

Protocol Design in an Uncooperative Internet

Stefan R. Savage

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2002

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Stefan R. Savage

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Co-Chairs of Supervisory Committee:

Thomas E. Anderson

Brian N. Bershad

Reading Committee:

Thomas E. Anderson

Brian N. Bershad

David J. Wetherall

Date: _____

©Copyright 2002

Stefan R. Savage

In presenting this dissertation in partial fulfillment of the requirements for the Doctorial degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to ProQuest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Protocol Design in an Uncooperative Internet

by Stefan R. Savage

Co-Chairs of Supervisory Committee

Associate Professor Thomas E. Anderson
Computer Science and Engineering

Associate Professor Brian N. Bershad
Computer Science and Engineering

In this dissertation, I examine the challenge of building network services in the absence of cooperative behavior. Unlike local-area networks, large scale *administratively heterogeneous* networks, such as the Internet, must accommodate a wide variety of competing interests, policies and goals. I explore the impact of this lack of cooperation on protocol design, demonstrate the problems that arise as a result, and describe solutions across a spectrum of uncooperative behaviors. In particular, I focus on three distinct, yet interrelated, problems – using a combination of experimentation, simulation and analysis to evaluate solutions.

First, I examine the problem of obtaining unidirectional end-to-end network path measurements to uncooperative endpoints. I use analytic arguments to show that existing mechanisms for measuring packet loss are limited without explicit cooperation. I then demonstrate a novel packet loss measurement technique that sidesteps this requirement and provides implicit cooperation by leveraging the native interests of remote hosts. Based on this design, I provide the first experimental measurements of widespread packet loss asymmetry.

Second, I study the problem of robust end-to-end congestion signaling in an environment with competitive interests. I demonstrate experimentally that existing congestion signaling protocols have flaws that allow misbehaving receivers to “steal” bandwidth from well-behaved clients. Fol-

lowing this I present the design of protocol modifications that eliminate these weaknesses and allow congestion signals to be explicitly verified and enforced.

Last, I explore the problem of tracking network denial-of-service attacks in an environment where attackers explicitly conceal their true location. I develop a novel packet marking approach that allows victims to reconstruct the complete network path back to the victim. I evaluate several versions of this technique analytically and through simulation. Finally, I present a potential design for incorporating this mechanism into today's Internet in a backwards compatible manner.

Table of Contents

List of Figures	v
List of Tables	vii
Chapter 1: Introduction	1
1.1 Goals	3
1.1.1 Active network measurement in an uncooperative environment	3
1.1.2 Robust congestion signaling in a competitive environment	5
1.1.3 IP Traceback in a malicious environment	6
1.2 Contributions	7
1.3 Overview	8
Chapter 2: Background	9
2.1 Trust	10
2.2 Piggybacking	11
2.3 Incentives	12
2.4 Enforcement	14
2.5 Summary	15
Chapter 3: Active Network Measurement	16
3.1 Packet loss measurement	18
3.1.1 ICMP-based tools	19
3.1.2 Measurement infrastructures	20
3.2 Loss deduction algorithm	21
3.2.1 TCP basics	21

3.2.2	Forward loss	22
3.2.3	Reverse Loss	24
3.2.4	A combined algorithm	24
3.3	Extending the algorithm	26
3.3.1	Fast ACK parity	26
3.3.2	Sending data bursts	27
3.3.3	Delaying connection termination	29
3.4	Implementation	29
3.4.1	Building a user-level TCP	30
3.4.2	The Sting prototype	31
3.5	Experiences	33
3.6	Summary	35
Chapter 4:	Robust Congestion Signaling	36
4.1	Vulnerabilities	38
4.1.1	TCP review	38
4.1.2	ACK division	39
4.1.3	DupACK spoofing	41
4.1.4	Optimistic ACKing	42
4.2	Implementation experience	45
4.2.1	ACK division	45
4.2.2	DupACK spoofing	46
4.2.3	Optimistic ACKing	47
4.2.4	Applicability	48
4.3	Solutions	49
4.3.1	Designing robust protocols	49
4.3.2	ACK division	50
4.3.3	DupACK spoofing	50
4.3.4	Optimistic ACKing	52

4.4	Summary	54
Chapter 5:	IP Traceback	56
5.1	Related work	58
5.1.1	Ingress filtering	59
5.1.2	Link testing	60
5.1.3	Logging	62
5.1.4	ICMP Traceback	62
5.2	Overview	63
5.2.1	Definitions	63
5.2.2	Basic assumptions	65
5.3	Basic marking algorithms	66
5.3.1	Node append	66
5.3.2	Node sampling	67
5.3.3	Edge sampling	68
5.4	Encoding issues	72
5.4.1	Compressed edge fragment sampling	72
5.4.2	IP header encoding	79
5.4.3	Assessment	81
5.5	Limitations and future work	82
5.5.1	Backwards compatibility	82
5.5.2	Distributed attacks	83
5.5.3	Path validation	83
5.5.4	Attack origin detection	84
5.6	Summary	85
Chapter 6:	Conclusion	86
6.1	Future Work	88

List of Figures

3.1	Data seeding phase of basic loss deduction algorithm.	22
3.2	Hole filling phase of basic loss deduction algorithm.	23
3.3	Example of basic loss deduction algorithm.	25
3.4	Example of basic loss deduction algorithm with fast ACK parity.	27
3.5	Mapping packets into fewer sequence numbers by overlapping.	28
3.6	Sample output from the sting tool.	31
3.7	Unidirectional loss rates observed across a twenty four hour period.	32
3.8	CDF of the loss rates measured over a twenty-four hour period.	33
4.1	Sample time line for a ACK division attack.	40
4.2	Sample time line for a DupACK spoofing attack.	43
4.3	Sample time line for optimistic ACKing attack.	44
4.4	Time-sequence plot of TCP Daytona <i>ACK division</i> attack.	46
4.5	Time-sequence plot of TCP Daytona <i>DupACK spoofing</i> attack.	47
4.6	Time-sequence plot of TCP Daytona <i>optimistic ACK</i> attack.	48
4.7	Time line for a data transfer using a cumulative nonce.	52
5.1	Network as seen from a victim, V , of a denial-of-service attack.	64
5.2	Node append algorithm.	68
5.3	Node sampling algorithm.	69
5.4	Edge sampling algorithm.	70
5.5	Compressing edge data using transitive XOR operations.	73
5.6	Fragment interleaving for compressed edge-ids.	74
5.7	Reconstructing edge-id's from fragments.	75

5.8 Compressed edge fragment sampling algorithm. 76

5.9 Encoding edge fragments into the IP identification field. 77

5.10 Experimental results for number of packets needed to reconstruct paths of varying lengths. 78

List of Tables

4.1	Operating system vulnerabilities to TCP Daytona attacks.	49
5.1	Qualitative comparison of existing schemes for combating anonymous attacks and the probabilistic marking approach I propose.	59

Acknowledgments

In retrospect, it seems quite improbable that this dissertation was ever written. No reasonable person would have wagered that the shy long-haired guy with so-so grades and a degree in history was a viable candidate for a PhD in computer science. Yet I have been fortunate enough to be surrounded by unreasonable people. I would like to thank them now.

During my tenure at UW I have had two wonderful advisors, Brian Bershad and Tom Anderson, who helped me in more ways than I can mention. I am first indebted to Brian, who took a chance on me in the beginning, drove me across the country to Seattle, got me into graduate school, taught me how to write a paper, how to give a talk, how to win an argument and was a never-ending source of support and inspiration – for these things I will always be grateful. I also could not have succeeded without Tom, who got me started in networking and provided great insight, guidance, enthusiasm and endless patience as I developed my research agenda and ultimately this dissertation.

In addition to my official advisors, I benefited from the “unofficial” mentoring of many other faculty in CSE. Anna Karlin taught me to like theory while John Zahorjan gave me a sense of ethics. Together they gave me PJ Harvey, late nights and loud music. David Wetherall was a partner in much of my work and stayed excited when no one else was. Ed Lazowska supported me in all things, above and beyond the call of duty, as he always does. Hank Levy was my academic grandfather and taught me that I could always do better.

I would also like to thank the CSE support staff, who were absolutely first rate and made it easy to get things done. I am especially indebted to Frankye Jones and Lindsay Michimoto, who helped me get through graduate school in spite of myself, Erik Lundberg, Jan Sanislo and Nancy Burr, who all helped me out in a crisis at one time or another, and Melody Kadenko-Ludwa who not only solved my problems on a regular basis, but also kept me informed about any and all goings on.

My fellow students guided me through school and taught me most of what I know. Its impossible

to thank all of them, but a few stand out. Dylan McNamee and Raj Vaswani took me under their collective wings early on and taught me to like coffee, Thai food, good movies and alternative music. Neal Lesh showed me the Zen of table tennis and Ruth Anderson helped me run over 12 miles. Geoff Voelker was a fellow Electric Cookie Monster and brought me to San Diego for the first time. Neal Cardwell was my comrade in arms in all things networking and musical, the hardest working conga-playing hacker in a tuxedo I will ever know. Przemek Pardyak provided some of the best and most comical debates I have ever had while Amin Vahdat and Wilson Hsieh kept me sane. I'd like to thank the SPIN group (David Becker, David Dion, Marc Fiuczynski, Charlie Garrett, Robert Grimm, Wilson Hsieh, Tian Lim, Przemek Pardyak, Yasushi Saito, and Gun Sirer) for the unique opportunity to help build a new system. Similarly, I would like to thank my networking partners (Amit Aggarwal, Neal Cardwell, Andy Collins, David Ely, and Eric Hoffman) for helping me learn from scratch.

Finally, I owe the greatest debt to my family. My parents always supported me unconditionally and gave me both the ambition to succeed and the understanding that its ok to fail too. My wife Tami was a constant source of love and support and I am deeply grateful for her patience and encouragement while I finished my degree.

Parts of this dissertation have been published previously as conference or journal papers. Chapter 3 is based on the paper *Sting: a TCP-based Network Measurement Tool* published in the Proceedings of the 1999 USENIX Symposium on Internet Systems and Technologies [Savage 99]. Chapter 4 is based on the paper *Congestion Control with a Misbehaving Receiver* published in ACM Computer Communications Review [Savage et al. 99a]. Finally, Chapter 5 is based on the paper *Practical Support for IP Traceback* versions of which appeared in the Proceedings of the 2000 ACM SIGCOMM Conference [Savage et al. 00] and ACM/IEEE Transactions on Networking [Savage et al. 01].

Chapter 1

Introduction

The collection of interconnected networks forming “the Internet” is one of the largest communications artifacts ever built. Millions of users, ranging from private individuals to Fortune 500 businesses, all depend on the Internet for day-to-day data communications needs – including e-mail, information search and retrieval, e-commerce, software distribution, customer service and supply chain management. However, the Internet achieved this scale in a very different manner from the Public Switched Telephone Networks (PSTN) that preceded it. Unlike the Bell System of old, the Internet is not a single network, but rather a loose confederation of several thousand *independent* networks that exchange data in a semi-cooperative fashion to present the “illusion” of a single entity. Moreover, while PSTN’s tend to be technologically homogenous, networks in the Internet are built from many different combinations of components supplied by thousands of different hardware and software vendors. Finally, unlike telephone networks, the Internet is not centrally controlled or administered. Instead, each content provider, network service provider and user is free to manage their own resources and network connectivity according to local policies.

The key technological elements underlying the Internet’s architecture are *packet switching* and *internetworking*. Packet switching allows data transmission to be decoupled from resource allocation – each chunk of data is encapsulated in a packet and sent hop-by-hop along some path to its destination. Internetworking, in particular the Internet Protocol (IP), provides a common network-layer substrate for communicating across heterogeneous network media. Together, these two technologies provide a loosely coupled environment in which many different networks can easily connect and interoperate without any central controlling authority. While the simplicity of this architecture has been essential to the Internet’s tremendous growth, it has also posed a number of unique challenges:

- *Protocol compatibility.* Since the Internet is composed of many heterogeneous communications elements it is impossible to guarantee that each will behave in an identical manner. Different vendors implement protocols independently and yet these implementations must somehow interact in a compatible manner – as Jon Postel famously wrote to protocol implementers, “Be liberal in what you accept, and conservative in what you send.” [Postel 81b, Braden 89].
- *Incremental deployability.* With thousands of different vendors and millions of users, it is impossible to upgrade any common component of the Internet universally. Consequently, all changes must be both incremental and backwards compatible. For example, common protocols such as the Transmission Control Protocol (TCP) and the Border Gateway Protocol (BGP) explicitly negotiate to determine which features are supported by each implementation [Postel 81c, Rekhter et al. 95].
- *Administrative heterogeneity.* Lacking centralized administration, the Internet is not run according to a well-defined set of rules or regulations. Each user, organization, or network service provider on the Internet may have its own unique social, political or economic motivations. Consequently, any particular communication service is ultimately governed only by the interests of the involved parties – which may range from fully cooperative, to disinterested, to competitive or even explicitly malicious.

These challenges, in combination, place considerable pressure on network protocol designers. Since any user is free to manipulate the network to satisfy their own goals, it is hard to depend on the presence of any service, on its correct operation, or on the accuracy of any service requests. The traditional means of solving such problems in distributed systems is through a central point of control that enforces system-wide invariants. Unfortunately, the Internet’s decentralized administrative structure does not provide a natural point to implement such a solution. Instead, these properties must be guaranteed in a distributed fashion – by protocols and services that are resilient to potential conflicts of interests among their users.

1.1 Goals

The goal of this dissertation is to study how existing protocols can be adapted to accommodate differences in motivation while still preserving sufficient backward compatibility to allow such changes to be incrementally deployed. My approach is to study by example. I explore the design space of solutions through several problems that cover the spectrum of competing interests – including uncooperative, competitive and malicious peer relationships. The following sections describe each of the specific problems in turn and the individual research challenges they pose.

1.1.1 Active network measurement in an uncooperative environment

A crucial issue in operating large networks or network services is being able to measure and troubleshoot the performance of the underlying network path used. In a homogenous network environment, the network itself might provide such a service and thereby guarantee the availability of network measurement information. However, in a heterogeneous Internet environment, the network layer provides few services and such measurements must be obtained end-to-end between pairs of hosts. For example, a client may measure end-to-end network performance to select among otherwise identical server replicas [Carter et al. 97, Francis et al. 01], or a site may use such measurements to reroute traffic around a congested network exchange point [RouteScience, SockeyeNetworks, Anderson et al. 01]. Collecting such end-to-end measurements requires cooperation from both endpoints – one host sends a network measurement probe and the target host responds accordingly. Among a small set of administratively homogenous hosts, it is easy to provide such functionality through a measurement service installed at every host or network element [Paxson et al. 98b, Almes 97]. However, this approach does not transfer well to the Internet since there is neither a mechanism nor an incentive to ensure that arbitrary remote sites will provide measurement services for the benefit of others.

Existing network path measurement tools, such as `ping`, estimate network characteristics such as packet loss and path latency by leveraging “built-in” features of the Internet Control Message Protocol (ICMP) [Postel 81a] such as the ability to “echo” packets from a remote host. This approach, while today’s “best practice”, has several critical limitations. First, this technique is increasingly undermined by network administrators who treat ICMP traffic differently from regular traffic. Since

ICMP is not required for the correct operation of most Internet-based services (e.g. Web, E-mail) and is seen as a potential security risk (including intelligence gathering [Vaskovich , Vivo et al. 99] and denial-of-service [CERT 96, CERT 97, CERT 98]), such traffic is frequently dropped or rate-limited at the border of many networks. The second problem is that ICMP-based tools can only measure round-trip path properties. Due to large disparities in directional traffic load (e.g. Web servers are net exporters of data) and common network routing policies that promote asymmetry, it is common that packets from client to server experience very different conditions than packets traveling the opposite path from server to client [Paxson 97b, Savage 99]. Understanding this asymmetry is essential to operational troubleshooting, traffic engineering and research. However, unidirectional path measurements generally require stateful measurements at both endpoints; a requirement that is seemingly impossible to satisfy without explicit cooperation between both parties.

The first part of this dissertation explores an alternative approach to network path measurement that avoids the limitations of ICMP and sidesteps the need for explicit cooperation. Since most Internet services are based on the standard Transmission Control Protocol (TCP), network measurement tools can avoid common filtering or rate-limiting by implicitly encoding network performance queries within legitimate TCP messages. In this manner, the goals of the remote endpoint – to provide a standard service (e.g. E-mail, Web, etc.) – are aligned with the needs of network path measurement. Moreover, by treating TCP as a “black box”, it is possible to exploit the protocol’s existing behavior to provide a new service – reliable asymmetric path measurements – *without* explicit cooperation from the remote host. In particular, I explore this approach to network measurement in the context of asymmetric packet loss measurement. In Chapter 3, I describe techniques for reliably measuring unidirectional packet loss rates to any Internet host providing a TCP-based service. I implement these techniques in a tool called *sting* and use it to collect the first measurements demonstrating asymmetry in end-to-end packet loss rates. Others have since extended my basic approach and implementation to measure bandwidth [Saroiu et al. 01], latency [Collins 01], packet reordering [Bellardo 01], and protocol compliance [Padhye et al. 01].

1.1.2 Robust congestion signaling in a competitive environment

The Internet is based on packet switching technology in order to leverage the efficiencies of “statistical multiplexing” [Clark 88]. Each host on the network can send data to arbitrary destinations without creating a circuit or reserving bandwidth. If multiple packets need to be transmitted over a given link at the same time, then one will go forward, while the next will be queued to wait its turn. In this way the network can be provisioned according to the *average* arrival rate, and queuing can absorb any short term transients. While this scheme is highly efficient under moderate load, when contention for a link persists, a condition known as congestion, the overall efficiency of the system can plummet and all network users can experience increased packet loss and queuing delay [Jacobson et al. 88].

Today’s Internet depends on a voluntary end-to-end congestion control mechanism to manage any scarce bandwidth resources. Each host must monitor the congestion on its path and limit its sending rate accordingly to approximate a “fair share” of any bandwidth bottleneck [Jacobson et al. 88]. While this good faith approach to resource sharing was appropriate during the Internet’s “kinder and gentler” days, it seems considerably less dependable in today’s competitive environment. In a homogenous environment, the network might “enforce” a bandwidth allocation among all hosts and thereby guarantee fairness and stability [Demers et al. 89, Shenker 94, Stoica et al. 99]. However, given the large number of disparate and competitive networks forming the Internet, such a solution seems unlikely to be deployed in the near future. Instead, we must address the potential for inequity arising from hosts with both the incentive and ability to “cheat” at the congestion signaling protocols in use today.

Fortuitously, most data on the Internet originates from content servers whose administrators have natural social and economic incentives to share bandwidth fairly among their customers. Consequently few, if any, of these servers violate the voluntary congestion control mechanisms incorporated in standard transport protocols (i.e. TCP). Unfortunately, receivers of data (i.e. Web clients) have the opposite incentives – their interest is reducing their own service time by maximizing their own share of the bandwidth at the expense of other competing clients.

In the second portion of this dissertation, I describe design weaknesses in the congestion signaling mechanism used by TCP and other similar protocols that allow misbehaving receivers to

compete unfairly for bandwidth. I demonstrate that simple protocol manipulations at the receiver can coerce a remote server into sending data at arbitrary rates. In Chapter 4, I demonstrate the seriousness of this weakness through a new protocol implementation, called *TCP Daytona*, that forces remote servers to use all available bandwidth when answering its requests. I further show that this weakness is not an innate property of end-to-end congestion control, but simply a limitation of the existing signaling methodology. By considering the competitive nature of the receiver in data retrieval applications it is possible to implement signaling mechanisms that can be explicitly validated and sender-side congestion control that enforces correct behavior. This work has subsequently been extended to include router-based congestion signaling as well [Ely et al. 01b].

1.1.3 *IP Traceback in a malicious environment*

Finally, as recent events demonstrate, Internet hosts are vulnerable to malicious denial-of-service attacks [CERT 00a]. By flooding a victim host or network with packets, an attacker can prevent legitimate users from communicating with the victim. Stopping these attacks is uniquely challenging because the Internet relies on each host to voluntarily indicate the origin of the packets it sends. In a homogenously administered network environment, the network itself might “enforce” the use of correct source address (and this does happen in some individual networks). However, once a packet escapes into the Internet it is no longer possible to enforce such an invariant. Attackers exploit this weakness and explicitly “forge” packets with incorrect source addresses. Consequently, it is frequently impossible to determine the path traveled by an attack – a requirement for strong operational countermeasures and for the gathering of targeted forensic evidence. The key difficulty in addressing this problem is designing a system that is both compatible with the existing architecture and one that does not depend on the correct behavior of endpoints (i.e. cannot be easily evaded by a determined attacker).

In the third part of this thesis, detailed in Chapter 5, I describe an efficient, incrementally deployable, and (mostly) backwards compatible network mechanism that allows victims to trace denial-of-service attacks back to their source by using a combination of random packet marking and mandatory distance calculation. This approach does not rely on end-host behavior, making it resistant to malicious end-host actions, and only requires a subset of the routers in a network to implement the

marking mechanism to be effective.

1.2 Contributions

The central hypothesis of this dissertation is that it is possible to design protocols that work in spite of uncooperative, competitive and malicious hosts by carefully and explicitly accommodating conflicts in motivation. Moreover, I argue that the converse is also true: designing protocols without attending to the potential conflicts between hosts increases the fragility of these protocols and can reduce the robustness of systems that use them. I demonstrate this hypothesis through proof by example and show further that it is possible to accommodate such environments while maintaining sufficient backwards compatibility to allow incremental and speedy deployment. In particular:

- I show that it is possible to measure unidirectional path performance in the absence of explicit cooperation from a network endpoint. I explore the limitations in existing approaches and then describe a technique that leverages the existing interests of Internet users to provide unidirectional packet loss measurements. I implement this approach and demonstrate that it is both accurate and has widespread applicability. Finally, I use the tool to conduct an initial measurement study demonstrating the presence of widespread asymmetry in packet loss rate.
- I show that one can build robust congestion signaling protocols in spite of endpoints that wish to compete for bandwidth on unfair terms. I first describe how existing congestion signaling protocols have significant weaknesses that allow misbehaving receivers to manipulate the rate at which data is sent. I verify this problem through an implementation that exploits weaknesses in TCP to consume unfair quantities of bandwidth. Finally, I show how simple modifications to the signaling protocol and the congestion control mechanisms can align the interests of receivers and senders – thereby enforcing correct behavior.
- I present a method for tracing denial-of-service attacks back through a network in spite of malicious attackers that actively seek to conceal their location. I describe the design tradeoffs inherent in providing such a capability. I develop analytic results concerning the efficacy of probabilistic marking methods and then explore the practical problems required for deploy-

ment. Through a combination of implementation and simulation I demonstrate the ability one such solution to track attacks over networks paths of varying length and composition.

1.3 Overview

The remainder of this dissertation is organized as follows. Chapter 2 provides background and discussion surrounding the problems of administrative heterogeneity and the approaches used to accommodate it. Chapter 3 discusses the application of this methodology to uni-directional network path measurement and demonstrates its value by measuring existing packet-loss asymmetry in today's Internet. In Chapter 4, I explore the problems posed by competitive peers to end-to-end congestion control mechanisms. Chapter 5 covers tracing the origin of spoofed denial-of-service attacks. Finally, Chapter 6 summarizes my results and contributions.

Chapter 2

Background

One of the original goals of the Internet architecture was to overcome the challenges of *network layer* heterogeneity [Clark 88]. At the time, each network technology used a distinct method for physical encoding, media access, addressing and routing. The Internet's designers realized that a common set of minimal network and transport protocols could be used to transparently interconnect networks based on different underlying technologies. Moreover, they reasoned, the same protocols could provide a standard communications substrate for a wide variety of network services and applications. These realizations, subsequently embodied in the IP and TCP protocols [Cerf et al. 98, Postel 81c, Postel 81b], provided the technical basis for *internetworking* which is widely credited with the rapid of growth of the Internet.

However, since each constituent network in the Internet is independently controlled, a byproduct of this success is ever-increasing *administrative* heterogeneity. This in turn threatens the robustness of the Internet's underlying protocols which were largely designed under the assumption that all hosts will cooperate towards a shared set of goals. In small inter-networks it is still possible to approximate a uniform administrative policy by negotiation and rough consensus among the participants. However, with tens of thousands of connected networks and millions of independent users, the Internet has grown to a point where it is naive to assume universal cooperation. In this environment, conflicts of Internet about how Internet resources should be managed are inevitable. While this challenge was observed as early as 1988 – as David Clark wrote, “Some of the most significant problems with the Internet today relate to the lack of sufficient tools for distributed management” [Clark 88] – there has not been any systematic examination of this problem and its impact on network service architecture. However, a number of approaches can be defined among the ad hoc solutions developed by service designers encountering these problems.

