Statistical Learning in Network Architecture

by

Robert Edward Beverly IV

B.S., Georgia Institute of Technology (1997)
S.M., Massachusetts Institute of Technology (2004)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author . . . .
Department of Electrical Engineering and Computer Science
June 9, 2008

Certified by . . . .
Dr. Karen R. Sollins
Principal Research Scientist
Thesis Supervisor

Accepted by . . . .
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Statistical Learning in Network Architecture

by

Robert Edward Beverly IV

Submitted to the Department of Electrical Engineering and Computer Science
on June 9, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The Internet has become a ubiquitous substrate for communication in all parts of society. However, many original assumptions underlying its design are changing. Amid problems of scale, complexity, trust and security, the modern Internet accommodates increasingly critical services. Operators face a security arms race while balancing policy constraints, network demands and commercial relationships. This thesis espouses learning to embrace the Internet's inherent complexity, address diverse problems and provide a component of the network's continued evolution.

Malicious nodes, cooperative competition and lack of instrumentation on the Internet imply an environment with partial information. Learning is thus an attractive and principled means to ensure generality and reconcile noisy, missing or conflicting data. We use learning to capitalize on under-utilized information and infer behavior more reliably, and on faster time-scales, than humans with only local perspective. Yet the intrinsic dynamic and distributed nature of networks presents interesting challenges to learning. In pursuit of viable solutions to several real-world Internet performance and security problems, we apply statistical learning methods as well as develop new, network-specific algorithms as a step toward overcoming these challenges. Throughout, we reconcile including intelligence at different points in the network with the end-to-end arguments.

We first consider learning as an end-node optimization for efficient peer-to-peer overlay neighbor selection and agent-centric latency prediction. We then turn to security and use learning to exploit fundamental weaknesses in malicious traffic streams. Our method is both adaptable and not easily subvertible. Next, we show that certain security and optimization problems require collaboration, global scope and broad views. We employ ensembles of weak classifiers within the network core to mitigate IP source address forgery attacks, thereby removing incentive and coordination issues surrounding existing practice. Finally, we argue for learning within the routing plane as a means to directly optimize and balance provider and user objectives.

This thesis thus serves first to validate the potential for using learning methods to address several distinct problems on the Internet and second to illuminate design principles in building such intelligent systems in network architecture.

Thesis Supervisor: Dr. Karen R. Sollins
Title: Principal Research Scientist

See first, think later, then test. But always see first. Otherwise you will only see what you were expecting.

- Douglas Adams

Acknowledgments

As sounding boards, purveyors of constructive criticism and sources of inspiration, my thesis advisors were unparalleled. In retrospect, I realize just how effective their subtle suggestions and prodding were in shaping my thinking and honing my ideas. For their support and guidance, I must thank:

- **Karen Sollins:** Karen's unassuming nature belies her huge body of experience and keen understanding of systems. Karen allowed me to pursue the research I found most interesting and encouraged useful tangents while redirecting me as necessary. She showed me how dead-ends are inevitable when tackling difficult problems, yet incredibly important to the final research product. Moreover, she was always ready with practical advise on everything from research to Scottish dance. I am indebted to Karen as a mentor, advisor and colleague these past six years.

- **Dave Clark:** Having been involved in just about every facet of the Internet in one way or another, Dave's perspective was unparalleled. Dave often helped me to step back, extract and articulate larger lessons, and place my work in the proper context. My ability to get Dave excited about a piece of research became my litmus test and I thank him for suffering through the bad ideas while encouraging the good ones.

- **Tommi Jaakkola:** Tommi's machine learning class was my first introduction to statistical learning theory and served as the impetus for tackling ideas in this thesis. I am extremely grateful, therefore, to have had Tommi on my committee to help bridge the gap between networking and learning.

At MIT, I was lucky enough to work with some extraordinarily talented individuals:

- **Mike Afergan:** Although he was never able to adequately explain either his supernaturally quick matriculation or the Red Sox victory, Mike brightened every part of my graduate career. Mike's ability to distill complicated problems down to their relevant parts and insightful bits was invaluable in challenging my ideas and making them stronger. I will miss bouncing ideas around in the office and poking fun at academia while in the gym – thanks for your friendship.

- **Steve Bauer:** Steve was my partner in crime for several measurement projects and papers. During this time, I realized how talented Steve is as a researcher. Steve always lent a willing ear to any crazy idea that might pop into my head and I'll remember all of the long-Friday lunches deep in discussion over some piece of arcana. I truly hope, and expect, to work with Steve again someday in the future.

- **John Wroclawski:** John has an uncanny knack for quickly honing in on not only strengths, but also weaknesses in any research. While frustrating, the end product is always stronger and this thesis is better as a result of my discussions with John. I thank him for always pushing me and my research to be better.

Within the Advanced Network Architecture group, many unique and interesting students, especially Simson Garfinkel, Richard Hansen, George Lee, Ji Li, Nishanth Sastry and Tim Shepard, were great colleagues. Arthur Berger and Bruce Davie deserve special recognition. Arthur was fantastic to work with and contributed significant rigor to my research and our papers. Bruce brought his expertise in operational carriers to all of our discussions and encouraged relevant research directions. Becky Shepardson and Susan Perez helped in more ways than I can remember or acknowledge.

Much of this work would not have been possible without funding from the National Science Foundation, in particular awards CNS-0137403, CNS-0626904 and CCF-0122419. I thank the NSF for their tireless dedication to the pursuit of fundamental research. Later bits of work were funded by Cisco through their university research program. I am grateful for Cisco's support in pursuing this research and their excellent feedback.

I would be remiss not to thank Neil Gershenfeld at the Media Lab. Neil is a rare breed, combining real smarts with a research agenda that includes a future most of us cannot envision. It was a privilege to work with Neil. I wish Neil, Kerry Lynn and all of the Internet 0 folks the best.

Upstairs, Nick Feamster, Srikanth Kandula, Dina Katabi, Sachin Katti, Nate Kushman, Hariharan Rahul and Stan Rost all improved my research by reviewing early drafts, listening to practice talks and generally contributing their keen research sense. Hari Balakrishnan's graduate networking class showed me how much I did not yet know, and how fun networking research can be. Hari was a fantastic teacher always ready with significant research insights.

Prior to MIT, I had the pleasure of working with Randy Nicklas, Kevin Thompson, Greg Miller and Vint Cerf – all of whom were more than supportive and contributed to where I am today. k claffy at CAIDA has been an inspiration for many of my initiatives over the past ten years and I must recognize her dedication and support. Also in the real world, Jeff Barrows, John Curran and Marty Hannigan were great friends and kept me grounded in reality while my head was off in academia.

At the onset of this thesis, I met a girl at random who took my breath away. She still does; my wife Kelly is my best friend and confidant. Thanks for everything, pup.

It is with great humility that I dedicate this thesis to my grandparents, people of great character who worked tirelessly to provide the opportunities I have enjoyed. Any difficulties

I encountered in writing this thesis pale in comparison to their sacrifices.

This same strength of character pervades every aspect of my parents and I am forever indebted to them for instilling these same values in me despite my frequent resistance. Even when I did not believe in myself, your confidence in my ability always brought me through. You broadened my world, introduced me to everything and let me choose my own path. You rode bikes and played soccer with me, took me for walks and gave me an appreciation for the journey, the trees and flowers along the way. You've inspired me in more ways than you will ever know. My love to you forever.

Contents

List of Figures

List of Tables

We are in great haste to construct a magnetic telegraph from Maine to Texas; but Maine and Texas, it may be, have nothing important to communicate.

- Henry David Thoreau

Chapter 1

Introduction

A central question systems architects face is how to abstract, modularize and best place functionality. The strength of an architecture often lies in its longevity – the ability to support new or unanticipated demands. A canonical example in communication networks, representing two ends of the functionality design spectrum, is the comparison between the telephone network and the Internet. Neither design is necessarily superior; in fact much of the Internet is built on top of transport circuits in the phone network. A key recognition is that each is suited for different requirements and designed upon different assumptions.

While legacy networks such as the telephone system centralize the bulk of system functionality, the Internet is unique in distributing functionality to the network edge. For example, the Internet Protocol (IP) provides no guarantees of reliability, ordering, bandwidth, latency, etc. Hosts or services which require such guarantees must implement them, thereby alleviating the whole of the network from the burden of unneeded functionality. First articulated by Saltzer, Reed and Clark as the "end-to-end argument" [131], these design principles are part of several architectural enablers identified in a retrospective analysis of the Internet's success [32, 41].

The end-to-end arguments have permitted the Internet to scale through several orders of magnitude and allowed new, unforeseen applications such as the world wide web, peer-to-peer (P2P) overlays, video, and IP telephony to emerge. However, many of the design assumptions underpinning the original Internet architecture are changing or have already changed.

A survey of the Internet's evolution shows a tremendous shift in users, uses, nodes, applications and societal perceptions [27, 12]. Businesses, governments and even emergency services rely on the Internet as a common platform for distributed communication. Yet the Internet's importance belies modest origins. The Internet made a remarkable transition from an academic to a commercial network [83]. With roots in academia, the Internet includes few or loose notions of security, auditing, pricing or management. While recognized early on, these issues intentionally received secondary attention [41] and now have profound implications. As a piece of critical infrastructure, the modern Internet faces problems of security, complexity, resilience and trust. A crucial consideration is how the Internet architecture can face existing problems, continue to evolve, resolve future conflict and accommodate new demands [42].

1.1 The Role of Learning in an Evolving Internet

This thesis espouses *learning* to not only address many of the aforementioned challenges, but also provide a general framework for network architecture in a complex world. Our high-level sentiment is echoed by the knowledge plane [43] which recognizes failings in the current Internet and argues for a distributed, cognitive system architecture.

We therefore consider where, if at all, *intelligence* should be placed in the network and what form it should take. Because of the dynamic, complex and multi-party nature of the Internet, we argue that any intelligence must include *predictive learning* where the network can react to new or unknown situations, data and environments[1]. Indeed, the Internet is characterized by many entities with conflicting interests and strategies, often commercially driven, working through cooperative competition. Other entities are actively hostile and adapt quickly to maximize malicious opportunities. In contrast to approaches that use memoization or pure algorithmic optimization with complete knowledge, we explicitly recognize and embrace the inherent complexity of the modern Internet.

1.1.1 Why Intelligence?

The success of the Internet begs the question of why future architectures should include intelligence. The Internet's heritage resulted in a design with significant advantages, but also weaknesses under modern stresses. Broadly, stress on the current architecture, in the form of scale, security, robustness and policy constraints *demands* increased intelligence. Addressing diverse strains in a common and robust way is vital to the Internet's continued evolution:

- ***Security:*** The original lack of emphasis on security, trust, identity and accountability has led to unsolicited commercial email (spam), social engineering (phishing), denial-of-service (DoS) and other attacks on users, hosts and the infrastructure itself.

- ***Complexity:*** An ever increasing number of nodes, users and critical services attach to the Internet. Meanwhile, functionality such as routing remains mired in legacy notions. Implementing policy, commercial relationships, reconciling network demands with economic factors and providing resilience is accomplished via available but inefficient, indirect and brittle mechanisms. As a consequence, the Internet is less robust due to incomplete network views, routing fragility and manual configuration.

[1] The precise meanings of "intelligence" and "learning" are the source of much debate and many theories. We use them as umbrella terms here, but more generally view learning as using statistics to make decisions. Chapter 2 refines this notion concretely.

- ***Trust:*** An undesirable artifacts of the Internet's commercial transition is its inability to accommodate "tussles." As first expressed by Clark et al. [45], Internet protocols must change to allow for the natural tussles that emerge as a result of both economic, social and technical stakeholders. For example, the widespread success of P2P overlays, where traditional clients also act as servers, raises tensions between users, free-riders, networks, operators and copyright holders.

Intelligence provides a natural means to combat these inter-related strains on the architecture. A predictive notion of learning promises networks that are adaptable, efficient and resilient in ways not currently possible. Such an intelligent network could perform diagnostics, recover from failures and stave off malicious attacks.

There exists a wealth of information at various levels and granularity on user, node and network activity, reputation, behavior, etc. Unfortunately, this data is hard to gather and aggregate with existing tools and available network support. In addition, the data is sufficiently complex so that it is difficult to make useful inferences, diagnoses or decisions. Our research leverages increased intelligence to extract, analyze and use information to the benefit of the modern network, even amid a potentially hostile environment.

1.1.2 Where to Place Intelligence?

Given the motivation to include additional intelligence, where should that intelligence be placed? Again, changing assumptions motivate seemingly non-traditional choices. For example, in a world where all nodes are potential victims of attack, all nodes require protection. Thus, including attack mitigation intelligence within the network, and inducing a cost on the network and its participants, is a reasonable design choice that does not violate end-to-end principles. Throughout this thesis and in our examples, we reconcile including intelligence at different points in the network while remaining consistent with the end-to-end arguments.

- ***Edge Intelligence:*** The basic Internet architecture has enabled new applications to emerge without additional network support. Several of the problems we tackle follow an end system design perspective and endow end-nodes with additional intelligence. While end-nodes already contain a variety of intelligent functionality, we show that there is ample room for these nodes to gather and use available data in non-traditional ways to their advantage. Because end-node intelligence is compatible with the current Internet architecture, such schemes have the added advantage of immediate deployment without implementation hurdles.

- ***Core Intelligence:*** Some classes of problem are best solved by adding intelligence to the network core, however. The core of the network affords a broader and more complete view; intelligence can be placed on collections of users, traffic, etc. to better

> optimize global and local performance or maintain security[2]. Consider a piece of email as one trivial example. Many inferences are possible on that email by the receiver. However, information such as how many other emails the sender has sourced, if the sender is receiving incoming emails as well as sending, etc. expands the inference space and therefore permits *better decisions*. Many of the security and optimization problems the Internet faces require this sort of collaboration and global scope, thus motivating our exploration into core intelligence.

As the Internet continues to grow in scale and scope, it penetrates more deeply and broadly, across users and societies. The problems the Internet faces are global problems and are therefore not well-solved by humans with only local perspective. Automated intelligence can be especially useful in situations where it is not possible to wait for human intervention, for example attacks, faults, worm propagation, etc. Further, as the network penetrates more deeply, there is a smaller fraction of users expert in networking. All of these factors combine to strongly motivate intelligence at *different levels* within the network.

1.2 Why Machine Learning?

In order to realize an intelligent network, we turn to learning methods, in particular machine learning. Several properties of the Internet fit well with machine learning. The Internet contains many data sources and data points in the form of users, networks, packets, flows and more. Often, these data points are relatively cheap to obtain through either active or passive measurement. However, the data is typically noisy; the Internet's structure implies partial, hidden, malicious and conflicting information. Machine learning uses statistics to build models and form decisions, and is therefore particularly adept at dealing with the uncertainty common to many of the aforementioned problems. For instance, one or more agents[3] may collect measurements which are used to form predictions, classifications or decisions. Whereas traditional algorithms cannot easily handle the conflicting measurements we would expect of Internet agents, machine learning is amenable to synthesizing data into a model that generalizes well. This generality comes at the expense of potential errors, however we show in §1.2.2 that many Internet problems can tolerate occasional predictive error.

Despite these advantages, other fundamental characteristics of the Internet present challenges to realizing the utopia of a cognitive network. The Internet is both highly dynamic and distributed. Traffic, topologies and policies shift in time, suggesting that traditional supervised learning methods, i.e. "train-then-test," require an internal notion of when the underlying environment has changed and how to adapt in an efficient manner. Learning must be on-line and continual. For many tasks, models and learning information needs to

[2] Core intelligence is contentious; we intend to be provocative by motivating such designs as sound.

[3] Again, we loosely define agents abstractly as intelligent hosts, users, networks or collections of networks.

be propagated between users, nodes or networks to be effective. An important aspect of this work is identifying areas of network architecture amenable to statistical learning techniques.

1.2.1 For the Machine Learning Reader

Learning in systems design offers the potential to elegantly solve complex problems, yet is often impractical due to algorithmic speed, convergence, scalability or simply relative benefit. Many of the problems we face are non-traditional from an artificial intelligence perspective. In contrast to supervised learning in stationary environments, networks are highly dynamic and require continuous on-line learning. Often, artificial intelligence deals with single agents acting autonomously; networks by definition may need distributed learning. Thus, the intrinsic characteristics of the Internet suggest interesting challenges to machine learning.

An appealing feature of this thesis is in the novel application of learning methods to problems in networks. Through our examples and experiments, we find suitable algorithms, models and parameters. While we draw upon the available machine learning research, we contribute valuable insights into the performance of particular methods and their generality. Our hope is that this research serves as part of the valuable feedback loop in advancing learning research via practical application. In particular, §4.3 details an IP clustering algorithm we develop that incorporates network-specific knowledge to overcome several important domain challenges.

1.2.2 For the Network Architecture Reader

Network architects often speak of learning in the context of memorization, for example a routing protocol that "learns" a path to a destination. Instead, we emphasize our predictive notion of learning: the goal of a learning task is to generalize to unseen data or new situations.

Introducing learning into the network architecture is anathema to many purists. It is easy to improperly apply learning to problems we do not understand, or apply learning as a poor solution to particularly hard problems. If there is no underlying structure in the problem, no amount of learning is useful. For instance, consider building a model from names and phone numbers in the telephone book. Regardless of the learning algorithm, the model will not be able to predict an unknown person's phone number. Similarly, we do not advocate learning as a blanket solution, but instead incorporate domain and problem specific knowledge to solve many common, highly complex tasks. Thus, the system architect must understand when useful structure exists, but is too complex to exploit without learning.

A further source of discomfort for network builders is that learning methods imply moving to a probabilistic world without correctness guarantees. The benefit of implementing learning must exceed its cost. The learning machinery and models will make errors, in the form of bad predictions, incorrect classifications, etc. which may be realized as dropped packets, disconnected networks or other undesirable properties. However, we note that the original Internet architecture provides few deterministic guarantees – packets are silently discarded or reordered, routes change or disappear and the true original source of traffic cannot be ascertained. Our research therefore strives to give system designers guidelines as to when to apply learning.

We view the utility of learning in two different lights: i) as a performance optimization; and ii) when the network is under duress. As a means to improve performance, learning works well on tasks such as service selection, routing, etc. where failures and errors are not fatal. For instance, consider an agent picking a service from among a set of identical, replicated resources. An incorrect prediction, for instance a poor server, leads only to degraded service. The agent may thus try a prediction, receive an answer, and then refine its model. Additionally, there may be design tradeoffs where errors on low-priority traffic, users or applications are acceptable in the name of achieving better performance for high-priority services. Finally, and perhaps most importantly, errors are permissible when the network is under attack or duress. Affecting a small percentage of legitimate users is preferable to leaving an entire network incapacitated. The increasing complexity and sophistication of attacks, for example adaptable malicious attackers, thus *requires* learning and cooperation.

1.3 Relation to Prior Work

While there is a wealth of both network and machine learning research, there is relatively little work that combines the two. Throughout this multi-disciplinary thesis, we place our contributions in the context of related work. However in this subsection, we outline general research initiatives which apply learning to the network or Internet domain.

Unsolicited commercial email (spam) is simultaneously a great success story and failure for learning at the application layer to handle untrusted, unauthenticated messages. Spam filters [104] learn key features and characteristics of a users' desired email in order to discern unwanted email. Large email installations often aggregate feedback from all users as a form of collaborative filtering to better classify junk email [135]. Content filtering amounts to a classification problem in the domain of text retrieval. We similarly draw upon the rich literature of prior work in text categorization, e.g. [80], [157] and [156].

Intrusion detection is a rich space similar in spirit to spam classification: determining if traffic is malicious. Researchers actively debate whether it is feasible to classify "bad" traffic when "bad" is a subjective definition. In fact, an April Fool's Internet RFC [13] pokes fun at such schemes by proposing an "evil bit" in the IP header. However, the abundance of malicious attacks are driving commercial and research development toward ways to block traffic classes.

Some of the first consideration for cognitive networks are brought forth in [43] and [53]. An observation from their research is that users often have little information available as to the root cause of a network failure. With the Internet's many administrative domains and failure points, problems may span from an improperly configured computer to a backbone fiber cut to congestion on the remote server. Recently, Lee investigates distributed diagnosis and inference using Bayes' nets in [92] and [94]. Other approaches use reinforcement learning [96] to find and repair network faults.

Anomaly detection is the proactive dual of failure diagnosis and bears similarities to intrusion detection, provided intrusions are the exception rather than the norm. Ahmed et al. [4] use minimum volume sets to find traffic anomalies on the Abilene network while [90] use feature distributions to find anomalies. Jin et al. use heuristics [79] to determine behavior and intent of hosts appearing in IP "gray space."

Other researchers examine machine learning for passive traffic inference. Many modern applications do not use well-known port numbers [74] and are in effect "hidden." Further, the widespread deployment of filters has driven developers to use the web port (TCP port 80) for non-web applications. Offline, unsupervised clustering algorithms are thus useful to deduce running applications in packet traces [56]. With machine learning clustering on traffic features such as packet size, delay variation, etc. researchers have for example shown that a large amount of traffic is P2P [85].

The research in §4.2 uses regression methods to exploit the natural structure in the Internet address space. While our work is the first we are aware of to use this technique, other researchers have since followed a similar approach. Mirza et al. use support vector regression (SVR) to predict TCP throughput in [106].

1.4 Research and Thesis Overview

This thesis makes several broad observations on the current state of the Internet and trends likely to influence its future and evolution. Issues of scale, security, efficiency and complexity are prominent in our examination. From these observations, we posit learning as a key component in an increasingly complex and malicious Internet. Specifically, we take statistical learning as a class of algorithms to exploit the wealth of available, but under-utilized, information in the network. Chapter 2 provides an introduction to machine learning concepts used throughout.

Our research takes a bottom-up approach by applying learning in non-obvious ways to several real-world network problems, outlined in Table 1.1. The problems we consider span the design space continuum between upper and lower-layer network learning. In particular, we demonstrate the utility and practicality of learning i) at the application layer; ii) in end-nodes; and finally iii) within the core of the network. Our work progresses from simple, autonomous learners operating in a traditional supervised mode to increasingly complex models with distributed on-line learning, non-stationary environments, etc. Throughout the research we appeal to the end-to-end arguments to guide decisions of when and where

Table 1.1: Outline of Networking Problems Examined in this Thesis

Complexity ↓

Network Problem	**Learning Task**	**Learning Model**
Overlay Formation	Neighbor Selection	Application-layer, supervised classification (§3.2)
Service Selection	Latency Prediction	End-nodes, supervised regression (§4.2)
Email Attacks	Email Classification via Traffic Analysis	End-nodes, supervised classification (§4.4)
Spoofed Source Attacks	Accurate Classification	Distributed, ensemble of weak learners (§5.2)
Efficient Routing Policy	Constrained Optimization	Distributed, on-line learning (§5.3)

to place intelligent functionality.

Because learning implies creating a model based on observed phenomenon, our work involves a substantial measurement component. We ask questions such as what information is available to agents and how agents might obtain additional data to substantiate predictions and future decisions. Our measurements further validate the research beyond theoretical schemes, bridging the gap to operational systems.

Chapter 3 applies learning at the application layer to address the neighbor selection task in P2P overlays. We show that learning enables nodes to locate and attach to peers likely to answer future queries without a priori knowledge of the queries. Most importantly, we employ feature selection in this distributed setting to reduce communication cost, thereby removing one obstacle to realizing self-reorganizing overlays.

In Chapter 4, we create and test two operational end-host systems that make use of ignored data. We observe that Internet addressing is hierarchical, but discontinuous and fragmented, suggesting that learning can discover additional structure beyond what is available in routing tables or registries. We develop a network-specific clustering algorithm to find common partitions across the entire Internet address space. Using this clustering method, we endow agents with the ability to predict round-trip latencies to random Internet destinations without any network or coordinated support (§4.2). Further, we adapt our algorithm to accommodate structural and temporal dynamics.

In §4.4, we create a packet flow classification technique which detects traffic originating from remote, resource constrained hosts. This method provides the basis for "SpamFlow," a novel spam detection tool that relies on neither content nor reputation analysis. In addition to providing high spam classification accuracy, we detect the majority of false negatives missed by traditional content filters. By using learning to exploit a fundamental weaknesses in sourcing spam, we show that SpamFlow is adaptable and not easily subvertible.

We examine distributed learning within the network in Chapter 5. To ground our discussion in a practical and current threat, we apply learning to the IP source address validation problem. Current mitigation techniques are hampered by incentive issues; new attacks based on IP source forgery appear continually. Our work exploits learning to allow the eventual recipient to form a classification decision on incoming packets, thereby removing the incentive problem. Whereas existing techniques require global participation, we demonstrate that the network core has a sufficiently broad view to filter the majority of forged traffic with minimal collateral impact.

Section 5.3 takes routing as a final example of intelligence in the core. Because of the lack of separation between reachability and policy in Internet routing, operational networks are forced to drive behavior with low-level configuration. The resulting network is complex, fragile and prone to pathologies and non-obvious failures. Given that providers have a rich set of business goals, we propose an intelligent routing substrate that implements policy. Through simulation, we examine the role of learning in realizing such future routing plane architectures.

