. For example run 4 seems to have very high failure rates however it is analyzed – but other runs with the same parameters as this run do not show anomalously low MTFs. Thus presumably some other variable which was not observed controls this effect. I speculate that this is simply the round-trip propagation across the network which I have subjectively observed to be quite variable. However, I do not have data to confirm or refute this hypothesis.

6.7 Performance considerations

We find that, although the LAN segment we have monitored in experiments is fairly busy because it houses one of our department's main mail and file servers, the thumbprinting program typically uses no more than a few percent of the processor time on the machine it is running on.

While the thumbprint mechanism described here has many applications, we are

focusing specifically on the assigning of signatures to interactive login sessions. So although the total amount of traffic crossing a large internetwork may be enormous, the portion of the traffic which is interesting is quite small.

For example, we looked at the traffic statistics for the NSFNET internetwork.[23] For November 1994, the combined rlogin and telnet traffic of 1.024×10^{12} bytes, accounted for only 4.56% of the total traffic. Distributed evenly over the month, we find the data rate to be 3.95×10^5 bytes per second. Furthermore, if we use a machine which performs 50 million instructions per second, this would allow us to use 126 instructions for each byte.

While the assumption that the traffic is distributed evenly is unrealistic, the fact that a single, moderately powered workstation could, in the steady state, apply 126 instructions to every byte of telnet and rlogin data crossing the NSFNET is remarkable. Furthermore, while the amount of traffic across the NSFNET doubled between November 1993 and November 1994, the traffic for telnet and rlogin increased at only about half that rate.

Similarly, a T1 data line can carry 1.9×10^5 bytes per second in total, while a T3 line carries 5.6×10^6 bytes per second. If we make the assumption that only 5% of these bytes are rlogin and telnet (as on the NSFNET) then our 50 MIP machine dedicated to this task has about 5200 instruction per byte on the T1 line, and 178 instructions per byte on the T3 line.

These calculations are of course simplistic - they neglect the fact that some work must be done examining headers of other protocols to determine that they must be ignored. We are also not in a position to assess the capabilities of suitable network interfaces. Nonetheless, the fact that upwards of a hundred instructions are available per byte on average in several contemporary network settings is very encouraging as

to the applicability of this method, given an implementation on a machine dedicated to the purpose.

Measurements indicate that our existing thumbprinting code handles ethernet traffic at 0.35 ± 0.06 MBps on a Sun 4/280. However, this is general purpose code which performs many error checks and compiles a variety of statistics. Code optimized for thumbprinting could probably run several times faster.

Chapter 7

Other Applications

I developed the thumbprinting techniques described here to trace connections over networks. However, I can envisage a number of other applications for the technology. In this chapter I briefly outline some of those. Thumbprinting is potentially useful whenever a series of texts must be compared to see if some of them are *almost* the same.

7.1 Tracing mail

It would obviously be extremely useful to be able to trace email. Firstly, some attacks are based on malformed email. Secondly, email may have contents, which while not an attack in the security sense, are nefarious in some way or constitute evidence of a crime (for example, an email death threat to the President).

Unfortunately, the present Simple Mail Transport Protocol (SMTP)[24] is extremely trusting, and consequently it is very easy to forge mail. See for example the helpful discussion in *Phrack*, Issue 41. The results are only subtly different from the real thing, and are very difficult to trace back to their source.

The sequence of mail forgery goes as follows. In the simplest version, the hacker on host H telnets to port 25 on host A. He then claims to the SMTP server there that he is entering mail from host R. He then enters the mail. This does not work particularly well because the SMTP software senses that something is wrong because a connection from H is claiming to be from R, and changes the form of the header,

thus revealing the mail as a forgery.

In a more complex but much better scheme, the hacker on H telnets to A and claims to be H transferring mail which has been received from R. SMTP takes this at face value and marks the From: line of the mail as good_userR. The fact that the mail travelled via machine H is also recorded, and may be grounds for suspicion that the mail is not genuine. Alternatively, it may well go unnoticed. In any event, should the mail later fall under suspicion there is no way to find the true origin of the mail - it originated as a telnet connection from H, and there the trail goes cold.

Further elaborations on this are to have the mail routed via several more sites before reaching the final destination.

Clearly, we can thumbprint mail, and we can particularly pick out any connections to port 25 and thumbprint those too. If we do this successfully, we can help in tracing mail a lot, in that we can reliably identify which machine a mail message actually originated on, and then we can perhaps use our telnet thumbprinting capacity to identify on which machine the message was actually typed.

The main differences from thumbprinting interactive telnet connections are firstly that mail travels asynchronously. Thus the beginning of a thumbprint must have some appropriate reference to the mail message. Secondly, mail messages mutate as they travel, acquiring more and more lines in the header. Thus a thumbprint must be based on that section of the mail which is invariant.

So the natural thing to do is give one thumbprint for an entire SMTP connection, but to parse out variant parts of the header (such as the Received: lines) before thumbprinting.