2.1 Trust

The simplest, and most pervasive, approach is to only communicate with cooperative users. Generally, this approach is based on a binary worldview in which users fall into one of two categories:

- **Friends.** Will implement a protocol or service correctly and in common interest with all peers.
- **Enemies.** Seek to gain unauthorized access remote computing resources, violate their integrity, eavesdrop on confidential communications and generally disrupt service.

If communication is restricted only to friends then, by definition, a cooperative environment will be maintained and existing protocols and services will operate correctly.

Of course, there is no general way to determine whether a particular user is a truly a friend or an enemy, and so network administrators develop static trust policies that define which users are trusted, and therefore are assumed to be friends, and which are not. For example, a company's employees might be trusted, while customers might not be. Once this initial categorization has been made, a variety of cryptographic mechanisms are brought to bear to guard the integrity of the categories. Trusted users are provided with passwords or other authentication tokens that are used to provide proof that they should be treated as friendly, while untrusted users are unable to provide such evidence. In addition, the communications channel may be cryptographically encoded to provide strong guarantees of confidentiality, integrity, freshness, and non-repudiation for any messages sent between trusted users [Schneier 96]. This basic trust-based approach is at the heart of most network security protocols, including the IPSEC standard [Kent et al. 98], the Secure Shell protocol [Ylonen et al. 00] and the Secure Socket Layer [Dierks et al. 99], and is quite effective at providing access control among known users.

However, trust-based mechanisms have several serious limitations. First, these mechanisms only protect the *differentiation* between trusted and untrusted users. They do **not** ensure that trusted users are in fact friends. Nothing prevents a trusted user from violating a protocol or service specification at any time – it is simply assumed that they will never do so. As the number of users grows large, this faith in trust becomes increasingly fragile. This is especially true for corporate information

security applications since it is widely believed that employees are the source of the most serious computer security breaches.

The second limitation of trust-based mechanisms is that they only accommodate two opposing points in the spectrum of potential conflicts: fully cooperative and fully adversarial. In practice, there are many in-between states, such as users who are non-cooperative or competitive, but non-adversarial. For example, a user may be generally trustworthy, yet unwilling to cooperate with other users in detecting and blocking unwanted e-mails. Similarly, while a customer and its Internet Service Provider may generally trust one another, they may have competing interests about how the customers traffic is routed – the customer would prefer for its packets to take the shortest path to all destinations, while the service provider may have peering agreements with other providers that make such a routing disadvantageous [Norton 01]. Such distinctions are not well captured or addressed using trust-based mechanisms.

Finally, trust-based mechanisms can be expensive to deploy and administer at large scale. Credentials must be created and securely distributed to each participant (usually requiring some kind of out-of-band channel such as postal mail or a personal meeting). This data must be distributed consistently to all pairs of potentially communicating hosts and must be periodically reviewed, renewed and occasionally revoked. As a consequence, trust mechanisms are usually only deployed bilaterally within a single organization, or unilaterally between a single organization and its customers (e.g. e-commerce).

2.2 Piggybacking

It can be extremely difficult to introduce a new service or protocol in the Internet. To be widely useful it must be deployed by a large number of users, each of whom may see little or no benefit until a critical mass is reached, and perhaps not even then. This problem is exacerbated in the case of services that do not have widespread appeal or interest. If a remote network has no interest in cooperating to provide a service, then it is difficult to extend the service to include those users. One approach to this problem is to *piggyback* a new service upon an existing service of greater importance and wider availability. For example, the Alex distributed file system [Cate 92], provides global hierarchical Unix-like file system built upon the widely deployed File Transfer Protocol (FTP) [Pos-

tel et al. 85]. Individual file servers in the Alex system are only required to provide FTP services and usually have no idea they are part of a larger structure.

This approach is particularly well suited to the challenges of Internet-wide network measurement. For a wide variety of operational and application-specific purposes it is useful to measure the performance and behavior of traffic between two points on a network. However, the Internet does not provide any standard network measurement services and few users are willing to deploy network measurement software for the benefit of outside parties. As a result, piggybacking is frequently the only method available for obtaining network measurements. The most well-known examples of this approach are the `ping` and `traceroute` tools which leverage the behavior of the existing Internet Control Message Protocol (ICMP) to obtain end-to-end and hop-by-hop measurements of packet loss and latency.

There are several requirements for this approach to be successful. First, the protocol or service being exploited must have sufficient value that remote users will support it independently of any new service (e.g. Web services, e-mail). Second, piggybacking upon this service should not create an undue burden for the target of this use (e.g. exploiting the relay feature of the SMTP mail protocol to send unsolicited e-mail causes an undue burden and is usually blocked very quickly as a result). Finally, the existing service must have sufficient functionality that the new service can be implemented in terms of it.

Obviously, piggybacking is only useful in the case of an uncooperative user and does not provide any means for controlling competitive or adversarial users. In fact, the same opportunistic techniques used for piggybacking can be used by competitive or malicious users to achieve their own ends.

2.3 Incentives

Another class of approaches is attuned to the conflicts that arise when users compete over shared resources and attempts to accommodate them explicitly through pseudo-economic means. Under this approach, users are compensated appropriately for their actions, whether rewards for behaving in a cooperative fashion or penalties for greedy behavior, leading each users self-interest to reinforce robust network-wide behavior.

The most common venue for this approach is the problem of fairly allocating shared bandwidth among users. When bandwidth is plentiful all users may send data as fast as they desire, however in times of scarcity they must send more slowly or other users will suffer. One approach is to construct router packet scheduling policies, such as Fair-Queuing [Demers et al. 89], that prevent any user from consuming more than their fair share, thereby eliminating the *incentive* for a potentially uncooperative user to send faster than they should [Shenker 94]. Another approach is to standardize a stable and roughly fair distributed congestion control behavior, such as TCP's exponential backoff during congestion and linear increase during bandwidth availability [Jacobson et al. 88]. Using analytic models of such algorithms [Padhye et al. 98], it is possible for the network to observe a network flow and, over time, determine whether it is "friendly" (i.e. conformant to the standard congestion control behavior) or not. If the flow is misbehaved, it is penalized accordingly through artificial rate-limiting – again eliminating any incentive to attempt cheating the system [Floyd et al. 99a, Manajan et al. 01]. Finally, instead of assuming that "fairness" is the most important global goal, some researchers have suggested treating bandwidth as an economic market and constructing bidding protocols for mediating access to it [Gibbens et al. 99, Key et al. 99, Lavens et al. 00]. Under these schemes, bandwidth becomes more expensive during times of congestion, leading each user to only bid as much as the bandwidth is worth – thereby maximizing the total utility of the network. This creates an incentive structure that not only prevents the rational user from sending more quickly than necessary, but also accommodates the reality that some users and some applications are more important than others. In addition to bandwidth sharing, similar schemes are being explored for sharing storage in peer-to-peer file-sharing systems [Mojonation 01].

These incentive-based approaches are still in their infancy, but appear promising for addressing conflicts between users with competitive interests. However, they are not appropriate for all conflicts of interest. For example, adversarial users are out to punish their enemy rather than optimize their own resource usage. Consequently, incentive structures that assume greedy self-interest will have little leverage in this situation. For the same reason, a user who has no interest in a service or resource cannot be enticed to participate by providing them more of it.

2.4 Enforcement

Finally, for addressing the problems of adversarial conflicts, the only clear solution is to dynamically detect and stop malicious actions as they occur, thereby enforcing cooperative behavior. Common examples of this approach include the network firewall, intrusion detection systems and virus detectors. All define a set of malicious actions which are evaluated against arriving network traffic. If network traffic is misbehaved then an appropriate countermeasure (e.g. blocking those packets from entering the network) is taken to stop or mitigate the malicious behavior. Enforcement-style approaches have been explored for a variety of situations including preventing remote host fingerprinting [Smart et al. 00], blocking certain classes denial-of-service attacks [Greene et al. 01], normalizing the control signals in TCP/IP packets [Handley et al. 01] and for validating intra-domain packet forwarding [Bradley et al. 98].

There are several requirements for enforcing correct behavior on a protocol or service. First, it must be possible to define correct behavior. Second, it must be possible to reliably distinguish correct behavior from malicious behavior. This can be accomplished by defining known “correct” behavior (e.g. a firewall ruleset contains the set of allowable packet contents), known “incorrect” behavior (e.g. an intrusion detection system contains a list of disallowed packet contents) or by some dynamic challenge mechanism. Finally, the “enforcer” must be in a position to prevent attackers from accomplishing their goal.

These seemingly simple requirements can be very hard to accommodate in practice. Many higher-level services are sufficiently complex that a formal description of correct behavior may not exist, or be feasible to create. Moreover, protocols that are not designed to allow enforcement may not contain sufficient information to distinguish correct actions from those of an adversary. Finally, for certain kinds of attacks, such as denial-of-service, the ideal location for enforcement actions may not be within the domain of the victim. For example, wide-area network routing is vulnerable to malicious attacks in which false routes are advertised into the network – either to divert traffic for eavesdropping or to deny service. Unfortunately, since each network is allowed to manage their routing policy independently there are few invariants upon which to establish a “correct” behavior. Moreover, wide-area network routing protocols do not contain sufficient information to evaluate whether a router advertisement is suspicious or not. Finally, a false routing advertisement for a

victim's network will impact how many other networks reach the victim. Consequently, there is nothing the victim can do directly to enforce the correct behavior – the correct behavior must be enforced by those other networks.

2.5 Summary

As the Internet grows in scale, so too grows the potential for resource conflicts among its users. There is little previous work that explicitly examines how such conflicts of interest may impact existing network protocols and services. However, there are several distinct approaches that I have synthesized from individual attempts to address some of these problems. Most common among these is the static trust approach, which statically limits the scope of users in order to (ideally) approximate a homogenous environment. This solution is by far the best understood and, as well, the most limited.

Less well developed are the piggybacking, incentive and enforcement approaches, which are protocol design methodologies that are oriented towards particular types of user conflicts. Piggybacking allows new services to be deployed in environments where users have not interest in cooperating to implement the service. By implementing the new service transparently in terms of an existing service cooperation can be obtained implicitly. In situations where users compete over shared resources, a more appropriate solution is to dynamically reward or punish a user thereby creating strong incentives for cooperative behavior. Finally, to control the actions of malicious users a network must validate and enforce the “correctness” of service requests and protocol signaling. In this dissertation I have focused predominantly on exploring these approaches and demonstrating how far they may be leveraged in different contexts.

Chapter 3

Active Network Measurement

This thesis considers three examples of uncooperative behavior: uncooperative, competitive and malicious. In this chapter, I consider an example of the first: how to obtain accurate end-to-end packet path measurements with an uncooperative endpoint.

Network measurements are absolutely essential for managing the performance and availability of any distributed system as well as for designing future distributed services. For example, most content providers employ some form of network measurement to monitor the performance of their servers and service providers use similar measurements to monitor their key services and to detect failures and congestion. As well, end-to-end network measurement is key for new distributed services that seek to optimize the use of the network. For example, many content delivery systems utilize such measurements to optimize the selection of “nearby” replicas or cached copies [Johnson et al. 01]. Similar methods are used by multi-player interactive games to select low-latency servers [Gameranger 01] and by Internet Service Providers to optimize the problem of network route selection [RouteScience , SockeyeNetworks]. Finally, end-to-end network measurements are the basic source of data for researchers to examine the dynamics of Internet behavior [Paxson 97b, Paxson 97a, Padhye et al. 01, Saroiu et al. 01, Savage et al. 99b].

There are two distinct approaches to network measurement. Passive network measurements, such as packet traces, are those which can be inferred simply by monitoring existing traffic as it passed an engineered measurement point. Passive measurements are ideal for understanding user workloads, but are limited for operational monitoring of a network because there is no control over what aspects of the network are measured, when the measurements take place, or how they are collected. By contrast, active network measurements involve injecting probe packets into the network and observing how, if and when they are delivered to their destination. These probes are used as estimates of the conditions that other packets may experience while traveling from one

host to another. Active measurements are ideal for monitoring network infrastructures because they provide the user with precise control over what, when and how a measurement takes place. This flexibility makes active measurements the prevailing method for optimizing and troubleshooting interactions between distributed applications and the Internet infrastructure.

In general, active end-to-end network measurement requires the cooperation of three parties: the initiating source host, the remote target host and the intervening network. The source host must correctly issue probe packets into the network, record any response packets received, and maintain state about the number and timing of each. The target host must cooperate by responding to these probes promptly, in a consistent manner, and with enough information to identify key network characteristics such as loss and delay. Finally, the network itself must cooperate by forwarding probe packets and responses as through they were regular traffic.

Unfortunately, the Internet architecture was not designed with performance measurement as a primary goal and therefore has few “built-in” services that support this need [Clark 88]. Moreover, there is no requirement that the network or the target host cooperate for this purpose. It is quite common for networks and servers to treat measurement probes in a manner quite different from normal application traffic. Consequently, today’s measurement tools must either “make do” with the imperfect services provided by the Internet, or deploy substantial new infrastructures geared towards measurement. Finally, the common services used for network measurement do not contain sufficient information to differentiate conditions that occur en route from the source host to the remote host from those conditions that are experienced in the reverse direction. This distinction is increasingly critical as network path properties are highly asymmetric and performance/availability issues are frequently localized to a particular direction.

Resolving these problems raise a number of interesting challenges. What mechanisms are necessary for unidirectional network measurements? How can these mechanisms be implemented and deployed on the existing Internet? How can remote hosts be convinced to cooperate in providing a measurement service? What can be done to ensure that the network will also cooperate?

To examine these questions, in this chapter I present a network measurement approach, explored in the context of packet loss measurement, that does not require explicit cooperation from the network or the remote end-hosts that are being measured. Instead, I show how *implicit cooperation* can be obtained by overloading an existing TCP-based services to extract essential measurements.

Since hosts and networks alike have a strong interest in providing reliable and efficient content delivery services (e.g. Web, E-mail), we can leverage these services to “coerce” cooperation from the existing Internet without requiring any additional deployment of services. I present a new tool, called *sting*, that uses TCP to measure the packet loss rates between a source host and some target host. Unlike traditional loss measurement tools, *sting* is able to precisely distinguish which losses occur in the forward direction on the path to the target and which occur in the reverse direction from the target back to the source. Moreover, the only requirement of the target host is that it run some TCP-based service, such as a Web server.

My experiences show that this approach is very powerful and is able to provide high-quality measurements to arbitrary points on the Internet. Using an initial prototype, I show that there is strong packet loss asymmetry to popular content providers – a result that previously would have been infeasible to obtain.

The remainder of this chapter is organized as follows: In section 3.1 I review the current state of practice for measuring packet loss. Section 3.2 contains a description of the basic loss deduction algorithms used by *sting*, followed by extensions for variable packet size and inter-arrival times in section 3.3. I briefly discuss my implementation in section 3.4 and present some preliminary experiences using the tool in section 3.5.

3.1 Packet loss measurement

The rate at which packets are lost can have a dramatic impact on application performance. For example, it has been shown that for moderate loss rates (less than 15 percent) the bandwidth delivered by TCP is proportional to $1/\sqrt{\text{lossrate}}$ [Mathis et al. 97]. Consequently, a loss rate of only a few percent can limit TCP performance to well under 10Mbps on most paths. Similarly, some streaming media applications only perform adequately under low loss conditions [Carle et al. 97]. For example, the popular RealPlayer software suite is frequently configured to drop video playback to a single frame per second during periods of any substantial packet loss.

Not surprisingly, there has always been a long-standing operational need to measure packet loss; the popular *ping* tool was developed less than a year after the creation of the Internet. These tools, and those derived from the same methodologies have been used for the last 20 years to conduct both

operational and research measurements of loss rates in the network [Paxson 97a, Bolot 93, Savage et al. 99b, CAIDA 00]. In the remainder of this section we'll discuss two dominant methods for measuring packet loss: tools based on the Internet Control Message Protocol (ICMP) [Postel 81c] and peer-to-peer network measurement infrastructures.

3.1.1 ICMP-based tools

Common ICMP-based tools, such as `ping` and `traceroute`, send probe packets to a host, and estimate loss by observing whether or not response packets arrive within some time period. There are two principle problems with this approach:

- *ICMP filtering.* ICMP-based tools rely on the near-universal deployment of the *ICMP Echo* or *ICMP Time Exceeded* services to coerce response packets from a host [Postel 81a, Braden 89]. Unfortunately, malicious use of ICMP services has led to mechanisms that restrict the efficacy of these tools. Several host operating systems (e.g. Solaris) now limit the rate of ICMP responses, thereby artificially inflating the packet loss rate reported by `ping`. For the same reasons many enterprise networks (e.g. microsoft.com) filter ICMP packets altogether. Some firewalls and load balancers respond to ICMP requests on behalf of the hosts they represent, a practice I call *ICMP spoofing*, thereby precluding real end-to-end measurements. Finally, many service provider networks now rate limit all inbound ICMP traffic to limit the impact of “Smurf” attacks based on ICMP [CERT 98, Hancock 00]. It is increasingly clear that ICMP’s future usefulness as a measurement protocol will be reduced [Rapier 98].
- *Loss asymmetry.* The packet loss rate on the forward path to a particular host is frequently quite different from the packet loss rate on the reverse path from that host. There are multiple reasons for this. First, the client/server architecture embodied in most Internet applications tends to present very different traffic loads on the network – servers are net producers of data, while clients tend to be predominantly consumers. Second, the growth of hosting and collocation services have aggregated and concentrated content servers in the network, while the development of wholesale and retail consumer access services (e.g. ZipLink, AOL) have achieved the same ends with clients. Finally, the “hot-potato” routing policies used by

most major Internet networks naturally produce asymmetric routes where the set of routers traversed from client to servers is different from the return path from server to client. Unfortunately, without any additional information from the receiver, it is impossible for an ICMP-based tool to determine if its probe packet was lost or if the response was lost. Consequently, the loss rate reported by such tools is really:

$$1 - ((1 - loss_{fwd}) \cdot (1 - loss_{rev}))$$

Where $loss_{fwd}$ is the loss rate in the forward direction from source host to target host and $loss_{rev}$ is the loss rate in the reverse direction. Loss asymmetry is important, because for many protocols the relative importance of packets flowing in each direction is different. In TCP, for example, losses of acknowledgment packets are tolerated far better than losses of data packets [Balakrishnan et al. 97]. Similarly, for many streaming media protocols, packet losses in the opposite direction from the data stream have little or no impact on overall performance. Finally, the ability to measure loss asymmetry allows a network engineer to detect and localize network bottlenecks which may not be evident from round-trip measurements.

3.1.2 Measurement infrastructures

In contrast, wide-area peer-to-peer measurement infrastructures, such as NIMI and Surveyor, deploy measurement software at both the sender and the receiver to correctly measure one-way network characteristics [Paxson 97b, Paxson et al. 98b, Almes 97]. Such approaches are technically ideal for measuring packet loss because they can precisely observe the arrival and departure of packets in both directions. The obvious drawback is that the measurement software is not widely deployed and therefore measurements can only be taken between a restricted set of hosts. My work does not eliminate the need for such infrastructures, but allows their measurements to be extended to include parts of the Internet that are not directly participating. For example, access links to Web servers can be highly congested, but they are not visible to current measurement infrastructures.

Finally, there is some promising work that attempts to derive per-link packet loss rates by correlating measurements of multicast traffic among many different receiving hosts [Caceres et al. 99]. The principle benefit of this approach is that it allows the measurement of N^2 paths with $O(N)$ messages. The slow deployment of wide-area multicast routing currently limits the scope of this

technique, but this situation may change in the future. However, even with universal multicast routing, multicast tools require software to be deployed at many different hosts, so, like other measurement infrastructures, there will likely still be significant portions of the commercial Internet that can not be measured with them.

My approach is similar to existing tools in that it only requires participation from the sender. However, by using TCP for probing the path rather than ICMP, there are several key advantages. First, using TCP eliminates the network filtering problem. Because TCP is essential to most popular Internet services (e.g. Web and e-mail), providers have no incentive to block or limit its use and the probes more closely match the network conditions encountered by application TCP packets. Second, unlike ICMP, TCP's behavior can be exploited to reveal the direction in which a packet was lost. In the next section I describe the algorithms used to accomplish this.

3.2 *Loss deduction algorithm*

To measure the packet loss rate along a particular path, it is necessary to know how many packets were sent from the source and how many were received at the destination. From these values the one-way loss rate can be derived as:

$$1 - (\text{packetsreceived}/\text{packetssent})$$

Unfortunately, from the standpoint of a single endpoint, one cannot observe both of these variables directly. The source host can measure how many packets it has sent to the target host, but it cannot know how many of those packets are successfully received. Similarly, the source host can observe the number of packets it has received from the target, but it cannot know how many more packets were originally sent. In the remainder of this section I will explain how TCP's error control mechanisms can be used to derive the unknown variable, and hence the loss rate, in each direction.

3.2.1 *TCP basics*

Every TCP packet contains a 32 bit sequence number and a 32 bit acknowledgment number. The sequence number identifies the bytes in each packet so they may be ordered into a reliable data stream. The acknowledgment number is used by the receiving host to indicate which bytes it has

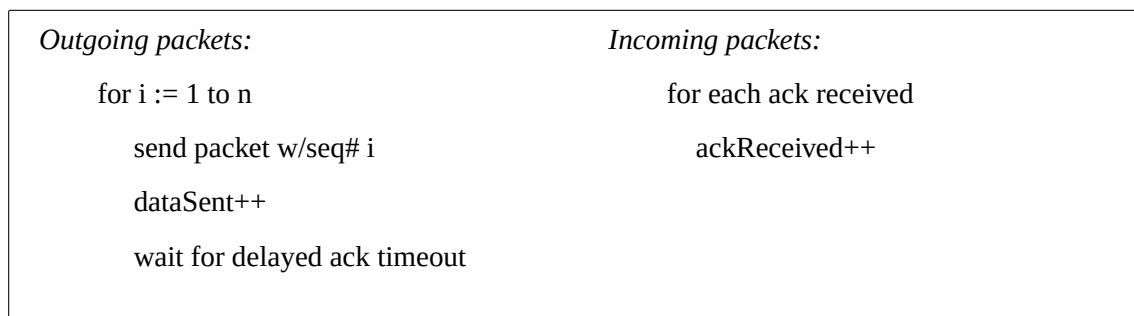


Figure 3.1: Data seeding phase of basic loss deduction algorithm.

received, and indirectly, which it has not. When in-sequence data is received, the receiver sends an acknowledgment specifying the next sequence number that it expects and implicitly acknowledging all sequence numbers preceding it. Since packets may be lost, or reordered in flight, the acknowledgment number is only incremented in response to the arrival of an in-sequence packet. Consequently, out-of-order or lost packets will cause a receiver to issue duplicate acknowledgments for the packet it was expecting.