Chapter 6 summarizes our findings and key contributions. We provide guidelines for the practical application of learning in system problems and conclude by discussing the future of learning and networking.

1.5 Key Contributions

Learning provides a general framework for a wide class of problems that have, or likely will, arise in networking. Our research demonstrates viable learning-based approaches to several real-world and difficult networking tasks, but also allows us to draw insight on the larger conclusions and implications for learning within the Internet. Our key contributions include:

- Learning to optimize network performance
  - ◇ Demonstration of learning to address the neighbor selection problem in distributed P2P systems, revealing the power of feature selection to reduce communication complexity.
  - ◇ Application of learning to agent-centric latency prediction, giving insight into the Internet's IP address space structure.
  - ◇ Introduction of a new notion of QoS to resolve exploration vs. exploitation convergence issues in a dynamic network. To this end, we show the natural and strong link between such a routing infrastructure and learning.
- Learning to mitigate security threats
  - ◇ Development of "SpamFlow," illustrating novel means to capitalize on available, but unused information, exploit attack weaknesses and prevent evasion.

    - ◇ A machine learning-based IP source spoofing prevention mechanism that removes existing incentive and coordination issues.
    - ◇ Use of an ensemble of weak learners to perform distributed classification within the routing substrate. Our approach allows agents to combine and synthesize multiple weak agent votes into a more accurate validity assertion.
    - ◇ Demonstration of tuning parametric models to meet specific problem requirements, for instance biasing against false positives in a security context.

- Learning as a fundamental element in network architecture
    - ◇ Development of an IP-specific on-line, non-stationary clustering algorithm.
    - ◇ Guidelines for system's designers in the practical application of learning to Internet problems. Examination of distinct real-world problems and their decomposition to enable a statistical approach.

Beyond addressing current stressors and enabling new functionality, our hope is that this work provides a step toward an architecture that naturally accommodates future "tussles" and allows mediation within the system as part of its natural design. Thus, this thesis serves first to validate the potential for using learning methods to address several distinct problems on the Internet and second to illuminate design principles in building such intelligent systems in network architecture.

1.6 Bibliographic Notes

Parts of Chapter 3 appeared as *Machine Learning for Efficient Neighbor Selection in Unstructured P2P Networks*, Robert Beverly and Mike Afergan in the *Proceedings of the 2nd USENIX Tackling Computer Systems Problems with Machine Learning Techniques Workshop* [19]. Portions of work from Chapter 4 appeared as *SVM Learning of IP Address Structure for Latency Prediction*, Robert Beverly, Karen Sollins and Arthur Berger in the *ACM SIGCOMM Workshop on Mining Network Data* [25]. SpamFlow in Chapter 4 was published as a technical report [24] and is under current submission.

Bring new eyes to a world or even new lenses, and presto - new world.

- John Steinbeck

Chapter 2

Learning Background

Learning is a rich domain with active research in learning theory and application. A great deal of machine learning research focuses on "traditional" topics such as vision, path planning, natural language processing, pattern recognition and medical diagnosis. Machine learning has been applied with great success in financial markets for both prediction and fraud detection. Recently, biology has seen a renaissance in using machine learning for DNA sequencing and classification.

This chapter presents an overview of key machine learning concepts and methods used throughout the thesis. For each learning technique, we provide an example of the method applied to a networking problem. These examples are intended to relate techniques within the context of communication networks and Internet architecture and illustrate the relative advantages of different methods. Naturally, we cannot hope to provide a complete introduction to machine learning. Thus, we omit detail in favor of focusing on high-level concepts.

These algorithms perform well on problems we examine, but are by no means complete. Learning methods perform best with underlying domain knowledge. A natural component of our research is in understanding applicable techniques, models and parameters which suit a given networking problem in addition to domain-specific tailoring of these methods.

2.1 Types of Learning

What does it mean to "learn?" Avoiding the philosophical debate, this thesis defines learning on the basis of an agent or agents that act based on prior experience and available inputs. Most generally, learning implies an agent or actor observing events, possibly with delayed outcomes, building a model from observation and then forming predictions using the model. Within this framework are many subtle details and variations. Instances of these variations include whether the agent's observations are within its control, delay in receiving feedback, dynamics in the underlying environment, noisy feedback, etc.

Rather than simply memorizing outcomes, learning implies generality: the ability to form a rational prediction given inputs or environments not previously encountered. In the context of networking, a router is often said to ”learn” the route to a remote destination. The definition of learning in this thesis is stronger. The router is merely assimilating information and not forming predictions over the path to destinations for which is has no routing information. Similarly, a host does not learn the IP address of a web server, it queries the domain name system.

The inputs to the learner may be Boolean, discrete or continuous. Similarly, outputs are Boolean (classification)[1], discrete (ranking) or continuous (regression). Some examples of high-level learning tasks and how they might map to intelligence network behavior include:

1. *Classification*, e.g. the ability to classify and block malicious traffic, email or nodes. Typically a binary decision.

2. *Prediction*, e.g. the ability to infer network conditions, paths, etc. in order to optimize performance or meet constraints. Analogous to classification, but with real valued predictions.

3. *Class discovery*, e.g. the ability to determine logical groupings, clusters, etc for example to determine anomalous events.

4. *Reorganization*, e.g. the ability adapt and reconfigure the routing system, ad-hoc topology or overlay structure to optimize some metric of performance.

The performance of a learning agent is separated into structural and approximation error. While approximation error is error resulting from limited understanding of the underlying data, structural error is the result of a limited set of models. Put another way, structural error is the error we obtain with an infinite amount of training data, but a model unable to capture structure. Thus, choosing appropriate models and model classes is important in effective application of machine learning techniques. Three widely accepted divisions of learning are:

- *Unsupervised*: A model is built from observations where ground-truth is not available to the agent. The most common form of unsupervised learning is clustering. Clustering is one approach to estimate the prevalence of different applications on the Internet such as peer-to-peer, web, etc. from traffic communication patterns [85].

- *Supervised*: A model is built from observations on known data. The model balances fit to the known data in such a way as to attempt to ensure accurate predictions on new data. Common examples include voice and handwriting recognition.

- *Reinforcement*: A model is built from delayed observations to maximize a (discounted) long-term reward function. An example of reinforcement learning is a chess playing

[1] Multi-class classification may be reduced to multiple binary classifications, or computed directly.

agent that learns the outcome of a long series of decisions at the end of play rather than at each step of play.

This thesis focuses on the latter two techniques because they are most applicable to networking problems and it is possible to evaluate their performance. In contrast, unsupervised learning such as clustering is difficult to evaluate since it is generally impossible on the Internet to know an actual or correct partitioning[2]

2.2 Key Concepts

2.2.1 Notation

To discuss learning concepts and formulations, we adopt the following notation throughout the thesis. We denote vectors using bold fonts and matrices with bold capital variables. Let $\mathbf{x}_t \in \mathbb{R}^d$ represent the t'th instance of d-dimensional real data in a learning problem. Thus, n data points of dimension d may be compactly represented as the n-x-d matrix $\mathbf{X}$. With each data instance $\mathbf{x}_t$, there is an associated label y_t. For binary classification, labels are generally $y_t \in \{\pm 1\}$ while regression typically implies $y_t \in \mathbb{R}$. We denote predictions or estimations using a hat, e.g. $\hat{y}_t$.

In supervised learning, pairs of $(\mathbf{x}_t, y_t)$ training data are used to build a model. Supervised learning finds a function f, or parameters $\boldsymbol{\theta}$ for a fixed function, that maps new data instances to either discrete or continuous labels:

$$f : \mathbf{x}_t \in \mathbb{R}^d \rightarrow \hat{y}_t \tag{2.1}$$

The model f is then used to evaluate new and unseen data points where the label is not known. A common technique in building supervised learning models is to take a data set and split it into training and test points. The accuracy of the model is then evaluated against predictions on the test data, i.e. whether $\hat{y}_t = y_t$. The performance of a particular function may be evaluated in a number of ways. In binary classification, the common zero-one loss function is simply:

$$V(y_t, \hat{y}_t) = \frac{(1 - y_t\hat{y}_t)}{2} \tag{2.2}$$

such that a mistake on the t'th prediction induces one unit of loss while correct predictions have zero loss. In regression, loss is commonly absolute or square error:

$$V(y_t, \hat{y}_t) = (y_t + \hat{y}_t)^2 \tag{2.3}$$

We parameterize the prediction function f by $\boldsymbol{\theta}$. The learning task is then divided into finding "good" functions and optimizing over $\boldsymbol{\theta}$ in a principled manner. Throughout this

[2] One might use unsupervised learning in simulation where a repeatable experiment can be evaluated.

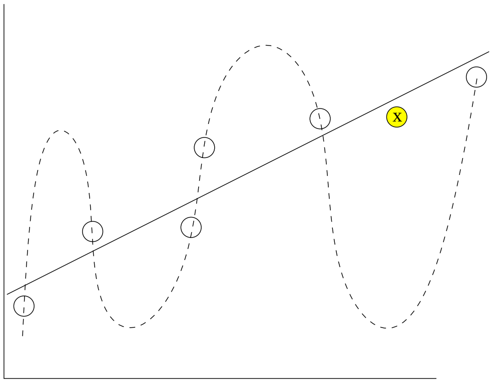


Figure 2-1: Over-fitting: A 4th degree polynomial produces zero training error, but performs poorly on unseen data. The linear solution in contrast generalizes well.

thesis we discuss suitable model selection, but note that learning attempts to find models that *generalize* well to new data instances the agent has yet to encounter.

2.2.2 Generality

A basic assumption in supervised learning is that finding a model which minimizes training error will produce a model that performs well on unseen (test) data. Such an assumption is reasonable if training and test samples are identically distributed and taken independently at random. To meet this criteria, we randomly permute our datasets in any supervised learning problem and evaluate the learning performance over multiple, independent trials.

However, minimizing training error alone may not ensure generality. Generality is a key concept in learning. The task of learning is not to mimic previous observations, i.e. memorization, but rather to find a model that is a suitable predictor of previously unseen data instances. Over-fitting is the classic symptom of a model that does not generalize well. Once the training error no longer approximates the test error, the model is said to overfit. Models that overfit to the input training data produce very low error in the training phase, but then subsequently perform poorly when forming predictions on new data points. For instance, an n degree polynomial can perfectly fit any n training points with zero error, but is likely to misclassify new data.

Figure 2-1 demonstrates an example of over-fitting. Here, empty circles represent training points. A 4th degree polynomial fits the training points with no error, but leads to over-fitting. A new data point x (filled circle) produces high error even though it is drawn from the same distribution as the training points. In contrast, the Figure shows that a linear solution, generalizes well in this example well despite higher training error. A prediction using the straight line regression on the new data point is more accurate than the over-fitting model.

A good test of whether a model will generalize well is to determine its performance under minor perturbations. For example, the model should not change drastically if one of the training inputs is removed or modified. The stability of the model under these small changes makes intuitive sense as a single data point that changes the overall solution will be unlikely to perform well on unseen data. A common test of generality is cross validation or leave-one-out validation. In leave-one-out validation, the learning algorithm successively picks a single training point to remove. The overall model is then built from an average estimation of the cross-validation.

Thus, learning algorithms often attempt to balance the empirical performance of a solution with the model complexity. This tradeoff is known as regularization.

2.2.3 Bayesian Inference

In many contexts, obtaining causal probabilities is easier than diagnostic probabilities. A classic example is in medical diagnosis where doctors can approximate the probability of a symptom given a disease (causal), but cannot estimate the probability of a disease given a symptom (diagnostic). Bayes' rule provides a convenient mechanism to obtain the posterior probability of a hypothesis $\hat{y}$ from causal data $\mathbf{X}$:

$$P(\hat{y}|\mathbf{X}) = \frac{P(\mathbf{X}|\hat{y})P(\mathbf{X})}{P(\hat{y})} \tag{2.4}$$

We often wish to evaluate the probability of multiple hypotheses $\hat{y}^i$ given some observed data $\mathbf{X}$, i.e. $P(\hat{y}^i|\mathbf{X})\forall i$. In the binary case, for example, there are two potential hypotheses $\hat{y}^1 = 1$ and $\hat{y}^2 = -1$. A prediction is made using all of the hypotheses weighted by their probabilities. With underlying knowledge of the data given a particular hypothesis $(P(\mathbf{X}|\hat{y}^i))$, Bayes' rule gives the most likely hypothesis. We employ the common simplifying strategy of assuming that each piece of data $\mathbf{x}_j \in \mathbf{X}$ is statistically independent to obtain the naïve Bayesian classifier:

$$P(\hat{y}^i|\mathbf{X}) = \frac{\prod_j P(\mathbf{x}_j|\hat{y}^i)P(\hat{y}^i)}{P(\mathbf{X})} \tag{2.5}$$

While the data is typically not independent, the causal data is often independent given the hypothesis in realistic scenarios. Naïve classifiers of this sort work surprising well in practice [91]. Note that for classification, the denominator $P(\mathbf{X})$ is the same for each hypothesis. Therefore the observed data prior $P(\mathbf{X})$, which may be difficult to obtain, does not need to be explicitly computed for comparisons. Such classifiers have the advantage of producing a maximum-likelihood guess along with a degree of confidence even in the fact of incomplete or conflicting data.

Consider the problem of predicting whether the `cnn.com` web site is reachable[3] as depicted in Figure 2-2. There may be a multitude of reasons why the CNN web site is down from the perspective of a particular network. Given knowledge of congestion, connectivity

[3] See also our work on robust TCP/IP fingerprinting [17] as an additional application of a naïve Bayesian classifier for network inference.

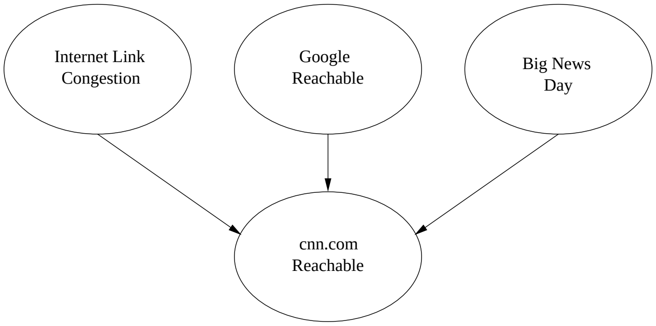


Figure 2-2: One representation of a network reachability inference problem as a naïve Bayesian classifier.

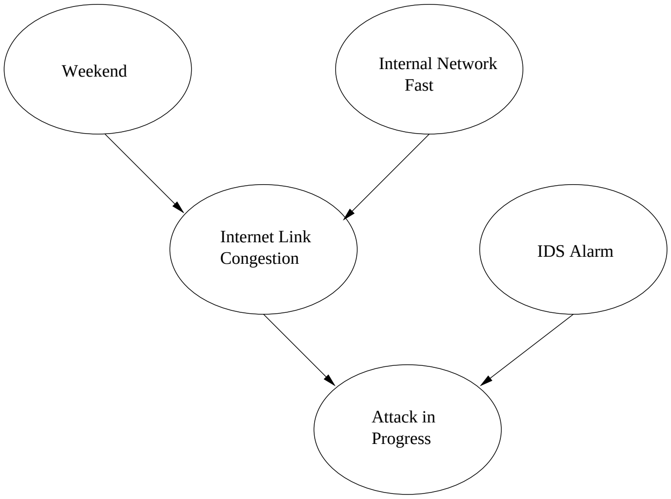


Figure 2-3: Bayesian network representation of a denial-of-service attack inference problem

to other sites and whether there is a major news announcement, the agent forms a belief on the ability to reach `cnn.com`. In this naïve formulation, each of the causal pieces of knowledge are assumed to be independent. This graphical model is a special case of Bayesian networks.

2.2.4 Bayesian Networks

Bayesian networks, or simply Bayes' nets are probabilistic graphical models to efficiently represent a full joint probability distribution by exploiting independence. A Bayes' net is a directed acyclic graph where nodes represent variables and edges represent dependence. Figure 2-3 shows a toy example of a Bayes' net representation of whether a denial-of-service attack is in progress.

This example considers discrete Boolean variables, but continuous Bayes' nets are just as feasible. The directed edges indicate dependence; in this case "Internet link congestion" depends on its two parent nodes: "weekend" and "internal network fast." The network captures independence. For example, "weekend" and "IDS alarm" are independent, demon-

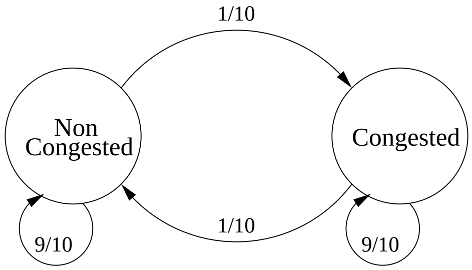


Figure 2-4: Hidden Markov Model for inferring congestion state

strating how a designer's assumptions and understanding of the underlying problem domain shape the network[4]. The net also represents marginal independence. "Internet link congestion" and "IDS alarm" are independent given nothing is known about "attack in progress." However, if the learner determines that the network is under attack, the two are dependent.

The important property of Bayes' nets is the ability to compactly represent the full joint probability table, in this case $P(weekend, internalfast, congestion, IDS, attack)$. In general, with n Boolean variables, the joint probability distribution requires 2^n values, or 32 entries. For large problems, exponential scaling with the number of variables is not feasible and hence the importance of identifying and exploiting independence in the problem. In our example, only $1 + 1 + 4 + 1 + 4 = 11$ probabilities need to be stored in order to answer any question.

2.2.5 Hidden Markov Models

Hidden Markov Models (HMMs) allow us to tackle learning problems involving dynamics, in particular reasoning over time or states. Environments amenable to HMMs are those with partial or noisy precepts and changing states. A canonical example where HMMs have been applied with good success is in speech recognition systems.

HMMs have a set of unobservable states (Q), transition probabilities between states (A), an alphabet of symbols (Σ) and probabilities that a particular symbol $\sigma \in \Sigma$ is emitted in state $q \in Q$. To build an HMM, the environment is sliced into representative discrete units. In order to make the problem tractable, HMMs include the Markov assumption which simply says that the current state depends only on a finite number of previous states. For instance, a first order Markov model has states where the current state depends only on the previous state (has a single parent).

Whether the Markov assumption is reasonable depends on the domain, but often provides a reasonable means to approximate the solution as physical processes often have independence between previous and future states.

With an HMM, several inferences are possible. Given a time series of observed evidence, one may determine the most likely explanation of the set of hidden states which generated

[4] It is also possible to learn the best Bayes' net, but we do not present this variation here.

Table 2.1: HMM symbol emission probabilities

	ACK	DupACK	OOAck	SACK
C	0.7	0.1	0.1	0.1
NC	0.97	0.01	0.01	0.01

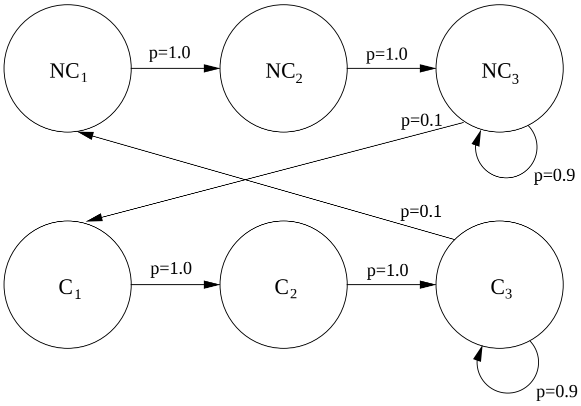


Figure 2-5: HMM with additional congestion state assumptions

the symbols. Alternatively, the HMM can predict the current or future state given the evidence.

As a simple illustrative example of HMMs applied to a network problem, consider a passive observer attempting to determine whether an end-to-end Internet connection is congested. Researchers often wish to make inferences from passive observation, such as passive traces collected from an Internet link. For example, Liu et al. use HMMs for a similar inference problem in hybrid wired/wireless networks in [97]. From such passively collected traces, the observer has no insight into the true states of the end TCP stacks; these correspond to the hidden states.

We will model the congestion inference problem as the HMM in Figure 2-4. The system has only two hidden states, congested or non-congested ($Q = \{C, NC\}$). Let the alphabet be $\Sigma = \{ACK, DupACK, OOAck, SACK\}$ corresponding respectively to evidence of normal, duplicate, out-of-order and selective TCP acknowledgments. For the sake of this example, we assume that the HMM emits symbols according to the probabilities in Table 2.1.

In this model, a TCP stack stays in its current state with probability 0.9 and switches to a different state with probability 0.1. Given evidence $x_i \in \Sigma$ over time steps $i = 0, ..., n$, we can use the HMM to determine what the most likely sequence of states the TCP connection encountered, infer the current state or predict the next state.

The congestion HMM is quite simple and assumes that the transition probabilities are the same in any time step. While this approximation may suffice, a more accurate model might model temporal congestion properties by assuming that a congested connection remains congested for at least two time steps. Similarly, Figure 2-5 shows that a non-congested link remains non-congested for at least two time steps.

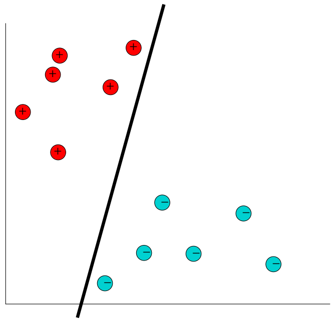


Figure 2-6: A linear binary classifier on separable data

2.2.6 Linear Binary Classifiers

A common learning task is to differentiate between a finite number of classes. In linear binary classification, the task is find $\boldsymbol{\theta}$ and θ_0 that maps input instances $\mathbf{x} \in \mathbb{R}^d$ to binary labels:

$$f(\mathbf{x}; \boldsymbol{\theta}, \theta_0) = \operatorname{sign}(\boldsymbol{\theta}^T \mathbf{x} + \theta_0) \tag{2.6}$$

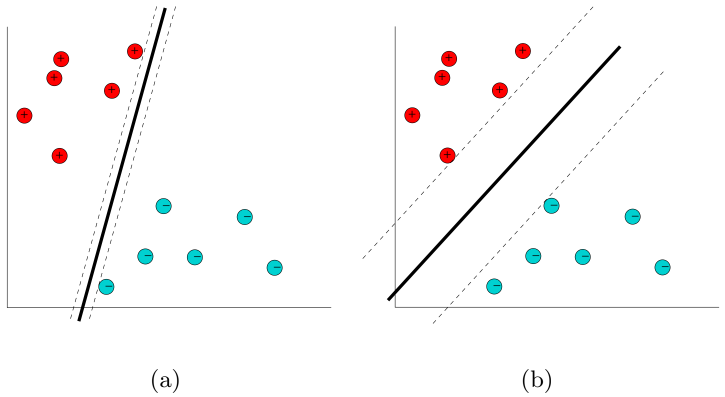
Geometrically, $\boldsymbol{\theta}^T \mathbf{x} + \theta_0 = 0$ defines a d dimensional hyper-plane decision boundary. If there exists a decision boundary such that all data points are classified with zero loss, the points are separable. Figure 2-6 depicts positive and negatively labeled two-dimensional data points along with a separating decision boundary. Positive samples are above the decision boundary where $y_t \left(\boldsymbol{\theta}^T \mathbf{x}_t + \theta_0\right) > 0$.

Note that if the points are linearly separable, there are an infinite number of possible decision boundaries. To select a solution, the notion of margin allows us to incorporate a model generality. Consider normal projections from the decision boundary to each data point. Informally, the minimum distance normal projection defines the margin.

Figure 2-7 shows the same data set with two different decision boundaries. The dashed lines indicate the margin. Each boundary provides a suitable decision function. However, the margin in Figure 2-7(b) is larger than the margin in Figure 2-7(a). Assuming that new, unclassified data points come from the same distribution on which the decision boundary is found, maximizing the minimum margin ensures generality. To see this, note that in Figure 2-7(a), a new data point next to the data point defining the minimum margin is likely to be misclassified. In contrast, the same new data point is not misclassified with a decision boundary that maximizes the minimum margin. Support Vector Machines (SVM) [146] attempt to find models by maximizing the minimum margin directly.