The kind of ability outlined here raises difficult policy issues. Viz, there exist several servers throughout the world which provide the explicit service of anonymity

to internet citizens. The user sends a message to the server which maps the user to a handle. It then generates mail from this handle to the ultimate destination of the mail. All traces of the original user are eradicated from the message, and only the server has the mapping between actual email addresses and handles. Such servers are frequently used by people who wish to exchange mail or make Usenet postings on subjects with which they do not wish to have their name associated. The kind of tracing ability proposed here would have the potential to pierce this screen.

This raises a variety of specters: for example, suppose a large company used its access to the tracing system to discover whether any of the email to, say, an AIDS support mailing list, was coming from its employees. The company could then use this information to cut off health insurance to the affected individuals.

7.2 Recovering damaged documents

An actual example problem which was described to me recently was the following. A large set of mail messages had been extracted from a mailing list server. They had then been processed and categorized into a convenient structure for browsing. Minor editing and correction was performed on some messages. During this process, much of the message headers were eradicated, including the Message-id: line. For various reasons, it was now necessary to re-associate the 'cleaned' messages with the original ones. How could this be done?

This is an example of where thumbprinting's error tolerance is an asset. It would be relatively straightforward to parse all these messages, thumbprint the bodies, sort all messages by thumbprint, and then re-associate the original messages with the cleaned ones.

Similar examples might occur in recovering files from a disk which had some disk errors.

Chapter 8

Conclusions

8.1 Results to date

The main result of this thesis is that it is easily possible, on an ethernet, to save summaries of interactive connections which can be stored in only a few tens of bytes per minute per connection. In the case where these connections are at least a minute and a half long and have moderate data flows, it is then possible to compare these summaries later and identify whether two connections have the same content or not with very low probability of error. This is true even when one of the sets of data being compared has passed through a tortuous route to Europe and back on the internet.

The present state of the art is to use four independent thumbprints and thumbprint on a ten second interval. More thumbprint components than this are probably not useful. Principal component analysis is used to select the best thumbprint vectors. Given two bytes per thumbprint component, plus another four bytes to store the total number of characters, this is 12 bytes per interval, or 72 bytes per minute of connection.

Alternatively, storing 28 bytes per minute for a one minute long interval gives excellent results on longer connections, but will fare less well on very short connections.

8.2 Future work

I am actively working to extend this result in various ways – this topic has proven to be considerably larger than can easily be contained in a single Master's Thesis.

Firstly, I am still doing research to fine tune the statistical algorithms to give the best performance possible. I am currently implementing a more sophisticated adaptive algorithm to try to group intervals in more complex ways to increase the power of the scheme for short connections.

I am also studying ways to break up the connection into pieces that do not depend on time, but rather on content based triggers. Success at this would obviate the need to synchronize geographically separated thumbprint stations. The main difficulty with this scheme is finding content based triggers that occur in a reasonably uniform way. From an algorithmic point of view, we would no longer have to worry about characters overflowing from one interval to the next, but would instead have to worry about two intervals being accidentally combined into one due to a trigger being missed. With an additive thumbprint scheme this is not hard to compensate for.

Also under way is an effort to design a general purpose protocol for communication between tracing systems. This will be as independent as possible from particular assumptions about the way in which a tracing system worked. It will incorporate thumbprinting as one possible way to describe the data in a connection.

Once this is done, it is my intent to build a prototype system to implement these ideas and make it available to the Internet community. We anticipate that this system, where implemented, will be capable of reliably tracking intruders who do not take adequate precautions to avoid it.

The main vulnerabilities of such a system will be, firstly, parts of the system being replaced by Trojan horses, and secondly, intruders encrypting their connections differently in each link of the extended connection chain. Such encryption need not be strong at all - a simple Playfair or Viginere cipher would more than suffice. While both of these are within the capability of the more talented members of the intruder

community, we believe that a tracing system such as this could raise the entry price paid to become an intruder, and, where deployed, would increase the risks and inconvenience of penetrating computers for all intruders. Such a system would not be a panacea, but might be a deterrent.

Appendix A

Product of uniform distributions

We calculate the probability density function (pdf) of

$$z = x_1 x_2 \dots x_n \tag{A.1}$$

assuming that the x_i are distributed $U(0,1)$. Throughout, we take $f(x)$ to be the pdf of x, and $F(x)$ to be the cumulative frequency distribution (cfd) of x. To begin, we define

$$Y = \log(z); y_i = \log(x_i) \tag{A.2}$$

so that

$$Y = \sum_{i=1}^{n} y_i \tag{A.3}$$

Both Y and the y_i have range $(-\infty, 0)$. We can easily calculate $F(y_i)$ for each i, since

$$F(y_i) = \text{Prob}(\log x_i \leq y_i) = \text{Prob}(x_i \leq e^{y_i}) = e^{y_i} \tag{A.4}$$