3.2.2 Forward loss

Deriving the loss rate in the forward direction, from source to target, is straightforward. The source host can observe how many data packets it has sent, and then can use TCP's error control mechanisms to query the target host about which packets were received. Accordingly, I divide my algorithm into two phases:

- *Data-seeding.* During this phase, the source host sends a series of in-sequence TCP data packets to the target. Each packet sent represents a binary sample of the loss rate, although the value of each sample is not known at this point. At the end of the data-seeding phase, the measurement period is concluded and any packets lost after this point are not counted in the loss measurement.
- *Hole-filling.* The hole-filling phase is about discovering which of the packets sent in the previous phase have been lost. This phase starts by sending a TCP data packet with a sequence number *one greater* than the last packet sent in the data-seeding phase. If the target responds

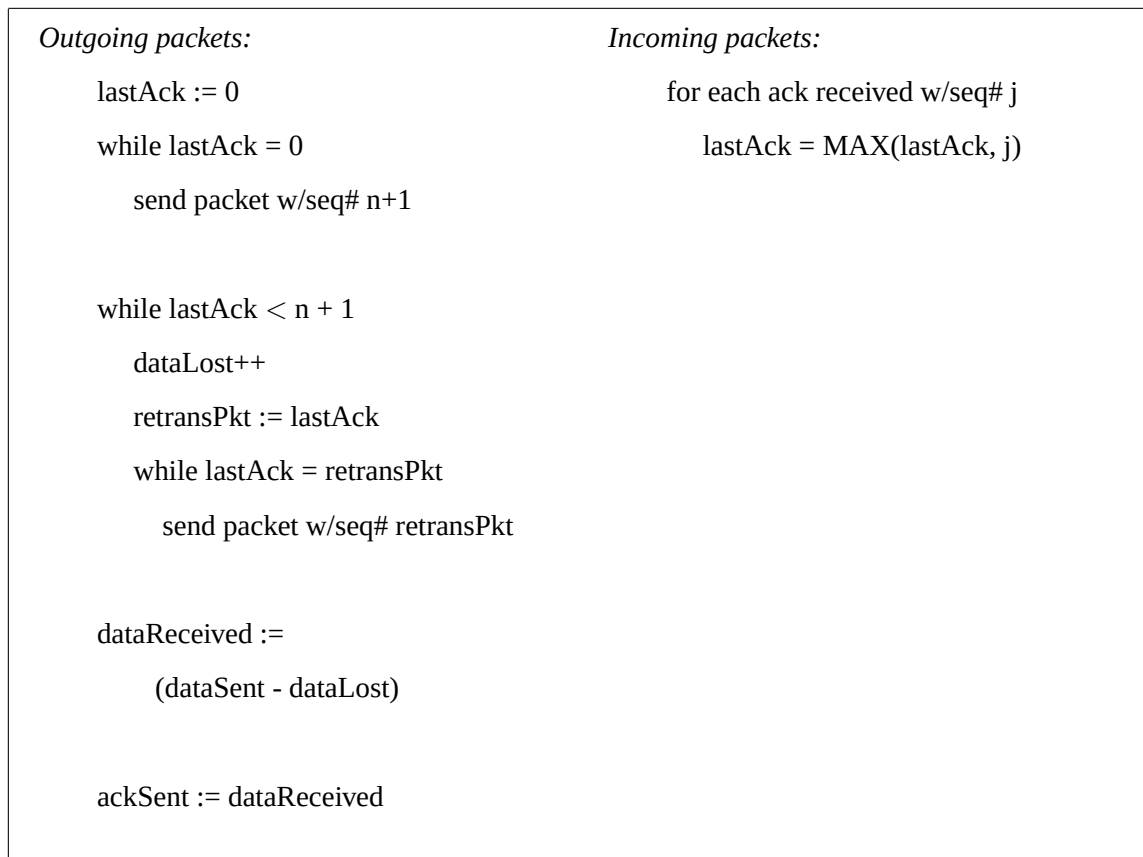


Figure 3.2: Hole filling phase of basic loss deduction algorithm.

by acknowledging this packet, then no packets have been lost. However, if any packets have been lost there will be a “hole” in the sequence space and the target will respond with an acknowledgment indicating exactly where the hole is. For each such acknowledgment, the source host retransmits the corresponding packet, thereby “filling the hole”, and records that a packet was lost. This procedure is repeated until the last packet sent in the data-seeding phase has been acknowledged. Unlike data-seeding, hole-filling must be reliable and so the implementation must timeout and retransmit its packets when expected acknowledgments do not arrive.

3.2.3 *Reverse Loss*

Deriving the loss rate in the reverse direction, from target to source, is somewhat more problematic. While the source host can count the number of acknowledgments it receives, it is difficult to be certain how many acknowledgments were sent. The ideal condition, which I refer to as *ACK parity*, is that the target sends a single acknowledgment for every data packet it receives. Unfortunately, most TCP implementations use a *delayed acknowledgment* scheme that does not provide this guarantee. In these implementations, the receiver of a data packet does not respond immediately, but instead waits for an additional packet in the hopes that the cost of sending an acknowledgment can be amortized [Braden 89]. If a second packet has not arrived within some small timeout (the standard limits this delay to 500ms, but 100-200ms is a common value) then the receiver will issue an acknowledgment. If a second packet does arrive before the timeout, then the receiver generally issues an acknowledgment immediately.¹ Consequently, the source host cannot reliably differentiate between acknowledgments that are lost and those which are simply suppressed by this mechanism.

An obvious method for guaranteeing ACK parity is to insert a long delay after each data packet sent. This will ensure that a second data packet never arrives before the delayed acknowledgment timer forces an acknowledgment to be sent. If the delay is long enough, then this approach is quite robust. However, the same delay limits the technique to measuring packet losses over long time scales. To investigate shorter time scales, or the correlation between the sending rate and observed losses, another mechanism must be used. I will discuss alternative mechanisms for enforcing ACK parity in section 3.3.

3.2.4 *A combined algorithm*

Figures 3.1 and 3.2 contain simplified pseudo-code for the algorithm as I have described it. Without loss of generality, I assume that the sequence space for the TCP connection starts at 0, each data packet contains a single byte (and therefore consumes a single sequence number), and data packets are sent according to a periodic distribution. When the algorithm completes, I calculate the packet

¹While TCP standards documents indicate that an TCP receiver should not delay more than one acknowledgement, there are a number of implementations that will not acknowledge a second packet immediately.

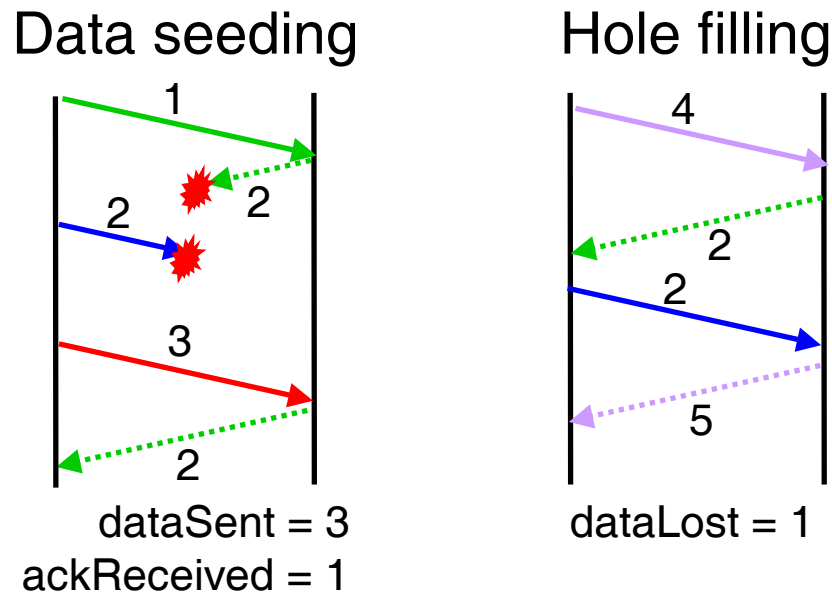


Figure 3.3: Example of basic loss deduction algorithm.

loss rate in each direction as follows:

$$Loss_{fwd} = 1 - (\text{dataReceived}/\text{dataSent})$$

$$Loss_{rev} = 1 - (\text{ackReceived}/\text{ackSent})$$

Figure 3.3 illustrates a simple example. In each time-line the left-hand side represents the source host and the right-hand side represents the target host. Right-pointing arrows are labeled with their sequence number and left-pointing arrows with their acknowledgment number. Here, the first data packet is received, but its acknowledgment is lost. Subsequently, the second data packet is lost. When the third data packet is successfully received, the target responds with an acknowledgment indicating that it is still waiting to receive packet number two. At the end of the data seeding phase, the source host knows that three data packets have been sent and one acknowledgement has been received.

In the hole filling phase, a fourth packet is sent and the source host receives a corresponding acknowledgment indicating that the second packet was lost. The loss is recorded and then the missing packet is retransmitted. The subsequent acknowledgment for the fourth packet indicates that the other two data packets were successfully received. Consequently, the following packet loss rate estimations can be calculated:

$$Loss_{fwd} = 1 - (2/3) = 33\%$$

$$Loss_{rev} = 1 - (1/2) = 50\%$$

These results are correct since during the measurement phase two of three packets sent to the target are received and one of two acknowledgements are received.

3.3 Extending the algorithm

The algorithm I have described is fully functional, however it has several unfortunate limitations, which I now remedy.

3.3.1 Fast ACK parity

First, the long timeout used to guarantee ACK parity restricts the tool to examining background packet loss over relatively large time scales. To examine losses over shorter time scales, or explore correlations between packet losses and packet bursts sent from the source, the long delay requirement must be eliminated.

An alternative technique for forcing ACK parity is to take advantage of the *fast retransmit* algorithm contained in most modern TCP implementations [Stevens 94]. This algorithm is based on the premise that since TCP always acknowledges the last in-sequence packet it has received, a sender can infer a packet loss by observing duplicate acknowledgments. To make this algorithm efficient, the delayed acknowledgment mechanism is *suspended* when an out-of-sequence packet arrives. This rule leads to a simple mechanism, shown in Figure 3.4, for guaranteeing ACK parity: during the data seeding phase the first sequence number is skipped, thereby ensuring that all data packets are sent, and received, out-of-sequence. Consequently, the receiver will immediately respond with an

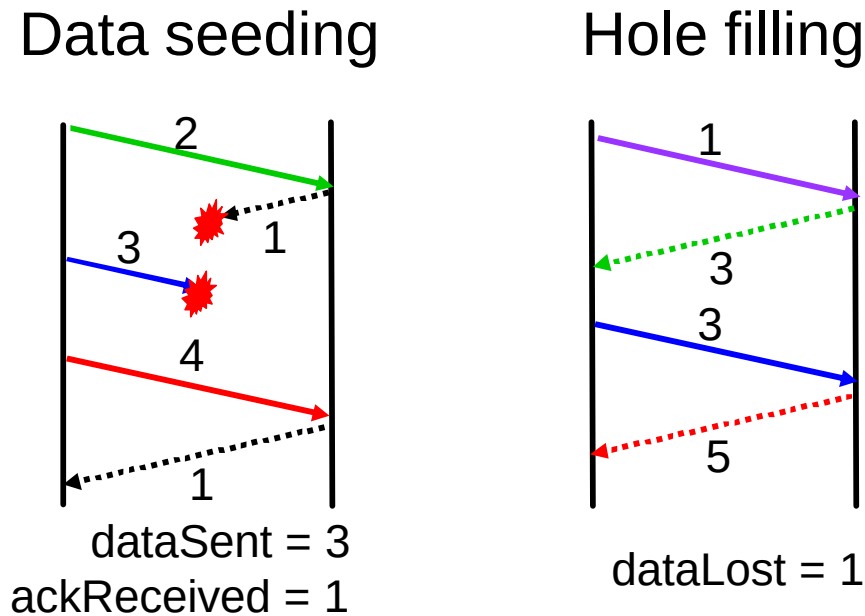


Figure 3.4: Example of basic loss deduction algorithm with fast ACK parity.

acknowledgment for each data packet received. The hole filling phase is then modified to transmit this first sequence number instead of the next in-sequence packet.

3.3.2 Sending data bursts

The second limitation is that large packets cannot be sent. The reason for this is that the amount of buffer space provided by the receiver is limited. Many TCP implementations default to 8KB receiver buffers. Consequently, the receiver can accommodate no more than five 1500 byte packets, a number too small to be statistically significant. While one could simply create a new connection and restart the tool, this limitation prevents the investigation of loss conditions during larger packet bursts.

Luckily, most TCP implementations *trim* packets that overlap the sequence space that has already been received. Consequently, if a packet arrives that overlaps a previously received packet, then the receiver will only buffer the portion that occupies “new” sequence space. By explicitly overlapping the sequence numbers of probe packets, every other large packet can be mapped into a

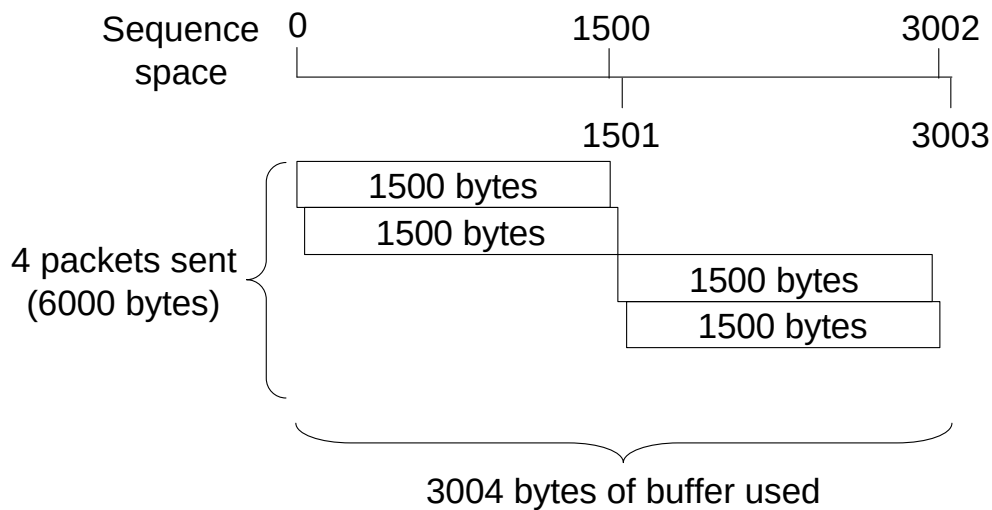


Figure 3.5: Mapping packets into fewer sequence numbers by overlapping.

single byte of sequence space, and hence only one byte of buffer at the receiver. Consequently, the effective buffer space at the receiver can be roughly doubled.

Figure 3.5 illustrates this technique. The first 1500 byte packet is sent with sequence number 1500, and when it arrives at the target it occupies 1500 bytes of buffer space. However, the next 1500 byte packet is sent with sequence number 1501. The target will note that the first 1499 bytes of this packet have already been received, and will only use one bytes of buffer space. The next packet is sent with sequence number 3002, effectively following the last byte of the second packet and restarting the pattern. This technique maps every other packet into a single sequence number, thereby halving the buffering limitation. For example, of the 6000 bytes transmitted in Figure 3.5, only 3004 bytes must be buffered by the receiver. However, this approach only permits data bursts to be sent in one direction – towards the target host. Coercing the target host to send arbitrarily sized bursts of data back to the source is more problematic since TCP’s congestion control mechanisms normally control the rate at which the target may send data. I have investigated techniques to remotely bypass TCP’s congestion control [Savage et al. 99a] but they are not suited for common

measurement tasks as they represent an overall security risk.

3.3.3 *Delaying connection termination*

One final problem is that some TCP servers do not close their connections in a graceful fashion. TCP connections are full-duplex – data flows along a connection in both directions. Under normal conditions, each “half” of the connection may only be closed by the sending side (by sending a FIN packet). The algorithms implicitly assume this is true, since it is necessary that the target host respond with acknowledgments until the testing period is complete. While most TCP-based servers follow this termination protocol, some Web servers simply terminate the entire connection by sending a RST packet – sometimes called an *abortive release*. Once the connection has been reset, the sender discards any related state so any further probing is useless and the measurement algorithms will fail.

To ensure that the algorithms have sufficient time to execute, I have developed two ad hoc techniques for delaying premature connection termination. First, I ensure that the data sent during the data seeding phase contains a valid Hyper Text Transfer Protocol (HTTP) request [Berners-Lee et al. 96]. Some Web servers (and even some “smart” firewalls and load balancers) will reset the connection as soon as the HTTP parser fails. Second, I use TCP’s *flow control* protocol to prevent the target from actually delivering its HTTP response back to the source. TCP receivers implement flow control by advertising the number of bytes they have available for buffering new data (called the *receiver window*). A TCP sender is forbidden from sending more data than the receiver claims it can buffer. By setting the source’s receiver window to zero bytes the HTTP response is kept “trapped” at the target host until measurements have been completed. The target will not reset the connection until its response has been sent, so this technique will inter-operate with such “ill-behaved” servers.

3.4 **Implementation**

In principle, it should be straightforward to implement the loss deduction algorithms I have described. However, in most systems it is quite difficult to do so without modifying the kernel and developing a portable application-level solution is quite a challenge. The same problem is true for any user-level implementation of TCP. The principle difficulty is that most operating systems do not

provide a mechanism for redirecting packets to a user application and consequently the application is forced to coordinate its actions with the host operating system's TCP implementation. In this section I will briefly describe the implementation difficulties and explain how my current prototype functions.

3.4.1 *Building a user-level TCP*

Most operating systems provide two mechanisms for low-level network access: *raw sockets* and *packet filters*. A raw socket allows an application to directly format and send packets with few modifications by the underlying system. Using raw sockets it is possible to create custom TCP segments and send them into the network. Packet filters allow an application to acquire *copies* of raw network packets as they arrive in the system. This mechanism can be used to receive acknowledgments and other control messages from the network. Unfortunately, another copy of each packet is also relayed to the TCP stack of the host operating system; this can cause some difficulties. For example, if sting sends a TCP SYN request to the target, the target responds with a SYN/ACK packet of its own. When the host operating system receives this SYN/ACK it will respond with a RST because it is unaware that a TCP connection is in progress.

One solution to this problem would be to use a secondary IP address for the sting application, and implement a user-level proxy ARP service [Postel 84]. This would be simple and straightforward, but has the disadvantage that users of sting would need to request a second IP address from their network administrator. For this reason, I have resisted this approach.

Another solution, which I implemented in Digital Unix version 3.2, is to use the standard Unix `connect()` service to create the connection, and then hijack the session in progress using the packet filter and raw socket mechanisms. Unfortunately, this solution is not always sufficient as the host system can also become confused by acknowledgments for packets it has never sent. In the Digital Unix implementation I was forced to change one line in the kernel to control such unwanted interactions.²

The cleanest solution is to leverage the proprietary firewall interfaces provided by many host operating systems (e.g. Linux, FreeBSD, Windows 2000) to filter incoming or outgoing packets.

²I modified the ACK processing in `tcp_input.c` so the response to an acknowledgment entirely above `snd_max` is to drop the packet instead of acknowledging it.

```
# sting www.audiofind.com

Source = 128.95.2.93
Target = 207.138.37.3:80
dataSent = 100
dataReceived = 98
acksSent = 98
acksReceived = 97
Forward drop rate = 0.020000
Reverse drop rate = 0.010204
```

Figure 3.6: Sample output from the sting tool.

Blocking incoming packets can be used to prevent selected incoming TCP packets from reaching the host operating systems protocol stack. Conversely, blocking outgoing traffic can be used to suppress the responses of the host operating system. Which of these is appropriate depends on where it is implemented in the network protocol pipeline. Inbound filtering must occur after any packets are intercepted by the packet filter so it does not block probe packets, and outbound filtering must not block packets sent from a raw socket.

3.4.2 *The Sting prototype*

The current implementation of sting is based on raw sockets and packet filters running on FreeBSD 3.x and Linux 2.x. I implement the complete TCP session initiation protocol in user-level and outbound firewall filters are used to suppress any responses from the host operating system. These techniques are quite powerful and have been used since to create a variety of user-level TCP tools: including tools to test TCP congestion control behavior [Padhye et al. 01], measure bottleneck bandwidth [Saroiu et al. 01], estimate packet re-ordering [Bellardo 01] and finally, a transparent migration of the entire TCP/IP protocol stack into user space [Ely et al. 01a].

Figure 3.6 shows the output presented by sting. From the command line the user can select the

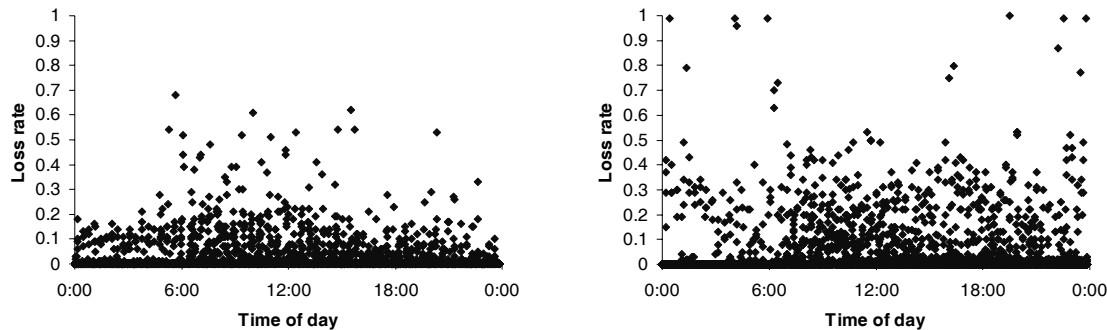


Figure 3.7: Unidirectional loss rates observed across a twenty four hour period.

inter-arrival distribution between probe packets (periodic, uniform, or exponential), the distribution mean, the number of total packets sent, as well as the target host and port. By default, sting sends 100 probe packets according to a uniform inter-arrival distribution with a mean of 100ms. My implementation verifies that the wire time distribution conforms to the expected distribution according to the Anderson-Darling tests for uniformity [Paxson et al. 98a].

I have tested this implementation in several ways. First, I have tested the tool in a purely controlled environment using an emulated network [Rizzo 97]. I have varied the forward and reverse loss rates in this network independently and verified that sting reports the correct results. Second, I have empirically compared the results of sting to results obtained from `ping` to a variety of test sites. Using the derivation for `ping`'s loss rate presented in section 3.1 I have verified that the results returned by each tool are compatible. Finally, I have tested sting with a large number of different host operating systems, including Windows 95, Windows NT, Solaris, Linux, FreeBSD, NetBSD, AIX, IRIX, Digital Unix, and MacOS. While I occasionally encounter problems with poor TCP implementations (e.g. some laser printers), Network Address Translation boxes, and Load Balancers the tool is generally quite stable.

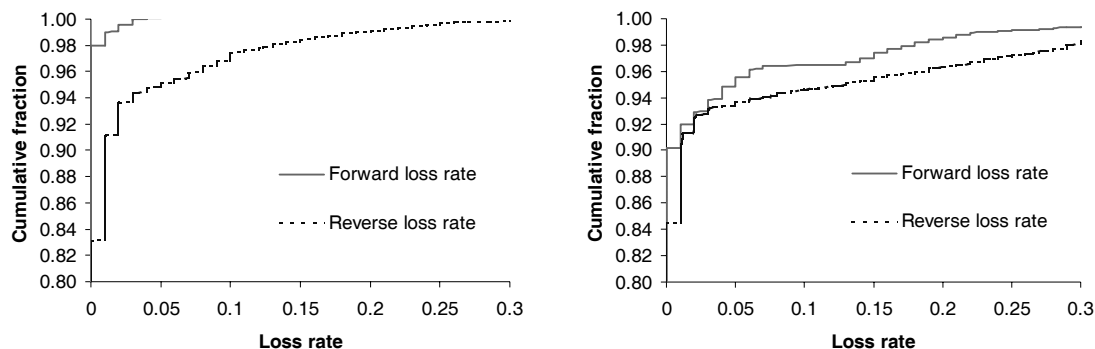


Figure 3.8: CDF of the loss rates measured over a twenty-four hour period.

3.5 Experiences

Anecdotally, user experiences with sting have been very positive. I have had considerable luck using it to debug network performance problems on asymmetric access technologies (e.g. cable modems) and I have also used it as a day-to-day diagnostic tool to understand the source of Web latency. Other groups have used it to monitor congestion on oceanic ISP links, investigate the prevalence of ICMP rate-limiting [SLAC 99], and for debugging broken CSU/DSU units on network access links. In the remainder of this section I present some preliminary results from a broad experiment to quantify the character of the loss seen from one site to the rest of the Internet.

For a twenty four hour period, I used sting to record loss rates from the University of Washington to a collection of 50 remote web servers. Choosing a reasonably-sized, yet representative, set of server sites is a difficult task due to the diversity of connectivity and load experienced at different points in the Internet. However, it is well established that the distribution of Web accesses is heavy-tailed; a small number of popular sites constitute a large fraction of overall requests, but the remainder of requests are distributed among a very large number of sites [Breslau et al. 99]. Consequently, I have constructed the target set to mirror this structural property – popular servers and random servers. Half of the 50 servers in the set are chosen from a list of the top 100 Web sites as advertised by `www.top100.com` in May of 1999. This list is generated from a collection of proxy logs and trace files. The remaining 25 servers were selected randomly using an interface

provided by Yahoo! Inc. to select pages at random from its on-line database [Yahoo! Inc].

For this experiment I used a single centralized data collection machine, a 200Mhz Pentium Pro running FreeBSD 3.1. I probed each server roughly once every 10 minutes.

Figures 3.7 shows scatter-plots showing the overall distribution of loss rates, forward and reverse respectively, during the measurement period. Each point on this scatter-plot represents a measurement to one of 50 Web servers. The plot on the left represents loss occurring on the path to the Web server, while the plot on the right represents loss occurring on the path from the Web server. Not surprisingly, overall loss rates increase during business hours and wane during off-peak hours. However, it is also quite clear that forward and reverse loss rates vary independently. Overall the average reverse loss rate (1.5%) is more than twice the forward loss rate (0.7%) and at many times of the day this ratio is significantly larger.

This reverse-dominant loss asymmetry is particularly prevalent among the popular Web servers. Figure 3.8 graphs a discrete cumulative distribution function (CDF) of the loss rates measured to and from the 25 popular servers (shown on the left) and the 25 randomly selected servers (shown on the right). In the popular server set, less than 2 percent of the measurements to these servers ever record a lost packet in the forward direction. In contrast, 5 percent of the measurements see a reverse loss rate of 5 percent or more, and almost 3 percent of measurements lose more than a tenth of these packets. On average, the reverse loss rate is more than 10 times greater than the forward loss rate in this population. One explanation for this phenomenon is that Web servers generally send much more traffic than they receive, yet bandwidth is provisioned in a full-duplex fashion. Consequently, bottlenecks are much more likely to form on paths leaving popular Web servers and packets are much more likely to be dropped in this direction.