2.2.7 Support Vector Machines

SVM classifiers find an optimal separating hyperplane that maximizes the margin between two classes of data points. The hyperplane separator is orthogonal to the shortest line



(a) (b)

Figure 2-7: Selection of decision boundary (solid) affects margin (dashed)

between the convex hulls of the two classes. Because this separator is defined only on the basis of the closest points on the convex hull, SVMs generalize well to unseen data. These closet points, those that define the margin, are termed "support vectors." Additional data points do not affect the final solution unless they redefine the margin. The primal form of an SVM optimization is:

$$\text{minimize } \frac{1}{2}\|\boldsymbol{\theta}\|^2 \text{ s.t. } y_t\left(\boldsymbol{\theta}^T\mathbf{x}_t + \theta_0\right) \geq 1 \; \forall \; t = 1 \ldots n \tag{2.7}$$

To prevent over-fitting and balance between minimizing loss and model generality, SVMs also include regularization. Regularization also allows the algorithm designer to include domain-specific knowledge into the learning algorithm to penalize or encourage particular properties. The more general primal form of Eq. 2.7 is:

$$\min \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{t=1}^{n} V(y_t, \hat{y}_t) \text{ s.t. } y_t\left(\boldsymbol{\theta}^T\mathbf{x}_t + \theta_0\right) \geq 1 - V(y_t, \hat{y}_t) \; \forall \; t = 1 \ldots n \tag{2.8}$$

where C controls the tradeoff. Large values of C imply that the final solution places a strong burden on incorrect solutions at the cost of more complex models, while small C allow more errors. The constraints can be subsumed into the minimization problem using Lagrange multipliers to give the SVM dual form:

$$J(\boldsymbol{\theta}^*, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}\geq 0} \|\boldsymbol{\theta}\| - \sum_{t=1}^{n} \alpha_t \left[y_t\left(\boldsymbol{\theta}^T\mathbf{x}_t + \theta_0\right) - 1\right] \tag{2.9}$$

which is solved by obtaining $\boldsymbol{\theta}^*$ as a function of $\boldsymbol{\alpha}$ and then maximizing over the Lagrange multipliers. It can be shown that the dual form is then to maximize:

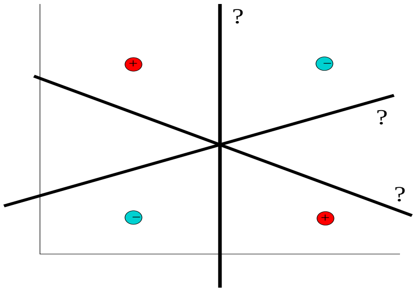


Figure 2-8: Separating XOR inputs. No linear separator exists that correctly classifies all four points with zero error.

$$\sum_{t=1}^{n} \alpha_t - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \text{ s.t. } C \geq \alpha_t \geq 0, \sum_{t=1}^{n} \alpha_t y_t = 0 \tag{2.10}$$

2.2.8 Kernels and Non-Linearity

Many real-world problems include data that is not linearly separable. A canonical example that illustrates the problem well are the four XOR[5] data points in Figure 2-8. No linear separator exists that correctly classifies all four points with zero error.

However, the data is often linearly separable once it has been transformed from its original input space to a new, higher-dimensional, feature space. Let $\boldsymbol{\phi}(\mathbf{x})$ be a transformation on the input space to a feature space. Particular instances of transformations known as kernels have important mathematical properties.

Thus, while the separator may be linear in some higher-dimensional feature space, it need not be linear in the input space. For example, the second degree polynomial kernel for a two dimensional input space $\mathbf{x} = [x_1, x_2]$ is:

$$\boldsymbol{\phi}(\mathbf{x}) = [1, x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2] \tag{2.11}$$

Notice that the inner product of this second degree polynomial $\boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{x'})$ is $(1 + (\mathbf{x}^T\mathbf{x}'))^2$. The inner product in the feature space does not need to be computed, a feature known as the "kernel trick." The kernel trick replaces the dot products $\mathbf{K}(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. By Mercer's theorem, any positive, semi-definite kernel can be expressed as a dot product in a high-dimensional space.

The transformation to a higher-dimensional space could easily lead to over fitting, but is prevented by the regularization penalty. The SVM formulation on the feature space is then a similar minimization problem as Eq.2.10. Thus, for appropriate kernels, the maximization problem becomes:

[5] XOR refers to the Boolean exclusive or function

$$\sum_{t=1}^{n}\alpha_t - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{K}(\phi(x_i),\phi(x_j)) \text{ s.t. } C \geq \alpha_t \geq 0, \sum_{t=1}^{n}\alpha_t y_t = 0 \tag{2.12}$$

2.2.9 Feature Selection

Feature selection finds a subset of all features which provide the most discrimination power [157]. The first feature selection method we consider is mutual information (MI). Mutual information attempts to use combinations of feature probabilities to assess how much information each feature, i.e. word token, contains about the classification. MI evaluates the mutual information score $I(\theta_i; y)$ for each feature $\theta_i \in \hat{\mathbf{X}}$ and sequentially picks the highest scoring features independently of the classifier. The MI score is in the range $[0, 1]$ and will be zero if the feature is completely independent of the label or one if they are deterministically related.

$$I(\theta_i; y) = \sum_{\theta_i \in \{0,1\}} \sum_{y \in \{\pm 1\}} \hat{P}(\theta_i, y) \log_2 \frac{\hat{P}(\theta_i, y)}{\hat{P}(y)\hat{P}(\theta_i)} \tag{2.13}$$

Secondly, we use greedy forward fitting (FF) feature selection. Forward fitting feature selection simply finds, in succession, the next single feature that minimizes training error. Therefore, training error decreases monotonically with the number of features. Feature selection proceeds in rounds. For an error function $V(f(\boldsymbol{\theta}), \cdot)$, find the next feature θ_i from the remaining features not previously selected in $X_1, \ldots, X_{i-1}$. Thus, we evaluate each potential feature i for the fixed set of $i-1$ features. Formally:

$$\theta_i \leftarrow \underset{j}{\operatorname{argmin}}\, V(f(\hat{\mathbf{X}}, x_j), \mathbf{y}) \forall x_j \notin X_1, \ldots, X_{i-1} \tag{2.14}$$

We can also express feature selection more precisely using matrix notation. In round i let $\mathbf{S}^j$ be an f x i binary selection matrix with $s_{j,i} = 1$ and $s_{k \neq j, i} = 0$. The $i-1$ columns of $\mathbf{S}^j$ are set in previous rounds. Recall that the data $\mathbf{X}$ is an n x f matrix containing n examples with f features each. Let:

$$\mathbf{Z}^j = \mathbf{X}\mathbf{S}^j \tag{2.15}$$

Thus, $\mathbf{S}^j$ selects feature j in round i. Let $F = \{1 \ldots f\}$ indicate the set of all possible features. We denote θ^i as the set of best features in round i. Then, for a prediction function $f(\cdot)$ and error function $V(\cdot)$, find:

$$\underset{j \in F - \theta^{i-1}}{\operatorname{argmax}}\; V(f(\mathbf{Z}^j), \mathbf{Y}) \tag{2.16}$$

The feature that best minimizes the error in round i is j, so we update the set of best features: $\theta^i = \theta^{i-1} + j$. Training error is typically an effective proxy for test error. One can employ, for instance, SVM accuracy as the error function $f(\cdot)$, although forward fitting can be used with any model and error function.

In some contexts it may be most important only to discover good features while in others it may be important to understand the relationship between them. Forward fitting is computationally expensive because it requires computing a combinatorial number of possible feature combinations. Mutual information is much faster, but does not always find the optimal features. A weakness of MI is that it may choose two features that are themselves closely dependent and thus the second feature provides little additional classification power. In contrast, forward fitting will continually seek the next best performing feature without this potential dependence.

2.3 Key Terms

- *Agent:* a logical model or abstraction that describes software that acts. An agent may act on behalf of a user or program and may interact with another user or other program.

- *Over-fitting:* Developing a complex model that yields excellent predictions on the input training data, but generalizes poorly to unseen data.

- *Regularization:* A penalization parameter against model complexity in order to enable generality.

- *Stability:* A metric of generalization whereby the derived model does not change significantly by perturbing the training set, i.e. by removing a training example.

- *Input space:* raw data points represented as d dimensional vectors $\mathbf{x} \in \mathbb{R}^d$.

- *Feature space:* a higher dimensional space $f > d$ in which data points are linearly separable.

- *Kernel:* A mapping from an input space to a feature space that allows a linear classifier to solve a non-separable problem in the input space. By using kernels, the inner product of feature spaces does not need to be explicitly computed.

- *True/false positive/negative:* A positive is a correct prediction $\hat{y}_t = y_t$. A true positive (TP) is a correct prediction where the label is 1, i.e. $\hat{y}_t = y_t = 1$. True negatives (TN) are correct predictions where $\hat{y}_t = y_t = -1$. A false positive (FP, also known as a type I error) is an incorrect prediction of 1, i.e. $\hat{y}_t = 1 \neq y_t$. A false negative (FN), $\hat{y}_t = -1 \neq y_t$ is also known as a type II error.

- *Accuracy:* is simply the ratio of correctly classified test samples to total samples. Let P be the number of positive samples in the data set ($P = TP + FN$) and N the number of negative samples, ($N = TN + FP$).

$$Accuracy = \frac{TP + TN}{P + N} \tag{2.17}$$

- *Precision:* measures the number of positive predictions that were truly positive.

$$Precision = \frac{TP}{TP + FP} \tag{2.18}$$

- *Recall:* synonymous with sensitivity and true positive rate. Recall provides a metric of how many positive samples were predicted positive. All three measures are important in assessing performance; for instance, a model may have high accuracy, but poor recall.

$$Recall = \frac{TP}{P} \tag{2.19}$$

- *Specificity:* determines how well the classifier identifies negative cases. The false positive rate is $1 - specificity$.

$$Specificity = \frac{TN}{FP + TN} \tag{2.20}$$

- *Receiver Operating Characteristic:* a plot that compares the false positive rate versus the true positive rate as a function of the model parameters. Provides a graphical and intuitive means to understand how parameters affect performance.

2.4 Summary

Unfortunately, there are no hard and fast rules as to which learning technique is superior. Learning performance is typically domain, model and parameter dependent. The inputs in many applications are sufficiently complex and not well-understood such that different models perform differently based on their ability to extract useful causal, dependence and generality information from the data. Further, the domain is often characterized by randomness and marked by theoretical and practical uncertainty, i.e. the domain is not completely understood and practical details may prohibit complete understanding.

Therefore, any bootstrapping or inclusion of domain specific information within the algorithm can aid performance significantly. For example, we consider an IP-specific clustering algorithm in §4.3. Throughout this thesis, we seek to understand the models and parameters that lend themselves to networking problems. Table 2.2 provides a summary of techniques described in this Chapter along with the relative strengths and weaknesses as they relate to networking problems.

Table 2.2: Summary of learning methods considered

Technique	**Domain**	**Network Example**	**Strengths**	**Weaknesses**
Naïve Bayes	Probabilistic classifier	Spam filter	Produces most likely estimation. Easy to train, estimate each distribution independently.	Strong independence assumption
Bayes' Net	Probabilistic inference	Attack detection	Captures independence. Produces probabilistic output.	Difficult to determine network structure. Can't exploit missed causal connections
SVM/SVR	Large data sets containing many variables with unknown distribution	Latency prediction	Generalizes well by penalizing model complexity. Easily allows non linear input features to be transformed into higher dimension where separable.	Difficult to determine kernel function. Can be computationally intensive.
HMM	Dynamic, time-series	TCP congestion inference	Incorporates non-stationarity	Markovian assumption
Mutual Information	Determine correlation between features independent of classifier	Neighbor selection	Simple, computationally fast	Performs worse than forward fitting
Forward Fitting	Greedily determine best features using classifier	Neighbor selection	Avoids duplicate features that provide no additional discrimination	Computationally expensive. Potential for over-fitting.

At the first balloon flight, Ben Franklin was asked "What good could a balloon be?" to which he replied: "What good is a newborn baby?"

Chapter 3

Application-Layer Learning

In this Chapter, we follow a traditional application-layer perspective on placing learning and intelligent functionality. Specifically, we consider unstructured Peer-to-Peer (P2P) networks as a nice example of applications built on top of the Internet to permit new functionality. In P2P overlays, nodes act as both producers and consumers of data, effectively filling the role of servers and clients simultaneously. Further, many P2P overlays provide the ability to search for content in the network. Thus, P2P networks provide functionality at the application layer in the absence of that functionality in the network itself.

Because of the separation between the application layer's needs and the network's support, applications must make inferences and will often attempt to optimize local or approximate global metrics. Thus, learning is a natural fit for many newer applications that organize such overlays. This Chapter shows that there is ample room for networked applications to use learning to their advantage.

3.1 Introduction

Previous research has considered the concept of network self-reorganization, for instance in mobile and P2P environments. We formulate the neighbor selection task in P2P networks as a machine learning problem. Specifically, how a node can effectively (with high success) and efficiently (with few queries) determine the suitability of another node. Using captured file sharing data from a live P2P network, we demonstrate that nodes equipped with suitable prediction mechanisms can organize for both local and global performance improvements. We use forwarding fitting feature selection in conjunction with Support Vector Machines (SVMs), to predict suitable neighbors with over 90% accuracy while requiring minimal queries (<2% of features). Our technique efficiently chooses neighbors that successfully answer not only current, but also future queries.

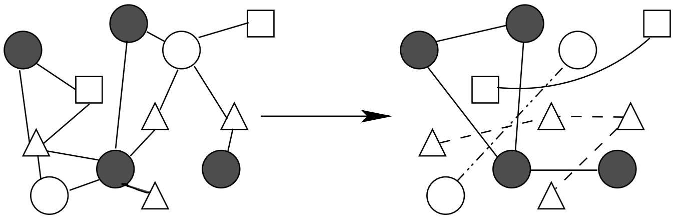


Figure 3-1: Self-reorganization within a peer-to-peer overlay. Nodes organically connect and may optimize their connections based on utility, e.g. connecting to nodes with similar interests, low load, etc.

3.2 Problem: Overlay Neighbor Selection

Simple unstructured Peer-to-Peer (P2P) overlay networks are both popular and widely deployed [86, 126]. Nodes issue queries, for example file keywords, that propagate through the overlay and receive answers from peers able to satisfy the query. Because unstructured overlays allow nodes to interconnect organically with minimal constraints, they are well-suited to self-reorganization. Specifically, prior research investigates reorganization for improved query recall, efficiency and speed, as well as increased system scalability and resilience [9, 18, 37, 142, 144]. Figure 3-1 graphically depicts the basic intuition behind self-reorganization within an overlay where nodes with common interest or type preferentially interconnect.

In practice however, the benefit of reorganization is often lower than the cost in a classic exploration versus exploitation paradox. A critical question prior research does not address is how nodes within a self-reorganizing P2P system can determine the suitability of another node, and hence whether or not to connect, in real-time. Nodes must classify potential peers as good or poor attachment points both effectively (with high success) and efficiently (with few queries). This task is the overlay neighbor selection problem.

Given an omniscient oracle able to determine a node's future queries and the fraction of those queries matched by other nodes, neighbor selection is readily realizable. Naturally, nodes do not have access to such an oracle. This work seeks to understand how to emulate, in a distributed fashion with minimum communication cost, the functionality of an online oracle as a component of a self-reorganizing network. In particular, nodes within a self-reorganizing network face two primary challenges: minimizing load induced on other network participants (exploration) and locating neighbors that are likely to answer *future* queries (exploitation).

We abstract these challenges into a distributed, uncoordinated, per-node machine learning classification task. Based on minimal queries, nodes predict whether or not to connect to a peer neighbor node. A key insight is that minimizing the induced load on other nodes maps to a feature selection problem. We wish to build an effective classification model by finding a small set of highly discriminatory queries.

Table 3.1: Gnutella datasets

DataSet	Nodes	Contains
Beverly, et al.	1,500	Queries, Files, Timestamps
Goh, et al.	4,500	Queries, Files, Timestamps

Our analysis uses live Gnutella [62] data to understand the efficacy and parameterization of machine learning techniques for neighbor selection. We determine the accuracy, precision and recall performance of SVMs for this task and empirically determine a training size with high prediction performance. We then experiment with forward fitting and mutual information feature selection algorithms as a means of finding queries with high discrimination power.

3.2.1 Learning Approach

Locating peer nodes in unstructured P2P networks is accomplished in a variety of ways. In Gnutella-like overlays, bootstrap nodes maintain pointers into the network while every node advertises the IP addresses of its neighbors. However, it is not obvious how a node can, in real-time, determine whether or not to connect to another node. Exploring other nodes incurs cost and presents a paradox: *the only way to learn about another node is to issue queries, but issuing queries makes a node less desirable and the system less scalable.*

A key insight of our research is that efficient neighbor selection maps to machine learning feature selection. We ask: "for a node i, does there exist an efficient method, i.e. a small set of key features, by which i can optimize its choice of neighbors?" While feature selection is traditionally used in machine learning to reduce the computational complexity of performing classification, we use it in a novel way. Specifically, we use feature selection to *minimize the number of queries*, which equates to network traffic and induced query load.

Representing the Dataset

We experiment on real, live Gnutella datasets from two independent sources: our own measurement of 1,500 nodes and a public repository of approximately 4,500 nodes from Goh, et al. [63]. Table 3.1 describes the data. Both datasets include timestamped queries and files offered across all nodes. Encouragingly, our experiments yield similar results using either dataset, lending additional credence to our methodology. We consider only the Goh dataset here.

While our datasets focus on Gnutella, we believe that they are sufficiently representative of general P2P usage (in particular, the Gnutella network is estimated to contain approximately 3.5M users [126]). It is reasonable to believe that most general-purpose networks will see comparable usage patterns. Many additional motivations for using Gnutella as a reference P2P network, including size, scope and growth, are given in [143].

Let N be the set of all nodes in the data set and $n = |N|$. Associated with each node is a list of queries it issues and the names of files it shares. For example, the first query of the first node is "`ann her lee hate womack i`" while the first file stored on the first node is "`The Doors - Light My Fire.mp3`." Unstructured P2P file sharing overlays typically perform tokenization where the query and file name strings are separated on white-space. From the resulting tokens, non-alphanumerics, stop-words and short words are removed. We similarly follow this tokenization procedure according to the Gnutella protocol.

Let $\mathbf{q}_{i,k}$ and $\mathbf{f}_{i,k}$ be vectors representing the k'th tokenized query and file of node i respectively. Thus, $\mathbf{q}_{1,1} = \{ann, lee, hate, womack\}$ and $\mathbf{f}_{1,1} = \{doors, light, fire\}$. We index the r'th token using array notation, thus $q_{i,k}[r]$ is the r'th token of the k'th query from node i, e.g. $q_{1,1}[2] = lee$. The set of all query and file tokens for node i is the union of the tokens of i's individual queries and files:

$$\mathbf{q}_i = \bigcup_k \mathbf{q}_{i,k} \tag{3.1}$$

$$\mathbf{f}_i = \bigcup_k \mathbf{f}_{i,k} \tag{3.2}$$

We represent the set of all unique file tokens as $\mathbf{F} = \bigcup \mathbf{f}_i$.

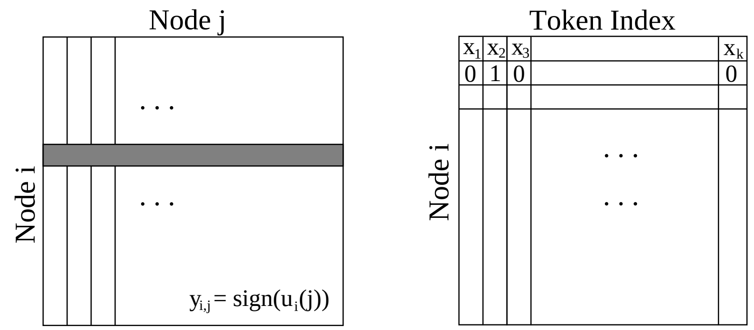
To qualitatively compare peers, we introduce the notion of a utility function. Given $\mathbf{q}_{i,k}$ and $\mathbf{f}_{i,k}$ for every node i, we can evaluate whether a potential neighbor has positive utility. Nodes are individual, selfish utility maximizers, i.e. their decisions reflect a desire to increase utility without regard for global welfare. Utility may be a function of many variables including induced query load, query success rate, etc. However, we are primarily interested not in the specific mechanics of a utility-based self-reorganizing network, but rather the neighbor selection task. Therefore, in this work, we define $u_i(j)$, node i's utility in connecting to node j, simply as the number of successful queries from i matched by j. A single query from i matches a single file held by j if and only if all of the query tokens are present in that file name[1]. For example, the first node would not match the two token query "`doors fire`," but would match "`doors fire light`." Formally:

$$u_{i,k}(j) = \begin{cases} 1 & \text{if } \forall r \exists l \text{ s.t. } q_{i,k}[r] \in \mathbf{f}_{j,l} \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

and

$$u_i(j) = \sum^{k} u_{i,k}(j) \tag{3.4}$$

[1] We also used more sophisticated decreasing marginal utility functions that consider both query matches and induced system load along with an ϵ-equilibrium analysis for non-strict utility maximizers, but omit results here.



(a) Adjacency Matrix (b) File Store Matrix

Figure 3-2: (a) The binary n-x-n adjacency matrix indicates whether node i wishes to connect to node j based on utility $u_i(j)$. (b) We assign a unique index k to all file store tokens and form a Boolean per-node word token presence matrix $\mathbf{X}$.

We represent the dataset with the two matrices in Figure 3-2, an adjacency and word token matrix:

- *Adjacency Matrix* $\mathbf{Y}$: An n-x-n pair-wise connectivity matrix where $\mathbf{Y}_{i,j} = \text{sign}\,(u_i(j))$. Because our dataset includes all queries and files of every node, our omniscient simulator definitively knows how many queries of each node are matched by every other peer. Thus, $\mathbf{Y}_{i,j} = +1$ indicates that node i wants to connect to node j.

- *File Store Matrix* $\mathbf{X}$: Using all file store tokens, F, we assign each token a unique index k. The word token Boolean matrix indicates the presence or absence of a given file token for every node in the system. $\mathbf{X}_{i,j} = 1 \iff F_j \in \mathbf{f}_i$.

From the adjacency and file store matrices we create a per-node matrix, $[\mathbf{Y}(i,:)^T, \mathbf{X}]$, as shown in Figure 3-3. Note that we compute the adjacency and file store token matrices explicitly only in order to evaluate the performance of our neighbor selection algorithm; our scheme **does not** require complete knowledge of all files and queries in the network. Rather, we employ the omniscient oracle only to evaluate the performance of our neighbor prediction algorithm.

Formulating the Learning Task

Given the hypothetical off-line (oracle) representation of a node's state as depicted in Figure 3-3, we now turn to the problem of classification. Note that the problem we face is slightly non-standard – we have a separate classification problem for each node. That is, the features that are optimal can, and likely will, be different from node to node. In addition, the optimal features need not match the node's queries. For instance, while a node may issue queries for "lord of the rings," the single best selective feature might be "elves." This example provides some basic intuition of how our system finds peers that are capable

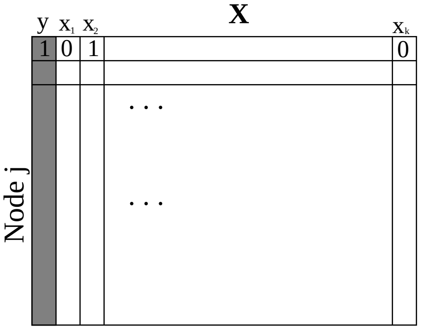


Figure 3-3: Representing the *single node i*: The i'th row of the adjacency matrix (fig 3-2a) is the first column (shaded) and represents node i's connection preferences (class labels). To this the file store token matrix ($\mathbf{X}$) is horizontally concatenated.

of answering future queries. By connecting to peers using "elves" as a selection criterion, a future query for "the two towers" is likely to succeed given interest-based locality.

Figure 3-4 shows how the conjoined matrices as shown in Figure 3-3 are split and used as input to machine learning algorithms. Some number of nodes, significantly fewer than the total number of nodes n, are selected at random to serve as training samples. The learner is given the word token features present for each training node ($\hat{\mathbf{X}}$ a row subset of $\mathbf{X}$) along with the corresponding classification labels ($\mathbf{y}$). For our neighbor selection task, the label is a binary decision variable, $y \in \{\pm 1\}$ where $y = +1$ indicates a good connection and $y = -1$ a poor connection. The class labels are computed by calculating each node's utility as described previously. We consider the size and complexion of the training data in the next Section. Using the training data, the learner develops a model that uses a small number of features $\boldsymbol{\theta} \in \hat{\mathbf{X}}$ in order to predict future connection decisions. We evaluate the efficacy of this model against the test data, i.e. whether the model correctly predicts the unknown y connection labels in the test set. Thus, in the testing phase, the input to the classifier is the small number features $(\theta_1, \cdots, \theta_d)$ where $d \ll k$, without either the labels (y) or the full word tokens $(x_1, \cdots, x_k)$.

Notice that the features we train and test on do not include any queries Q from our dataset. Only the y labels depend on the queries. Thus, successful connection predictions imply the ability to predict queries *yet to be asked.*[2]

We find that some nodes in the dataset have queries that are matched by very few other nodes. Therefore, a prediction model that deterministically predicts not to connect will yield a high accuracy – and thus a misleading result. Consider a node whose query is matched by only 1 of 500 other nodes. A classifier that always predicts not to connect gives an accuracy of 499/500 = 99.8%. To better assess our neighbor selection scheme without bias or skewing the results, we randomly select 50 nodes that have at least 20%

[2] Additionally, our prediction accuracy implies correlation between a single node's file store and queries, a result we analyze in detail in [18].