Then

$$F(Y) = \int_{\sum y_i \leq Y} \exp\left(\sum y_i\right) \prod dy_i \tag{A.5}$$

This integral can be effected by making the change of variables

$$P = \sum_{i=1}^{n} y_i \tag{A.6}$$

$$q_i = y_i - y_1 \quad \forall i \neq 1 \tag{A.7}$$

$$\tag{A.8}$$

If we denote the linear transformation defined in these equations by M, then (A.5) can be rewritten as

$$F(Y) = \int_{-\infty}^{Y} e^P \det(M^{-1}) A \, dP \tag{A.9}$$

where $\det(M^{-1})$ is the Jacobean of the variable transformation, and A is a factor coming from integrating over the $n-1$ variables q_i (on which the integrand did not depend). It is possible, though a little tricky, to directly evaluate A and $\det(M^{-1})$. It is easier to sidestep this work by noting that, since M is a linear transformation, $\det(M^{-1})$ must be constant. A, on dimensional grounds, must be proportional to P^{n-1}. Thus

$$F(Y) = C \int_{-\infty}^{Y} P^{n-1} e^{P} dP \tag{A.10}$$

where C is an unknown constant. This integral is a standard form,[25] and the result is

$$F(Y) = Ce^{Y} \left[\sum_{i=0}^{n-1} (-1)^{n-1-i} \frac{(n-1)! Y^{i}}{i!} \right] \tag{A.11}$$

The requirement that $F(Y) = 1$ at $Y = 0$ then fixes the unknown constant C at

$$C = \frac{(-1)^{n-1}}{(n-1)!} \tag{A.12}$$

Since $Y = \log z$ we can deduce the cfd for z as

$$F(z) = z \sum_{i=0}^{n-1} (-1)^{i} \frac{(\log z)^{i}}{i!} \tag{A.13}$$

Finally, differentiating this wrt z gives the pdf for z, which we earlier called $U^{n}(z)$

$$U^{n}(z) = \frac{(-\log z)^{n-1}}{(n-1)!} \tag{A.14}$$

From an implementor's perspective, it is easiest to use this in the form of $\log F(Y)$ which is close to a linear function in the region of interest and so can be efficiently approximated as a lookup table with linear interpolation.

References

[1] Fraser, B. (CERT). Private Communication, 1994.

[2] Van Wyck, K. (ASSIST). Private Communication, 1994.

[3] R. Bace. A New Look at Perpetrators of Computer Crime. In *Proc. 16th Department of Energy Computer Security Group Conference*, 1994.

[4] P. Neumann and D. Parker. A Summary of Computer Misuse Techniques. In *Proc. 12th National Computer Security Conference*, pages 396–407, 1989.

[5] C. Stoll. *The Cuckoo's Egg.* Doubleday, 1987.

[6] C. Stoll. Stalking the Wily Hacker. *Communications of the ACM*, 31:???, 1988.

[7] K. Hafner and J . Markoff. *Cyberpunk.* Simon and Schuster, 1991.

[8] Phrack. Electronic Journal, 1994. Ftp to ftp.fc.net.

[9] 2600 - The Hacker's Quarterly, 1994.

[10] The Knightmare. *Secrets of a SuperHacker.* Loompanics Press, 1994.

[11] J.I. Schiller. Secure Distributed Computing. *Scientific American*, pages 72–76, November 1994.

[12] S. Snapp, *et al.* DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype. In *Proc. 14th National Computer Security Conference*, 1991.

[13] C. Ko, *et al.* Analysis of an Algorithm for Distributed Recognition and Accountability. In *Proc 1st ACM Conf. on Computer and Communications Security, Fairfax, VA*, pages 154–164, 1993.

[14] H. Jung, *et al.* Caller Identification System in the Internet Environment. In *Proc. 4th Usenix Security Symposium*, 1993.

[15] Wadell, S. Private Communication, 1994.

[16] Abell, V. lsof. Version 3.10. Computer Software, 1994. Ftp to vic.cc.purdue.edu.

[17] L.T. Heberlein, K. Levitt, and B. Mukherjee. Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks. In *Proc. 15th National Computer Security Conference*, pages 262–271, 1992.

[18] Palasek, B. Private Communication, 1994.

[19] C. Stanfill and B. Kale. Parallel Free-Text Search on the Connection Machine System. *Communications of the ACM*, 29:1229, 1986.

[20] W. Krzanowski. *Principles of Multivariate Analysis.* Clarendon Press, Oxford, 1988.

[21] C. Chatfield and A. Collins. *Introduction to Multivariate Analysis.* Chapman and Hall, London, 1980.

[22] Solomon, M. and Wimmers, E. Telnet Terminal Type Option., 1983. Request for Comments RFC 884.

[23] Merit Network Information Center. NSFNET Statistics, 1994. Ftp to ftp.merit.edu.

[24] Crocker, D. Standard for the Format of Arpa Internet Text Messages., 1982. Request for Comments RFC 822.

[25] M. Abramowitz and I. Stegun, editors. *Handbook of Mathematical Functions.*, page 71. Dover, New York, 1965.