There are similar, although somewhat different results in the random server population. Overall the loss rate is increased in both directions, but the forward loss rate has increased disproportionately. It is likely that this effect is related to the lack of dedicated network infrastructure at these sites. Many of the random servers obtain network access from third-tier ISP's that serve large user populations. Consequently, unrelated Web traffic being delivered to other ISP customers directly competes with the packets sent to these servers.

3.6 Summary

This chapter presented an approach for conducting end-to-end network measurement without requiring explicit cooperation from the hosts being probed or the underlying network. By engineering network measurement tools to leverage the protocols and services required by the remote host being measured, cooperation can be assured implicitly.

I have designed and implemented a prototype system for measuring unidirectional packet loss rates using this approach. I have developed two algorithms, *data seeding* and *hole filling* for exploiting the behavior of the Transmission Control Protocol (TCP) to achieve this end. I have also implemented several ad-hoc techniques for mitigating the limitations and inconsistencies in existing TCP implementations. Finally, I have provided initial measurements showing the presence of wide-spread loss asymmetry to popular Internet content sites.

Chapter 4

Robust Congestion Signaling

In this chapter I consider an example of competitive behavior: network clients that seek to consume more than their fair share of scarce bandwidth resources. I explore this problem in the context of delivering accurate end-to-end signaling of network congestion.

In today's Internet, voluntary end-to-end congestion control mechanisms are the primary means used to allocate scarce bandwidth resources between users. Each sending endpoint moderates its data transfer rate according to feedback from the receiver concerning bandwidth availability. This rate is decreased upon indications of network congestion, and increased in the absence of such signals. If all senders follow the same increase/decrease algorithms and all receivers generate congestion signals in the same manner then the resulting system will be both stable and "roughly" fair in its bandwidth allocation. However, while this approach is technically sound, it implicitly assumes that all users are motivated to cooperate in support of the same network-wide goals. If a user is instead motivated to increase their own bandwidth at the expense of others then this approach becomes much more fragile.

Obviously, if the sender violates the underlying congestion control regime, then it may send data more quickly than well-behaved hosts – possibly forcing competing traffic to be delayed or discarded. The potential congestion resulting from misbehaving senders is well understood and has been widely addressed in the literature. Solutions include mandatory per-flow bandwidth reservation [Zhang et al. 93], fair per-flow packet scheduling policies [Demers et al. 89, Shenker 94, Stoica et al. 99], and policing mechanisms for detecting and limiting overly aggressive flows [Floyd et al. 99a]. Unfortunately, these solutions have proven difficult to implement, scale, manage and deploy. In practice, such policing is rarely implemented, and yet major content providers rarely disavow the use of congestion control. A reasonable hypothesis for this "good behavior" is that while content providers have the capability to violate congestion control there is insufficient incentive to justify

the action. Since most popular data on the Internet is sent from a relatively small number of sites, it is likely that a content provider's primary competition for bandwidth is itself. Consequently, the aims of congestion control – sharing bandwidth fairly among multiple flows – are naturally aligned with the goals of the provider – providing good service to all of its clients.

A less obvious, and potentially more serious, vulnerability arises from misbehaving *receivers*. Curiously, the division of trust between sender and receiver has not been studied previously in the context of congestion control. While both sender and receiver must cooperate to implement the congestion control function, in many environments the interests of sender and receiver may differ considerably – creating significant incentives to violate this “good faith” doctrine. For example, in today's content retrieval applications, the primary motivation for a receiver is to minimize its own transfer latency – even at the expense of other clients. As I will show, using minor modifications to today's congestion signaling protocols, a receiver can implicitly control the data rate by manipulating the contents and timing of the feedback it provides to the sender. While the possibility of this problem has been hinted at previously [Allman et al. 99, Paxson et al. 99], the ease of exploiting this vulnerability and the potential impact have not been fully appreciated. This is especially concerning since the population of receivers is extremely large (all Internet users) and has both the incentive (faster Web surfing) and the opportunity (open source operating systems) to exploit this vulnerability.

In this chapter, I explore these issues in the context of the popular Transmission Control Protocol (TCP). I present three kinds of results:

First, I identify several real vulnerabilities that can be exploited by a malicious receiver to defeat TCP congestion control. I show that this can be done in a manner that does not break end-to-end reliability semantics and that relies only on the standard behavior of correctly implemented TCP senders. I validate these results using a modified TCP implementation, called “TCP Daytona”, and confirm that the common TCP implementations used on live Web servers possess these vulnerabilities.

Second, I show that these weaknesses are not a fundamental property of congestion signaling in general, but simply a side-effect of assuming symmetric interests between sender and receiver. I describe an alternative TCP congestion signaling mechanism that eliminates this behavior – without requiring that the receiver be trusted in any manner. By explicitly validating congestion signals at

the sender and penalizing flows with invalid signals, a receiver can only *reduce* the data transfer rate by misbehaving, thereby eliminating the incentive to do so.

Finally, I explore the practical implementation issues for deploying such a fix. While the pure form of my signaling approach provides strong controls on receiver behavior, it requires changes to the protocol software on both the sender and the receiver – a difficult requirement to meet in practice. Because this work has serious practical ramifications in an Internet that depends on trust to avoid congestion collapse, I also describe backwards-compatible mechanisms that can be implemented entirely at the sender to mitigate the effects of untrusted receivers.

The remainder of this chapter is organized as follows: In section 4.1 I review the relevant behavior of TCP’s congestion control protocol and illustrate the vulnerabilities therein. Next, in section 4.2 I describe a series of experiments that validate these vulnerabilities. Finally, in section 4.3 I explore potential solutions and alternative congestion signaling mechanisms that remove these vulnerabilities.

4.1 Vulnerabilities

By systematically considering sequences of message exchanges, I have identified several vulnerabilities that allow misbehaving receivers to control the sending rate of unmodified, conforming TCP senders. This section describes these vulnerabilities and techniques for exploiting them. In addition to denial-of-service attacks, these techniques can be used to enhance the performance of the attacker’s TCP sessions at the expense of behaving clients.

4.1.1 TCP review

While a detailed description of TCP’s error and congestion control mechanisms is beyond the scope of this chapter, I describe the rudiments of their behavior below to allow those unfamiliar with TCP to understand the vulnerabilities explained later. For simplicity, we consider TCP without the Selective Acknowledgment option (SACK) [Mathis et al. 96], although the vulnerabilities I describe also exist when SACK is used.

TCP is a connection-oriented, reliable, ordered, byte-stream protocol with explicit flow control. A sending host divides the data stream into individual segments, each of which is no longer than

the Sender Maximum Segment Size (SMSS) determined during connection establishment. Each segment is labeled with explicit sequence numbers to guarantee ordering and reliability. When a host receives an in-sequence segment it sends a cumulative acknowledgment (ACK) in return, notifying the sender that all of the data preceding that segment's sequence number has been received and can be retired from the sender's retransmission buffers. If an out-of-sequence segment is received, then the receiver acknowledges the next contiguous sequence number that was *expected*. If outstanding data is not acknowledged for a period of time, the sender will timeout and retransmit the unacknowledged segments.

TCP uses several algorithms for congestion control, most notably *slow start* and *congestion avoidance* [Jacobson et al. 88, Stevens 94, Allman et al. 99]. Each of these algorithms controls the sending rate by manipulating a congestion window (*cwnd*) that limits the number of outstanding unacknowledged bytes that are allowed at any time. When a connection starts, the slow start algorithm is used to quickly increase *cwnd* to reach the bottleneck capacity. When the sender infers that a segment has been lost it interprets this as an implicit signal of network overload and decreases *cwnd* quickly. After roughly approximating the bottleneck capacity, TCP switches to the congestion avoidance algorithm which increases the value of *cwnd* more slowly to probe for additional bandwidth that may become available.

I now describe three attacks on this congestion control procedure that exploit a sender's vulnerability to non-conforming receiver behavior.

4.1.2 ACK division

TCP uses a byte granularity error control protocol and consequently each TCP segment is described by sequence number and acknowledgment fields that refer to byte offsets within a TCP data stream. However, TCP's congestion control algorithm is implicitly defined in terms of segments rather than bytes. For example, the most recent specification of TCP's congestion control behavior, RFC 2581, states:

During slow start, TCP increments *cwnd* by at most SMSS bytes for each ACK received that acknowledges new data.

...

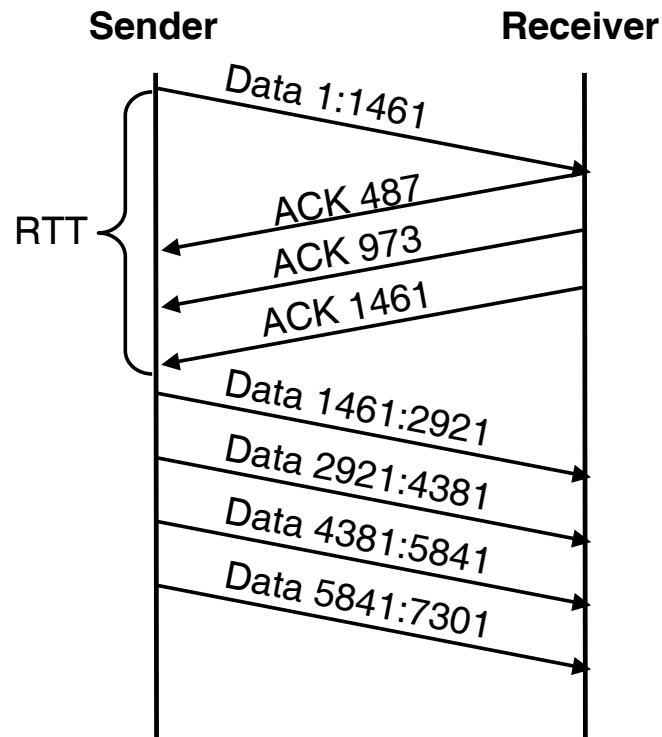


Figure 4.1: Sample time line for a ACK division attack.

During congestion avoidance, *cwnd* is incremented by 1 full-sized segment per round-trip time (RTT).

The incongruence between the byte granularity of error control and the segment granularity (or more precisely, SMSS granularity) of congestion control leads to the following vulnerability:

Attack 1:

Upon receiving a data segment containing N bytes, the receiver divides the resulting acknowledgment into M , where $M \leq N$, separate acknowledgments – each covering one of M distinct pieces of the received data segment.

This attack is demonstrated in Figure 4.1 with a time line. Here, each message exchanged between sender and receiver is shown as a labeled arrow, with time proceeding down the page. The labels indicate the type of message, data or acknowledgment, and the sequence space consumed. Since each of the M divided ACKs cover data that was sent and previously unacknowledged, they “count” as valid acknowledgments. This leads the TCP sender to grow the congestion window at a rate that is M times faster than usual. In this example, the sender begins with $cwnd=1$, which is incremented by one for each ACK received. After one round-trip time, $cwnd=4$, instead of the expected value of $cwnd=2$.

The receiver can control this rate of growth by dividing the segment at arbitrary points – up to one acknowledgment per byte received (when $M = N$). At this limit, a sender with a 1460 byte SMSS could *theoretically* be coerced into reaching a congestion window in excess of the normal TCP sequence space (4GB) in only four round-trip times! ¹ Moreover, while high rates of additional acknowledgment traffic may increase congestion on the path to the sender, the penalty to the receiver is negligible since the cumulative nature of acknowledgments inherently tolerates any losses that may occur.

4.1.3 DupACK spoofing

TCP uses two algorithms, *fast retransmit* and *fast recovery*, to mitigate the effects of packet loss. The fast retransmit algorithm detects loss by observing three duplicate acknowledgments and it immediately retransmits what appears to be the missing segment. However, the receipt of a duplicate ACK also suggests that segments are leaving the network. The fast recovery algorithm employs this information as follows (again quoted from RFC 2581):

Set $cwnd$ to $ssthresh$ plus $3 \cdot SMSS$. This artificially “inflates” the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

..

For each additional duplicate ACK received, increment $cwnd$ by $SMSS$. This artificially

¹Of course the practical transmission rate is ultimately limited by other factors such as sender buffering, receiver buffering and network bandwidth.

inflates the congestion window in order to reflect the additional segment that has left the network.

There are two problems with this approach. First, it assumes that each segment that has left the network is full sized – again an unfortunate interaction of byte granularity error control and segment granularity congestion control. Second, and more important, because TCP requires that duplicate ACKs be *exact* duplicates, there is no way to ascertain which data segment they were sent in response to. Consequently, it is impossible to differentiate a “valid” duplicate ACK, from a forged, or “spoofed”, duplicate ACK. For the same reason, the sender cannot distinguish ACKs that are accidentally duplicated by the network itself from those generated by a receiver [Allman et al. 99]. In essence, duplicate ACKs are a signal that can be used by the receiver to force the sender to transmit new segments into the network as follows:

Attack 2:

Upon receiving a data segment, the receiver sends a long stream of acknowledgments for the last sequence number received (at the start of a connection this would be for the SYN segment).

Figure 4.2 shows a time line for this technique. The receiver forges multiple duplicate ACKs for sequence number 1. The first four ACKs for the same sequence number cause the sender to retransmit the first segment. However, *cwnd* is now set to its initial value plus $3 \times \text{SMSS}$, and increased by SMSS for each additional duplicate ACK, for a total of 4 segments (as per the fast recovery algorithm). Since duplicate ACKs are indistinguishable, the receiver does not need to wait for new data to send additional acknowledgments. As a result, the sender will return data at a rate directly proportional to the rate at which the receiver sends acknowledgments. After a period, the sender will timeout. However, this can easily be avoided if the receiver acknowledges the missing segment and enters fast retransmit again for a new, later, segment.

4.1.4 Optimistic ACKing

Implicit in TCP’s algorithms is the assumption that the time between a data segment being sent and an acknowledgment for that segment returning is at least one round-trip time. Since TCP’s

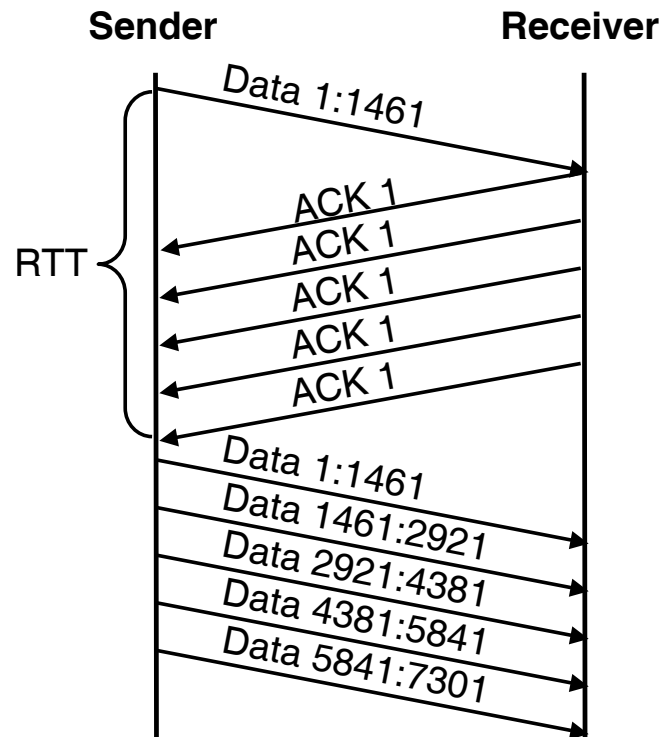


Figure 4.2: Sample time line for a DupACK spoofing attack.

congestion window growth is a function of round-trip time (an exponential function during slow start and a linear function doing congestion avoidance), sender-receiver pairs with shorter round-trip times will transfer data more quickly.

However, the protocol does not use any mechanism to enforce its assumption. Consequently, it is possible for a receiver to *emulate* a shorter round-trip time by sending ACKs optimistically for data it has not yet received:

Attack 3:

Upon receiving a data segment, the receiver sends a stream of acknowledgments anticipating data that will be sent by the sender.

This technique is demonstrated in Figure 4.3. The ACK for the second segment is sent before

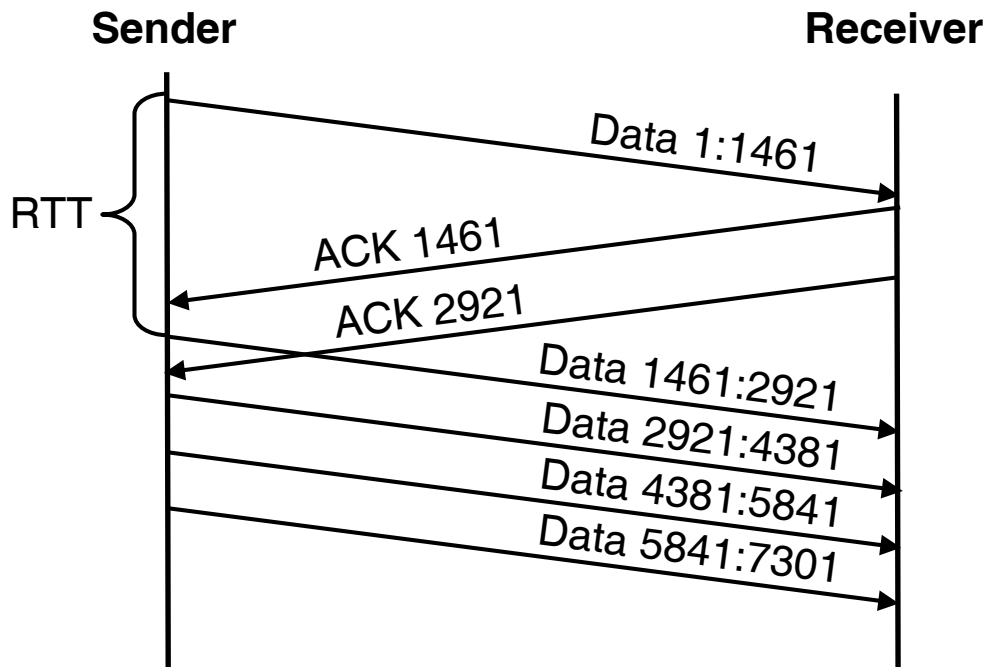


Figure 4.3: Sample time line for optimistic ACKing attack.

the segment itself is received, leading the receiver to grow *cwnd* more quickly than otherwise. At the end of this example, *cwnd*=3, rather than the expected value of *cwnd*=2. Note that while it is easy for the receiver to anticipate the correct sequence numbers to use in each acknowledgment (since senders generally send full-sized segments), this accuracy is not necessary. As long as the receiver acknowledges new data the sender will transmit additional segments. Moreover, if an ACK arrives for data that has not yet been sent, this is generally ignored by the sending TCP – allowing a sender to be arbitrarily aggressive in its generation of optimistic ACKs.

Unlike the previous attacks, this technique does not necessarily preserve end-to-end reliability semantics – if data from the sender is lost it may be unrecoverable since it has already been acknowledged. However, new features in protocols such as HTTP-1.1 allow receivers to request particular byte-ranges within a data object [Fielding et al. 99]. This suggests a strategy in which data is

gathered on one connection and lost segments are then collected selectively with application-layer retransmissions on another. Optimistic ACKing could be used to ramp the transfer rate up to the bottleneck rate immediately, and then hold it there by sending acknowledgments in spite of losses. This ability of the receiver to conceal losses is extremely dangerous because it eliminates the only congestion signal available to the sender. A malicious attacker could conceal *all* losses and therefore lead a sender to increase *cwnd* indefinitely – possibly overwhelming the network with useless packets.

4.2 Implementation experience

To exploit the vulnerabilities described above, three modifications were made to the TCP subsystem of Linux 2.2.10. This resulting TCP implementation, referred to facetiously as “TCP Daytona”, provides extremely high performance at the expense of its competitors. I demonstrate these abilities with time sequence plots of packet traces for both normal and modified receiver TCP’s. Needless to say, this implementation is intentionally not “stable”, and would likely lead to congestion collapse if it were widely deployed.

4.2.1 ACK division

The TCP Daytona ACK division algorithm adds 24 lines of code that divide each new outgoing ACK into many ACKs for smaller extents of the sequence space. Half of the new code is dedicated to ensuring that the number of outgoing ACKs is no more than should be needed to coerce a sender in slow start to saturate the test machine’s 100Mbps Ethernet interface.

Figure 4.4 shows client-side TCP sequence number plots of the test machine making an HTTP request for the `index.html` object from `cnn.com`, with and without the ACK division attack enabled. This figure spans the entire transaction, beginning with the TCP handshake that starts at 0ms and ends at around 70ms, when the HTTP request is sent. The first HTTP data from the server arrives at around 140ms.

This figure shows that, when this attack is enabled, the many small ACKs sent around 140ms convince the Web server to unleash the entire remainder of the document in a single burst; this data arrives exactly one round-trip time later. By contrast, with the normal TCP implementation, the

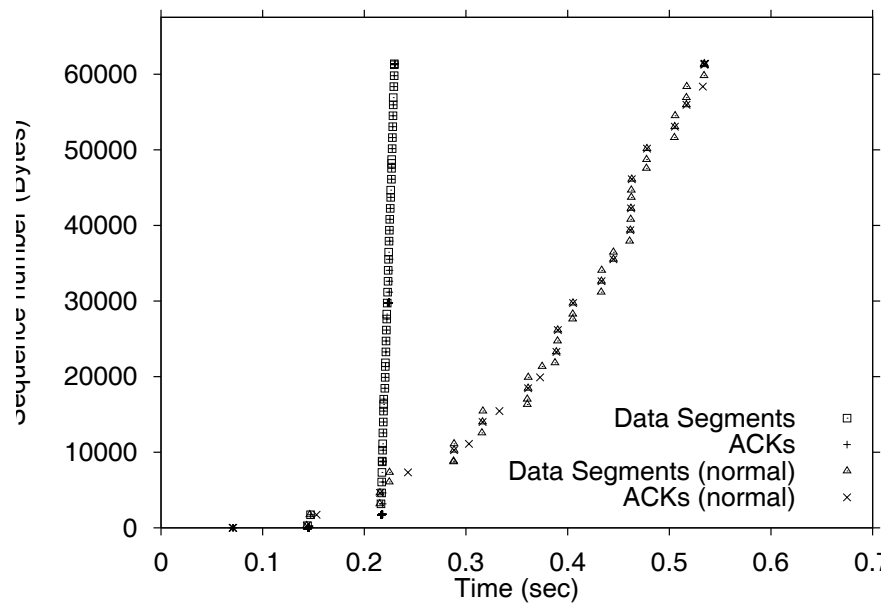


Figure 4.4: Time-sequence plot of TCP Daytona ACK division attack.

server spreads out the data over the next four round-trip times. In general, as this figure suggests, this attack can convince a TCP sender to send all of its data in a single burst.

4.2.2 DupACK spoofing

The TCP Daytona DupACK spoofing attack is implemented by 11 lines of code that cause the receiver to send sufficient duplicate ACKs such that the sender (re-)enters fast recovery and fills the receiver's advertised flow control window each round-trip time.

Figure 4.5 shows another client-side plot of the same HTTP request, this time with the DupACK spoofing attack superimposed on a normal transfer. The many duplicate ACKs that the receiver sends at around 140ms cause the sender to enter fast recovery and transmit the rest of the data, which arrives at around 210ms. Were there more data, the flurry of duplicate ACKs sent at 210ms-230ms would elicit another burst from the sender. Since there is no more new data, the sender simply fills in the hole it perceives; this segment arrives at around 290ms. This figure illustrates how the DupACK spoofing attack can achieve performance essentially equivalent to the ACK division attack – namely, both attacks can convince the sender to empty its entire send buffer in a single burst.