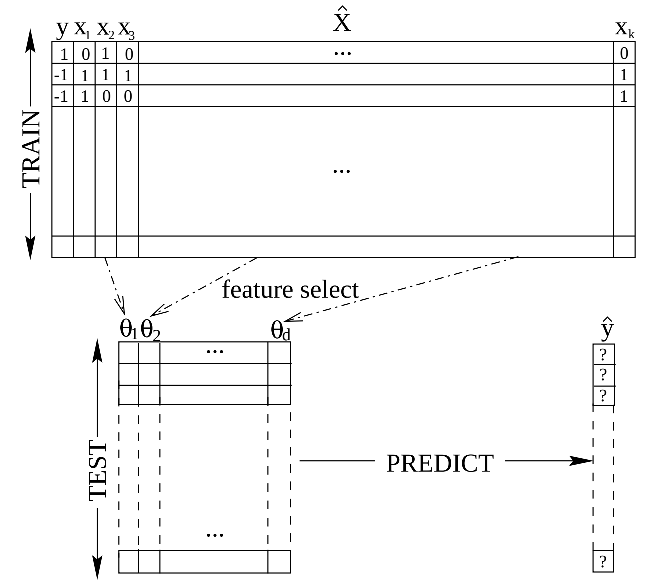


Figure 3-4: The machine learning task: from randomly selected training samples, find the best model and features ($\boldsymbol{\theta} \subset \hat{\mathbf{X}}$) to minimize training error. With this small set ($d \ll k$) of features, predict neighbor suitability (y class label).

positive labels, i.e. a non-trivial number of suitable potential peers. In this way, we choose to evaluate the nodes that are **most difficult** to accurately perform predictions with and thereby stress our approach.

These nodes include $k = 37,000$ unique file store tokens. We provide both average and variance measures across these nodes from our dataset. Thus, we wish to show that our scheme is viable for the vast majority of all nodes.

We note that incorrect predictions in our scheme are not fatal. In the overlays we consider, a node that attaches to a peer who in fact provides no utility can simply disconnect that peer. Therefore, while the statistical methods we employ provide only probabilistic measures of accuracy, they are well-suited to the neighbor selection task.

Methodology Summary

Our evaluation methodology simulates the following algorithm on nodes from our dataset and measures prediction accuracy. For a node $i \in N$ in the system that wishes to optimize its connection topology:

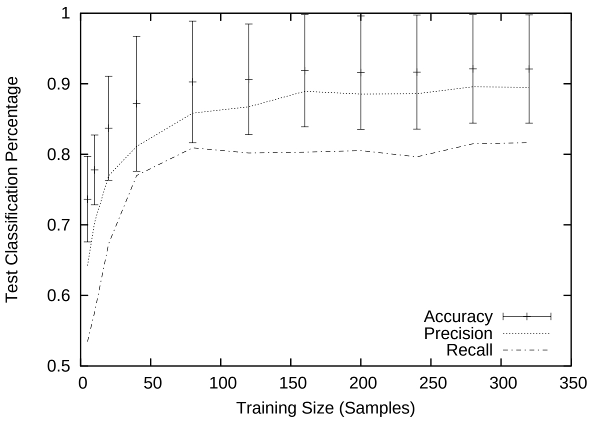


Figure 3-5: SVM Neighbor prediction: classification performance versus number of training samples

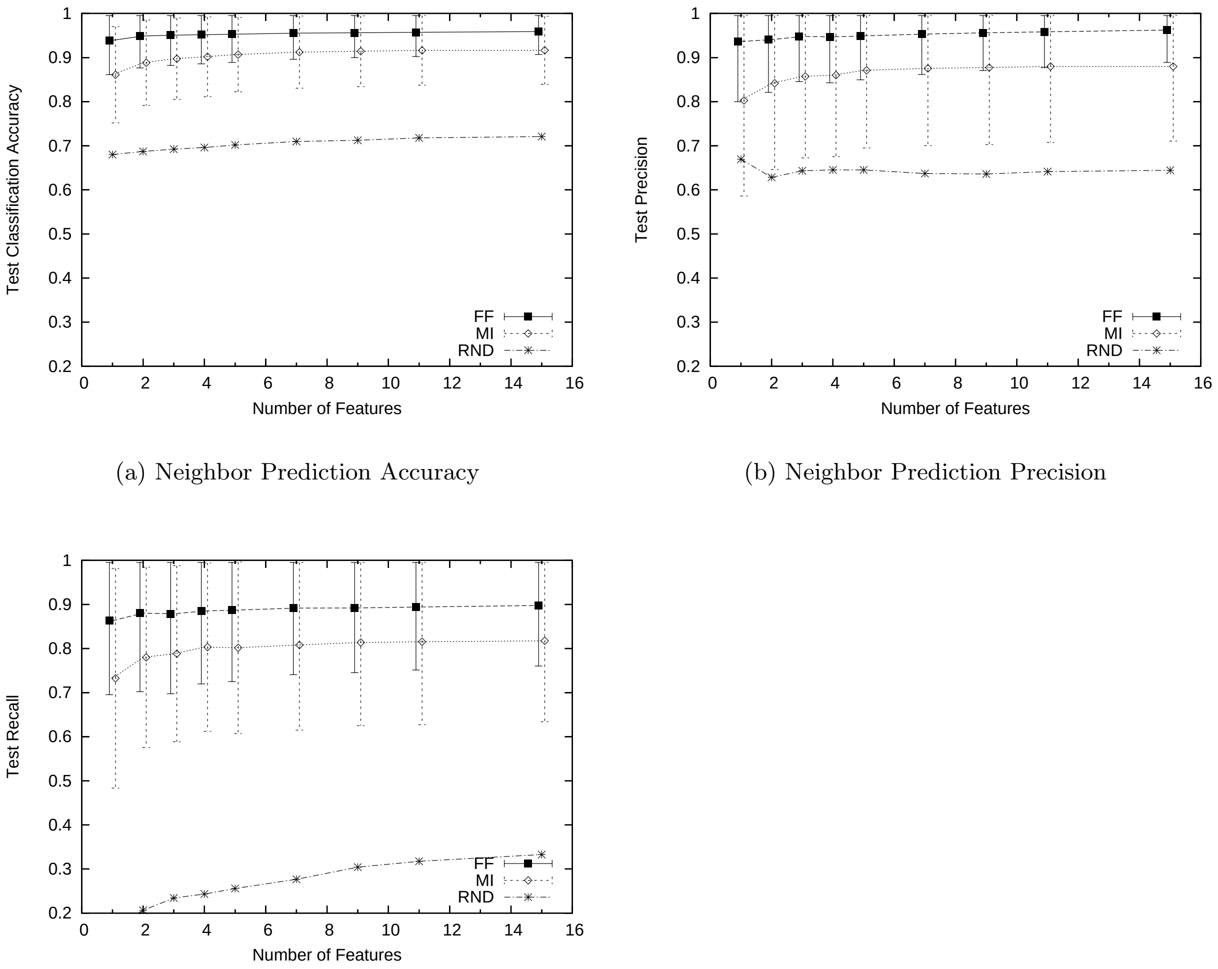
Summary of Methodology

1. Randomly select $T \subset N$ other peers as trainers, where $|T| \ll |N|$
2. Receive file tokens $\mathbf{x}_t$ from each $t \in T$
3. Determine utility of each training peer $y_t = \text{sign}\,(u_i(t))$
4. Let $X = \bigcup \mathbf{x}_t$ and $k = |X|$
5. Find features $\theta_1, \cdots, \theta_d \subset X$, where $d \ll k$, which best predict $\hat{y}_t = y_t \forall t \in T$
6. Issue $\boldsymbol{\theta}$ to test set $M \in N - T$, predict whether to connect to each peer $j \in M$

3.2.2 Results

In many respects, our problem is most akin to text categorization and we draw upon the rich literature of prior work espousing machine learning theory. One of the earlier works in application of SVMs to text categorization is from Joachims [80]. Yang and Liu examine several text categorization methods against standard new corpora, including SVMs and Naïve Bayes [156]. Our results similarly find SVMs outperforming Naïve Bayes for our application; we omit Naïve Bayes results here. Finally, as in Yang's comparative study [157], forward fitting also outperforms mutual information feature selection on our dataset.

SVMs are a natural selection for neighbor selection since the problem is separable, we may need to consider high-dimensional data, and the underlying distribution of the data points and independence are not fully understood. We use the MySVM package for our experiments [80]. To reduce possible dependence on the choice of training set, all results we



(a) Neighbor Prediction Accuracy

(b) Neighbor Prediction Precision

(c) Neighbor Prediction Recall

Figure 3-6: Neighbor selection prediction performance for three different feature selection algorithms: Mutual Information (MI), Forward Fitting (FF), and Random (RND) with an SVM model. The points represent average performance across nodes in our dataset, while the error bars show the standard deviation.

present are the average of five independent experiments. We randomly permute the order of the dataset so that, after splitting, the training and test samples are different between experiments. In this way, we ensure generality, a critical measure of learning effectiveness. We evaluate model performance on the basis of classification accuracy, precision and recall (see §2.3 for the ML definition of these terms). All three measures are important in assessing performance; for instance, a model may have high accuracy, but poor recall as mentioned in the previous section when nodes have few suitable peers in the entire system.

An important first consideration is the sensitivity of the model to the number of training points (step 1 in §3.2.1). In Figure 3-5 we plot the test accuracy, precision and recall versus the number of training points. Test error decreases significantly in the range [10,70]. Test error is minimized around 150 points; beyond 100 training points, accuracy, precision and recall all remain fairly stable. Because we are interested in obtaining the best combination of the three performance metrics while minimizing the number of training points, we select 100 training points for the remainder of our experiments.

In order to find a small number of highly discriminative tokens (step 5 in §3.2.1), we turn to feature selection methods (detailed in §2.2.9). Recall that we use feature selection in a novel manner, not to reduce the computation complexity of classification, but rather to minimize communication cost for neighbor selection in the network. Figure 3-6 summarizes the most meaningful results from our feature selection and SVM experiments (step 6 in §3.2.1). The experiment is run five times for each configuration and the average numbers are presented in the graph along with the standard deviation error range. We include random feature selection as a baseline measure. The primary observations from these graphs are:

- *We are able to make accurate predictions with as few as 5 features.* This is a surprisingly good result. Recall that we are handicapped from the fact that a) we consider only the file features not the query features, even though the queries are what generated the results; and b) there are 37,000 single-word features and thus a massive number of multi-word combinations.

- *Forward-fitting performs better than mutual information.* In particular, we see that even with one feature, FF has a lower test error than MI.

- *The random feature selector performs poorly.* While we expect randomly chosen features to perform the worst, it provides validation of our results and methodology and proved to be a surprisingly useful benchmark in building our system.

3.3 Discussion

Here we examine the larger conclusions that can be drawn from our findings:

- *While our input space is of high dimension, it can be effectively summarized with very few parameters.* The summarization aspect of our problem inspired us to consider an

SVM approach. Forward fitting reveals that a small number of features effectively correlates to accurate predictions. Our findings are in stark contrast to the standard motivation for SVMs in text classification problems where the inputs are independent and the primary motivation for feature selection is to reduce the computational complexity.

- *SVMs allow us to accurately model the problem with little or no underlying understanding of the inputs.* While we have a general understanding of the data, we face the challenge that we do not totally understand the relationship between the features and moreover the nature of these relationships vary from node to node. Nodes in a real system face this same challenge. Therefore SVMs, which make no underlying assumptions regarding the data, are particularly well-suited for our problem.

- *For our problem, forward fitting outperforms mutual information.* The literature is mixed on the relative merit of mutual information and forward fitting. In our problem, we were interested in seeing if, as enough features were added via MI, the combination of features could outperform FF, where features are selected in a greedy fashion. Empirically this was not the case.

  One reason FF performs well is the high degree of correlation between features in the dataset. For example, if a user has songs by "Britney Spears" both "Britney" and "Spears" may be descriptive features. However, simple MI will not take into account that once it adds "Britney", adding "Spears" will not improve prediction performance. In future work, we plan to investigate the ability to remove correlated features found via MI by computing feature-to-feature MI. Conversely, forward fitting will likely only add one of these terms, moving on to a more descriptive second term. The danger of forward fitting of course is over-fitting but we do not observe over-fitting in practice.

- *For our problem, SVMs do not suffer significantly from over-fitting.* As witnessed by varying the number of training points, SVMs are robust against over-fitting. While some over-fitting is present in our empirical results with forward fitting, in the context of our particular problem it has little impact. In particular, we are attempting to minimize the number of features with as little impact on prediction performance as possible. Therefore, the fact that too many features leads to worse classification performance is not as problematic as it may be for other problems.

- *Our neighbor selection algorithm is computationally practical.* Nodes can use the SVM prediction we describe in a completely decentralized fashion. While forward fitting gives the highest accuracy, it requires training many SVMs. Nodes with limited computational power can use MI to achieve comparable accuracy. In future work, we may consider FF over multiple features at once. While this method of forward fitting may be more expensive computationally, it could run as a background process on a user's desktop (say overnight) and thus not be prohibitively expensive in practice.

- *Our neighbor selection algorithm is practical in real networks.* While we simulate and model the operation of nodes using machine learning algorithms to predict suitable neighbors, our scheme is viable in practice. P2P systems, and Gnutella in particular, utilize a system of caches which store IP addresses of nodes in the overlay thereby allowing new clients to locate potential peers. Our experiments show that randomly selecting $\simeq 100$ peers on which to train suffices to build an effective classifier. Because we randomly permute the set of training nodes in each experiment, the density of "good neighbors" in 100 peers is sufficiently high for accurate future predictions.

- *Efficient neighbor selection is a general problem.* While we focus only on P2P overlays, minimizing the communication overhead via feature selection methods such as those in our algorithm may generalize to tasks in other networks or applications.

3.4 Conclusions

Overlay and content delivery networks (CDNs) are architecturally provocative as they demonstrate the power of distributed intelligence. By operating in the face of a continually changing environment, overlays must adapt and act intelligently, often assuming roles traditionally held by the routing infrastructure. We examine pushing this intelligence both out individual end-nodes in this Chapter and down into the network later in the thesis.

Our investigation in feature selection methods suggests several addition avenues. We plan to investigate the ability to remove correlated features found via MI by computing feature-to-feature MI. Additionally, we wish to observe the efficacy of running FF over multiple features at once in the neighbor selection task.

In this Chapter, we examine efficient neighbor selection in self-reorganizing P2P networks. While self-reorganizing overlays offer the potential for improved performance, scalability and resilience, their practicality has thus far been limited by a node's ability to efficiently determine neighbor suitability. We address this problem by formulating neighbor selection into a machine learning classification problem. Using a large dataset collected from the live Gnutella network, we examine the efficacy of SVM classifiers to predict good peers. A key insight of our work was that nodes can use feature selection methods in conjunction with these classifiers into order to *reduce the communication cost* inherent in neighbor selection. Using forward fitting feature selection, we successfully predicted suitable neighbor nodes with over 90% accuracy using only 2% of features.

By addressing the efficiency of neighbor selection in the formation of self-reorganizing networks, we hope our work serves as a step forward in bringing self-reorganizing architectures to real-world fruition.

It is easier to introduce new complications than to resolve the old ones.

- Neal Stephenson

Chapter 4

End-Node Intelligence

4.1 Introduction

At the highest level, the purpose of any network is to service user needs and demands as expressed by hosts. We abstract the interaction between users, hosts and the Internet in Figure 4-1. End-nodes have available resources, but issue queries to resolve instances of incomplete information. This abstraction is quite general; a specific example of a query in response to user demand is the Domain Name System (DNS) [120]. A user accessing the `web.mit.edu` web page implicitly induces a DNS query to resolve the name to a machine readable address. The host meets this demand by using multiple DNS query resolvers in the network, selecting from among the responses and forming decisions on behalf of the user. Thus, DNS represents a network resource with complete and valid information.

Other functionality, however, is not in the network or not exposed to end-nodes. For instance, an end-host does not know the route or latency to a destination, what other hosts the destination is interacting with or any measures of reputation. In the absence of network support for such functionality, hosts must make inferences based on their available data. This Chapter explores end-node intelligence and learning. Specifically we examine ways to effect more intelligent decisions; adapting the user's queries and decision function. To

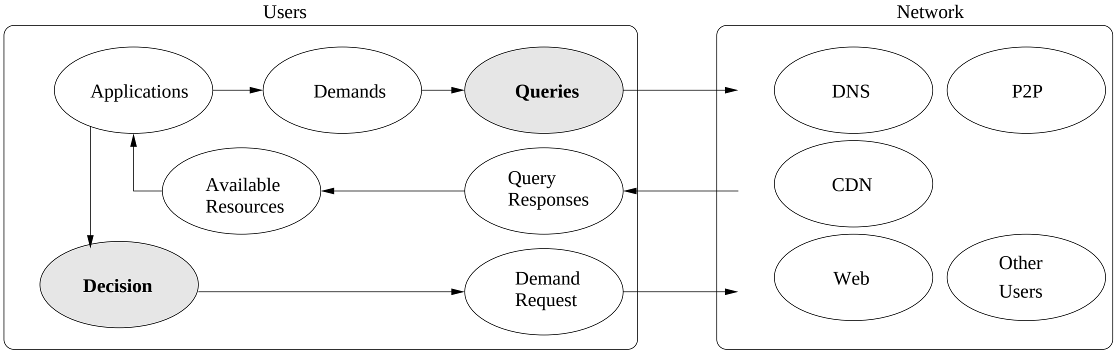


Figure 4-1: Intelligence in the Internet architecture in the context of user demands

ground our thinking, we consider two real-world problems in detail and apply agent-centric machine learning as a solution:

1. Network Latency Prediction (§4.2): We consider determining the latency to unknown IP addresses based on previous interaction with the network. While the Internet address space is fragmented and discontinuous, we use kernel methods (§2.2.7) to find structure upon which to learn. We transform the IP feature space into a feature space and perform support vector regression. We obtain an estimation accuracy within 30% of the true value for approximately three-quarters of the latency predictions on a large, live Internet data set. We obtain this performance without any prior interaction with the target, using only 20% of our data samples for training.

2. Transport-Layer Spam Detection (§4.4): We investigate the discriminatory power of email *transport-layer* characteristics, i.e. the TCP/IP packet stream. With a corpus of messages and corresponding packets, we extract TCP features such as latency, retransmissions, congestion window, etc. While legitimate (ham) mail flows are well-behaved, spam traffic exhibits TCP behavior indicative of congestion, resource contention and large geographic distance. We build "SpamFlow" to exploit these transport characteristics and effect greater than 90% classification accuracy. SpamFlow correctly identifies 78% of the false negatives from a popular content filter – demonstrating the power in combining techniques. By exploiting fundamental weaknesses in sourcing spam, SpamFlow is adaptable and not easily subvertible.

4.2 Problem 1: Network Latency Prediction

With oracle knowledge of all nodes on the network, learning is unnecessary and predictions over, e.g. path performance, traffic or bot-net membership, become perfect. Unfortunately, the size of the Internet precludes complete information. Yet, the Internet's physical, logical and administrative boundaries [60, 71] provide useful structure.

A natural source of Internet structure is Border Gateway Protocol (BGP) routing data [128, 102]. Krishnamurthy and Wang [89] suggest using BGP to form clusters of topologically close hosts thereby allowing a web server to intelligently replicate content for heavy-hitting clusters. Unfortunately, BGP data is often unavailable, incomplete or at the wrong granularity to represent appropriate structure along a particular problem dimension. For instance, many service providers advertise a single large routing aggregate, yet internally demultiplex addresses to geographically disparate locations. Rather than using BGP, we focus on an agent's ability to *infer network structure* from its available data.

Thus, while Internet addressing is hierarchical, it is discontinuous and fragmented, suggesting that learning can discover additional structure beyond what is available in routing tables or registries. In this Section, we exploit IP address structure in order to endow net-

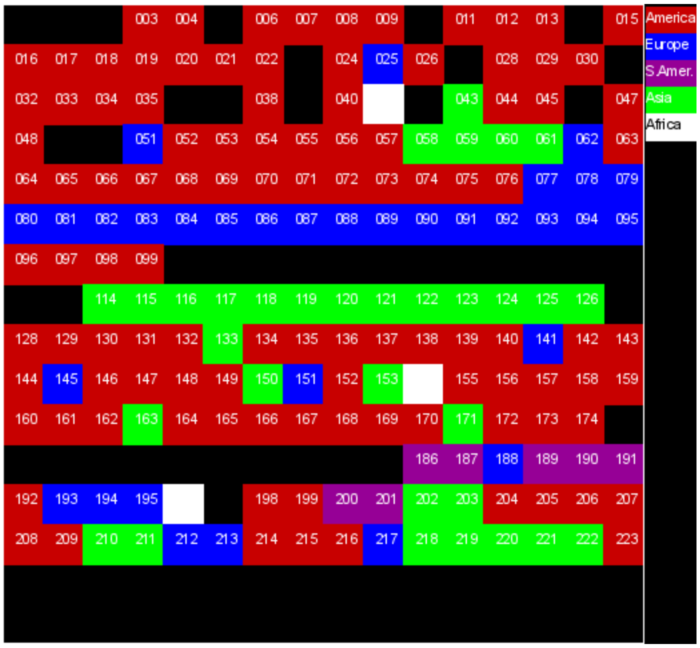


Figure 4-2: Global allocation of /8 IP address prefixes by IANA: distinct colors represent different geographical regions. Even at a low-granularity, allocations are discontinuous and fragmented.

work agents with predictive abilities. Based on prior network interaction, we examine the ability to predict latency to random hosts an agent has not previously interacted with.

4.2.1 Internet Structure

IP addresses must be globally unique and are therefore assigned by regional registries which are in turn governed by a central body, the Internet Assigned Numbers Authority (IANA). Wherever possible, addresses are delegated in a semi-organized and hierarchical fashion with the intention that networks, organizations and geographic regions receive contiguous blocks of IP address space. Contiguous allocations permit aggregation of routing announcements, preserve router memory and reduce BGP convergence time.

Figure 4-2 is a graphical representation of the global IP address allocation at the /8 granularity, i.e. how large, size 2^{24}, pieces of address space are delegated. The Figure is a 16x16 grid representing the geographical allocation of all 256 /8 prefixes. Generally, these large prefixes are allocated by IANA to regional registries. Color in the figure indicates the region: red is North America, blue is Europe, green is Asia, purple is South America and white is Africa.

While Internet IP address space maintains hierarchy and structure, it has evolved organically over time. As a result the address space is discontinuous, variable and fragmented [30, 102] as shown in our graphical representation and as evidenced by the approximately 250,000 BGP entries in the global routing table [72].

The semi-structured IP address space is sufficiently complex to motivate a learning approach. While an agent's latency to machines on a particular subnetwork may be within a tight bound, there exist other subnetworks with an identical bound that are numerically distant in the IP space. Analysis of geographic locality in BGP prefixes [59] finds that autonomous systems commonly advertise multiple noncontiguous prefixes corresponding to a single location. Because of this address discontinuity, our research investigates the use of kernel functions to transform the Internet address space into a feature space amenable to support vector [146] learning methods.

4.2.2 Applications

The ability to learn and predict network latencies to random destinations is potentially useful in a variety of practical applications. For example, our results provide an estimation accuracy granularity suitable for:

1. **Service Selection:** To balance load and optimize performance, a resource may be distributed over a set of geographically distributed servers. These servers may coordinate to form service selection decisions on the basis of current network performance and the origin of a request as well as the object requested [127]. Service selection is also an important problem in peer-to-peer (P2P) networks where popular data is replicated among many nodes. A search in a P2P file sharing network may result in many potential peers offering the file. Latency prediction enables an alternate architecture where intelligence is shifted to the end-nodes. For instance, consider a web service existing in multiple, distributed locations advertised via a set of DNS address records. An intelligent resolver agent's first choice for the given resource can be guided by our learning algorithm. The client predicts which server is closest from among the set of all potential addresses for the given resource. Note that an incorrect prediction is not fatal; nothing precludes the agent from selecting a different server if the first proves to be a poor choice. Both clients and servers benefit in such an architecture without explicit coordination.

2. **User-directed Routing:** Currently network end-nodes have no control over the route their data takes through the network to a destination. However, the continued adoption of IPv6, with multiple per-provider logical interfaces, and research efforts such as NIRA [154], RON [7] and deflections [155], are poised to give nodes coarse routing control. In an IPv6 world with its provider-assigned addressing model, hosts will have a combinatorial number of interfaces. When forming decisions on how to best send traffic to a particular destination, learning algorithms can significantly narrow the host's search space. Similarly, a mobile device choosing from many different possible wireless networks could form decisions based on prior interactions with each [93]. In fact, any agent can build a "routing table" without formal participation in a

routing protocol or receiving routing announcements. Latency prediction potentially benefits any overlay system, providing efficient construction of distributed hash tables or multicast trees.

3. **Resource Scheduling:** Web-servers endowed with predictive abilities might tailor content depending on the anticipated latency of the remote end-point or perform opportunistic scheduling [8]. Additionally, the grid computing community would like to predict transfer times in order to perform distributed scheduling efficiently [122].

4. **Network Inference:** Researchers frequently use structural models of the Internet including routing tables. However, publicly available routing tables [105] provide only a highly aggregated view and from limited vantage points. In many cases, it would be useful to understand the internal structure and address assignments of individual networks. A classification algorithm such as we propose could be used to infer detailed topological properties of networks.

4.2.3 Related Work

Because of the broad range of potential applications, Internet latency prediction has a long history of prior research. We present relevant efforts here.

Using recursive DNS queries, King [67] estimates latencies between arbitrary pairs of Internet hosts. King's method assumes DNS servers are in close proximity to the hosts they are authoritative for and requires an active probe. Vivaldi [52] is a scheme which defines a synthetic coordinate system in order to predict latencies. Vivaldi is a distributed algorithm that requires nodes to query each other to establish their relative position in the coordinate space. Meridian [149] is a distributed network location system using concentric ring queries rather than a virtual coordinate system.

The iPlane project [98] similarly embraces our notion of finding and utilizing network structure. However, iPlane is intended as a service for many agents rather than a single-agent system. iPlane continually measures the Internet using traceroutes, bandwidth measurements, etc. and performs clustering operations to infer structure.

These existing approaches utilize active queries, landmarks and synthetic coordinate systems. The key difference between this prior work and our research is that our effort is designed for individual, intelligent network agents and does not presume any additional network infrastructure, overlays or network-layer assistance. An agent in the network can be any device that is attempting to make decisions based upon previous interactions with the network. Our intent is that every autonomous agent in the network build an independent view of the network in order to form predictions that maximize individual utility. Our technique is predictive on the basis of prior learning: an agent forms a latency estimate for a random, remote end-node with which it has never previously interacted.

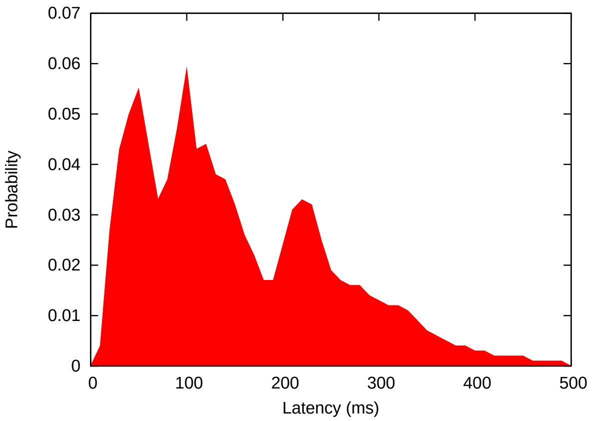


Figure 4-3: Probability mass function of 30,000 node latencies in data set

4.2.4 Secondary Network Structure

For Internet-scale networks, an agent cannot maintain complete information on all nodes. However, we hypothesize that agents can exploit the inherent IP address locality to their advantage. An agent in the network can be any device that is attempting to make decisions based upon previous interactions with the network[1].

Many properties of interest may be reasonably approximated by a distribution shared among members of the node's subnetwork. For example, latency, congestion, throughput, etc. are more likely to be correlated between two nodes within the same subnetwork as compared to two random nodes from the entire network. Given a general expectation of consistency that is inversely proportional to the numerical difference between node addresses, it is not necessary to maintain per-host information[2]. Rather an agent may form reasonable predictions from available, observed data that is sparse in relation to the entire network.

To collect data for our experiments, we use a simple active measurement procedure. IPv4 addresses are 32-bit integers, typically represented using "dotted-quad" notation as four octets: `A.B.C.D`. We select unsigned 32-bit integers at random until one is found as a valid IP address in a public global routing table. Based on the approximately 1.8B publicly advertised addresses, filtering with the BGP table reduces our search space by approximately half. If the randomly selected destination responds to ICMP echo requests, i.e. "ping," we record the average of five ping times from our measurement host as the round-trip latency. Our data set consists of approximately 30,000 randomly selected $(IP, Latency)$ pairs.

Figure 4-3 displays the probability mass function of latencies as observed in our data set. The distribution is non-trivial, with multiple modes likely corresponding to geographic

[1] Our intent is that every autonomous agent in the network build an independent view of the network in order to form predictions that maximize individual utility.

[2] Intra-network consistency is naturally not absolute, but our work is concerned with providing a *most likely prediction* for applications that can compensate for occasional errors.

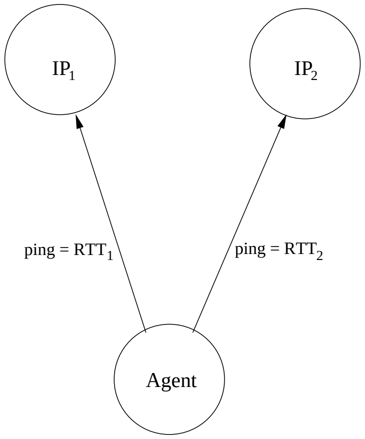


Figure 4-4: Assessing the correlation between address distance and RTT

regions. Is it reasonable to believe that latency, congestion and throughput are much more likely to be consistent amongst destinations all within the same subnetwork as compared to random nodes drawn from the entire network?

Before attempting to learn, we examine our initial hypothesis: sufficient secondary network structure exists upon which to learn. Without this secondary structure, we cannot expect to partition and cluster portions of the Internet. We focus on network latency in this discussion, however other network properties may provide sufficient structural basis.

Let the distance between two addresses be $d = |IP_1 - IP_2|$, i.e. simply the absolute numeric difference between the two unsigned 32-bit integers. To understand the correlation between round-trip latency and distance, we perform active measurement and gather real data from Internet address pairs as shown in Figure 4-4.

Algorithm 4.1 provides our measurement procedure which finds pairs of active hosts separated by d. For a given distance, find a random address *addr* which exists in the global BGP routing table. If *addr* responds to an ICMP echo request (ping), we attempt to find a responsive neighboring address at a distance of d apart by using a sliding window ($swin$) with different initial offsets. Once two suitable nodes are found, we measure and record the average round-trip latency over five trials. The procedure repeats until it finds *count* unique pairs. We use this procedure to asses the correlation between address distance and RTT. Specifically, we are interested in $Pr\,(RTT_1 = (1-\epsilon)RTT_2|d)$.

Figure 4-5 shows the probability that the round-trip time (RTT) latency for two random d-distant IP addresses disagrees within ϵ percent error. In addition to several d values, we include $rand$, the probability that two randomly selected addresses, irrespective of their distance apart, share the same latency. Two random addresses have less than a 10% chance of agreeing within 10% of each other. In contrast, adjacent addresses ($d = 2^0$) have a greater than 80% probability of latencies within 20%. While the latency of virtually all pairs of addresses separated by a class C ($d = 2^8$) agree within 50%, fewer than half of the random addresses agree in this bound.

Algorithm 4.1 *gather*$(count, d)$: Gather latencies to *count* random d-distant address pairs

```
$R$, a longest-match IP routing table
$D$, set of $(ip_1, rtt_1, ip_2, rtt_2)$ tuples
$swin \leftarrow 5$, pair search window
**while** $count > 0$ **do**
  **repeat**
    $addr \leftarrow [0, 2^{32}]$
  **until** $addr \in R$
  **if** (ping($A$) == true) **then**
    **for** $i \leftarrow -swin$ to $swin$ **do**
      $d_1 \leftarrow addr + i$
      $d_2 \leftarrow d_1 + d$
      **if** (ping($d_1$) && ping($d_2$)) **then**
        $D \leftarrow (d_1$, avgrtt($d_1$), $d_2$, avgrtt($d_2$))
        $count \leftarrow count - 1$
```

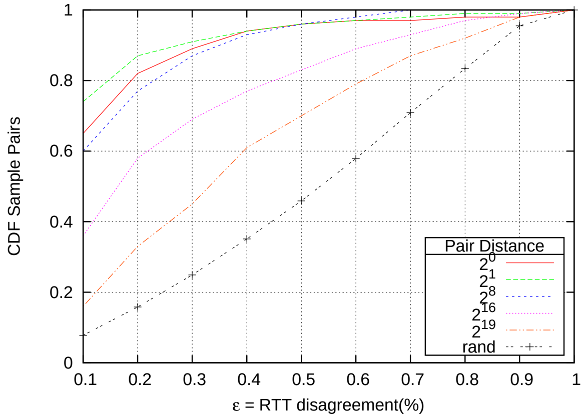


Figure 4-5: Cumulative probability that the RTT of a pair of d-distant IP addresses agrees within ϵ percent. Two random addresses have less than a 10% chance of having RTTs that agree by 10%.

4.2.5 Learning Approach

We use SVM regression to produce a latency prediction. SVMs are attractive for this problem as they allow us to experiment with various kernel functions to accommodate the unknown and discontinuous IP address structure seen even at low-granularities (Figure 4-2). Further, SVMs are known to perform well on many practical learning problems because of their regularization parameters as detailed in §2. Later in this Chapter we devise an alternate prediction algorithm (§4.3).

Learning algorithms require optimization along several dimensions. We begin by analyzing the training complexion: which features of the IP address provide the most discriminatory power and what size training set generalizes well. Given a suitable training set, we examine prediction error and error distribution.

IP addresses are simply unsigned 32-bit integers. We transform the IP addresses into a 32 dimension input space where each bit of the address corresponds to a dimension. Thus, the input to the SVM machine is an IP address bit vector $\mathbf{x}$ while the labels y are floating point round-trip latency numbers.

To reduce possible dependence on the choice of training set, all results we present are the average of five independent experiments. We randomly permute the order of the data set so that, after splitting, the training and test samples are different between experiments. In this way, we ensure the generality, a critical measure of the effectiveness of learning algorithms.

Performance is measured in terms of deviation from the true latency in the data set. The most naïve approach is to simply always predict the mean latency of the training samples: $f(\mathbf{x}) = \bar{y} = \frac{\sum y_i}{|y|}$. The mean latency of our data set is $\bar{y} = 122$ms. A mean prediction strategy with an absolute loss function, $V(f(\mathbf{x}), y) = |\bar{y} - y|$, yields a mean prediction error of approximately 70ms. Thus, results lower than 70ms metric indicate effective learning.

4.2.6 Results

Training Complexion

The selection of training points is crucial to any learning algorithm. 30,000 addresses out of the approximately 1.8B advertised in the global routing table is quite sparse. If our test points are close to points in the training set, we expect the learning to over-perform. We wish to ensure that our training set is suitably well-distributed in order to generalize to random predictions. One metric of distribution is address dispersion. To compute address dispersion, we find the numeric difference between each address and the next closest address. For the set of 30,000 addresses A, the minimum dispersion of address i is:

$$min_i = \underset{j}{\operatorname{argmin}} \left(|i - j|\right) \ \forall j \in \{A - i\} \tag{4.1}$$

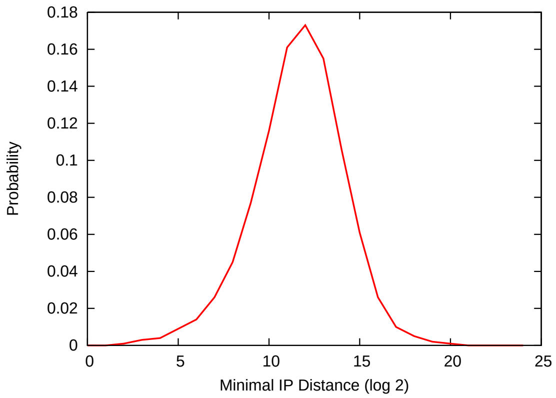


Figure 4-6: Probability mass function of IP address dispersion in data set

Figure 4-6 shows the probability mass function of IP address dispersion in our data set, i.e. $\Pr(\lfloor \log_2 min_i \rfloor = x) \; \forall i \in \{A\}$. Approximately 82% of the addresses have a minimal separation of 2^{10} or greater.

Kernel Selection

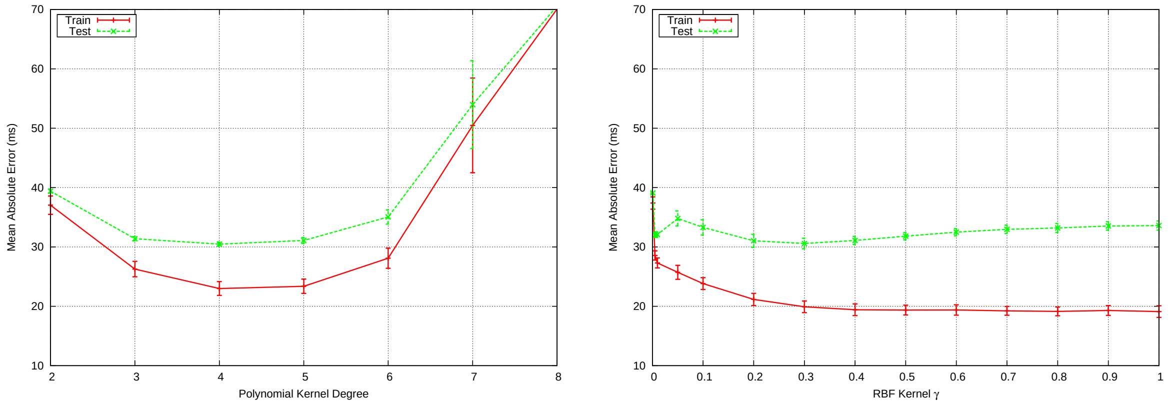
Next, we consider the problem of selecting an appropriate kernel. For a training set size of 2000 points, we evaluate the prediction loss, or mean prediction error, for the test samples as a function of kernel parameters. Figure 4-7(a) plots both training and test error for a polynomial kernel as a function of polynomial degree. Included in the plot are error bars that indicate the performance standard deviation over 10 independent trials; the deviation is tight up to the sixth degree polynomial, after which the model performs poorly. For polynomial kernels, the test error closely approximates the training error across the range of polynomial degrees. Both the minimum training and test error is achieved with a fourth degree polynomial.

Next, we examine a radial basis kernel, $e^{-\gamma||x-\hat{x}||^2}$ and vary the γ parameter which controls the width of the Gaussian. Figure 4-7(b) shows that the SVR training error continues to decrease, while the test error decreases until $\gamma = 0.3$. After this point, the test error begins increasing, a symptom of over-fitting where the test error is no longer reflective of the training error. As the model becomes more complex, both training and test error improve up to a point, then diverge.

Because the polynomial and radial basis kernels achieve similar best performance, approximately 30ms test error, we elect to use the more efficient polynomial kernel. The remainder of the experiments use a fourth degree polynomial.

Problem Dimension

The selection of training complexion is crucial to creating a machine that generalizes well and operates efficiently. We examine the informational content of each bit, or "feature," in



(a) Polynomial Kernel

(b) Radial Basis Kernel

Figure 4-7: Kernel selection, training and test performance as a function of kernel type and parameters

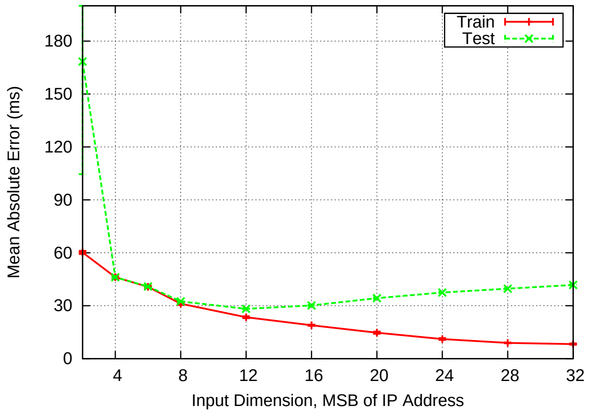


Figure 4-8: Latency prediction mean error vs. input dimension, i.e. number of most significant bits of input IP address.

the IP address. Let $\boldsymbol{\theta}$ be a feature vector where $\theta_i \in \mathbf{x}$. Intuitively, the most significant bits correspond to large networks and should provide the most discriminatory power. Here "most-significant features" correspond directly to BGP prefix masks, i.e. `192.160.0.0/12`. We run the regression SVM algorithm against our data set using 4000 points for training while varying the number of input features. For example, the first 12 features of IP address `192.168.1.1` is the bit vector $\boldsymbol{\theta}$ =`110000001010`.

We plot the SVM training and test mean error as a function of input space dimensionality in Figure 4-8. We see that four or fewer bits yields virtually no information; the mean error is no better than in the mean prediction strategy. However, with only four more bits of input address, the regression achieves a mean error around 32ms. The optimal number of most significant features is between 12 and 14, after which test error begins to increase.

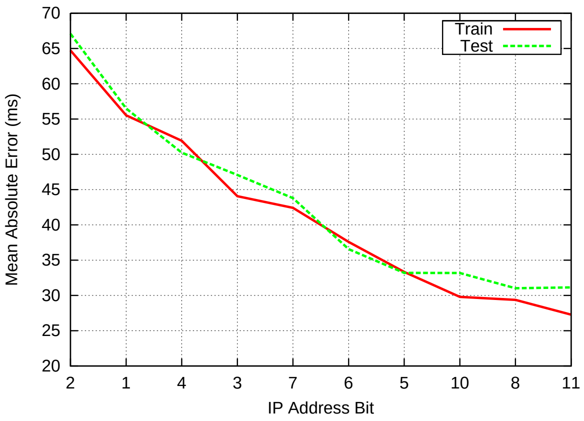


Figure 4-9: Feature selection: mean error vs. sequential bit chosen via greedy strategy.

Notice that training error decreases monotonically as more input bit features are added and training error no longer generalizes to the achieved validation error. This divergence between training and test error is symptomatic of over-fitting. Unsurprisingly, the least significant bits of the IP address add no discretionary strength.

Next, we use greedy forward fitting feature selection (§2.2.9) to determine which bits of the IP address are most valuable to the regression algorithm. The feature selection criterion is training error. Figure 4-9 depicts the prediction error as a function of features found in greedy feature selection. Again, we plot both training and test error, but now as a function of the set of features chosen. The i'th x-axis data point indicates which bit feature is chosen in the i'th round of feature selection. For example, the best three features are, in order, 2 1 4, while the best five features are 2 1 4 3 7. Note that the training error closely approximates the test error, with only a small deviation on the ninth and tenth best features where training error decreases but test error does not. The fact that there is little over-fitting among the top ten features agrees closely with Figure 4-8 since the best features also have high bit-order significance.

Interestingly, feature selection reveals that the single most powerful bit in an IP address is the second bit. Intuitively we might expect the power of a bit to correspond to its bit-level significance. To see why the second bit has more discriminatory power than, for instance, the first, we reexamine our geographic illustration of the IP address allocation from Figure 4-2. The first IP address bit essentially horizontally divides the figure in half, resulting into two groups. Using the second bit divides the figure into two discontiguous groups as depicted in Figure 4-10. Here, we use yellow lines to show the division imposed by the second bit and have illustrated the one induced group by shading it.

By using the second bit, the algorithm is in actually better grouping North American and European allocations. Table 4.1 shows the grouping induced by selection on the first and second bits respectively. Whereas the division on the first IP address bit almost evenly divide /8's allocated to North America, dividing on the second bit better groups not only

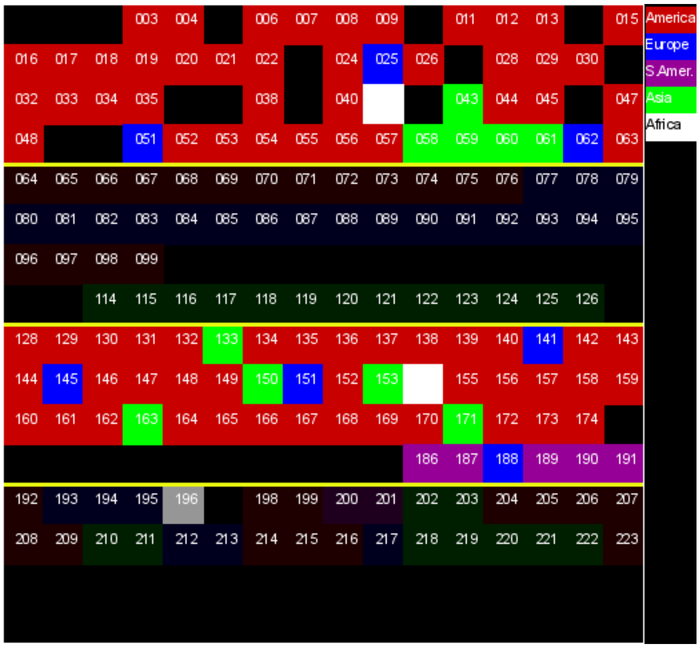


Figure 4-10: Global allocation of /8 IP address prefixes, yellow lines represent a division of the IP space by second most significant bit, resulting in a shaded and non-shaded region. These regions have more geographic commonality than if the space were divided by the first most significant IP bit.

America, but also European addresses. Quite clearly, by selecting the second IP address bit, feature selection is identifying the correct address structure, illustrating the power of the statistical technique in uncovering structure that may not be intuitively evident.

Training Size

Given the best features found via feature selection, we next attempt to optimize the balance between training and test size. Figure 4-11 shows the mean absolute error in milliseconds as a function of training size along with a 70ms line indicating the learning bound. Using 4000 of the data points as training, we obtain an average error of 26.6ms across all latency

Table 4.1: Induced geographical groupings by selecting first or second IP address bit

Address Bit	First Octet Range	US /8s	Europe /8s
1	0- 127	56	21
	128- 255	51	10
2	0-63 128-191	77	7
	64-127 192-255	30	24

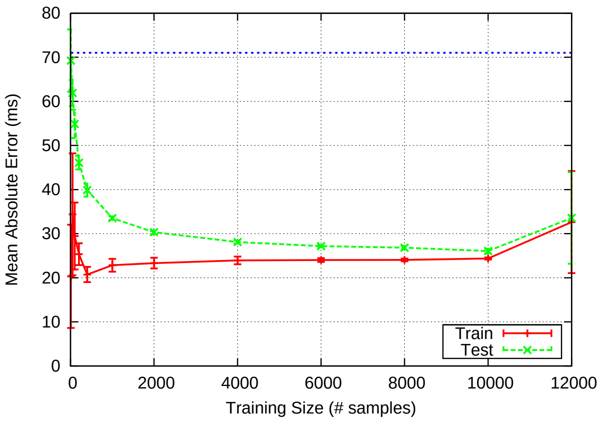


Figure 4-11: Latency prediction mean error vs. training size. The 70ms line represents a naïve mean prediction strategy.

predictions in the test set. 6000 training points yields 25ms average error. Given the inherent noise in the network, the ability to predict latencies within 25ms of their actual value is readily acceptable for applications such as resource scheduling, service selection and user-directed routing.

Regression Performance

Using the first 12 features, i.e. bits, from 6000 of our samples for training yields a good balance between performance and exploitation. Given this training set, we examine the distribution of latency prediction errors. While the previous Section demonstrates a mean error of 25ms, it is important to understand the character of the errors.

11% of the predictions are within 2ms of the correct value, while more than 80% are within 40ms of our measured latency. The distribution has a long tail however, indicating that while the majority of predictions are quite close, there is a relatively even distribution of infrequent errors greater than 60ms. Figure 4-12 presents the cumulative distribution of ratios between predicted and measured latencies. Ratios less than one indicate an underestimate of latency while those larger than one indicate an overestimate. With our SVM prediction method, approximately 61% of the estimates are within a 20% error, i.e. with ratios between 0.8 and 1.2, while approximately three-quarters of the predictions are within 30% of the actual value.

4.2.7 Ranking Performance

Many practical problems do not require an absolute prediction, but rather a preference or rank. In the case of network latency prediction, for example, a prediction of preference is sufficient for an agent seeking a low-latency neighbor. The agent's prediction accuracy is less important than selecting the destination with the lowest latency. For example, an

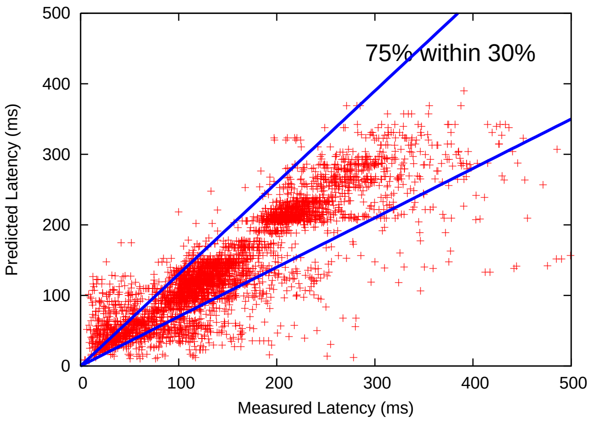


Figure 4-12: Latency estimation performance: scatter plot of predictions versus true latencies. Our regression works well across the range of values.

agent may only use regression predictions in order to discern a degree of preference, e.g. finding "nearby" server. This relative preference is often as useful as an exact floating point prediction.

Consider an application layer overlay that is forming connections guided by a latency heuristic, for example a P2P overlay that is attempting to minimize the difference between the overlay path's latency and the latency of the true end-to-end path. Alternatively, a service might be provided by multiple, geographically diverse servers all of which are advertised through DNS. An intelligent agent might select the server with the lowest latency.