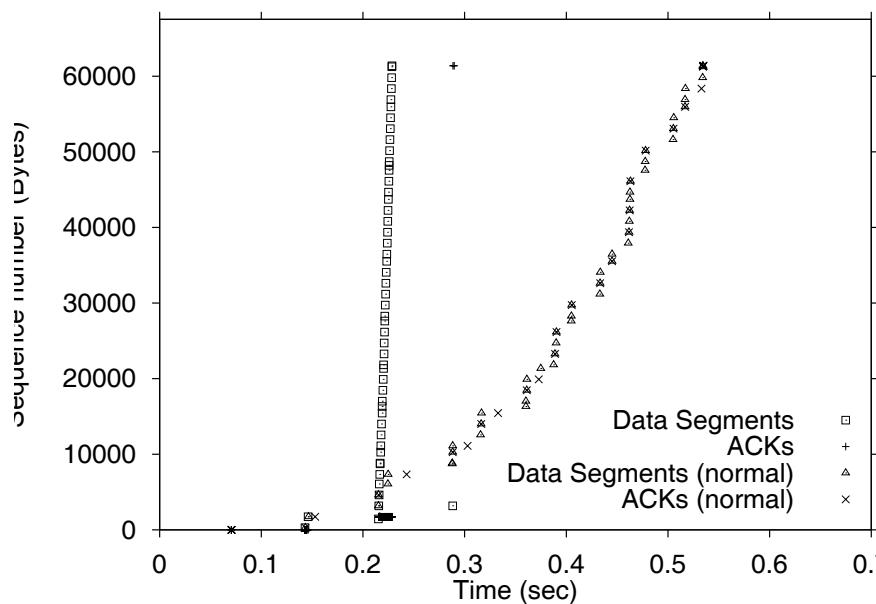


Figure 4.5: Time-sequence plot of TCP Daytona *DupACK spoofing* attack.

4.2.3 Optimistic ACKing

The TCP Daytona implementation of optimistic ACKing consists of 45 lines of code. Because acknowledging data that has not arrived is a fundamentally tricky business, I chose a very simple implementation as a proof of concept. When a TCP connection for an HTTP or FTP client receives its first data, a timer is set to expire every 10ms. Any interval would do, but I chose 10ms because it is the smallest interval that Linux 2.2.10 supports on the Intel PC platform. Whenever this periodic timer expires, or a new data segment arrives, the receiver sends a new optimistic ACK for one MSS beyond the previous optimistic ACK.

Figure 4.6 shows the optimistic ACK algorithm in action transferring the same `index.html`, again with a normal transfer superimposed. Note that after the first few data segments arrive at around 140ms, the receiver sends a steady stream of ACKs, where each ACK is sent about 10ms-70ms *before* the corresponding data arrives! The result is that the data transfer employing optimistic ACKs completes in approximately half the normal transfer time. Though this is a modest gain relative to the other attacks, a more bold optimistic ACKing scheme could achieve far greater throughput by acknowledging data at a more rapid pace.

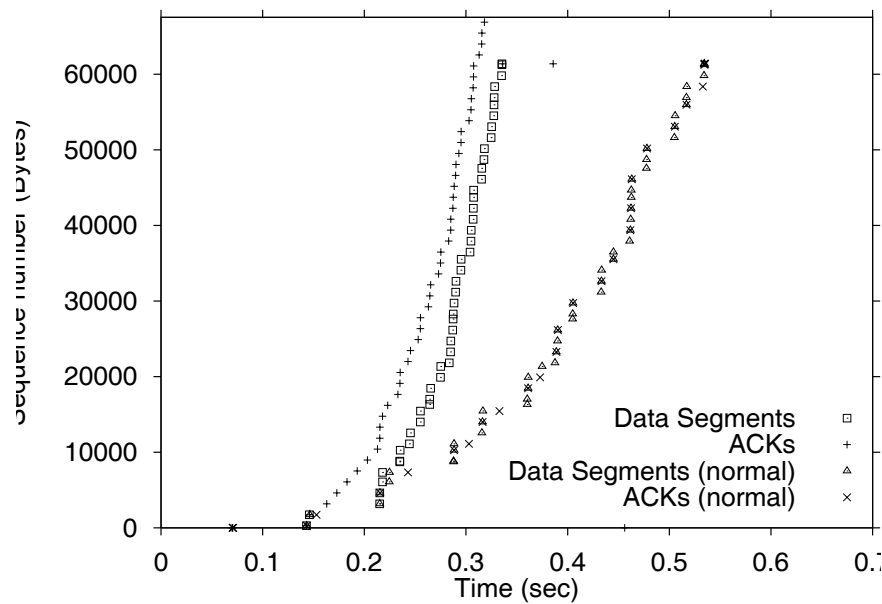


Figure 4.6: Time-sequence plot of TCP Daytona *optimistic ACK* attack.

4.2.4 Applicability

In order to verify that common TCP implementations have these vulnerabilities, each attack was tested against a set of nine Web servers running a diverse array of popular server operating systems. The operating system of the target server was identified in two ways. First, in all but three cases the server was operated by the OS vendor of the associated operating system (e.g. www.microsoft.com). Second, the *nmap* tool was used to match characteristic TCP implementation fingerprints and verify that the target server was running the expected operating system and version [Vaskovich].

Table 4.1 shows which TCP implementations are vulnerable to each attack. The attacks are all widely applicable, with three exceptions. First, Linux 2.2 is not vulnerable to the ACK division attack because it increases its congestion window only if at least one whole previously-unacknowledged segment is acknowledged. Second, Linux 2.0 refuses to count duplicate acknowledgments until *cwnd* is greater than three. Consequently, the DupACK attack will fail if initiated on connection startup. Finally, Windows NT appears to have a bug that causes it to rarely, if ever, enter fast recovery. This bug renders NT immune to attacks that rely on extra duplicate acknowledgments.

Table 4.1: Operating system vulnerabilities to TCP Daytona attacks.

	ACK Division	DupACK Spoofing	Optimistic Acks
Solaris 2.6	Y	Y	Y
Linux 2.0	Y	Y (N)	Y
Linux 2.2	N	Y	Y
Windows NT4/95	Y	N	Y
FreeBSD 3.0	Y	Y	Y
DIGITAL Unix 4.0	Y	Y	Y
IRIX 6.x	Y	Y	Y
HP-UX 10.20	Y	Y	Y
AIX 4.2	Y	Y	Y

4.3 Solutions

As demonstrated in the previous section, TCP's current specification has several vulnerabilities that allow a misbehaving receiver to control the sender's transmission rate. While it is impossible to *force* a receiver to behave correctly, it is both possible and desirable to remove its incentive to misbehave. That is, we wish to ensure that a misbehaving receiver can not obtain data faster than a behaving one. In this section I describe simple modifications to the TCP protocol that, without changing the nature of congestion control, allow the verification of what has historically been an implicit contract between sender and the receiver – that each acknowledgment faithfully and unambiguously reflects data that has been successfully transferred to the receiver.

4.3.1 Designing robust protocols

TCP's vulnerabilities arise from a combination of unstated assumptions, casual specification and a pragmatic need to develop congestion control mechanisms that are backward compatible with previous TCP implementations. In retrospect, if the contract between sender and receiver had been defined explicitly these vulnerabilities would have been obvious.

This work is inspired by Abadi and Needham’s paper, *Prudent Engineering Practice for Cryptographic Protocols*, which presents a set of design rules that are surprisingly germane to this problem [Abadi et al. 96]. In particular I reprint their first three principles below:

Principle 1. Every message should say what it means: the interpretation of the message should depend only on its content.

Principle 2. The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.

Principle 3. If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal’s name explicitly in the message.

4.3.2 ACK division

This vulnerability arises from an ambiguity about how ACKs should be interpreted – a violation of the second principle. TCP’s error-control allows an ACK to specify an arbitrary byte offset in the sequence space while the congestion control specification assumes that an ACK covers an entire segment.

There are two obvious solutions: either modify the congestion control mechanisms to operate at byte granularity or guarantee that segment-level granularity is always respected. The first solution is virtually identical to the “byte counting” modifications to TCP discussed in [Allman 98, Allman 99]. If *cwnd* is not incremented by a full SMSS, but only proportional to the amount of data acknowledged, then ACK division attacks will have no effect. The second, perhaps simpler, solution is to only increment *cwnd* by one SMSS when a valid ACK arrives that covers the entire data segment sent. As mentioned earlier, this technique is employed in the latest versions of Linux (2.2.x) at the time of this writing.

4.3.3 DupACK spoofing

During fast recovery and fast retransmit, TCP’s design violates the first principle – the meaning of a duplicate ACK is implicit, dependent on previous context, and consequently difficult to verify.

TCP assumes that all duplicate ACKs are sent in response to unique and distinct segments. This assumption is unenforceable without some mechanism for identifying the data segment that led to the generation of each duplicate ACK. The traditional method for guaranteeing association is to employ a *nonce* [Schneier 96]. We present a simple version of such a nonce protocol below (I will extend it shortly):

Singular Nonce:

I introduce two new fields into the TCP packet format: Nonce and Nonce reply. For each segment, the sender fills the Nonce field with a unique random number generated when the segment is sent. When a receiver generates an ACK in response to a data segment, it echoes the nonce value by writing it into the Nonce Reply field.

The sender can then arrange to only inflate *cwnd* in response to duplicate ACKs whose Nonce Reply value corresponds to a data segment previously sent and not yet acknowledged.

Note that the singular nonce, as described so far, is similar to the Timestamps option [Jacobson et al. 92], with two important differences. First, the Nonce field preserves association for duplicate ACKs, while the Timestamps option does not (preferring instead to reuse the previous timestamp value). Second, and more important, because Timestamps is a *option*, a receiver has the choice to not participate in its use. Misbehaving clients cannot be relied upon to voluntarily participate in their own policing. For the same reason, it is not reasonable to rely on other TCP options, such as proposed extensions to SACK [Floyd et al. 99b], to eliminate this vulnerability.

Unfortunately, the fix requires the modification of clients and servers and the addition of a TCP field. While it is the only complete solution I have discovered, there are sender-only heuristics which can mitigate, although not eliminate, the impact of the DupACK spoofing attack in a purely backward compatible manner. In particular, the sender can maintain a count of outstanding segments sent above the missing segment. For each duplicate acknowledgment this count is decremented and when it reaches zero any additional duplicate acknowledgments are ignored. This simple fix appears to limit the number of segments wrongly sent to contain no more than $cwnd - MSS$ bytes. Unfortunately, a clever receiver can acknowledge the missing segment and then repeat the process indefinitely unless other heuristics are employed to penalize this behavior (e.g. by refusing to enter fast retransmit multiple times in a single window as suggested in [Floyd 95]).

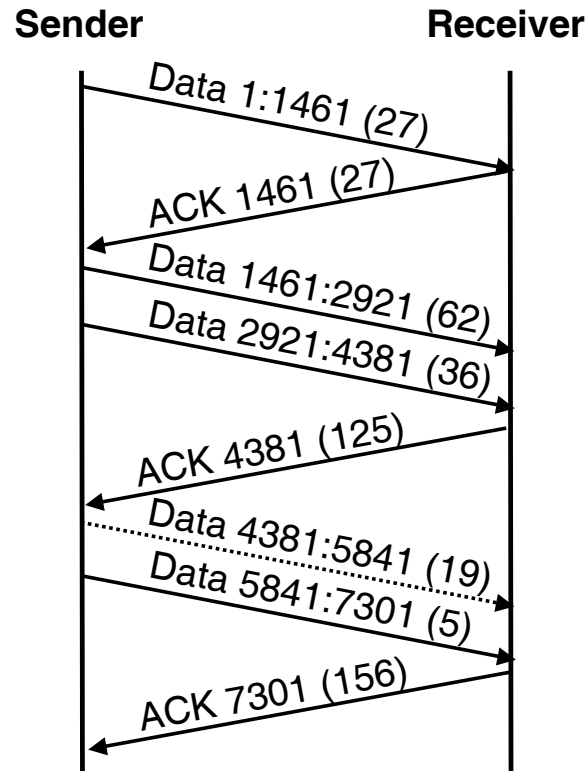


Figure 4.7: Time line for a data transfer using a cumulative nonce.

4.3.4 Optimistic ACKing

The optimistic ACK attack is possible because ACKs do not contain any proof regarding the identity of the data segment(s) that caused them to be sent. In the context of the third principle described earlier, a data segment is a principal and an ACK is the message of concern.

This problem is also well addressed using a nonce. If a nonce can't be guessed by the receiver, then ACKs with valid nonces imply that a full round-trip time has taken place (man-in-the-middle attacks notwithstanding).

However, the singular nonce I have described is imperfect because it does not mirror the cumulative nature of TCP. Acknowledgments can be delayed or lost, yet the cumulative property of TCP's sequence numbers ensures that the most recent ACK can cover all previous data. In contrast, the

singular nonce only provides evidence that a single segment was received. A misbehaving sender could still mount a denial of service attack by concealing lost data, yet still sending back ACKs with valid nonces. To address this deficiency we describe a *cumulative nonce* as follows:

Cumulative Nonce:

For each segment, the sender fills the Nonce field with a unique random number generated when the segment is sent. Each side maintains a nonce sum representing the cumulative sum of all in-sequence acknowledged nonces. When a receiver receives an in-sequence segment it adds the value contained in its Nonce field to this sum. When a receiver generates an ACK in response to a data segment, it either echoes the current value of the nonce sum (for in-sequence data) or echoes the nonce value sent by the sender (for out-of-sequence data).

The sender can then efficiently verify that the data acknowledged by the receiver has, in fact, been successfully transferred.

An example of this protocol is depicted in Figure 4.7. The nonce values are shown in parenthesis and it is assumed that each side starts with a nonce sum of zero. The dotted line indicates a data segment that was dropped. The second ACK (acknowledging bytes 4380 and below) demonstrates the cumulative effect of the nonce, proving that the receiver has in fact seen all three segments ($125=27+62+36$). The fourth data segment is lost (indicated by a dotted line) and the third ACK attempts to conceal this loss by acknowledging a later segment. However, the ACK will be rejected by the sender, since it cannot provide the proper nonce sum (149) for the data it purports to acknowledge.

A potential complication can occur if the segment boundaries differ between the initial transmission and a subsequent retransmission. Such an occurrence might occur during dynamic path MTU changes. There are several implementation strategies to address this situation, but the simplest is to randomly subdivide the original nonce value in a way that the sum of the new nonce values is still consistent with the original transmission. For example, if a 1460 byte segment is initially transmitted with a nonce value of 14, but subsequent retransmissions are limited to 536 bytes by a path MTU change, then one might retransmit the data in three packets, with nonce values of 7, 3 and 4.

While it is difficult to prevent loss concealment without a cumulative nonce, there are interim sender-side modifications that can approximate a singular nonce and thereby limit the impact of optimistic ACKing attacks. If the sending TCP randomly varies the size of outgoing segments by a small amount (e.g. [SMSS-15 bytes .. SMSS bytes]), a misbehaving receiver will be unable to correctly anticipate the segment boundaries. Consequently, the exact segment boundaries encode a form of nonce and the sending TCP can filter out optimistic ACKs as those that do not fall on the appropriate sequence numbers (this assumes that receivers acknowledge all of the data they receive). As an added disincentive, the sender could send a RST for any ACK that acknowledged data not yet sent. This strategy does not prevent the receiver from concealing loss, but it can mitigate the effects of optimistic ACKs (which is a more attractive attack for the average user).

4.4 Summary

This chapter described how a competitive receiver can manipulate the TCP congestion signaling algorithm to increase their own bandwidth at the expense of other users. In addition, I describe protocol modifications that allow the sender to detect such manipulations and enforce a fair behavior. This work highlights two results that I believe are significant yet not widely appreciated:

- TCP, which was originally designed for a cooperative environment, contains several vulnerabilities that stem from the assumption that receiver and sender have complementary interests. A greedy receiver can exploit these weaknesses to obtain improved service at the expense of other network clients. I have described ACK division, DupACK spoofing and Optimistic ACK mechanisms and implemented them to demonstrate that the attacks are both real and widely applicable.
- The design of TCP can be modified, without changing the nature of the congestion control function, to eliminate these vulnerabilities and “force” cooperative behavior from a receiver. By explicitly validating congestion signals provided by the receiver and penalizing invalid signals, the sender can eliminate the receiver’s incentive to “cheat”. I have described the workings of a new Cumulative Nonce approach that accomplishes this in a simple yet effective

manner. I have also identified and described sender-only modifications that can be deployed immediately to reduce the scope of the vulnerabilities without receiver-side modifications.

Finally, this work can readily be extended to other protocols. While the Cumulative Nonce was defined in the context of TCP, it could be adapted to any sender-based congestion control scheme. While not part of this thesis, follow-on work has extended robust congestion signaling to the network-based signaling environment [Ely et al. 01b]. This approach might also prove fruitful for unreliable transports, for example, either those that are explicitly TCP-friendly, such as RAP [Rejaie et al. 99], or other rate adaptive mechanisms, like those employed by RealAudio. A Cumulative Nonce could also be used more widely to aid in the design of other kinds of protocols. This is because it effectively defines a sequencing mechanism between untrusted parties that, because it is lightweight, idempotent and cumulative, is well suited to network environments.

Chapter 5

IP Traceback

In this chapter I consider the problem of malicious behavior. In particular, I explore the problem of tracking spoofed denial-of-service attacks to their source.

Denial-of-service attacks consume the resources of a remote host or network, thereby denying or degrading service to legitimate users. Typically, this involves sending large numbers of spurious packets to towards a victim. The victim or the intervening network is overwhelmed and legitimate clients are unable to use the victim's services. Such attacks are among the hardest security problems to address because they are simple to implement, difficult to prevent, and very difficult to trace. In the last several years, Internet denial-of-service attacks have increased in frequency, severity and sophistication. Howard reports that between the years of 1989 and 1995, the number of such attacks reported to the Computer Emergency Response Team (CERT) increased by 50 percent per year [Howard 98]. A 1999 CSI/FBI survey reports that 32 percent of respondents detected denial-of-service attacks directed against their sites [Computer Security Institute et al. 99] and more a more recent empirical study shows that roughly 4,000 such attacks occur every day [Moore et al. 01]. Finally, attackers have recently developed tools to coordinate distributed attacks from many separate sites [CERT 00a] and the highly publicized attacks of February 2000 have demonstrated the power of this approach.

Unfortunately, mechanisms for dealing with denial-of-service have not advanced at the same pace. Most work in this area has focused on *tolerating* attacks by mitigating their effects on the victim [Spatscheck et al. 99, Banga et al. 99, Karn et al. 99, Meadows 99, Cisco Systems 97]. This approach can provide an effective stop-gap measure, but does not eliminate the problem nor does it discourage attackers. The other option, and a focus of this thesis, is to trace attacks back towards their origin – ideally stopping an attacker at the source.

Tracking a denial-of-service attack in the Internet is particularly challenging due to the host-

centric nature of the Internet architecture. In the PSTN's, the network layer provides a bi-directional reliable connection service and explicitly manages the addressing and routing of each connection. At any given time, the network contains centralized state indicating the source, destination and complete network path used by each call. By contrast, the Internet provides an unreliable unidirectional datagram service with only a minimal packet forwarding mechanism at the network layer – each router is only responsible for knowing the “next hop” towards each destination. All higher-level functions, including reliability, sequencing and even host addressing, are provided by hosts working in concert. In particular, the Internet relies on each host to voluntarily specify its source address in each IP packet so the receiver can respond in return. Attackers exploit this vulnerability and routinely disguise their location using incorrect, or “spoofed”, IP source addresses. Since Internet routers do not maintain any connection state, once such a packet enters the Internet its true origin is lost and a victim is left with little useful information.

In this chapter I address the operational goal of identifying the machines that *directly* generate attack traffic and the network path this traffic subsequently follows. I call this the *traceback problem* and it is motivated by the operational need to control and contain attacks. In this setting, even incomplete or approximate information is valuable because the efficacy of measures such as packet filtering improve as they are applied further from the victim and closer to the source. I present a new approach to the traceback problem that addresses the needs of both victims and network operators. My solution is to probabilistically mark packets with partial path information as they arrive at routers. This approach exploits the observation that attacks generally comprise large numbers of packets. While each marked packet represents only a “sample” of the path it has traversed, by combining a modest number of such packets a victim can reconstruct the entire path. This allows victims to locate the approximate source of attack traffic without requiring the assistance of outside network operators. Moreover, this determination can be made even after an attack has completed. Both facets of this solution represent substantial improvements over existing capabilities for dealing with flooding-style denial-of-service attacks. Finally, the combination of random packet marking with a deterministic hop-count signaling mechanism prevents an adversary from spoofing the traceback mechanism itself and leading the victim down the incorrect path.

A key practical deployment issue with any modification of Internet routers is to ensure that the mechanisms are efficiently implementable, may be incrementally deployed, and are backwards

compatible with the existing infrastructure. I describe a traceback algorithm that adds little or no overhead to the router's critical forwarding path and may be incrementally deployed to allow traceback within the subset of routers supporting my scheme. Further, I demonstrate that the necessary path information can be encoded in a way that peacefully co-exists with existing routers, host systems and more than 99% of today's traffic.

The rest of this chapter is organized as follows: In Section 5.1, I describe related work concerning IP spoofing and solutions to the traceback problem. Section 5.2 outlines my basic approach and section 5.3 characterizes several abstract algorithms for implementing it. In Section 5.4 I detail a concrete encoding strategy for one algorithm that can be implemented within the current Internet environment. I also present experimental results demonstrating the effectiveness of this solution. In section 5.5 I discuss the main limitations and weaknesses of my proposal and potential extensions to address some of them.

5.1 Related work

It has been long understood that the IP protocol permits anonymous attacks. In his 1985 paper on TCP/IP weaknesses, Morris writes:

“The weakness in this scheme [the Internet Protocol] is that the source host itself fills in the IP source host id, and there is no provision in ... TCP/IP to discover the true origin of a packet.” [Morris 85]

In addition to denial-of-service attacks, IP spoofing can be used in conjunction with other vulnerabilities to implement anonymous one-way TCP channels and covert port scanning [Morris 85, Bellovin 89, Heberlein et al. 96, Vivo et al. 99].

There have been several efforts to reduce the anonymity afforded by IP spoofing. Table 5.1 provides a subjective characterization of each of these approaches in terms of management cost, additional network load, overhead on the router, the ability to trace multiple simultaneous attacks, the ability trace attacks after they have completed, and whether they are preventative or reactive. I also characterize my proposed traceback scheme according to the same criteria. In the remainder of this section I describe each previous approach in more detail.

Table 5.1: Qualitative comparison of existing schemes for combating anonymous attacks and the probabilistic marking approach I propose.

	Management overhead	Network overhead	Router overhead	Distributed capability	Post-mortem capability	Preventative/reactive
Ingress filtering	Moderate	Low	Moderate	N/A	N/A	Preventative
Link testing						
Input debugging	High	Low	High	Good	N/A	Reactive
Controlled flooding	Low	High	Low	Poor	N/A	Reactive
Logging	High	Low	High	Excellent	Excellent	Reactive
ICMP Traceback	Low	Low	Low	Good	Excellent	Reactive
Marking	Low	Low	Low	Good	Excellent	Reactive

5.1.1 Ingress filtering

One way to address the problem of anonymous attacks is to eliminate the ability to forge source addresses. One such approach, frequently called *ingress filtering*, is to configure routers to block packets that arrive with illegitimate source addresses [Ferguson et al. 00]. This requires a router with sufficient power to examine the source address of every packet and sufficient knowledge to distinguish between legitimate and illegitimate addresses. Consequently, ingress filtering is most feasible in customer networks or at the border of Internet Service Providers (ISP) where address ownership is relatively unambiguous and traffic load is low. As traffic is aggregated from multiple ISPs into transit networks, there is no longer enough information to unambiguously determine if a packet arriving on a particular interface has a “legal” source address. Moreover, on many deployed router architectures the overhead of ingress filter becomes prohibitive on high-speed links.

The principal problem with ingress filtering is that its effectiveness depends on widespread, if not universal, deployment. Unfortunately, a significant fraction of ISPs, perhaps the majority, do not implement this service – either because they are uninformed or have been discouraged by the administrative burden¹, potential router overhead and complications with existing services that depend on source address spoofing (e.g. some versions of Mobile IP [Perkins 96] and some hybrid

¹Some modern routers ease the administrative burden of ingress filtering by providing functionality to automatically check source addresses against the destination-based routing tables (e.g. `ip verify unicast reverse-path` on Cisco’s IOS). This approach is only valid if the route to and from the customer is symmetric – generally at the border of single-homed stub networks.

satellite communications architectures). A secondary problem is that even if ingress filtering were universally deployed at the customer-to-ISP level, attackers could still forge addresses from the hundreds or thousands of hosts within a valid customer network [CERT 00a].

It is clear that wider use of ingress filtering would dramatically improve the Internet's robustness to denial-of-service attacks. At the same time it is prudent to assume that such a system will never be fullproof – and therefore traceback technologies will continue to be important.

5.1.2 *Link testing*

Most existing traceback techniques start from the router closest to the victim and interactively test its upstream links until they determine which one is used to carry the attacker's traffic. Ideally, this procedure is repeated recursively on the upstream router until the source is reached. This technique assumes that an attack remains active until the completion of a trace and is therefore inappropriate for attacks that are detected after the fact, attacks that occur intermittently, or attacks that modulate their behavior in response to a traceback (it is prudent to assume the attacker is fully informed). Below I describe two varieties of link testing schemes, *input debugging* and *controlled flooding*.