Sequential Rank Prediction

Sequential rank prediction assigns one of k ranks to each new instance. Formally, the ranking agent is given data points $\mathbf{x}_1 \ldots \mathbf{x}_n \in \mathbb{R}^d$. After assigning an estimated rank $\hat{r}_i$ to the i'th data instance, the agent receives the true rank r_i. The rank loss is then the absolute difference between the true and predicted rank: $V(f(\mathbf{x}_i), r_i)) = |\hat{r}_i - r_i|$ and the average rank loss is:

$$\frac{1}{n} \sum_{i=0}^{n} V(f(\mathbf{x}_i), r_i)) \tag{4.2}$$

We employ the PRank algorithm [51] to perform sequential prediction. PRank is a perceptron-based on-line ranking algorithm that maintains a vector $\mathbf{w} \in \mathbb{R}^d$ and a vector $\mathbf{b}^k$ of k thresholds. The value of each threshold is monotonically increasing. Each data point is mapped to a rank by computing the product $\mathbf{w} \cdot \mathbf{x}$ and finding the first threshold b_r such that the product is less than b_r. If the predicted rank is correct, there is no change to the vectors. However, if the prediction is incorrect, PRank updates both the thresholds and the vector $\mathbf{w}$ to "move" the inner-product closer to its true rank threshold.

Using the PRank algorithm, we rank each IP, latency instance in our data set. We

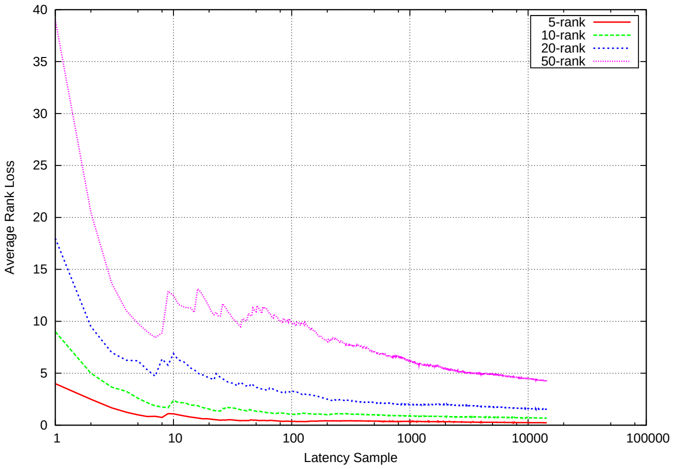


Figure 4-13: Latency ranking performance: average loss for different group sizes using PRank algorithm.

determine the true rank of each IP by dividing one second into k equally-sized latency regions. The rank of an IP is then $rank(\mathbf{x}_i) = \lfloor \frac{latency(\mathbf{x}_i)k}{1sec} \rfloor + 1$.

Figure 4-13 displays the average rank loss, Eq. 4.2, as a function of the sample number for rank sizes $k = 5, 10, 20, 50$. Clearly, the average loss decreases throughout the duration of the experiment, indicating that the algorithm's predicted rankings become closer on average the true ranks.

To understand how well the PRank algorithm performs in practice, we compare our results to a naïve baseline. The most trivial ranking algorithm assigns a rank at random from the k available ranks to each data instance.

Lemma 4.2.1. *The expected rank loss for random rankings among k ranks is $\frac{1}{3}(k - \frac{1}{k})$.*

Proof. Without any knowledge, each data instance is given rank $\hat{r}$ at random. Assume that the true rank of the data point is r. There is a $1/k$ chance that $\hat{r} = r$. There is also a $1/k$ chance that the object's true rank is one of $1 \ldots r-1$ (a higher rank) or $r+1 \ldots n$. Therefore, an object with rank r given a random predicted rank $\hat{r}$ has an expected loss of:

$$E[loss|rank(\mathbf{x}) = r] = \frac{1}{k}\left(\sum_{i=0}^{k-r} i + \sum_{i=0}^{r-1} i\right) \tag{4.3}$$

For a single object, and no a priori knowledge of the true rank r, the expected loss is then simply:

$$E[loss] = \frac{1}{k}\sum_{r=1}^{k}\left(E[loss|rank(\mathbf{x}) = r]\right) = \frac{1}{k^2}\sum_{r=1}^{k}\left(\sum_{i=0}^{k-r} i + \sum_{i=0}^{r-1} i\right) \tag{4.4}$$

By the arithmetic series relationships, these summations reduce to:

$$E[loss] = \frac{1}{k^2}\sum_{r=1}^{k}\left(\frac{1}{2}(k-r)(n-r+1)+\frac{1}{2}(r-1)(r)\right) \tag{4.5}$$

$$= \frac{1}{2k^2}\left[\sum_{r=1}^{k}k^2-\sum_{r=1}^{k}2kr+\sum_{r=1}^{k}k+\sum_{r=1}^{k}2r^2-\sum_{r=1}^{k}2r\right] \tag{4.6}$$

$$= \frac{1}{2k^2}\left[k^3-k^2(k+1)+k^2+2\left(\frac{k(k+1)(2k+1)}{6}\right)-k(k+1)\right] \tag{4.7}$$

$$= \frac{1}{3}(k-\frac{1}{k}) \tag{4.8}$$

□

Thus, for $k = 50$, we expect an average rank loss of ~ 16.7, a value much higher than the PRank loss of less than ten after only 100 samples and less than five after processing all samples.

Relative Rank Prediction

The most common instance of latency prediction is one where the agent is given a set of potential end points and must pick from among them. We cast this problem as *relative ranking.* This problem is subtly different from the sequential ranking problem. In contrast to sequential rank prediction, where each individual instance is assigned a rank, relative ranking assigns a strict total order over a group of instances.

Formally, the relative ranking task maps each of k data points $\mathbf{x}_1, \cdots, \mathbf{x}_k$ to labels $\hat{r}_1, \cdots, \hat{r}_k$ where $\hat{r}_i$ is an integer rank ($r_i \in \{1, 2, \cdots, k\}$) that describes preferences over $\mathbf{X}_{n,k}$. Each predicted rank for a set of k data points is unique, i.e. $\hat{r}_i \neq \hat{r}_j \forall i, j$.

As an example, consider three servers that are 200, 205 and 210ms distant from a agent. Sequential rank prediction might assign each a rank of 9,9 and 10 on a one to ten scale indicating low desirability. Relative ranking of the three servers, however, considers them as a group and might rank them as 1, 2 and 3. Thus, among the possible alternatives, a rank of 1 indicates not a high preference, but a prediction of the best choice given the options available to the agent.

Again, we consider the naïve guesser to determine the baseline performance of the relative ranking. Without loss of generality, say that the naïve learner picks an object for rank=$1, 2, \ldots k$ in order. Therefore, for a single experiment to determine the relative rank across a group of k objects, we have an expected error of:

$$E[loss] = \frac{1}{3}(k^2 - 1) \tag{4.9}$$

We first turn to adapting the previously described SVR method to the relative ranking task. We simply use the predicted latency to provide an ordering over $|k|$ sets of inputs.

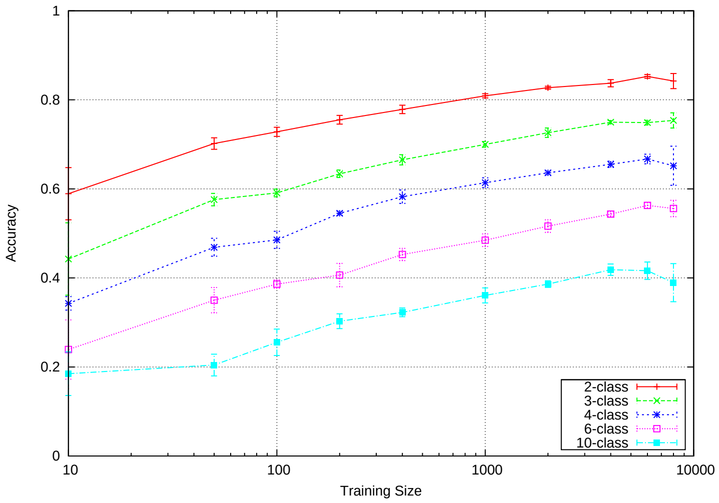


Figure 4-14: k-ranking performance versus log(training size), standard deviation indicated by error bars

Again we train the machine using various size training sets. Let a predicted rank of $\hat{r}_i = 1$ for data point $\mathbf{r}_i$ indicate that the i'th data has the lowest predicted latency. Let the data point with the true lowest latency be $r_j = 1$. If $i = j$, then $\hat{r}_i = r_j = 1$ and the prediction is correct. Otherwise, the prediction is incorrect.

Figure 4-14 shows the ranking accuracy measured as the ratio of correct first rank predictions to all predictions with standard deviation error bars. The most naïve ranking algorithm with perform with accuracy $1/k$ for k-ranking in this scenario. Thus, with 2-ranking, accuracy over 50% indicates learning. We see that at 6000 training points on the 2-ranking problem, the SVR achieves slightly greater than 85% accuracy.

We are also interested in how badly a ranking mis-prediction affects the agent. Let ε be the difference between the true latency to the 1st ranked data point (j) and the true latency to the predicted data point (i). In other words, if the prediction selects an incorrect server, what is the latency difference between the best and predicted servers. In Figure 4-15, we plot this mis-prediction error versus training size for various k-rankings. At 6000 training points, the mis-prediction error is always less than 30ms and is less than 20ms for k=10. Thus, this small amount of error incurred for mis-predictions is not significant.

Finally, we adapt the PRanking algorithm to the relative ranking problem. In each experiment, k points are taken at random from the data set. The PRank algorithm predicts a rank for each $\hat{r}_i, i = \{1, \ldots, k\}$. We then sort the $\hat{r}_i$ by increasing rank value. The predicted relative rank $\hat{r}'_i$ is determined by its position in the order. For example, for $k = 3$, assume that $\hat{r}_1 = 5, \hat{r}_2 = 9, \hat{r}_3 = 2$, then $\hat{r}'_1 = 2, \hat{r}'_2 = 3, \hat{r}'_3 = 1$.

Figure 4-16 shows the performance of the adapted PRank algorithm on the relative ranking task for various group k sizes. In this Figure, we again plot the fraction of correct first predictions, i.e. how often the agent is able to select the lowest latency host from among

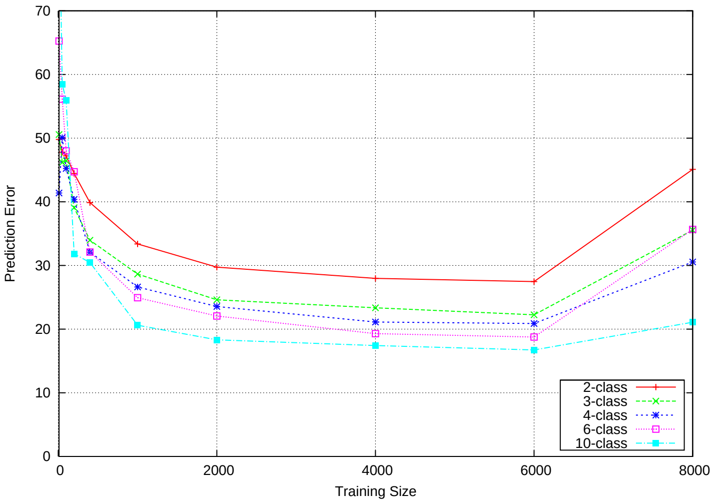


Figure 4-15: k-ranking mis-prediction error versus training size

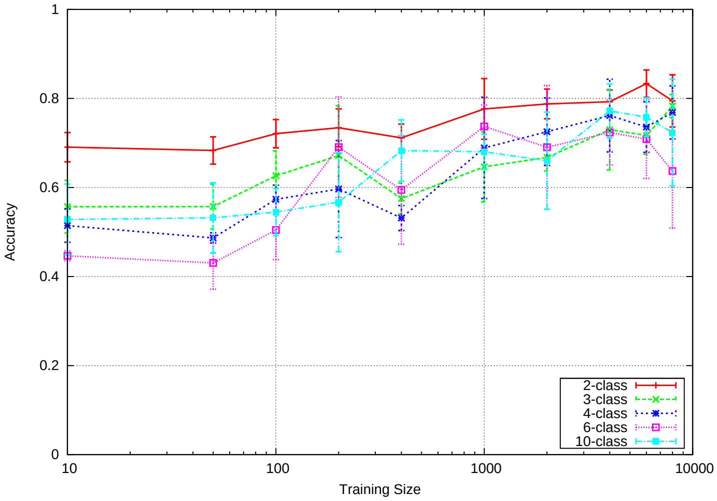


Figure 4-16: k-relative ranking performance using adapted PRank algorithm

those available to it. Thus, in the case of 3-class ranking, we determine not whether one of the three servers has a low latency, or a preferred latency, but rather whether it is the best choice among the three available servers.

Compared to Figure 4-14, we see that the adapted PRank algorithm outperforms using the SVR result to form relative rankings. Even among ten servers, our results show that the agent is able to select the best server with more than 70% accuracy and approximately 1000 training examples. Again, these results demonstrate the potential practical utility in applying prediction to the service selection task.

4.2.8 Discussion

- A significant cost is collecting the training points, however several points bear notice. First, the host may continually collect training data through its normal interaction with the network, for instance using latency from the TCP three-way handshake. In this mode of operation, the data are no longer random and hence the model may be overly specific to IP addresses within networks the host typically interacts with. Yet, this specificity is a desirable feature for most hosts and servers. Because SVMs are amenable to online learning, the model adapts to provide the best generality and performance given the workload the host expects to encounter.

- *The majority of the discriminatory power is comprised of the first eight bits of address.* This is a powerful result, but perhaps unsurprising given the traditional assignment of classful "net A" address blocks to large organizations and networks. Error continues to decrease to a minimum around 12 bits after which the additional features begin to over-fit the training data and hinder the regression performance.

As future work, we plan to explore alternate feature geometries in order to use linear kernels and better represent the IP address structure. We wish to explore other network applications of SVMs and construction of autonomous agents on network test beds. These agents will model our vision of how a web server, peer-to-peer node or IPv6 host might act intelligently.

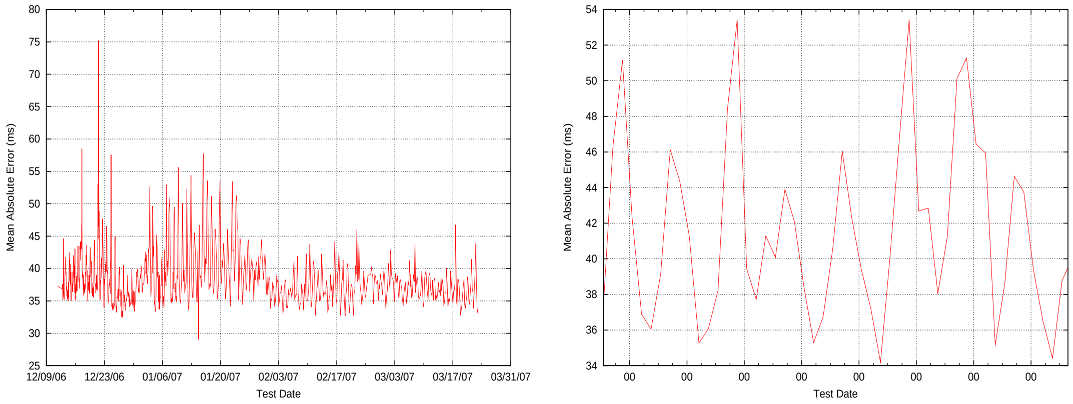
4.2.9 Model Degradation

The preceding discussion abstracts the Internet into a set of static latency measurements. Yet, networks are intrinsically dynamic. For instance, Internet routing and physical topology events change the underlying environment on large-time scales while congestion induces short-term variance. In addition to a single snapshot of latencies, we gather approximately three months of latencies to the same IP addresses. Every hour, from December, 2006 to March, 2007, our collection engine determines the round trip latency to each IP in the original dataset.

We are interested in understanding the model stability and model degradation over time. We train an SVM using the initial latency samples from the first hour in December and use the resulting model to predict latencies for the same IP addresses at subsequent points in time [3].

Figure 4-17(a) displays the mean latency prediction error, averaged over all predictions on each IP address. The set of IP addresses for which we form predictions does not change over time. Further, the SVM model does not change over time; we use the model produced by training on the initial training points. Therefore, this experiment attempts to isolate temporal performance differences.

[3] We exclude addresses which no longer responded during the duration of our data collection.



(a) Mean prediction error over collection period

(b) Week zoom of prediction error showing diurnal effects

Figure 4-17: SVM model performance and stability over time

Clearly, the performance differs over the three month duration, with a significant change visible from the beginning to end of January, 2007. Zooming in on a single week, we see pronounced diurnal effects. In Figure 4-17(b), the minimum error occurs around 02:00 in the morning Eastern Daylight Time[4]

In the next subsection, we consider the task of identifying changes and then how to react once a change is detected in §4.3. Our primary focus is on detecting changes at longer time scales rather than accommodating the diurnal effects. In future work, we plan to explore the use of HMMs to optimize predictive error over these short, predictable day-long behaviors.

4.2.10 Coping with Network Dynamics

When considering structural changes, we are concerned with a change in the mean error resulting from the prediction process. Assume that the process produces data points ($\mathbf{x}$) from a Gaussian probability distribution parameterized by a mean and variance, $\theta = N(\mu, \sigma)$. The probability density is:

$$p_\theta(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.10}$$

We first detect a change in a continuous process parameterized by $\theta_0 = N(\mu_0, \sigma)$ to a new state $\theta_1 = N(\mu_1, \sigma)$. Here, the variance ($\sigma$) remains constant while the mean changes to a known value μ_1. Change detection algorithms rely on the notion of the log-likelihood ratio as a sufficient statistic:

$$s_i = \ln \frac{p_{\theta_1}(x_i)}{p_{\theta_0}(x_i)} \tag{4.11}$$

[4] The time is relative to our collection point which is on EDT.

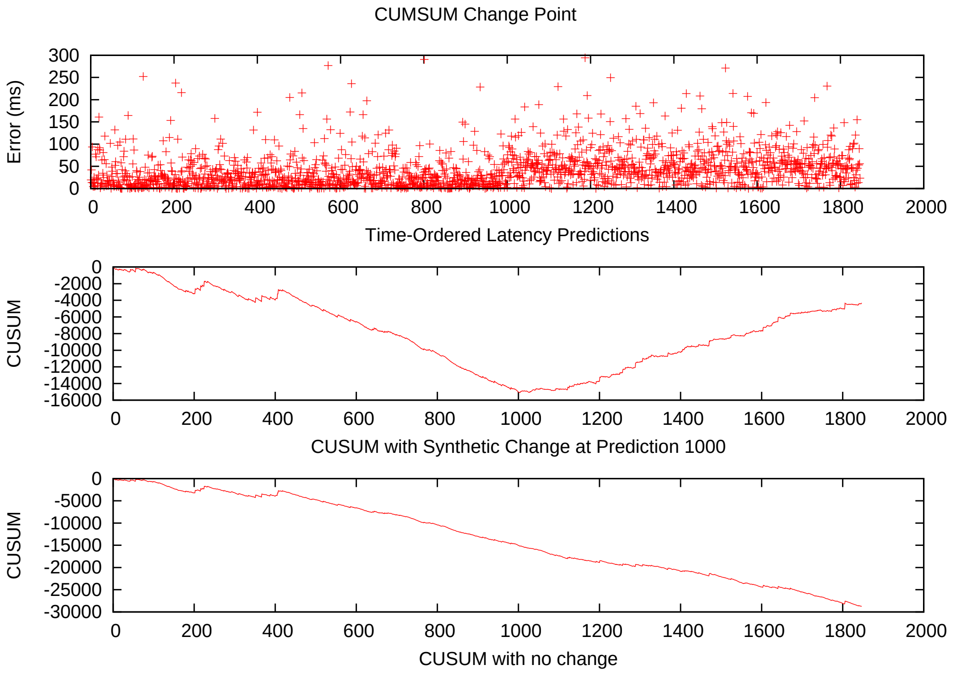


Figure 4-18: CUSUM algorithm: synthetic change of mean injected in real data at the 1000th prediction causes change in prediction error (top), value of S_k over all predictions (middle), value of S_k with no change (bottom).

which is simply the log of the ratio between the probability of a data point x_i in world θ_1 (after a change) to θ_0 (before change). If the probability that x_i is greater in θ_0, i.e. it is more probable that x_i came from the process before the change, the sign of s_i is negative. It can be shown in the Gaussian case that:

$$s_i = \frac{\mu_1 - \mu_0}{\sigma^2}\left(x_i - \frac{\mu_0 + \mu_1}{2}\right) \tag{4.12}$$

Let S_k be the cumulative sum of the s_i values up to point k:

$$S_k = \sum_{i=1}^{k} s_i \tag{4.13}$$

The cumulative sum algorithm notes that before a change, the value of S_k will drift negative. After the change, S_k increases from its minimum. To illustrate this behavior, we take the latency prediction machinery from the previous section. Figure 4-18 plots the error for each of the time ordered latency predictions. At point 1000 we introduce an additional 50ms of latency to each point to represent a structural change. The second pane in Figure 4-18 shows the value of the cumulative sum S_k and clearly illustrates the change, whereas the third pane is the same value without introducing the synthetic change.

To perform the actual change detection, we use the CUSUM decision rule g_k which must be above a threshold h for a change to have occurred at point k:

$$g_k = S_k - m_k \geq h \tag{4.14}$$

where m_k is simply defines the minimum value of S seen thus far:

$$m_k = \min_{1 \leq j \leq k} S_j \tag{4.15}$$

From Figure 4-18, we see that the CUSUM algorithm detects the change at point $k = 1096$, 96 steps after the actual change.

Naturally, we cannot assume, a priori, knowledge of how the processes' parameters will change. Thus, we use the well-known generalized likelihood ratio (GLR) algorithm which assumes the parameter θ_1 after change is unknown. As before, the log-likelihood ratio is as in Eq 4.13. Since θ_1 is unknown, g_k is a function of two independent unknown values:

$$g_k = \max_{1 \leq j \leq k} \sup_{|\theta_1 - \theta_0| > 0} S_j \tag{4.16}$$

We perform the double maximization by taking the derivative of Eq 4.16 with respect to μ_1. It can be shown that:

$$g_k = \frac{1}{2\sigma^2} \max_{1 \leq j \leq k} \frac{1}{k - j + 1} \left[\sum_{i=j}^{k} (x_i - \mu_0) \right]^2 \tag{4.17}$$

Figure 4-19 shows the GLR algorithm applied to the prediction error process in two instances, first when there is a synthetic change introduced into the data at point 1000 and second when there is no change.

Next, we consider the question of a change in the variance where the mean remains fixed. We use the Inclan algorithm [75]. Define the cumulative sum of squares:

$$C_k = \sum_{i=1}^{k} x_i^2 \tag{4.18}$$

For a window of N data points, let:

$$D_k = \frac{C_k}{C_N} - \frac{k}{N} \tag{4.19}$$

D_k will oscillate around 0 when there is no change in variance. If the variance does change, D_k will fluctuate over a threshold with high probability. Inclan [75] derive empirical boundary conditions on $\sqrt{(N/2)}|D_k|$ as a decision function. Specifically, the asymptotic critical value of $D{*}_{0.05} = 1.358$ for N large. Figure 4-20 applies the Inclan algorithm to our data set. The first pane shows the error from predictions, with synthetic variance disruption introduced at point 1000 by multiplying the true latency by 1.5. The second pane shows the value of D_k over the time-ordered latency predictions. The Inclan algorithm detects the

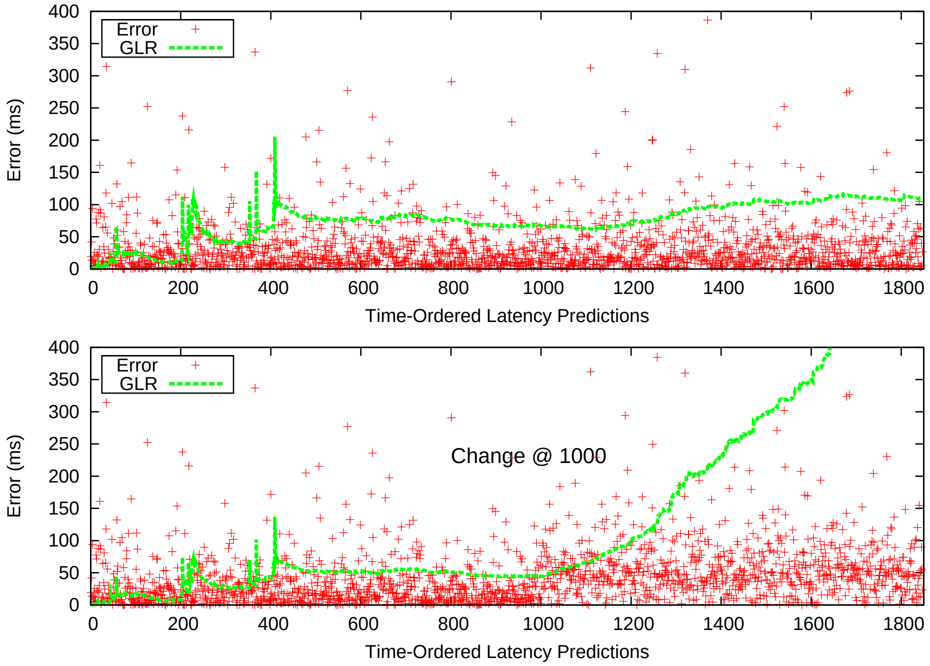


Figure 4-19: GLR algorithm: scatter plot of prediction errors with g_k value (top), same data with synthetic mean error injected at the 1000th prediction and the resulting spike in g_k (bottom).

change in variance at point 1110, 110 points in time after the actual change. In contrast, the third pane shows the Inclan algorithm over the original data. Without an artificial change, the Inclan algorithm does not detect a change in variance over the time window.

4.3 An IP Clustering Algorithm

Thus far, we have demonstrated network locality for latencies (§4.2.4). The presence of address locality leads to a natural question: is it possible to cluster the entire 32-bit IP address space into blocks where the members of any cluster share a common distribution? Such a division is useful in a variety of applications including service selection, user-directed routing, resource scheduling and network inference.

We have shown that learning network structure is effective in forming predictions in the presence of incomplete or partial information. However, our analysis has been overly simple, assuming a stationary environment. The Internet, in contrast, experiences frequent structural and dynamic changes. Many learning algorithms are not naturally amenable to on-line learning and adapting to Internet dynamics. Further, despite the wide array of learning algorithms, none are network or Internet centric, i.e. they do not incorporate *domain-specific knowledge.*

In this Section, we explicitly accommodate an Internet environment with a supervised IP clustering algorithm that imposes a partitioning over a k-bit IP address space. Given

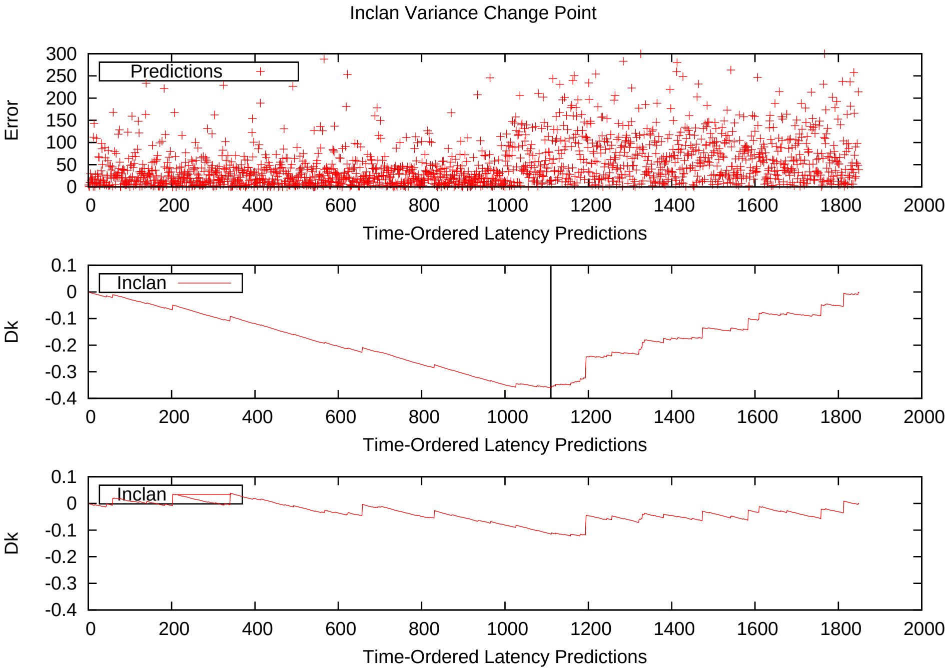


Figure 4-20: Inclan algorithm: synthetic variance change injected in real data at 1000th prediction and the resulting error scatter plot (top), value of D_k as a function of sample order with change detected at 1110th prediction (middle with vertical line). Value of D_k on data without artificial change (bottom). The bottom value of D_k correctly infers no change.

a data set that is sparse relative to the size of the 2^k space, we form clusters such that, with high probability, addresses within a partition share a common property. The resulting clusters provide the basis for accurate predictions on addresses for which the agent has no direct knowledge. Figure 4-21 provides a graphical intuition of an example clustering. For example, in the case of latency prediction, our algorithm uses training data to form clusters where the latencies within a cluster have a statistical guarantee of coming from a Gaussian distribution with a common mean.

Our primary contributions in this section include:

1. An on-line IP clustering algorithm that incorporates Internet-specific knowledge.

2. Validation of the algorithm's ability to detect both structural and dynamic network changes.

3. Application of clustering for latency prediction on synthetic and real data demonstrating superior performance compared to prior approaches.

Network Boundaries

First, we examine the network's notion of providing aggregation and administrative boundaries on power of two multiples. IP routing and address assignment uses the notion of a

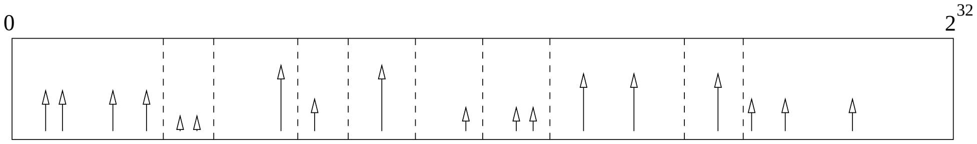


Figure 4-21: Example clustering: arrow location represents training sample IP addresses, length reflects latency. Dashed lines impose a partitioning on the address space that allows for accurate predictions.

prefix. The bit-wise AND (&) between a prefix p and a netmask m denotes the network portion of the address (m effectively masks the "don't care" bits of an address). We employ the common notation p/m as containing the set of IPv4 addresses inclusive of:

$$p/m := [p, p + 2^{32-m} - 1] \tag{4.20}$$

Thus, p/m contains 2^{32-m} addresses[5]. For example, the prefix 2190476544/24, corresponding to 130.144.5.0/24, includes 2^8 address from 130.144.5.0 to 130.144.5.255. The number of addresses within any prefix (p/m) is always a power of two. Our algorithm incorporates this invariant as described in §4.3.1.

But a prefix also implies a contiguous group of addresses under common administrative control. A naïve approach might assume that two numerically contiguous 32-bit IP addresses $a_1 = 318767103$ and $a_2 = 318767104$ are likely to be under the same control. However, by understanding the way in which networks are allocated, an educated observer might come to a different conclusion given that the dotted-quads are: $a_1 = 18.255.255.255$ and $a_2 = 19.0.0.0$. Both a_1 and a_2 may in fact belong to a larger aggregate, 18.0.0.0/7. However it is more likely that an address that differs from a_1 by -100, e.g. $a_3 = 18.255.255.155$, is part of the same network than a_2 which differs by only one.

Network Dynamics

Despite demonstrable benefit, the non-stationary nature of network problems presents a challenging environment for machine learning. In the context of latency prediction, the model may produce poor predictions due to either structural changes or dynamic conditions. A structural change might include a new link in the network which adds additional latency to a set of destinations, while congestion might temporarily influence predictions.

In the trivial case, an algorithm can remodel the world by purging old information and explicitly retraining. Completely re-learning the problem can be beneficial in some cases, but is typically expensive and unnecessary when only a portion of the network has changed. Further, even if a portion of the learned model is stale and providing inaccurate results, *forgetting stale training data may lead to even worse performance.* We desire an algorithm

[5] We use '$\frac{a}{b}$' to denote numerical division.

where the underlying model is easy to update on a continual basis and maintains acceptable performance during updates.

4.3.1 Clustering Algorithm

Given evidence of secondary structure, network boundaries and network dynamics, we consider an algorithm specific to clustering IP addresses.

Overview

Our algorithm takes as input a network prefix (p/m) and some number of training points (X) distributed within the prefix. The initial input is typically the entire IP address space (0.0.0.0/0) and all training points. Each training point $X = \{x_1, \ldots, x_n\}$ consists of an (IP, value) pair $x_i := (IP_i, val_i)$.

Define split s_i as inducing 2^i partitions, p_j, on p/m. Then for $j = 0, \ldots, 2^i - 1$:

$$p_j = p + j2^{32-(m+i)}/m + i \tag{4.21}$$

Let $x_i \in p_j$ iff the address of x_i falls within prefix p_j (Eq. 4.20). The general form of the algorithm is:

1. Compute mean of data point values: $\mu = \sum \frac{X(val_i)}{n}$
2. Add the input prefix and associated mean to a radix trie (§4.3.1): $R \leftarrow R + (p/m, \mu)$
3. Splits the input prefix to create potential partitions (Eq. 4.21): Let $p_{i,j}$ be the j'th partition of split s_i.
4. Let N be $x_i \in p_{i,j}$, let M be $x_i \notin p_{i,j}$.
5. Over each split granularity (i), compute the t-statistic for each potential partition j (§4.3.1): $t_{i,j} = ttest(N(val), M(val))$.
6. Find the partitioning that minimizes the t-test:
   $(\hat{\imath}, \hat{\jmath}) = \underset{i,j}{\operatorname{argmin}}\ t_{i,j}$
7. Recurse on the maximal partition(s) induced by $(\hat{\imath}, \hat{\jmath})$ while the t-statistic is less than $thresh$ (§4.3.1).

We refine the algorithm and present details in this Section, but first draw attention to several useful features of our learning technique:

- *Complexity:* A natural means to penalize complexity. Intuitively, clusters containing very specific prefixes, e.g. /30's, are likely over-fitting. Rather than attempting to tune traditional machine learning algorithms indirectly, limiting the minimum size of inferred prefixes corresponds directly to network generality.

- *Memory:* A natural means to bound memory consumption. Because the trie structure provides longest-match lookups, the algorithm can sacrifice accuracy for lower memory utilization by bounding tree depth or width.
- *Change Detection:* Allows for direct analysis on trie nodes. In a dynamic environment, change point analysis on these individual portions can determine if part of the underlying network has changed.
- *On-Line Learning:* When re-learning stale information, the longest match nature of the trie implies that once the information is discarded, in-progress predictions will use the next available longest match which is likely to be more accurate than an unguided prediction.
- *Active Learning:* For any given training data, the tree is likely unbalanced, providing a natural strategy to perform active learning. While guiding the learning this way decouples the training from testing, sparse portions or poorly performing portions of the trie are easily identified.

The last three points are especially important in providing an algorithm that represents a step forward in making on-line learning algorithms practical in dynamic networks.

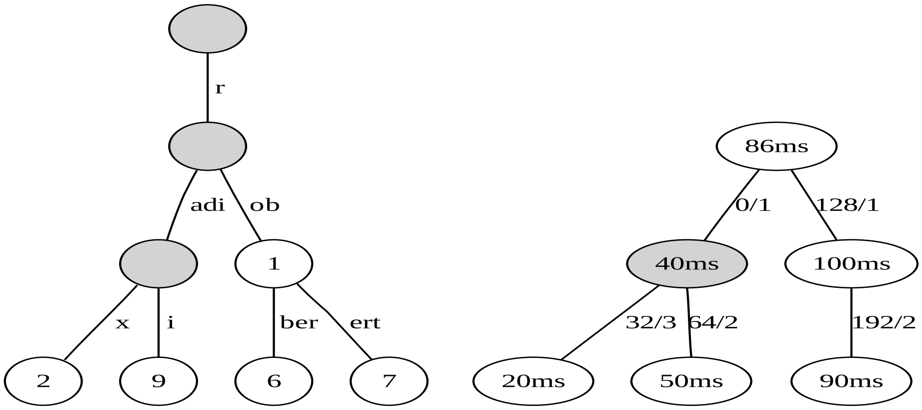
Cluster Data Structure

A radix tree, or Patricia trie [110], is a form of compressed trie that stores strings. In contrast to normal tries, the edges may be labeled with multiple characters. Radix trees are an efficient data structure for storing small sets of long strings that share common prefixes. An example of a radix tree on an English alphabet is given in Figure 4-22. The tree stores the following (string, value) pairs: `radix = 2, radii = 9, rob = 1, robert = 6, robber = 7`.

Radix trees support lookup, insert, delete and find predecessor operations in $O(k)$ time where k is the maximum length of all strings in the set. By using a binary alphabet, strings of $k = 32$ bits and nexthops as values, radix trees support IP routing table longest match lookup, an approach suggested by Sklower [136] and others. We adopt radix trees to store our algorithm's inferred structure and provide predictions.

Evaluating Potential Partitions

Student's t-test [64] is a popular test to determine the statistical significance in the difference between two sample means. We use the t-test in our algorithm to evaluate potential partitions of the address space at different split granularity. The t-test is useful in many practical situations where the population variance is unknown and the sample size too small to estimate the population variance.



(a) Radix tree encoding strings `radix = 2, radii = 9, rob = 1, robert = 6, robber = 7`

(b) Radix tree encoding IP prefixes and their associated mean latencies

Figure 4-22: Example radix trees, illustrating how prefixes with common mean latencies might be stored.

The t-test is founded on the t-distribution. The t-distribution depends on the degrees of freedom (*df*) where $df = n - 1$ for sample size n. As *df* grows large, the t-distribution tends toward the normal distribution. Figure 4-23 shows the normal distribution and t-distribution for varying degrees of freedom.

The t-test is a statistical hypothesis test. The null hypothesis is that two normally distributed populations have the same mean. Thus, the test statistic has a t-distribution if the null hypothesis is true. Given two sample means, $\bar{X}_1$ and $\bar{X}_2$, the t-test is the ratio of the difference between means and the standard error of the difference between means:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_{\bar{X}_1 - \bar{X}_2}} \tag{4.22}$$

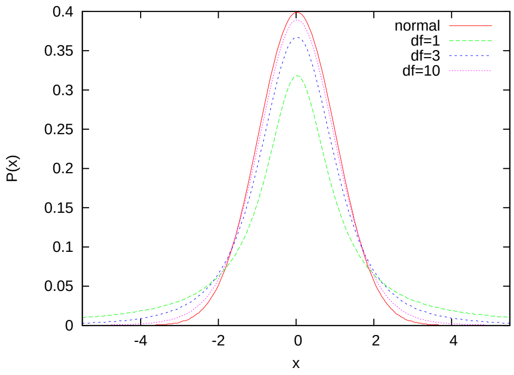


Figure 4-23: t-distribution probability densities for various degrees of freedom (*df*). As *df* grows large, the t-distribution approximates the normal distribution.

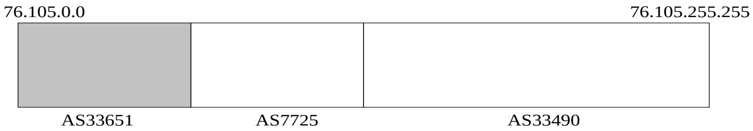


Figure 4-24: True allocation of `76.105.0.0/16`. For an inferred difference between shaded and non-shaded regions, we ensure a maximal prefix split.

where the standard error is an estimate of the standard deviation of the error in a measured process. As the means grow apart, the t statistic increases, while t decreases as the sample variances become smaller. For unequal sample sizes:

$$s_{\bar{X}_1 - \bar{X}_2} = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)} \tag{4.23}$$

where $\bar{X}_1 - \bar{X}_2$ is the difference between the sample means, n_1 and n_2 are the size of the two samples, and s^2 is the unbiased sample variance estimator. The t-test gives the probability that under the null hypothesis, i.e. that the two samples come from populations with equal means, a given t value or larger would arise by chance. Because the t-distribution is symmetrical, a two-sided test gives the probability that the absolute value of t or larger might arise by chance.

Maximal Prefix Splits

Assume the t-test procedure identifies a “good” partition of an IP address space. The partition defines two chunks (not necessarily contiguous), each of which contains data points with statistically different characteristics. We ensure that each chunk is valid (Appendix, definition 1) within the constraints in which networks are allocated. If a chunk of address space is not valid for a particular partition, it must be split further. Therefore, our algorithm creates *maximal valid prefixes* to ensure generality.

Consider the prefix 76.105.0.0/16 in Figure 4-24. Say the algorithm determines that the first quarter of this address space (shaded) has a statistically different mean latency from the rest. How can we use the domain-specific knowledge to partition the remaining three-quarters of addresses?

The first point of note is that the partition from 76.105.64.0 to 76.105.255.255 is not valid. One might divide the space into three equally sized 2^{14} valid prefixes. However, we wish to create maximally sized prefixes to capture the true hierarchy as well as possible given sparse data. In actuality, a current BGP table reveals that the prefix is split into at least three pieces, each advertised by a different autonomous system (AS). The IP address registries list 76.105.0.0/18 as being in Sacramento, CA, 76.105.64.0/18 as Atlanta, GA and 76.105.128.0/17 in Oregon.

Algorithm 4.2 describes our method for ensuring maximal valid prefixes. We provide a

Table 4.2: Examples of maximal IP prefix division

128.61.0.0 → 128.61.255.255	128.61.0.0 → 128.61.4.1	16.0.0.0 → 40.127.255.255
128.61.0.0/16	128.61.0.0/22	16.0.0.0/4
	128.61.4.0/31	32.0.0.0/5
		40.0.0.0/9

proof of correctness in the Appendix. The basic intuition is to determine the largest power of two chunk that could potentially fit into the address space. If a valid starting position for that chunk exists, the algorithm recurses on the remaining sections. Otherwise, it divides the maximum chunk into two valid pieces. Table 4.2 gives three example divisions.

Algorithm 4.2 *divide*(*start*, *end*): return maximal-sized (prefix,mask) list that divides *start* to *end* IPs

```
$R$, an IP prefix table
$max = \lfloor \log_2(end - start + 1) \rfloor$
$maxsize = 2^{max}$
$mask = 2^{32} - maxsize$
$first = start \ \& \ mask$
if ($first \neq start$) then
  $first \leftarrow first + maxsize$
if ($first + maxsize - 1 > end$) then
  $maxsize \leftarrow maxsize/2$
  $mask = 2^{32} - maxsize$
  $first = (start \& mask) + maxsize$
  $R \leftarrow R+$ divide($start, first - 1, results$)
  $R \leftarrow R+$ divide($first, first + maxsize - 1, results$)
  $R \leftarrow R+$ divide($first + maxsize, end, results$)
else
  if ($first \neq start$) then
    $R \leftarrow R+$ divide($start, first - 1, results$)
  $R \leftarrow (first, 32 - max)$
  if ($first + maxsize - 1 \neq end$) then
    $R \leftarrow R+$ divide($first + maxsize, end, results$)
return($R$)
```

Full Algorithm

Given the basic outline of the algorithm, the data structures, a method to create and evaluate partitions, and notion of maximal prefixes, we present the complete algorithm. Our formulation is based on a divisive approach; agglomerative techniques that build partitions up are a potential subject for further work. Algorithm 4.3 takes a prefix p/m along with the data samples for that prefix: $x_i \in X \forall x_{ip} \in p/m$.

The algorithm first computes the mean μ of the X values and adds an entry to radix table R containing p/m pointing to μ (lines 1-4). In lines 5-12, we create partitions p_j

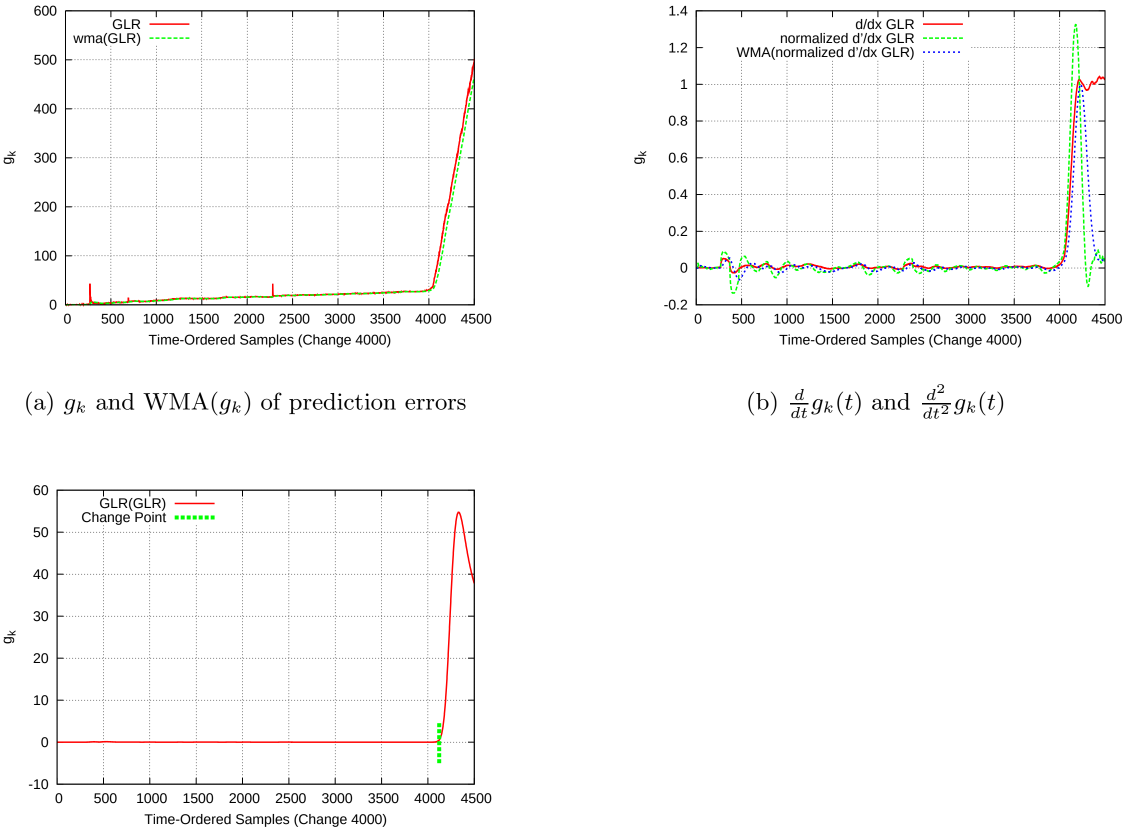
at a granularity of s_i as described in Eq. 4.21. For each $p_{i,j}$, line 13 evaluates the t-test between points within and without the partition. Thus, for s_3, we divide p/m into eighths and evaluate each partition against the remaining seven. We determine the lowest t-test value t_{best} corresponding to split i_{best} and partition j_{best}.

If no partition produces a split with t-test less than a threshold, the algorithm terminates that branch of splitting. Otherwise, lines 16-25 use algorithm 4.2 to divide the best partition into maximal valid prefixes, each of which is placed into the set P. Finally, the algorithm recurses on each prefix in P.

Algorithm 4.3 *split*$(p/m, X)$:

```
     $R$, an IP prefix table
     $b \leftarrow 32 - m$
     $\mu \leftarrow \text{mean}(X(val))$
     $R \leftarrow R + (p/m, \mu)$
 5:  for $i \leftarrow 1$ to 3 do
       for $j \leftarrow 0$ to $2^i - 1$ do
         $p_j \leftarrow p + j2^{b+i}/(m-i)$
         for $x \in X$ do
           if $x_{ip} \in p_j$ then
10:          $N \leftarrow N + x_{val}$
           else
             $M \leftarrow M + x_{val}$
           $t_{i,j} \leftarrow \text{ttest}(N, M)$
     $t_{best}, i_{best}, j_{best} \leftarrow \underset{i,j}{\text{argmin}}\ t_{i,j}$
15:  if $t_{best} < thresh$ then
       $last \leftarrow p + 2^b - 1$
       $start \leftarrow p + (j_{best})2^{b+i_{best}}$
       $end \leftarrow start + 2^{b+i_{best}} - 1$
       $P \leftarrow start/(m - i_{best})$
20:    if $start = p$ then
         $P \leftarrow P+$ divide($end + 1$, $last$)
       else if $end = last$ then
         $P \leftarrow P+$ divide($p$, $start - 1$)
       else
25:      $P \leftarrow P+$ divide($end + 1$, $last$)
         $P \leftarrow P+$ divide($p$, $start - 1$)
     for $p_d/m_d \in P$ do
       $X_d \leftarrow x_{ip,val} \in X \forall ip \in p_d/m_d$
       $R \leftarrow R+$ split($p_d/m_d, X_d$)
30:  return $R$
```

The output after training is a radix trie which defines clusters. Subsequent predictions are made by performing longest prefix matching on the trie.



(a) g_k and WMA(g_k) of prediction errors

(b) $\frac{d}{dt}g_k(t)$ and $\frac{d^2}{dt^2}g_k(t)$

(c) Impulse triggered change detection

Figure 4-25: Modified GLR to accommodate learning drift; synthetic change injected beginning at point 4000.

4.3.2 Handling Network Dynamics

An important feature of the algorithm is its ability to accommodate network dynamics. However, first the system must detect changes in a principled manner. Each node of the radix trie naturally represents a part of the network structure. Therefore, we may run traditional change point detection [11] methods on the prediction error of data points classified by a particular node. If the portion of the network associated with a node exhibits structural or dynamic changes, evidenced as a change in prediction error mean or variance respectively, we may associate a cost with retraining. For instance, pruning a node close to the root of the trie represents a large cost which must be balanced by the magnitude of the current errors on predictions under that node[6].

When considering structural changes, we are concerned with a change in the mean error resulting from the prediction process. Assume that predictions produces errors ($\mathbf{e}$) from a Gaussian probability distribution parameterized by a mean and variance, $\theta_0 = N(\mu_0, \sigma_0)$. As we cannot assume, a priori, knowledge of how the processes' parameters will change, we turn to the GLR test statistic (Eq. 4.17) which detects a statistical change from μ_0 (mean before change). Unfortunately, GLR is typically used in a context where μ_0 is well-known, e.g. manufacturing processes. Figure 4-25(a) shows g_k as a function of ordered prediction errors produced from our algorithm on real Internet data. Beginning at the 4000th prediction, we create a synthetic change by adding 50ms to the mean of every data point (thereby ensuring a 50ms error for a normally perfect prediction). We use a weighted moving average (WMA) to smooth effects of noise in the data. The change is clearly evident. Yet g_k drifts even under no change since μ_0 is estimated from training data error which is necessarily less than the error mean produced by forming test predictions.