Input debugging

Many routers include a feature called *input debugging*, that allows an operator to filter particular packets on some egress port and determine which ingress port they arrived on. This capability is used to implement a trace as follows: First, the victim must recognize that it is being attacked and develop an *attack signature* that describes a common feature contained in all the attack packets. The victim communicates this signature to a network operator, frequently via telephone, who then installs a corresponding input debugging filter on the victim's upstream egress port. This filter reveals the associated input port, and hence which upstream router originated the traffic. The process is then repeated recursively on the upstream router, until the originating site is reached or the trace leaves the ISP's border (and hence its administrative control over the routers). In the later case, the upstream ISP must be contacted and the procedure repeats itself. While such tracing is frequently performed manually, several ISPs have developed tools to automatically trace attacks across their own networks. One such system, called CenterTrack, provides an improvement over hop-by-

hop backtracking by dynamically rerouting all of the victim's traffic to flow through a centralized tracking router [Stone 00]. Once this reroute is complete, a network operator can then use input debugging at the tracking router to investigate where the attack enters the ISP network.

The most obvious problem with the input debugging approach, even with automated tools, is its considerable management overhead. Communicating and coordinating with network operators at multiple ISPs requires the time, attention and commitment of both the victim and the remote personnel – many of whom have no direct economic incentive to provide aid. If the appropriate network operators are not available, if they are unwilling to assist, or if they do not have the appropriate technical skills and capabilities, then a traceback may be slow or impossible to complete [Glave 98].

Controlled flooding

Burch and Cheswick have developed a link testing traceback technique that does not require any support from network operators [Burch et al. 00]. I call this technique *controlled flooding* because it tests links by flooding them with large bursts of traffic and observing how this perturbs traffic from the attacker. Using a pre-generated “map” of Internet topology, the victim coerces selected hosts along the upstream route into iteratively flooding each incoming link on the router closest to the victim. Since router buffers are shared, packets traveling across the loaded link – including any sent by the attacker – have an increased probability of being dropped. By observing changes in the rate of packets received from the attacker, the victim can therefore infer which link they arrived from. As with other link testing schemes, the basic procedure is then applied recursively on the next upstream router until the source is reached.

While the scheme is both ingenious and pragmatic, it has several drawbacks and limitations. Most problematic among these is that controlled flooding is itself a denial-of-service attack – exploiting vulnerabilities in unsuspecting hosts to achieve its ends. This drawback alone makes it unsuitable for routine use. Also, controlled flooding requires the victim to have a good topological map of large sections of the Internet in addition to an associated list of “willing” flooding hosts. As Burch and Cheswick note, controlled flooding is also poorly suited for tracing distributed denial-of-service attacks because the link-testing mechanism is inherently noisy and it can be difficult to discern the set of paths being exploited when multiple upstream links are contributing to the attack.

Finally, like all link-testing schemes, controlled flooding is only effective at tracing an on-going attack and cannot be used “post-mortem”.

5.1.3 *Logging*

An approach suggested in [Sager 98] and [Stone 00] is to log packets at key routers and then use data mining techniques to determine the path that the packets traversed. This scheme has the useful property that it can trace an attack long after the attack has completed. However, it also has obvious drawbacks, including potentially enormous resource requirements (possibly addressed by sampling) and a large scale inter-provider database integration problem. At this time of this writing, I was unaware of any commercial organizations using a fully operational traceback approach based on logging².

5.1.4 *ICMP Traceback*

Since this work was first published, a new traceback proposal has emerged based on the use of explicit router-generated ICMP traceback messages [Bellovin 00]. The principle idea in this scheme is for every router to sample, with low probability (e.g., 1/20,000), one of the packets it is forwarding and copy the contents into a special ICMP traceback message including information about the adjacent routers along the path to the destination. During a flooding-style attack, the victim host can then use these messages to reconstruct a path back to the attacker. This scheme has many benefits compared to previous work and is in many ways similar to the packet marking approach I have taken. However, there are several disadvantages in the current design that complicate its use. Among these: ICMP traffic is increasingly differentiated and may itself be filtered in a network under attack, the ICMP Traceback message relies on an input debugging capability (i.e. the ability to associate a packet with the input port and/or MAC address on which it arrived) that is not available in some router architectures, if only some of the routers participate it seems difficult to positively “connect” traceback messages from participating routers separated by a non-participating router, and finally, it requires a key distribution infrastructure to deal with the problem of attackers sending false ICMP Traceback messages. That said, that the scheme is clearly promising and I believe

²Historically, the T3-NFSNET *did* log network-to-network traffic statistics and these were used on at least one occasion to trace IP spoofing attacks to an upstream provider [Villamizar 00].

that hybrid approaches combining it with some of the algorithms I propose are likely to be quite effective.

5.2 Overview

Burch and Cheswick mention the possibility of tracing flooding attacks by “marking” packets, either probabilistically or deterministically, with the addresses of the routers they traverse [Burch et al. 00]. The victim uses the information in the marked packets to trace an attack back to its source. This approach has not been previously explored in any depth, but has many potential advantages. It does not require interactive cooperation with ISPs and therefore avoids the high management overhead of input debugging. Unlike controlled flooding, it does not require significant additional network traffic and can potentially be used to track multiple attacks. Moreover, like logging, packet marking can be used to trace attacks “post-mortem” – long after the attack has stopped. Finally, I have found that marking algorithms can be implemented without incurring any significant overhead on network routers. The remainder of this chapter focuses on fully exploring and characterizing this approach.

5.2.1 Definitions

Figure 5.1 depicts the network as seen from a victim V . Routers are represented by R_i , and potential attackers by A_i . For the purposes of this chapter, V may be a single host under attack, or a network border device such as a firewall or intrusion detection system that represents many such hosts. Every potential *attack origin* A_i is a leaf in a tree rooted at V and every router R_i is an internal node along a path between some A_i and V . The *attack path* from A_i is the unique ordered list of routers between A_i and V . For instance, if an attack originates from A_2 then to reach V it must first traverse the path R_6, R_3, R_2 , and R_1 – as shown by the dotted line in Figure 5.1.

The *exact traceback* problem is to determine the attack path and the associated attack origin for each attacker. However, solving this problem is complicated by several practical limitations. The exact attack origin may never be revealed (even MAC source addresses may be spoofed) and a wily attacker may send false signals to “invent” additional routers in the traceback path. I address these issues in section 5.5, but for now I restrict my discussion to solving a more limited problem. I define the *approximate traceback* problem as finding a candidate attack path for each attacker that contains

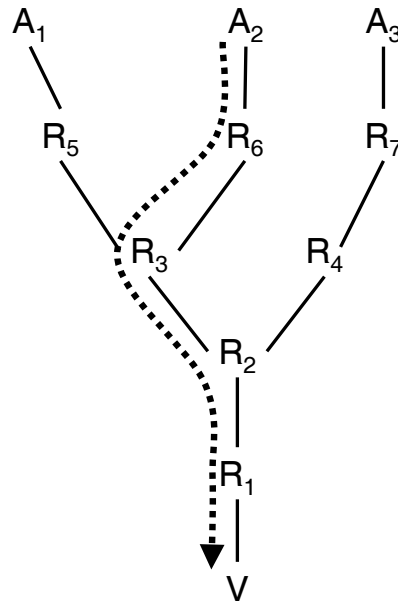


Figure 5.1: Network as seen from a victim, V , of a denial-of-service attack.

the true attack path as a suffix. I call this the *valid suffix* of the candidate path. For example, $(R_5, R_6, R_3, R_2, R_1)$ is a valid approximate solution to Figure 5.1 because it contains the true attack path as a suffix. I call a solution to this problem is *robust* if an attacker cannot prevent the victim from discovering candidate paths containing the valid suffix.

All marking algorithms have two components: a *marking procedure* executed by routers in the network and a *path reconstruction procedure* implemented by the victim. A router “marks” one or more packets by augmenting them with additional information about the path they are traveling. The victim attempts to reconstruct the attack path using only the information in these marked packets. The *convergence time* of an algorithm is the number of packets that the victim must observe to reconstruct the attack path.

5.2.2 *Basic assumptions*

The design space of possible marking algorithms is large, and to place this work in context I identify the assumptions that motivate and constrain my design:

- an attacker may generate any packet,
- multiple attackers may conspire,
- attackers may be aware they are being traced,
- packets may be lost or reordered,
- attackers send numerous packets,
- the route between attacker and victim is fairly stable,
- routers are both CPU and memory limited, and
- routers are not widely compromised.

The first four assumptions represent conservative assessments of the abilities of the modern attackers and limitations of the network. Designing a traceback system for the Internet environment is extremely challenging because there is very little that can be trusted. In particular, the attacker's ability to create arbitrary packets significantly constrains potential solutions. When a router receives a packet, it has no way to tell whether that packet has been marked by an upstream router or if the attacker simply has forged this information. In fact, the only invariant that one can depend on is that a packet from the attacker must traverse all of the routers between it and the victim.

The remaining assumptions reflect the basis for my design and deserve additional discussion. First, denial-of-service attacks are only effective so long as they occupy the resources of the victim. Consequently, most attacks are comprised of thousands or millions of packets. My approach relies on this property because each packet is marked with only a small piece of path state and the victim must observe many such packets to reconstruct the complete path back to the attacker. If many

attacks emerge that require only a single packet to disable a host (e.g. ping-of-death [CERT 96]), then this assumption may not hold (although even these attacks require multiple packets to *keep* a machine down).

Second, measurement evidence suggests that while Internet routes do change, it is extremely rare for packets to follow many different paths over the short time-scales of a traceback operation (seconds in my system) [Paxson 97b]. This assumption greatly simplifies the role of the victim, since it can therefore limit its consideration to a single primary path for each attacker. If the Internet evolves to allow significant degrees of multi-path routing then this assumption may not hold.

Third, while there have been considerable improvements in router implementation technology, link speeds have also increased dramatically. Consequently, I assert that any viable implementation must have low per-packet overhead and must not require per-flow state. Significantly simpler schemes than mine can be implemented if one assumes that routers are not resource constrained.

Finally, since a compromised router can effectively eliminate any information provided by upstream routers, it is effectively indistinguishable from an attacker. In such circumstances, the security violation at the router must be addressed first, before any further traceback is attempted. In normal circumstances, I believe this is an acceptable design point. However, if non-malicious, but information hiding, routing infrastructures become popular, such as described in [Goldberg et al. 99, Reed et al. 98], then this issue may need to be revisited.

5.3 Basic marking algorithms

In this section I describe a series of marking algorithms – starting from the most simple and advancing in complexity. Each algorithm attempts to solve the approximate traceback problem in a manner consistent with the assumptions.

5.3.1 Node append

The simplest marking algorithm – conceptually similar to the IP Record Route option [Postel 81b] – is to append each node's address to the end of the packet as it travels through the network from attacker to victim (see Figure 5.2). Consequently, every packet received by the victim arrives with a complete ordered list of the routers it traversed – a built-in attack path.

The node append algorithm is both robust and extremely quick to converge (a single packet), however it has several serious limitations. Principal among these is the infeasibly high router overhead incurred by appending data to packets in flight. Moreover, since the length of the path is not known *a priori*, it is impossible to ensure that there is sufficient unused space in the packet for the complete list. This can lead to unnecessary fragmentation and bad interactions with services such as MTU discovery [Mogul et al. 90]. This problem cannot be solved by reserving “enough” space, as the attacker can completely fill any such space with false, or misleading, path information.

5.3.2 Node sampling

To reduce both the router overhead and the per-packet space requirement, one can sample the path one node at a time instead of recording the entire path. A single static “node” field is reserved in the packet header – large enough to hold a single router address (i.e. 32 bits for IPv4). Upon receiving a packet, each router chooses to write its address in the node field with some probability p . After enough packets have been sent, the victim will have received at least one sample for every router in the attack path. As stated in section 5.2, I assume that the attacker sends enough packets and the route is stable enough that this sampling can converge.

Although it might seem impossible to reconstruct an ordered path given only an unordered collection of node samples, it turns out that with a sufficient number of trials, the order can be deduced from the relative number of samples per node. Since routers are arranged serially, the probability that a packet will be marked by a router and then left unmolested by all downstream routers is a strictly decreasing function of the distance to the victim. If p is constrained to be identical at each router, then the probability of receiving a marked packet from a router d hops away is $p(1 - p)^{d-1}$. Since this function is monotonic in the distance from the victim, ranking each router by the number of samples it contributes will tend to produce the accurate attack path. The full algorithm is shown in Figure 5.3.

Putting aside for the moment the difficulty in changing the IP header to add a 32-bit node field, this algorithm is efficient to implement because it only requires the addition of a write and checksum update to the forwarding path. Current high-speed routers already must perform these operations efficiently to update the *time-to-live* field on each hop. Moreover, if $p > 0.5$ then this algorithm is

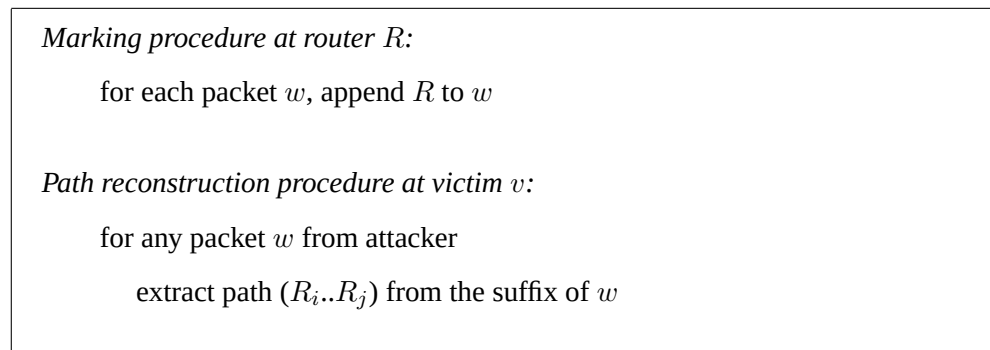


Figure 5.2: Node append algorithm.

robust against a single attacker because there is no way for an attacker to insert a “false” router into the path’s valid suffix by contributing more samples than a downstream router, nor to reorder valid routers in the path by contributing more samples than the difference between any two downstream routers.

However, there are also two serious limitations. First, inferring the total router order from the distribution of samples is a slow process. Routers far away from the victim contribute relatively few samples (especially since p must be large) and random variability can easily lead to misordering unless a very large number of samples are observed. For instance, if $d = 15$ and $p = 0.51$, the receiver must receive more than 42,000 packets on average before it receives a *single* sample from the furthest router. To guarantee that the order is correct with 95% certainty requires more than seven times that number.

Second, if there are multiple attackers then multiple routers may exist at the same distance – and hence be sampled with the sample probability. Therefore, this technique is not robust against multiple attackers.

5.3.3 Edge sampling

A straightforward solution to these problems is to explicitly encode *edges* in the attack path rather than simply individual nodes. To do this, two static address-sized fields, *start* and *end*, need to be reserved in each packet to represent the routers at each end of a link, as well as an additional small field to represent the distance of an edge sample from the victim.

Marking procedure at router R :

```

for each packet  $w$ 
  let  $x$  be a random number from  $[0..1)$ 
  if  $x < p$  then,
    write  $R$  into  $w.node$ 

```

Path reconstruction procedure at victim v :

```

let  $NodeTbl$  be a table of tuples (node,count)
for each packet  $w$  from attacker
   $z := \text{lookup } w.node \text{ in } NodeTbl$ 
  if  $z \neq \text{NIL}$  then
    increment  $z.count$ 
  else
    insert tuple ( $w.node,1$ ) in  $NodeTbl$ 
sort  $NodeTbl$  by count
extract path ( $R_i..R_j$ ) from ordered node fields in  $NodeTbl$ 

```

Figure 5.3: Node sampling algorithm.

When a router decides to mark a packet, it writes its own address into the start field and writes a zero into the distance field. Otherwise, if the distance field is already zero this indicates that the packet was marked by the previous router. In this case, the router writes its own address into the end field – thereby representing the edge between itself and the previous router – and increments the distance field to one. Finally, if the router doesn’t mark the packet then it always increments the distance field. This somewhat baroque signaling mechanism allows edge sampling to be incrementally deployed – edges are constructed only between participating routers.

The mandatory increment is *critical* to minimize spoofing by an attacker. When the packet arrives at the victim its distance field represents the number of hops traversed since the edge it contains was sampled.³ Any packets written by the attacker will necessarily have a distance greater

³It is important that distance field is updated using a saturating addition. If the distance field were allowed to wrap, then the attacker could spoof edges close to the victim by sending packets with a distance value close to the maximum.

Marking procedure at router R :

```

for each packet  $w$ 
  let  $x$  be a random number from  $[0..1)$ 
  if  $x < p$  then
    write  $R$  into  $w.start$  and 0 into  $w.distance$ 
  else
    if  $w.distance = 0$  then
      write  $R$  into  $w.end$ 
    increment  $w.distance$ 

```

Path reconstruction procedure at victim v :

```

let  $G$  be a tree with root  $v$ 
let edges in  $G$  be tuples (start,end,distance)
for each packet  $w$  from attacker
  if  $w.distance = 0$  then
    insert edge ( $w.start, v, 0$ ) into  $G$ 
  else
    insert edge ( $w.start, w.end, w.distance$ ) into  $G$ 
remove any edge ( $x, y, d$ ) with  $d \neq \text{distance from } x \text{ to } v \text{ in } G$ 
extract path ( $R_i..R_j$ ) by enumerating acyclic paths in  $G$ 

```

Figure 5.4: Edge sampling algorithm.

or equal to the length of the true attack path. Therefore, a single attacker is unable to forge any edge between themselves and the victim (for a distributed attack, of course, this applies only to the closest attacker) and the victim does not have to worry about “chaff” while reconstructing the valid suffix of the attack path. Consequently, since this algorithm abandons the sampling rank approach to distinguish “false” samples, arbitrary values can be used for the marking probability p .

The victim uses the edges sampled in these packets to create a graph (much as in Figure 5.1) leading back to the source, or sources, of attack. The full algorithm is described in Figure 5.4. Because the probability of receiving a sample is geometrically smaller the further away it is from

the victim, the time for this algorithm to converge is dominated by the time to receive a sample from the furthest router, $\frac{1}{p(1-p)^{d-1}}$ in expectation, for a router d hops away. However, there is also a small probability that a sample will be received from the furthest router, but not from some nearer router. This effect can be bounded to a factor of $\ln(d)$ by the following argument: I conservatively assume that samples from all of the d routers appear with the same likelihood as the furthest router. Since these probabilities are disjoint, the probability that a given packet will deliver a sample from some router is at least $dp(1-p)^{d-1}$. Finally, as per the well-known *coupon collector* problem, the expected number of trials required to select one of each of d equi-probable items is $d(\ln(d) + O(1))$ ⁴ [Feller 66]. Therefore, the number of packets, X , required for the victim to reconstruct a path of length d has the following bounded expectation:

$$E(X) < \frac{\ln(d)}{p(1-p)^{d-1}}$$

For example, if $p = \frac{1}{10}$, and the attack path has a length of 10, then a victim can typically reconstruct this path after receiving 75 packets from the attacker. While this choice of $p = \frac{1}{d}$ is optimal, the convergence time is not overly sensitive to this parameter for the path lengths that occur in the Internet. So long as $p \leq \frac{1}{d}$, the results are generally within a small constant of optimal. In the rest of this chapter I will use $p = \frac{1}{25}$ since few paths exceed this length [Carter et al. 97, Theilmann et al. 00, CAIDA 00]. For comparison, the previous example converges with only 108 packets using $p = \frac{1}{25}$.

This same algorithm can efficiently discern multiple attacks because attackers from different sources produce disjoint edges in the tree structure used during reconstruction. The number of packets needed to reconstruct each path is independent, so the number of packets needed to reconstruct all paths is a linear function of the number of attackers. Finally, edge sampling is also robust. It is impossible for any edge closer than the closest attacker to be spoofed, due to the mandatory increment operation used to establish the distance that marked packet has traveled. Conversely, in a distributed attack this also means that it is impossible to trust the contents of any edge *further* away than the closest attacker. As with the ICMP Traceback approach [Bellovin 00], an additional mechanism incorporating a shared secret is required to completely address the problem of attackers

⁴More exactly, the expression is $d(\ln(d) + \gamma)$, where γ represents Euler's constant. For simplicity, I ignore this small constant when describing the expectation, although I include its effect during calculations.

spoofing edges.

Of course, a significant practical limitation of this approach is that it requires additional space in the IP packet header and therefore is not backwards compatible. In the next section I discuss a modified version of edge-sampling that addresses this problem, albeit at some cost in performance and a reduction in robustness during large distributed attacks.

5.4 Encoding issues

The edge sampling algorithm requires 72 bits of space in every IP packet (two 32-bit IP addresses and 8 bits for distance to represent the theoretical maximum number of hops allowed using IP). It would be possible to directly encode these values into an MPLS label stack [Rosen et al. 00], to enable traceback within a single homogeneous ISP network. However, my focus is on a heterogeneous environment based purely on IP datagrams. One obvious approach is to store the edge sample data in an IP option, but this is a poor choice for many of the same reasons that the node append algorithm is infeasible – appending additional data to a packet in flight is expensive and there may not be sufficient space to append this data. This data could also be sent out-of-band – in a separate packet – but this would add both router and network overhead plus the complexity of a new and incompatible protocol.

Instead, I have developed a modified version of edge sampling that dramatically reduces the space requirement in return for a modest increase in convergence time and a reduction in robustness to multiple attackers. Following an analysis of the algorithm I explore the practical implementation issues and discuss one concrete encoding of this scheme based on overloading the 16-bit IP *identification* field used for fragmentation. Any solution involving such overloading necessarily requires compromises and I stress that my solution reflects only one design point among many potential implementation tradeoffs for this class of algorithm and *does not* necessarily reflect an optimal balance among them.

5.4.1 Compressed edge fragment sampling

I use three techniques to reduce per-packet storage requirements while preserving robustness. First, I encode each edge in half the space by representing it as the *exclusive-or* (XOR) of the two IP

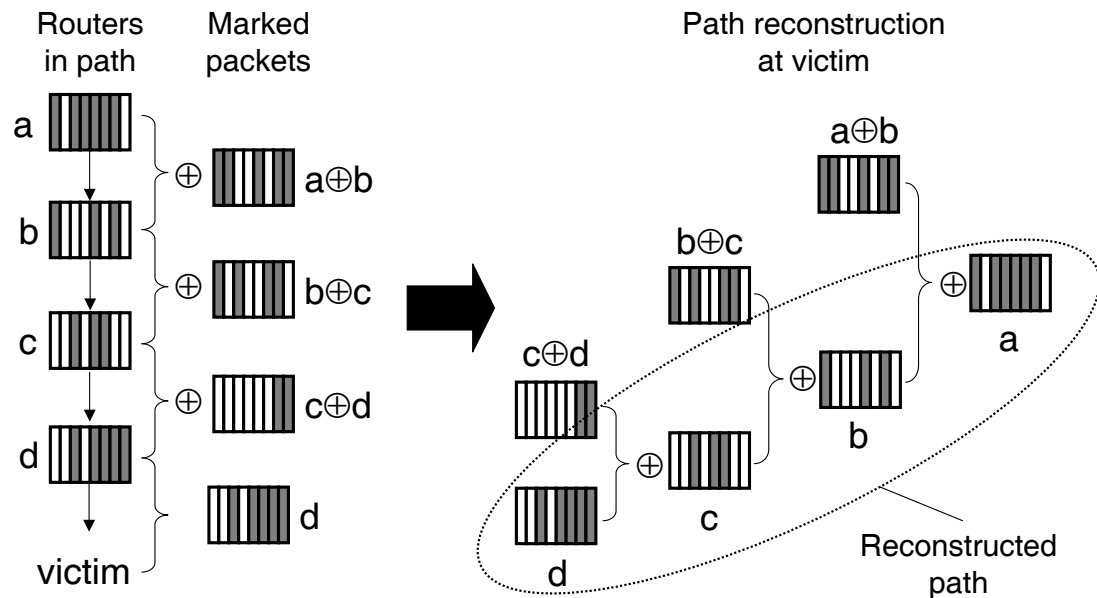


Figure 5.5: Compressing edge data using transitive XOR operations.

addresses making up the edge, as depicted in Figure 5.5.

When some router decides to mark a packet it writes its address, *a*, into the packet. The following router, *b*, notices that the distance field is 0 and (assuming it does not mark the packet itself) reads *a* from the packet, XORs this value with its own address and writes the resulting value, $a \oplus b$, into the packet. I call the resulting value the *edge-id* for the edge between *a* and *b*. The edge-ids in the packets received by the victim always contain the XOR of two adjacent routers, except for samples from routers one hop away from the victim, which arrive unmodified. Over time the victim receives the messages *d*, $c \oplus d$, $b \oplus c$, and $a \oplus b$. Since $x \oplus y \oplus x = y$, the message *d* from the final router can be used to decode the previous edge id, and so on, hop-by-hop until the first router is reached.

My second modification further reduces the per-packet space requirements by subdividing each edge-id into some number, *k*, of smaller non-overlapping fragments. When a router decides to mark a packet, it selects one of these fragments at random and stores it in the packet. I use a

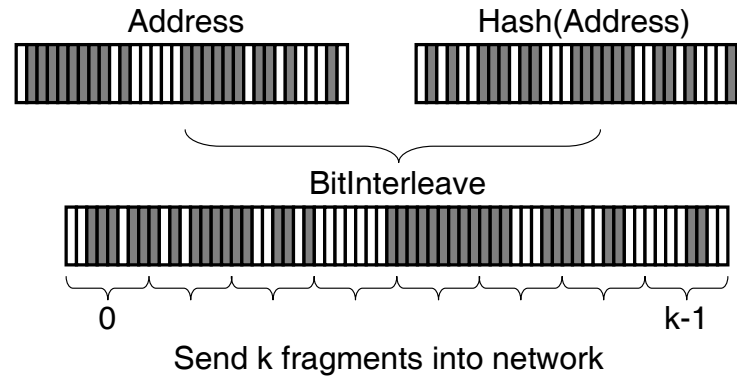


Figure 5.6: Fragment interleaving for compressed edge-ids.

few additional bits ($\log_2 k$) to store the offset of this fragment within the original address – this is necessary to ensure that different fragments from an edge-id can be recombined in the correct order. If enough packets are sent by the attacker, the victim will eventually receive all fragments from all edge-ids.

Finally, unlike full IP addresses, edge-id fragments are not unique and multiple fragments from different edge-ids may have the same value. If there are multiple attackers, a victim may receive multiple edge fragments with the same offset and distance. To reduce the probability that a “false” edge-id is accidentally reconstructed by combining fragments from different paths, I add a simple error detection code to the algorithm. Each router calculates a uniform hash of its IP address once, at startup, using a well-known function. As shown in Figure 5.6 this hash is interleaved with the original IP address (the original address on odd bits, the hash on even bits) thereby *increasing* the size of each router address, and hence each edge-id. The resulting quantity is divided into k fragments. When the router marks a packet it picks a random offset value between 1 and k , and writes both the offset and the associated fragment into the marked packet. Downstream routers use this offset to select the appropriate fragment to XOR – thereby encoding part of an edge. Finally, the victim can construct *candidate edge-ids* by combining all combinations of fragments at each distance with disjoint offset values.

When reconstructing a candidate edge, as shown in Figure 5.7, the victim combines k fragments

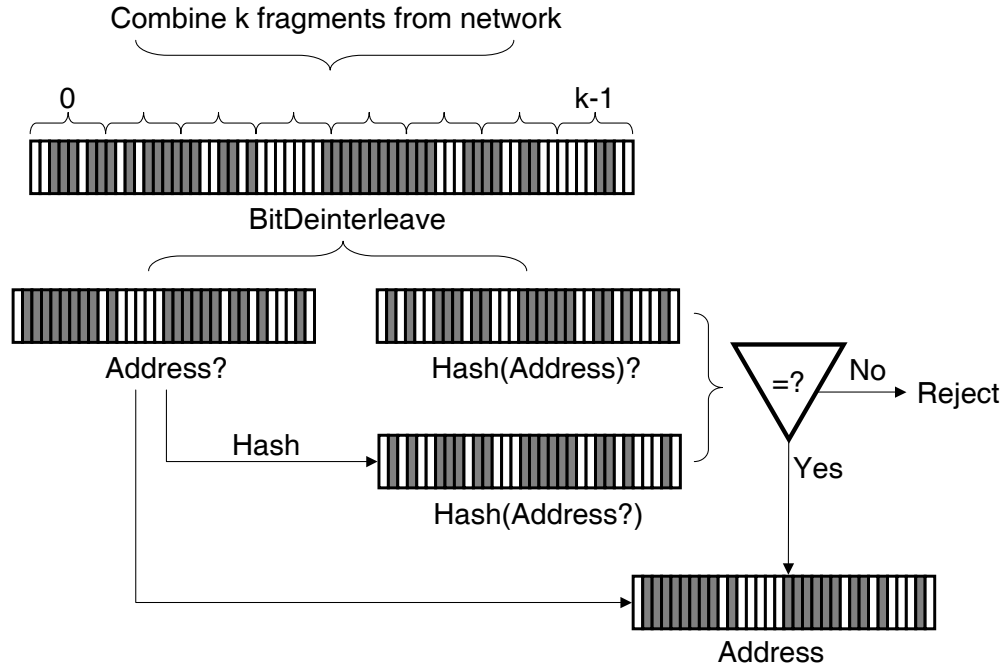


Figure 5.7: Reconstructing edge-id's from fragments.

to produce a bit string. By de-interleaving this string, the address portion and the hash portion are extracted. I recalculate the hash over this address portion using the same hash function used by the router. If the resulting hash is the same as the hash portion extracted, then the address is accepted as valid. This procedure protects against accidentally combining fragments of different edges. As the size of the hash is increased, the probability of a collision is reduced. I describe the full procedure in Figure 5.8.

The expected number of packets for this algorithm to converge is similar to the edge sampling approach, except now k fragments are needed for each edge-id, rather than just one, a total of kd fragments. If we again assume conservatively that each of these fragments is delivered equi-probably with probability $p(1 - p)^{d-1}$, the expected number of packets required for path reconstruction is bounded by:

$$E(X) < \frac{k \cdot \ln(kd)}{p(1 - p)^{d-1}}$$

Marking procedure at router R :

```

let  $R' = \text{BitIntereave}(R, \text{Hash}(R))$ 
let  $k$  be the number of non-overlapping fragments in  $R'$ 
for each packet  $w$ 
  let  $x$  be a random number from  $[0..1)$ 
  if  $x < p$  then
    let  $o$  be a random integer from  $[0..k - 1]$ 
    let  $f$  be the fragment of  $R'$  at offset  $o$ 
    write  $f$  into  $w.\text{frag}$ 
    write 0 into  $w.\text{distance}$ 
    write  $o$  into  $w.\text{offset}$ 
  else
    if  $w.\text{distance} = 0$  then
      let  $f$  be the fragment of  $R'$  at offset  $w.\text{offset}$ 
      write  $f \oplus w.\text{frag}$  into  $w.\text{frag}$ 
    increment  $w.\text{distance}$ 

```

Path reconstruction procedure at victim v :

```

let  $\text{FragTbl}$  be a table of tuples (frag,offset,distance)
let  $G$  be a tree with root  $v$ 
let edges in  $G$  be tuples (start,end,distance)
let  $\text{maxd} := \text{last} := 0$ 
for each packet  $w$  from attacker
   $\text{FragTbl.Insert}(w.\text{frag}, w.\text{offset}, w.\text{distance})$ 
  if  $w.\text{distance} > \text{maxd}$  then
     $\text{maxd} := w.\text{distance}$ 
for  $d := 0$  to  $\text{maxd}$ 
  for all ordered combinations of fragments at distance  $d$ 
    construct edge  $z$ 
    if  $d \neq 0$  then
       $z := z \oplus \text{last}$ 
    if  $\text{Hash}(\text{EvenBits}(z)) = \text{OddBits}(z)$  then
      insert edge  $(z, \text{EvenBits}(z), d)$  into  $G$ 
       $\text{last} := \text{EvenBits}(z)$ 
remove any edge  $(x, y, d)$  with  $d \neq \text{distance from } x \text{ to } v \text{ in } G$ 
extract path  $(R_i..R_j)$  by enumerating acyclic paths in  $G$ 

```

Figure 5.8: Compressed edge fragment sampling algorithm.

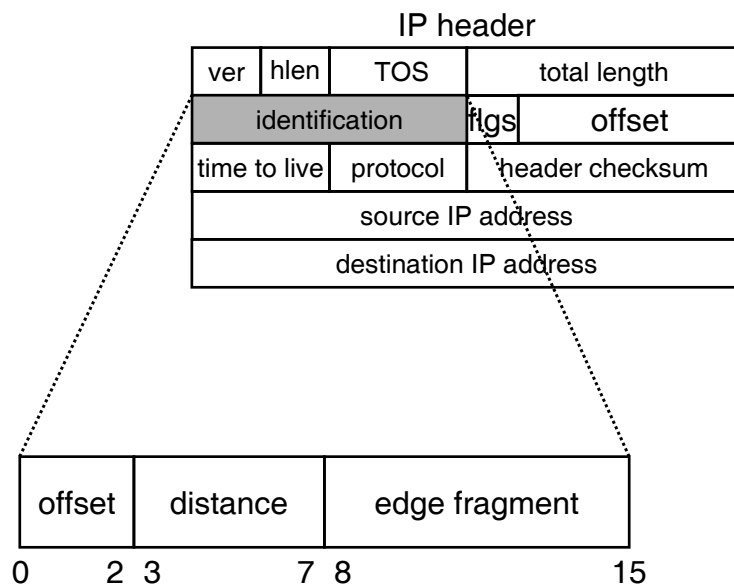


Figure 5.9: Encoding edge fragments into the IP identification field.

For example, if there are 8 fragments per edge-id, an attacker is 10 hops away, and $p = \frac{1}{25}$, then a victim can reconstruct the full path after receiving slightly less than 1,300 packets on average. Using techniques similar to those used to show sharp concentration results for the coupon collectors problem, it can be shown that the approximate the number of packets required to ensure that a path can be reconstructed with probability $1 - \frac{1}{c}$ is:

$$\frac{k \cdot \ln(kdc)}{p(1-p)^{d-1}}$$

packets. To completely reconstruct the previous path with 95% certainty should require no more than 2150 packets. Many denial-of-service attacks send this many packets in a few seconds.

Finally, I explore the robustness of this algorithm with respect to multiple attackers. For a random hash of length h , the probability of accepting an arbitrarily constructed candidate edge-id is $\frac{1}{2^h}$. In the event that there are m attackers, then at any particular distance d , in the worst case there may be up to m distinct routers.⁵ Consequently the probability that any edge-id at distance d

⁵In practice, the number of distinct routers is likely to be smaller for the portion of the path closest to the receiver, since many attackers will still share significant portions of their attack path with one another.

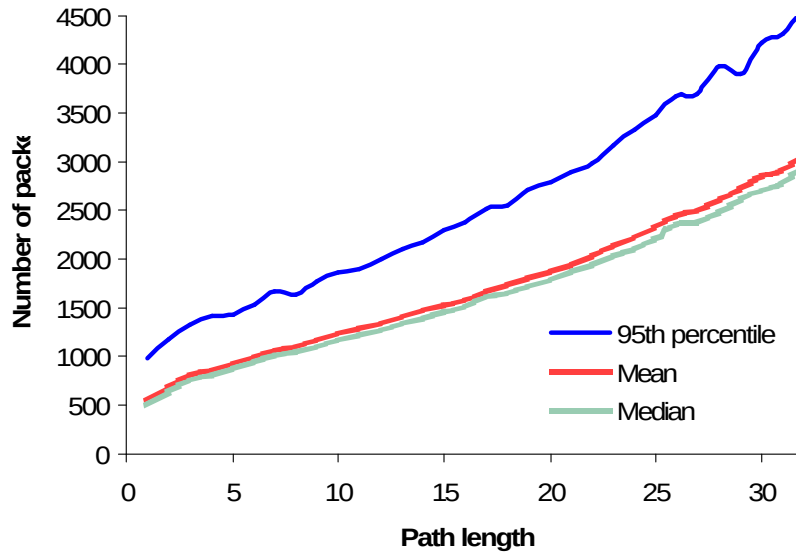


Figure 5.10: Experimental results for number of packets needed to reconstruct paths of varying lengths.

is accepted incorrectly is at most:

$$1 - \left(1 - \frac{1}{2^h}\right)^{m^k}$$

since there are m^k possible combinations of fragments in the worst case. For $h = 32$ and $k = 4$ this means that 100 distinct routers at the same distance (i.e. disjoint attack paths) will be resolved with no errors with a probability of better than 97%. For $h = 32$ and $k = 8$ (the values I use in my implementation), the same certainty can only be provided for 10 distinct routers at the same distance. The use of the XOR function further complicates reconstruction since all combinations of XOR values must be tried as attack paths diverge. This is somewhat mitigated as the probability of propagating an error from a single edge all the way to the attacker is extremely small because the resulting edge-id, when XORed with the previous edge-id, must again produce a correct hash.

The most significant drawback to this scheme is the large number of combinations that must be considered as the multiple attack paths diverge. While these combinations can be computed off-line, for large values of k and m even this can become intractable. For example, even with $k = 8$ and $m = 10$, if the separate attack paths diverge such that there are 10 completely independent edges per attacker, this will require roughly a billion combinations to be considered. Consequently, there

is a design tension in the size of k – per-packet space overhead is reduced by a larger k , while computational overhead and robustness benefits from a smaller k .

5.4.2 IP header encoding

To allow for practical deployment requires the “overloading” of existing header fields in a manner that will have minimal impact on existing users. This is a difficult task, especially given that even after prodigious effort 16 bits of space are still required. Nonetheless, it is possible to obtain this space by overloading the 16-bit IP identification field. This field is currently used to differentiate IP fragments that belong to different packets. I describe the proposed encoding below, and then discuss the issues of backwards-compatibility that it raises. However, because the issue of backwards-compatible encoding is largely separate from the choice of traceback algorithm, one could adopt any reasonable encoding that comes to light.

Figure 5.9 depicts one choice for partitioning the identification field: 3 offset bits to represent 8 possible fragments, 5 bits to represent the distance, and 8 bits for the edge fragment. I use a 32-bit hash, which doubles the size of each router address to 64 bits. This implies that 8 separate fragments are needed to represent each edge – each fragment indicated by a unique offset value. Finally, 5 bits is sufficient to represent 31 hops, which is more than almost all Internet paths [Carter et al. 97, Theilmann et al. 00, CAIDA 00].⁶

The observant reader will note that this layout is chosen to allow the highest performance software implementation of my algorithm, which already had a low per-packet router overhead. In the common case, the only modification to the packet is to increment its distance field. Because of its alignment within the packet, this increment precisely offsets the required decrement of the time-to-live field implemented by each router [Baker 95]. Consequently, the header checksum *does not need to be altered at all* and the header manipulation overhead could be even lower than in current software-based routers – simply an addition to the distance field, a decrement to the TTL field, and a comparison to check if either has overflowed. In the worst case, the algorithm must read the IP identification field, lookup an edge fragment and XOR it, and fold the write-back into the existing checksum update procedure (a few ALU operations). Of course, for modern ASIC-based routers

⁶It is also reasonable to turn off marking on any routers that cannot be directly connected to an attacking host (e.g. core routers). This both reduces the convergence time, and increases the “reach” of the distance field.

these optimizations are unnecessary.

Since the IP identification field is overloaded, I must address issues of backwards-compatibility for IP fragment traffic. Ultimately, there is no perfect solution to this problem and it is necessary to make compromises that disadvantage fragmented traffic. Fortunately, recent measurements suggest that less than 0.25% of packets are fragmented [Stoica et al. 99, Claffy et al. 00]. Moreover, it has long been understood that network-layer fragmentation is detrimental to end-to-end performance [Kent et al. 87] so modern network stacks implement automatic MTU discovery to prevent fragmentation regardless of the underlying media [Mogul et al. 90]. Consequently, I believe that this encoding will inter-operate seamlessly with existing protocol implementations in the vast majority of cases.

However, there is a small but real fraction of legitimate traffic that is fragmented, and consequently there can be a conflict over the use of the identification field. Normally if a packet is fragmented, its identification field is copied to each fragment so the receiver can faithfully reassemble the fragments into the original packet. The marking procedure can violate this property in one of two ways: by writing different values into the identification fields of fragments from the same datagram or by writing the same values into the identification fields of fragments from different datagrams. These two problems present different challenges and have different solutions.

First, a datagram may be fragmented *upstream* from a marking router. If the fragment is subsequently marked and future fragments from the same datagram are not marked consistently then reassembly may fail or data may be corrupted. While the simplest solution to this problem is to simply not mark fragments, an adversary would quickly learn to evade traceback by exploiting this limitation. In fact, some current denial-of-service attacks already use IP fragments to exploit errors in host IP reassembly functions [CERT 97]. Instead, I propose an alternative marking mechanism that uses a separate marking probability, q , for fragments. When a router decides to mark a fragment, it prepends a new ICMP “echo reply” header, along with the *full* edge data – truncating the tail of the packet. This ICMP packet is considered “marked” and its distance field is set to zero, thereby guaranteeing that the distance field reflects the number of edges traversed on the way to the victim. The packet is consequently “lost” from the standpoint of the receiver, but the edge information is delivered in a way that does not impact legacy hosts. Because the full edge sampling algorithm can be used, q can be more than an order of magnitude smaller than p and yet achieve the same conver-

gence time. This solution increases the loss rate of fragmented flows somewhat (more substantially for longer paths) but preserves the integrity of the data in these flows.

A more insidious problem is presented by fragmentation that occurs *downstream* from a marking router. If a marked packet is fragmented, but one of the fragments is lost, then the remaining fragments may linger in the victim's reassembly buffer for an extended period [Braden 89]. Future packets marked by the same router can have the same IP identification value and consequently may be incorrectly reassembled with the previous fragments. One possibility is to leave this problem to be dealt with by higher layer checksums. However, not all higher layer protocols employ checksums, and in any case it is dangerous to rely on such checksums because they are typically designed only for low residual error rates. Another solution is to set the *Don't Fragment* flag on every marked packet. Along rare paths that require fragmentation, this solution will degrade communication between hosts not using MTU path discovery, and may filter marked packets if a reduced MTU edge is close to the victim, but it will never lead to data corruption.

5.4.3 Assessment

I have implemented the marking and reconstruction portions of this algorithm and have tested it using a simulator that creates random paths and originates attacks. In Figure 5.10 I graph the mean, median and 95th percentile for the number of packets required to reconstruct paths of varying lengths over 1,000 random test runs for each length value. I assume a marking probability of $\frac{1}{25}$. Note that while the convergence time is theoretically exponential in the path length, all three lines appear linear due to the finite path length and appropriate choice of marking probability.

Using this approach most paths can be resolved with between one and two thousand packets, and even the longest paths can be resolved with a very high likelihood within four thousand packets. To put these numbers in context, most flooding-style denial of service attacks send many hundreds or thousands of packets each second. The analytic bounds described earlier are conservative, but in practice they appear to be no more than 30% higher than the experimental results.

5.5 *Limitations and future work*

There are still a number of limitations and loose ends in my approach. I discuss the most important of these here:

- backwards compatibility,
- distributed attacks,
- path validation, and
- approaches for determining the attack origin.

5.5.1 *Backwards compatibility*

The IP header encoding as I have described it has several practical limitations. It negatively impacts users that require fragmented IP datagrams and is incompatible with parts of IPsec [Kent et al. 98] (the Authentication Header provides cryptographic protection for the identification field and therefore the field cannot be safely modified by routers). These problems are hardly unique to my traceback technique and are inherent limitations that come about from attempting to co-exist with or co-opt protocol features that did not anticipate a new use. One way to partially address this issue, originally proposed by John Hawkinson, is to selectively enable traceback support in response to operational needs. A “request for traceback” from a particular network could be encoded as a BGP attribute in the network’s route advertisement. Routers receiving such an advertisement would enable traceback support on packets destined for that network. Since a network requesting such support is presumably already suffering under an attack, any minor service degradation for fragmented flows would be acceptable.

Finally, my scheme does not address implementation in IPv6, the proposed successor to IPv4, which does not have an identification field [Deering 98]. While I do not attempt to propose a complete encoding here, I believe that the same techniques I have proposed could also be employed within IPv6, perhaps by overloading the 24-bit *flow label* field (without any further modifications this would result in roughly a factor of three increase in the number of packets required to reconstruct a path).

5.5.2 *Distributed attacks*

For moderate distributed attacks, the implementation I have described has serious limitations due to the difficulty in correctly grouping fragments together. Consequently, the probability of mis-attributing an edge, as well as the amount of state needed to evaluate this decision, increases very quickly with the fan-out of an attack. There is ongoing work by several groups to develop improved marking algorithms to address this deficiency. Song and Perrig leverage the additional assumption of a network topology map to compress the representation of edge state – thereby vastly improving the robustness against distributed attack [Song et al. 01]. Dean, Franklin and Stubblefield also improve robustness by replacing the ad hoc XOR-based marking approach with one based on algebraic coding theory [Dean et al. 01]. There is significant future work in designing alternative encoding methods that scale their robustness as they receive more data.

5.5.3 *Path validation*

Some number of the packets sent by the attacker are unmarked by intervening routers. The victim cannot differentiate between these packets and genuine marked packets. Therefore an attacker could insert “fake” edges by carefully manipulating the identification fields in the packets it sends. While the distance field prevents an attacker from spoofing edges between it and the victim – what I call the *valid suffix* – nothing prevents the attacker from spoofing extra edges past the end of the true attack path.

There are several ways to identify the valid suffix within a path generated by the reconstruction procedure. With minimal knowledge of Internet topology one can differentiate between routers that belong to transit networks (e.g. ISPs) and those which belong to stub networks (e.g. enterprise networks). Generally speaking, a valid path will never enter a stub network and then continue into a transit network. Moreover, simple testing tools such as `traceroute` should enable a victim to determine if two networks do, in fact, connect. More advanced network maps [Cheswick et al. 00, Govindan et al. 00] can resolve this issue in an increasing number of cases.

A more general mechanism is to provide each router with a time-varying “secret” that is used to authenticate each marked packet (minimally, one bit in the IP header). When the victim wants to validate a router in the path, it could contact the associated network (possibly out of band, via

telephone or e-mail) and obtain the secret(s) used by the router at the time of the attack. To guard against replay, the secret must be varied relatively quickly and hashed with the packet contents. Since the attacker will not know the router's secret, the forged edge-id fragments will not contain a proper authentication code. By eliminating edge-ids for which the constituent fragments can not be validated, the candidate attack path can be pruned to only include the valid suffix. This rough idea is developed much further in Song and Perrig's traceback proposal [Song et al. 01].

5.5.4 *Attack origin detection*

While an IP-level traceback algorithm could be an important part of the solution for stopping denial-of-service attacks, it is by no means a complete solution. My algorithm attempts to determine the approximate origin of attack traffic – in particular, the earliest traceback-capable router involved in forwarding attack traffic from the source that directly generated it. As mentioned earlier, there are a number of reasons why this may differ from the true source of the attack: attackers can hide their true identities by “laundering” attacks through third parties, either indirectly (e.g. smurf attacks [CERT 98] or DNS reflectors [CERT 00b]) or directly via compromised “stepping stone” machines or IP-in-IP tunnels. While there is on-going work on following attackers through intermediate hosts [Zhang et al. 00, Staniford-Chen et al. 95], there are still significant challenges in developing a generally applicable and universally deployable solution to this problem. One interesting possibility enabled by the packet marking approach is to extend traceback across “laundering points”. For example, identifying marks could be copied from a DNS request packet into the associated DNS reply, thereby allowing the victim to trace the full causal path. However, this would also increase the required path length to be reconstructed in such cases – possibly exceeding the limited space in the length field.

Even in absence of such “laundering”, my approach does not reveal the actual host originating the attack. Moreover, since hosts can forge both their IP source address and MAC address the origin of a packet may never be explicitly visible. On shared media such as FDDI rings, this problem can only be solved by explicit testing. However, on point-to-point media, the input port a packet arrives on is frequently enough to determine its true origin. On other media, there may be a MAC address, cell number, channel, or other hint that would help to locate the attack origin. In principle, my algorithm could be modified to report this information by occasionally marking packets with a

special edge-id representing a link between the router and the input port on which the packet arrived (or other “hint” information). I have not explored the design of such a feature in any depth.

Finally, traceback is only effective at finding the source of *attack traffic*, not necessarily the *attacker* themselves. Stopping an attack may be sufficient to eliminate an immediate problem, but long term disincentives may require a legal remedy and therefore the forensic means to determine an attacker’s identity. Even with perfect traceback support, unambiguously identifying a sufficiently skilled and cautious attacker is likely to require cooperation from law enforcement and telecommunications organizations.

5.6 Summary

In this chapter I have argued that denial-of-service attacks motivate the development of improved traceback capabilities and I have explored traceback algorithms based on packet marking in the network. I have shown that this class of algorithm, best embodied in *edge sampling*, can enable efficient and robust multi-party traceback that can be incrementally deployed and efficiently implemented. As well, I have developed variant algorithms that sacrifice convergence time and robustness for reduced per-packet space requirements. Finally, I have described one potential deployment strategy using such an algorithm based on overloading existing IP header fields and I have demonstrated that this implementation is capable of fully tracing an attack after having received only a few thousand packets. This solution represents a valuable first step towards an automated network-wide traceback facility.

Chapter 6

Conclusion

The Internet is not a single network, but rather a collection of many different networks and hosts interconnected through the use of a few common protocols. Consequently, unlike traditional local area distributed systems, the Internet has the property that different system components are frequently under the control of different administrative authorities – each with their own policies and objectives. This characteristic has profound implications for network service designers since without the assumption of widespread cooperation important system-wide properties can no longer be engineered explicitly. Instead such properties must be established implicitly through observation, inference, incentives and careful protocol design.

Network service designers face twin challenges in such an environment: accommodating conflict and preserving backwards compatibility. Conflict is the inevitable price of scale and administrative heterogeneity. Two different Internet hosts may have different goals about whether to participate in a service or how to share a resource. In the extreme, one host may even desire to shut the other down. Accommodating such conflict requires understanding the spectrum of different interests that may be party to a communication and then designing a protocol that is either compatible with the interests of all parties or one that is at least robust against exploitation. A further complexity is that the large number of hosts and administrative domains implies that it is infeasible to “upgrade” the Internet en masse. Consequently, any solution to these problems must maintain backwards compatibility with the existing Internet protocols to have a realistic hope of deployment.

In this thesis, I have demonstrated that both challenges are surmountable. I have demonstrated several protocols that provide correct services in spite of uncooperative, competitive and even malicious hosts. Moreover, I have shown that these aims can be met while preserving sufficient compatibility to allow for deployment in the existing Internet.

First, I examined the problem of measuring one-way network path characteristics, such as packet

loss rate. These measurements require observations at both endpoints, but unfortunately there is little motivation for most Internet sites to cooperate in such experiments. By creatively exploiting the standard behavior of the existing TCP protocol, I demonstrated that it is possible to obtain these measurements implicitly, without requiring explicit cooperation. The key observation is that by aligning the goal of the content provider (to serve content) with that of the measurement client we can implement a TCP-based measurement service that is compatible with the goals of both hosts. This approach, and the underlying source code, has since been used in several measurement projects, including one to evaluate congestion control policies [Padhye et al. 01], measure link capacity [Saroiu et al. 01], round-trip latency [Collins 01] and estimate packet re-ordering [Bellardo 01].

Next, I examined the fragility of congestion control protocols. Today's Internet depends on every host to voluntarily limit the rate at which they send data. This good faith approach to resource sharing was appropriate during the Internet's "kinder and gentler" days, but is not dependable in today's competitive environment. I demonstrated that today's congestion control protocols have serious weaknesses that allow receivers of data (i.e. Web clients) to coerce remote servers into sending data at arbitrary rates. As one might suspect, many users would happily improve their own performance at the expense of others. However, this weakness is not a fundamental limitation of end-to-end congestion control, but is the accident of particular protocol design decisions. I show that a small set of backwards compatible protocol modifications can detect and punish attempts at "cheating" – thereby eliminating the incentive to do so. The approach underlying this work is not specific to TCP and has since been both validated and extended to "Explicit Congestion Notification" [Ely et al. 01b].

Finally, as we have become all too aware, the Internet is vulnerable to malicious denial-of-service attacks. These attacks present a unique challenge because the Internet architecture relies on each host to voluntarily indicate a packet's origin. Attackers ignore this convention and consequently determining the source of such attacks is difficult and time-consuming at best. To address this problem, I described an efficient, incrementally deployable, and (mostly) backwards compatible network mechanism that allows victims to trace denial-of-service attacks back to their source. Moreover, the protocol is designed to operate correctly in spite of explicit attempts to circumvent or disrupt it. This approach has become the basis for a several ongoing efforts to define traceback

mechanisms [Dean et al. 01, Song et al. 01].

6.1 Future Work

Distributed system design has many challenges – performance, scalability, availability and so forth. However, most of the techniques used to address these challenges require the participants to communicate or infer one another’s state (e.g. concerning load, cache contents, name resolution, etc). A key issue is how to protect this state from being corrupted. Traditionally, most systems have relied entirely on simple binary trust relationships to ensure robustness. Under this approach, “trusted” participants are provided cryptographic credentials that carry with them the assumption that all communications will be truthful, consistent and cooperative. All other participants are excluded from the system. The key benefit of this model is that it allows remote participants to be efficiently modeled with abstract state machines – the essence of almost all distributed protocols.

Unfortunately trust is a property that does not scale well. While it may be reasonable to trust 10 or 100 users within the same administrative domain, it is much less clear why 100,000 users among 20 different organizations are worthy of the same trust. To wit, there is a clear and widespread understanding that verification [Necula et al. 96, Lindholm et al. 97] is preferable to authentication [Microsoft 99] for ensuring the safety of mobile code. With hundreds of millions of users and thousands of different administrative domains, Internet-scale distributed systems cannot reasonably rely on trust, as established through cryptographic mechanisms, to ensure robustness. As a consequence, protocol designers *cannot* depend on traditional abstract state machines to model remote Internet hosts – hosts will be consistent to such a model only when it serves their own goals.

This dissertation has only scratched the surface of this problem. While I have demonstrated the viability of accommodating conflicts in specific end-to-end protocols, such as congestion signaling, there is significant creativity required to extend these solutions to other domains, such as network-ing routing. The remaining challenge, and a substantial open problem, is how to generalize these approaches into a reusable protocol design methodology that can guarantee system robustness even when participants cannot be trusted to behave consistently.

I believe there are three components to this research agenda. First, is a set of design principles for exposing, expressing, and testing remote protocol state. By exposing internal state to outside

scrutiny many behavioral properties can be validated. In this dissertation for example, both congestion signaling and IP traceback protocols include sufficient information that a skeptic may “test” the protocol for compliance with a shared behavioral contract and consequently enforce a consistent behavior. However, determining what state to expose and how to express remains non-obvious. The second part of any solution is a general methodology for determining whether a host’s behavior and state is consistent with the abstract state machine it claims to implement or not. This is a model checking problem, but is complicated significantly by the fact that the system being evaluated may be adversarial and act in a Byzantine fashion to confuse any analysis. Finally, if a behavioral property cannot be validated (e.g. because it is not dependent on external state) then its potential impact should be minimized. The final part of this research agenda then is a framework for controlling the spread and effect of un-validated data in a system.

These observations are increasingly understood in narrow communities, such as multi-player gaming and peer-to-peering computing, that have started to experience routine “cheating” accomplished via protocol violations. However, the problem is fundamental to any large scale distributed system in which the interests of different users are in conflict. If we are to rely on such systems, we must develop a new methodology for system design that explicitly accommodates conflicts and inconsistencies, and limits the impact of any abuse.

Bibliography

- [Abadi et al. 96] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.
- [Allman 98] M. Allman. On the Generation and Use of TCP Acknowledgments. *ACM Computer Communications Review*, 28(5):4–21, October 1998.
- [Allman 99] M. Allman. TCP Byte Counting Refinements. *ACM Computer Communications Review*, 29(3):14–22, July 1999.
- [Allman et al. 99] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.
- [Almes 97] G. Almes. Metrics and Infrastructure for Internet Protocol Performance. <http://www.advanced.org/surveyor/presentations.html>, 1997.
- [Anderson et al. 01] D. G. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 2001 ACM Symposium on Operating Systems Principles*, pages 131–145, Lake Louise, Canada, October 2001.
- [Baker 95] F. Baker. Requirements for IP Version 4 Routers. RFC 1812, June 1995.
- [Balakrishnan et al. 97] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, December 1997.
- [Banga et al. 99] G. Banga, P. Druschel, and J. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the 1999 USENIX/ACM Sym-*

posium on Operating System Design and Implementation, pages 45–58, New Orleans, LA, February 1999.

- [Bellardo 01] J. Bellardo. Personal Communication, 2001.
- [Bellovin 00] S. M. Bellovin. ICMP Traceback Messages. Internet Draft: draft-bellovin-itrace-00.txt, March 2000.
- [Bellovin 89] S. M. Bellovin. Security Problems in the TCP/IP Protocol Suite. *ACM Computer Communications Review*, 19(2):32–48, April 1989.
- [Berners-Lee et al. 96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, May 1996.
- [Bolot 93] J.-C. Bolot. End-to-end Packet Delay and Loss Behavior in the Internet. In *Proceedings of the 1993 ACM SIGCOMM Conference*, pages 289–298, San Francisco, CA, September 1993.
- [Braden 89] R. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, October 1989.
- [Bradley et al. 98] K. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. Olsson. Detecting Disruptive Routers: A Distributed Network Monitoring Approach. *IEEE Network*, 12(5):50–60, September 1998.
- [Breslau et al. 99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of the 1999 IEEE INFOCOM Conference*, pages 126–134, New York, NY, March 1999.
- [Burch et al. 00] H. Burch and B. Cheswick. Tracing Anonymous Packets to Their Approximate Source. In *Proceedings of the 2000 USENIX LISA Conference*, pages 319–327, New Orleans, LA, December 2000.

- [Caceres et al. 99] R. Caceres, N. Duffield, J. Horowitz, D. Towsley, and T. Bu. Multicast-Based Inference of Network-Internal Characteristics: Accuracy of Packet Loss Estimation. In *Proceedings of the 1999 IEEE INFOCOM Conference*, pages 371–379, New York, NY, March 1999.
- [CAIDA 00] Cooperative Association for Internet Data Analysis: Skitter Analysis. <http://www.caida.org/Tools/Skitter/Summary/>, 2000.
- [Carle et al. 97] G. Carle and E. W. Biersack. Survey of Error Recovery Techniques for IP-Based Audio-Visual Multicast Applications. *IEEE Network*, 11(6):24–36, November 1997.
- [Carter et al. 97] R. L. Carter and M. E. Crovella. Server Selection Using Dynamic Path Characterization in Wide-Area Networks. In *Proceedings of the 1997 IEEE INFOCOM Conference*, pages 1014–1021, Kobe, Japan, April 1997.
- [Cate 92] V. Cate. Alex – A Global File System. In *Proceedings of the USENIX File System Workshop*, pages 1–11, Ann Arbor, Michigan, 1992.
- [Cerf et al. 98] V. Cerf and R. Kahn. A Protocol for Packet Network Interconnection. *IEEE Transactions on Communications*, 22(5):637–648, May 1998.
- [CERT 00a] Computer Emergency Response Team Advisory CA-2000-01: Denial-of-Service Developments. <http://www.cert.org/advisories/CA-2000-01.html>, January 2000.
- [CERT 00b] Computer Emergency Response Team Incident Note IN-2000-04: Denial-of-Service Attacks using Nameservers. http://www.cert.org/incident_notes/IN-200-04.html, April 2000.
- [CERT 96] Computer Emergency Response Team Advisory CA-96.26: Denial-of-Service Attack via pings. <http://www.cert.org/advisories/CA-96.26.ping.html>, December 1996.

- [CERT 97] Computer Emergency Response Team Advisory CA-97.28: IP Denial-of-Service Attacks. <http://www.cert.org/advisories/CA-97.28.smurf.html>, December 1997.
- [CERT 98] Computer Emergency Response Team Advisory CA-98.01: "smurf" IP Denial-of-Service Attacks. <http://www.cert.org/advisories/CA-98.01.smurf.html>, January 1998.
- [Cheswick et al. 00] B. Cheswick and H. Burch. Internet Mapping Project. <http://cm.bell-labs.com/who/ches/map/index.html>, 2000.
- [Cisco Systems 97] Cisco Systems. Configuring TCP Intercept (Prevent Denial-of-Service Attacks). Cisco IOS Documentation, December 1997.
- [Claffy et al. 00] K. Claffy and S. McCreary. Sampled Measurements from June 1999 to December 1999 at the AMES Inter-exchange Point. Personal Communication, January 2000.
- [Clark 88] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proceedings of the 1988 ACM SIGCOMM Conference*, pages 106–114, Palo Alto, CA, September 1988.
- [Collins 01] A. Collins. Personal Communication, 2001.
- [Computer Security Institute et al. 99] Computer Security Institute and Federal Bureau of Investigation. 1999 CSI/FBI Computer Crime and Security Survey. Computer Security Institute publication, March 1999.
- [Dean et al. 01] D. Dean, M. Franklin, and A. Stubblefield. An Algebraic Approach to IP Traceback. In *Proceedings of the 2001 Network and Distributed System Security Symposium*, San Diego, CA, February 2001.
- [Deering 98] S. Deering. Internet Protocol, Version 6 IPv6. RFC 2460, December 1998.

- [Demers et al. 89] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queuing Algorithm. In *Proceedings of the 1989 ACM SIGCOMM Conference*, pages 1–12, Austin, TX, September 1989.
- [Dierks et al. 99] T. Dierks and C. Allen. The TLS protocol. RFC 2246, January 1999.
- [Ely et al. 01a] D. Ely, S. Savage, and D. Wetherall. Alpine: A User-Level Infrastructure for Network Protocol Development. In *Proceedings of the 2001 USENIX Symposium on Internet Technologies and Systems*, pages 171–183, San Francisco, CA, March 2001.
- [Ely et al. 01b] D. Ely, N. Spring, D. Wetherall, S. Savage, and T. Anderson. Robust Congestion Signaling. In *Proceedings of the 2001 International Conference on Network Protocols*, pages 332–341, Riverside, CA, November 2001.
- [Feller 66] W. Feller. *An Introduction to Probability Theory and Its Applications (2nd edition)*, volume 1. Wiley and Sons, 1966.
- [Ferguson et al. 00] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing. RFC 2827, May 2000.
- [Fielding et al. 99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.
- [Floyd 95] S. Floyd. TCP and Successive Fast Retransmits. <http://www.aciri.org/floyd/papers/fastretrans.ps>, May 1995.
- [Floyd et al. 99a] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, August 1999.
- [Floyd et al. 99b] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, and A. Romanow. An Extension to the Selective Acknowledgment (SACK) Option for TCP. Internet Draft, August 1999.

- [Francis et al. 01] P. Francis, S. Jamin, C. Jim, Y. Jin, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, October 2001.
- [Gameranger 01] <http://www.gameranger.com>, 2001.
- [Gibbens et al. 99] R. Gibbens and F. Kelly. Resource Pricing and the Evolution of Congestion Control. *Automatica*, 35:1969–1985, 1999.
- [Glave 98] J. Glave. Smurfing Cripples ISPs. Wired Technology News: (<http://www.wired.com/news/news/technology/story/9506.html>), January 1998.
- [Goldberg et al. 99] I. Goldberg and A. Shostack. Freedom Network 1.0 Architecture and Protocols. Zero-Knowledge Systems White Paper, November 1999.
- [Govindan et al. 00] R. Govindan and H. Tangmunarunkit. Heuristics for Internet Map Discovery. In *Proceedings of the 2000 IEEE INFOCOM Conference*, pages 1371–1380, Tel Aviv, Israel, March 2000.
- [Greene et al. 01] B. Greene and P. Smith. Cisco ISP Essentials: Essential IOS Features Every ISP Should Consider. Cisco white paper, version 2.9, June 2001.
- [Hancock 00] B. Hancock. Network Attacks: Denial of Service and Distributed Denial of Service. Exodus, Inc. Whitepaper, 2000.
- [Handley et al. 01] M. Handley, V. Paxson, and C. Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *Proceedings of the 2001 USENIX Security Symposium*, Washington, D.C., August 2001.
- [Heberlein et al. 96] L. T. Heberlein and M. Bishop. Attack Class: Address Spoofing. In *1996 National Information Systems Security Conference*, pages 371–378, Baltimore, MD, October 1996.

- [Howard 98] J. D. Howard. *An Analysis of Security Incidents on the Internet*. PhD dissertation, Carnegie Mellon University, August 1998.
- [Jacobson et al. 88] V. Jacobson and M. J. Karels. Congestion Avoidance and Control. *Proceedings of the 1988 ACM SIGCOMM Conference*, pages 314–329, August 1988.
- [Jacobson et al. 92] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323, May 1992.
- [Johnson et al. 01] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek. The Measured Performance of Content Distribution Networks. *Computer Communications*, 24(2):202–206, 2001.
- [Karn et al. 99] P. Karn and W. Simpson. Photuris: Session-Key Management Protocol. RFC 2522, March 1999.
- [Kent et al. 87] C. Kent and J. Mogul. Fragmentation Considered Harmful. In *Proceedings of the 1987 ACM SIGCOMM Conference*, pages 390–401, Stowe, VT, August 1987.
- [Kent et al. 98] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, November 1998.
- [Key et al. 99] P. Key, D. McAuley, P. Barham, and K. Lavens. Congestion Pricing for Congestion Avoidance. Technical Report MSR-TR-1999-15, Microsoft Research, February 1999.
- [Lavens et al. 00] K. Lavens, P. Key, and D. McAuley. An ECN-based End-to-End Congestion-Control Framework: Experiments and Evaluation. Technical Report MSR-TR-2000-70, Microsoft Research, June 2000.
- [Lindholm et al. 97] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, MA, 1997.

- [Manajan et al. 01] R. Manajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. Unpublished whitepaper, in review, July 2001.
- [Mathis et al. 96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, April 1996.
- [Mathis et al. 97] M. Mathis, J. Semke, and J. Mahdavi. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communications Review*, 27(3):67–82, July 1997.
- [Meadows 99] C. Meadows. A Formal Framework and Evaluation Method for Network Denial of Service. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, Mordano, Italy, June 1999.
- [Microsoft 99] Microsoft. How Software Publishers Can Use Authenticode Technology. Microsoft Whitepaper, 1999.
- [Mogul et al. 90] J. Mogul and S. Deering. Path MTU Discovery. RFC 1191, November 1990.
- [Mojonation 01] <http://www.mojonation.net>, 2001.
- [Moore et al. 01] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. In *Proceedings of the 2001 USENIX Security Symposium*, Washington, D.C., August 2001.
- [Morris 85] R. T. Morris. A Weakness in the 4.2BSD Unix TCP/IP Software. Technical Report Computer Science #117, AT&T Bell Labs, February 1985.
- [Necula et al. 96] G. C. Necula and P. Lee. Safe Kernel Extensions Without Run-Time Checking. In *Proceedings of the ACM Symposium on Operating Systems Design and Implementation*, pages 229–243, Seattle, WA, October 1996.

- [Norton 01] W. B. Norton. Internet Service Providers and Peering. Unpublished whitepaper, available upon request, 2001.
- [Padhye et al. 01] J. Padhye and S. Floyd. On Inferring TCP Behavior. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 287–298, San Diego, CA, August 2001.
- [Padhye et al. 98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of the 1998 ACM SIGCOMM Conference*, pages 303–314, Vancouver, BC, September 1998.
- [Paxson 97a] V. Paxson. End-to-end Internet Packet Dynamics. In *Proceedings of the 1997 ACM SIGCOMM Conference*, pages 139–152, Cannes, France, September 1997.
- [Paxson 97b] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997.
- [Paxson et al. 98a] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. Framework for IP Performance Metrics. RFC 2230, May 1998.
- [Paxson et al. 98b] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An Architecture for Large-Scale Internet Measurement. *IEEE Communications*, 36(8):48–54, August 1998.
- [Paxson et al. 99] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz. Known TCP Implementation Problems. RFC 2525, March 1999.
- [Perkins 96] C. Perkins. IP Mobility Support. RFC 2002, October 1996.
- [Postel 81a] J. Postel. Internet Control Message Protocol. RFC 792, September 1981.
- [Postel 81b] J. Postel. Internet Protocol. RFC 791, September 1981.
- [Postel 81c] J. Postel. Transmission Control Protocol. RFC 793, September 1981.
- [Postel 84] J. Postel. Multi-LAN Address Resolution. RFC 925, October 1984.

- [Postel et al. 85] J. Postel and J. Reynolds. File Transfer Protocol (FTP). RFC 959, October 1985.
- [Rapier 98] C. Rapier. ICMP and future testing (IPPM mailing list). <http://www.advanced.org/IPPM/archive/0606.html>, December 1998.
- [Reed et al. 98] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.
- [Rejaie et al. 99] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of the 1999 IEEE INFOCOM Conference*, pages 1337–1345, New York, NY, March 1999.
- [Rekhter et al. 95] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.
- [Rizzo 97] L. Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *ACM Computer Communications Review*, 27(1):31–41, January 1997.
- [Rosen et al. 00] E. C. Rosen, Y. Rekhter, D. Tappan, D. Farinacci, G. Fedorkow, T. Li, and A. Conta. MPLS Label Stack Encoding. Internet Draft: draft-ietf-mpls-label-encaps-08.txt (expires March 2000), July 2000.
- [RouteScience] RouteScience Technologies, Inc. <http://www.routescience.com>.
- [Sager 98] G. Sager. Security Fun with OCxmon and cflowd. Presentation at the Internet 2 Working Group, November 1998.
- [Saroiu et al. 01] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. Technical Report UW-CSE-01-06-02, University of Washington, June 2001.

- [Savage 99] S. Savage. Sting: a TCP-based Network Measurement Tool. In *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, pages 71–79, October 1999.
- [Savage et al. 00] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 295–306, Stockholm, Sweden, August 2000.
- [Savage et al. 01] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network Support for IP Traceback. *IEEE/ACM Transactions on Networking*, 9(3):226–237, June 2001.
- [Savage et al. 99a] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *ACM Computer Communications Review*, 29(5):71–78, October 1999.
- [Savage et al. 99b] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-end Effects of Internet Path Selection. In *Proceedings of the 1999 ACM SIGCOMM Conference*, pages 289–299, Boston, MA, August 1999.
- [Schneier 96] B. Schneier. *Applied Cryptography*. John Wiley and Sons, 2nd edition, 1996.
- [Shenker 94] S. Shenker. Making Greed Work in Networks: A Game-Theoretic Analysis of Switch Service Disciplines. In *Proceedings of the 1994 ACM SIGCOMM Conference*, pages 47–57, London, UK, August 1994.
- [SLAC 99] Icmp rate limiting. SLAC End-to-End Performance Monitoring Project: <http://www-iepm.slac.stanford.edu/monitoring/limit/limiting.html>, December 1999.
- [Smart et al. 00] M. Smart, G. R. Mlan, and F. Jahanian. Defeating TCP/IP Stack Fingerprinting. In *Proceedings of the 2000 USENIX Security Symposium*, Denver, CO, August 2000.
- [SockeyeNetworks] Sockeye Networks, Inc. <http://www.sockeye.com>.

- [Song et al. 01] D. Song and A. Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proceedings of the 2001 IEEE INFOCOM Conference*, Anchorage, AK, April 2001.
- [Spatscheck et al. 99] O. Spatscheck and L. Peterson. Defending Against Denial of Service Attacks in Scout. In *Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, pages 59–72, New Orleans, LA, February 1999.
- [Staniford-Chen et al. 95] S. Staniford-Chen and L. T. Heberlein. Holding Intruders Accountable on the Internet. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 39–49, Oakland, CA, May 1995.
- [Stevens 94] W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison Wesley, 1994.
- [Stoica et al. 99] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings of the 1999 ACM SIGCOMM Conference*, pages 81–94, Boston, MA, August 1999.
- [Stone 00] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *Proceedings of the 2000 USENIX Security Symposium*, Denver, CO, July 2000.
- [Theilmann et al. 00] W. Theilmann and K. Rothermel. Dynamic Distance Maps of the Internet. In *Proceedings of the 2000 IEEE INFOCOM Conference*, Tel Aviv, Israel, March 2000.
- [Vaskovich] F. Vaskovich. nmap. <http://www.insecure.org/nmap/>.
- [Villamizar 00] C. Villamizar. Personal Communication, February 2000.
- [Vivo et al. 99] M. Vivo, E. Carrasco, G. Isern, and G. O. Vivo. A Review of Port Scanning Techniques. *ACM Computer Communications Review*, 29(2):41–48, April 1999.
- [Yahoo! Inc] Yahoo! Inc. Random Yahoo! Link. <http://random.yahoo.com/bin/ryl>.

- [Ylonen et al. 00] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Transport Layer Protocol. Internet Draft, May 2000.
- [Zhang et al. 00] Y. Zhang and V. Paxson. Stepping Stone Detection. In *Proceedings of the 2000 USENIX Security Symposium*, Denver, CO, July 2000.
- [Zhang et al. 93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 7(5):8–18, September 1993.

Vita

Stefan Savage was born on June 11, 1969 in Paris, France and was raised in New York City, New York. In 1987 he attended Carnegie Mellon University, where he received his B.S. degree in Applied History in 1991. He attended the University of Washington starting in 1994 and received his Ph.D. degree in Computer Science in Engineering in 2002.