This GLR drift effect is problematic because we cannot easily determine a threshold for detecting a change. Any arbitrary threshold will eventually be reached even without a change as the g_k statistic drifts over time. While one threshold value might work in one particular situation, it could be falsely triggered or never triggered in other situations. Ideally, we would like a non-parametric means to set the GLR change point threshold.

To combat GLR drift, we take the derivative of the weighted moving average of g_k with respect to sample time. Doing so produces the step function in Figure 4-25(b). The intuition behind taking the first derivative lies in the fact that while g_k drifts, it drifts at a constant rate with no change. Thus, the first derivative of a line with constant slope is a constant. We use the weighted moving average to remove noise from the signal. After a change, the value of $\frac{d}{dt}g_k(t)$ changes to a new, higher-valued constant. Again, we run into the dilemma of setting an appropriate threshold to detect change. Instead, we impulse trigger on the change point by taking the second derivative as depicted in Figure 4-25(c). As the derivative of a constant is zero, we expect $\frac{d^2}{dt^2}g_k(t)$ to be zero except at the point of

[6] We do not attempt to tackle the problem of defining this cost, but rather point out that our algorithm naturally supports such cost balancing operations.

a change where there is an impulse (corresponding to the step in the first derivative).

This simple change to the standard GLR algorithm thus provides a principled means to perform change point analysis in our learning environment without having to set arbitrary thresholds.

Dynamic Internet changes, for instance congestion events, often occur on short time scales and present as a change in the variability of our prediction error. Experiments modeling variance detection with the Inclan [75] algorithm show promise, but we defer a complete exposition to later work.

4.3.3 Results

A distinct disadvantage of inference in the domain of Internet measurement is that it is difficult if not impossible to know ground truth. In order to better quantify the performance of our algorithm in a controlled environment, we first build a synthetic data set.

A Representative Synthetic Model

While several network simulators and topology generators exist [101, 29], none maps addresses to networks or nodes in a way characteristic of the Internet. Algorithm 4.4 presents our method to generate Internet-like address structure and associate latencies with prefixes.

The algorithm is invoked with the argument *basenets* to create a particular number of initial base networks corresponding to service provider independent space top-tier network blocks. Because the Internet is biased toward classful netblocks, we choose the base network to have a netmask of 8 with probability 0.01, 16 with probability 0.2 and 24 with probability 0.79 (lines 1-9). These values are empirically chosen to match distributions observed in real BGP data; we examine the prefix distribution next. Each base network is chosen at random to be a valid prefix p/m (lines 10-11). Next, each base prefix is evenly split into 2^i subnets $s/m+i$ for i taken from a normal distribution with mean $m+2$ and variance 1. Each subnet is assigned a random latency between $[0, 200]$ms and added to a prefix tree (lines 12-18). Finally, a quarter of the resulting prefixes are picked for an uneven divide in order to model the noise and randomness typical in real network structure. We pick a random point at which to "split" the prefix and then again invoke Algorithm 4.2 to create maximal sized prefixes (lines 19-29).

We do not make any quantitative claims as to how accurately our method reproduces the true Internet address structure. However, Figure 4-26 compares the prefix distributions observed in a routeviews [105] real Internet BGP table with prefixes generated by our algorithm. Our algorithm captures the predominance of prefixes with /24 netmasks as well as the modes at /16 and /8.

To simulate provider assigned space, for example providers that allocate portions of their own address space to customers, we choose half of the prefixes at random (lines 13-16). For each randomly chosen prefix, we pick a random "split" point. We use algorithm 4.2 again

Algorithm 4.4 *build*(*basenets*): Build a synthetic "Internet-like" routing table with random round-trip latencies

```
$R$, an IP prefix table
for $i \leftarrow 0$ to $basenets$ do
    $z \leftarrow [0, 1]$
    if $z \leq 0.01$ then
        $mask \leftarrow 8$
    else if $z \leq 0.2$ then
        $mask \leftarrow 16$
    else
        $mask \leftarrow 24$
    $prefix \leftarrow [0, 2^{32} - 2^4]$
    $prefix \leftarrow prefix$ & $(2^{32} - 2^{32-mask-2})$
    $subnet \leftarrow N(mask, 1)$
    $subnets \leftarrow 2^{subnet-mask-2}$
    $subnetsize \leftarrow 2^{32-subnet}$
    for $j \leftarrow 0$ to $subnets$ do
        $subprefix \leftarrow prefix + j * subnetsize$
        $rtt \leftarrow [0, 200]$
        $R \leftarrow R + (subprefix/subnet, rtt)$
repeat
    $r \leftarrow R, r \notin S$
    $S \leftarrow S + r$
until $|S| < |R|/2$
for $r$ in $S$ do
    $netsize \leftarrow 2^{32-r_{mask}}$
    $split \leftarrow [1, netsize - 1]$
    $P \leftarrow \text{divide}(r_{prefix}, r_{prefix} + split)$
for $r$ in $P$ do
    $rtt \leftarrow [0, 200]$
    $R \leftarrow R + (r_{prefix}/r_{mask}, rtt)$
return($R$)
```

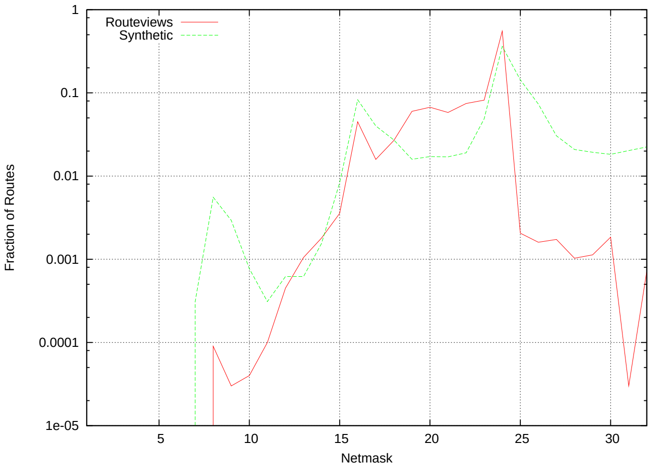


Figure 4-26: Real vs. synthetic prefix distribution: note peaks at classful prefix sizes in both models.

to form valid, maximally sized prefixes given this random split point (lines 17-20). Each resulting prefix is again added to the prefix tree with a random latency between zero and 200ms (lines 21-23).

To generate the synthetic data set, we use Algorithm 4.5 to randomly sample from the network address model of Algorithm 4.4. For each random address with longest prefix match p in the model, we assign a latency drawn from a Gaussian centered around p's latency: $N(p_{rtt}, 5ms)$.

Algorithm 4.5 $synthesize(R, num)$: Randomly sample addresses within a routing table. Assign each address an RTT according to a Gaussian distribution on its prefix's latency.

```
$R$, a synthesized IP prefix table
$D$, synthesized random data points
for $i \leftarrow 0$ to $num$ do
  repeat
    $a \leftarrow [0, 2^{32}]$
    $p \leftarrow \text{longestmatch}(R, a)$
  until $p \neq \emptyset$
  $l \leftarrow \text{gaussian}(p_{rtt}, 5)$
  $D \leftarrow D + (a, l)$
return$(D)$
```

We use Algorithm 4.5 create 20,000 $(ip, latency)$ data point pairs and run our IP clustering algorithm. To ensure generality, we randomly permute the data set so that the test and training sets are different between experiments. The training set defines the inferred network structure; based on the resulting radix trie, the algorithm performs longest prefix matching to form predictions. Figure 4-27 depicts the prediction performance on our synthetic data set averaged over five experiments as a function of training size. Encouragingly, the test error decreases monotonically across the extent of our data without any signs of

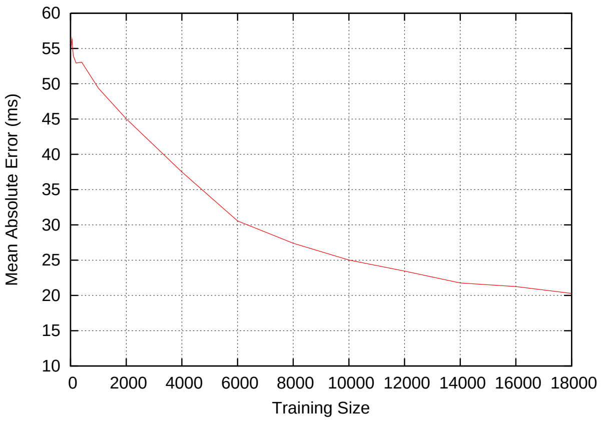


Figure 4-27: Latency prediction performance on synthetically generated data.

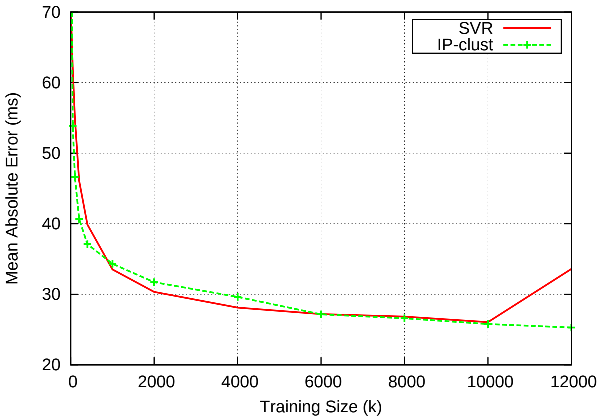


Figure 4-28: Prediction performance of IP clustering algorithm vs. support vector regression.

over fitting. With 18,000 training samples, the mean prediction test error decreases to under 20ms. With 8,000 or more training points, the prediction error is under 30ms, a range suitable for a broad range of network application tasks.

Performance on Internet Data

Given the performance on synthetic data, we study the performance of the algorithm on our real data set. Figure 4-28 depicts the mean prediction error as a function of training size for both the previous support vector regression (SVR) methodology [25] and our IP clustering algorithm. The clustering algorithm outperforms SVR across much of the range, particularly with few or many training examples. While the performance is comparable, it is achieved with faster, deterministic $0(k), k = 32$ performance and the ability to accommodate network dynamics.

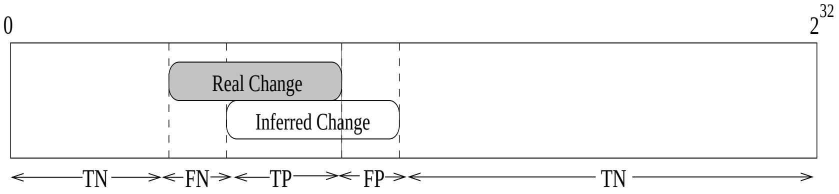


Figure 4-29: Change point game: overlap between the real and inferred change provide true negatives (TN), false negatives (FN), true positives (TP) and false positives (FP).

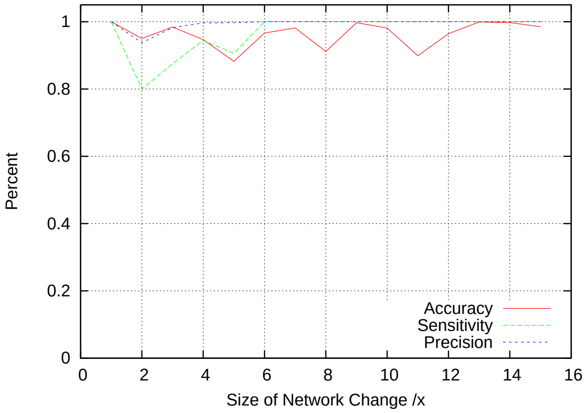


Figure 4-30: Change detection performance as a function of changed network size.

Performance under Dynamics

Finally, we consider performance under dynamic network conditions. To evaluate our algorithm's ability to handle a changing environment, we formulate the induced change point game of Figure 4-29. Within our real data set, we artificially create a mean change that simulates a routing event or change in the physical topology. We create this change only for data points that lie within a randomly selected prefix. Note that this prefix is picked purely randomly and is not a function of the learned tree model. The game is then to determine the algorithm's ability to detect the change for which we know the ground truth.

Let the shaded portion of the figure indicate the true change we create within the entire IPv4 address space. The non-shaded portion represents the algorithm's best prediction of where a change occurred. We take the fraction of overlap to indicate the false negatives, false positives and true positives. The remainder of the IP address space comprises the true negatives.

Figure 4-30 shows the performance of our change detection technique in relation to the size of the artificial change. For example, a network change of /2 represents one-quarter of the entire 32-bit IP address space. Again, for each network size we randomly permute our data set, artificially induce the change and measure the resulting error rates. We see that across the range of network changes our algorithm performs well, with greater than 80% accuracy, sensitivity and precision.

4.3.4 Conclusions and Future Work

This work describes an IP-specific clustering algorithm amenable to network dynamics experienced in an Internet environment. While this work represents a step forward in allowing for feasible learning within the network, it presents many topics for further study.

Our algorithm attempts to find appropriate partitions by using a sequential t-test. We have informally analyzed the stability of the algorithm with respect to the choice of optimal partition, but wish to apply a principled approach similar to random forests. In this way, we plan to form multiple radix tries using the training data sampled with replacement. We then may obtain predictions using a weighted combination of trie lookups for greater generality.

While we demonstrate the algorithm's ability to detect changed portions of the network, further work is needed in determining the tradeoff between pruning stale data and the cost of retraining. Properly balancing this tradeoff requires a better notion of utility and further understanding the time-scale of Internet changes. Our initial work on modeling network dynamics by inducing increased variability shows promise in detecting short-term congestion events. Additional work is needed to analyze the time-scale over which such variance change detection methods are viable.

Thus far, we examine synthetic dynamics on real data such that we are able to verify our algorithm's performance against a ground truth. In the future, we wish to also infer real Internet changes and dynamics on a continuously sampled data set.

Finally, our algorithm suggests at many interesting methods of performing active learning, for instance by examining poorly performing or sparse portions of the trie, which we plan to investigate going forward.

4.3.5 Proofs

Definition 4.3.1. *For b-bit IP routing prefixes $p/m; p \in \{0,1\}^b$ $m \in [0,b], b = 32$ is valid iff $p = p \ \& \left(2^b - 2^{b-m}\right)$.*

Lemma 4.3.2. *The largest valid prefix of $[s,e]$ has $n = 2^m$ addresses, $m = \lfloor \log_2(e-s-1) \rfloor$, iff $\exists z, (0 \leq z < e-s)$ such that prefix $(s+z) : b-m$ is valid and $s+z+n \leq e-1$.*

Proof. Suppose $\exists z, (0 \leq z < e-s)$ such that prefix $(s+z) : b-m$ is valid for $m = \lfloor \log_2(e-s-1) \rfloor + 1$. Then $s+z+n \leq e$, so $s + 2^m \leq e-1$ or $2^m \leq e-s-1$. By the definition of floor, $m > \log_2(e-s-1)$, therefore: $e-s-1 < 2^m \leq e-s-1$ which contradicts the supposition. □

Lemma 4.3.3. *If there exists no prefix of size $n = 2^m$ addresses, $m = \lfloor \log_2(e-s-1) \rfloor$, which is both valid and within $[s,e]$, then two prefixes of size $n = 2^{m-1}$ are the largest valid prefixes.*

Proof. Suppose $\nexists z, (0 \leq z < e - s)$ such that prefix $(s + z) : b - (m - 1)$ is valid and $s + z + n \leq e - 1$. Then, there is no address in the range with $m - 1$ least significant bits zero: $\nexists a \in [s, e], a \in \{0, 1\}^*\{0\}^{m-1}$. Therefore, there are at most $2^{m-1} - 1$ addresses in the range $[s, e]$. This contradicts the fact that there must be at least $2^m - 1$ addresses. Further, suppose $\nexists z', (z \leq z' < e - s)$ such that prefix $(s + z') : b - (m - 1)$ is valid and $s + z' + n \leq e - 1$. Then by the same reasoning, there must be fewer than $2(2^{m-1} - 1)$ addresses in $[s, e]$. Thus, $2^m \leq 2^m - 2$, which is a contradiction. □

4.4 Problem 2: Transport-Level Characteristics of Spam

By all estimates, unsolicited email (spam) is a pressing and continuing problem on the Internet. A consortium of service providers reports that across more than 500M monitored mailboxes, 75% of all received mail is spam, amounting to more than 390B spam messages over a single quarter [103]. Spam clogs mailboxes, slows servers and lowers productivity. Not only is spam annoying, it adversely affects the reliability and stability of the global email system [2].

Popular methods for mitigating spam include content analysis [100, 130], collaborative filtering [139, 121], reputation analysis [140, 138], and authentication schemes [6, 150]. While effective, none of these methods offer a panacea;spam is an arms race where spammers quickly adapt to the latest prevention techniques.

We propose a fundamentally different approach to identifying spam that is based on two observations. First, spam's low penetration rate requires spammers to send extremely large volumes of mail to remain commercially viable. Second, spammers increasingly rely on zombie "botnets," [49] large collections of compromised machines under common control, as unwitting participants in sourcing spam [77]. Botnet hosts are typically widely distributed with low, asymmetric bandwidth connections to the Internet. Combining these observations we make the following hypothesis: the network links and hosts which source spam are constrained. We ask whether *the transport level characteristics of email flows provide sufficient statistical power to differentiate spam from legitimate mail (ham).*

In investigating this hypothesis, we gather a live data set of email messages and their corresponding TCP [119] packets. We extract and examine per-email flow characteristics in detail. Based on the statistical power of these flow features, we develop "SpamFlow," a spam classifier. In contrast to existing approaches, SpamFlow relies on neither content nor reputation analysis; Figure 4-31 shows this relation. Using machine learning feature selection, SpamFlow identifies the most selective flow properties, thereby allowing it to adapt to different users and network environments.

By examining email at the transport layer, we hope to exploit a fundamental weakness in sourcing spam, the requirement to send large quantities of mail on resource constrained links. As the volume of spam is unlikely to abate, SpamFlow represents a new defense

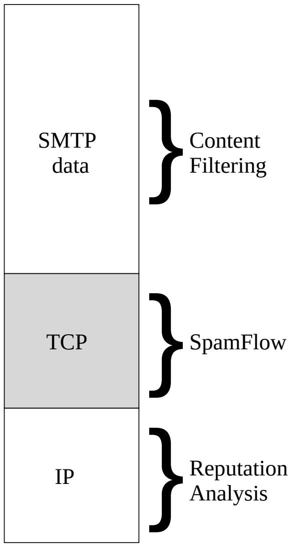


Figure 4-31: The relation of SpamFlow to existing spam mitigation schemes. SpamFlow analyzes only transport-layer packet headers and does not rely on the IP header or data payload.

against a significant source of unwanted mail. This section's primary contributions are:

1. Identification of flow features that exhibit significant probability differences between spam and ham. E.g. 99% of ham flows, compared to only 24% of spam flows, have an RTT $\leq$ 100ms.

2. SpamFlow, a classifier to learn and leverage these statistical differences for $>$ 90% accuracy, precision and recall without content or reputation analysis.

3. Correct identification of 78% of the false negatives generated by SpamAssassin [100]. Thus, SpamFlow may usefully be combined with existing techniques as a weighted vote.

Consequently, we hope this work serves to identify a new area of spam research and presents a working system which sources of spam cannot easily evade.

4.4.1 Insight from Mail Transport

The intuition behind our scheme is simple. Because spammers must send large volumes of mail, they transmit mail continuously, asynchronously and in parallel. In addition, the sources of spam are frequently large compromised "botnets," which are resource constrained and typically connected to the Internet by links with asymmetric bandwidths, e.g. aDSL and cable modems.

Therefore the flows that comprise spam TCP traffic exhibit behavior consistent with traffic competing for link access. Thus, there is reason to believe that a spammer's traffic is

(a)
```
Received: from vms044pub.verizon.net
From: "Dr. Beverly, MD" <b@ex.com>
Subject: thoughts

Dear Robert, I hope you have had
a great week!
```

(b)
```
Received: from unknown (59.9.86.75)
From: Erich Shoemaker <ried@ex.com>
Subject: Repl1ca for you

A T4g Heuer w4tch is a luxury statement on
its own. In Prest1ge Repl1cas, any T4g
Heuer...
```

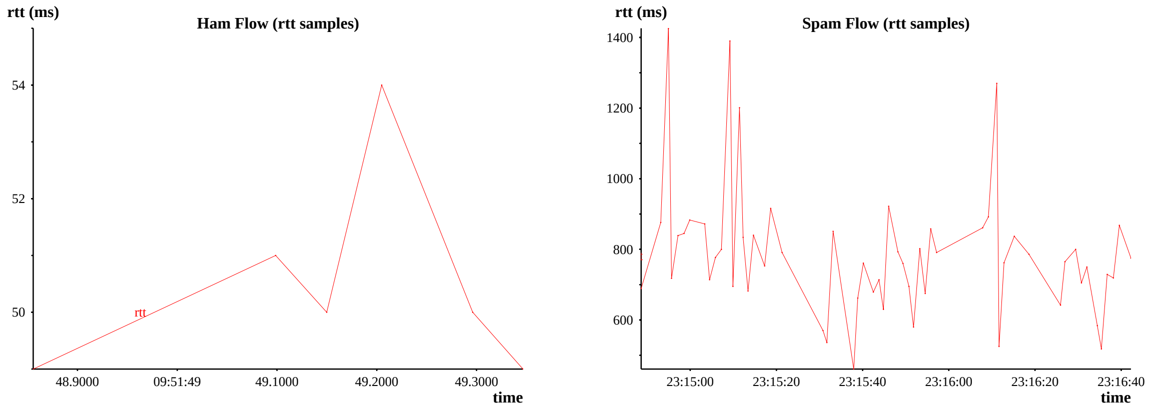


Figure 4-32: Comparison of (a) ham and (b) spam flow RTT estimation. Note that the RTT variance differs by an order of magnitude - an indication of flow congestion effects.

more likely to exhibit TCP timeouts, retransmissions, resets and highly variable round trip time (RTT) estimates. Figure 4-33 illustrates a compromised host sending mail to many remote mail servers simultaneously. The resulting contention on the host's connection to the Internet manifests as TCP congestion, loss and reordering effects.

Our initial impetus to investigate this hypothesis came from observing seemingly anomalous TCP socket behavior on an MTA. We discuss this real-world socket behavior next.

Socket Behavior

For intuition on the real-world behavior of SMTP [88] flows, we manually investigate connections on a busy mail server. A large fraction of sockets remain in particular TCP states for an unexpectedly long time. While [24] presents a detailed analysis, a brief summary is revealing:

SYN_SENT: Many sockets remain in the SYN_SENT state for an extended period of time, where the remote server sends a SYN, but fails to acknowledge our MTA's response. These connections may be indicative of mail server scans or congestion near the source.

LAST_ACK: Many sockets are stuck in the LAST_ACK state. Manual investigation shows senders successfully opening a TCP connection and sending SMTP commands. However, the senders do not reliably receive our MTA's acknowledgments, leading to many retransmissions. Often the remote server eventually closes its connection by sending a FIN.

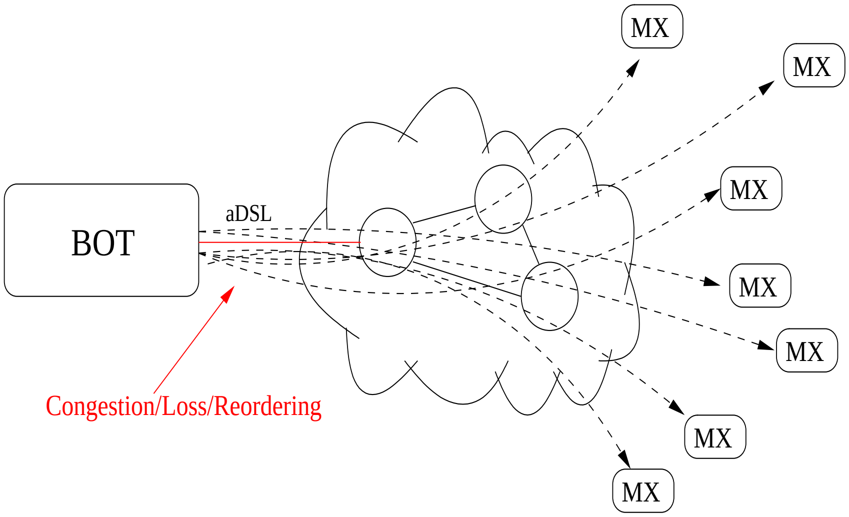


Figure 4-33: Compromised hosts unwittingly source spam as part of zombie "botnets." These bots are typically connected via low, asymmetric uplinks such as aDSL and are widely distributed across the Internet. The large volume of traffic combined with home connectivity can manifest as TCP packet stream congestion, loss and reordering.

Our MTA acknowledges the FIN and closes its connection, but the remote server never acknowledges this FIN. The appendix in §4.4.7(I), at the end of this section, contains a detailed example trace of the LACK_ACK situation.

FIN_WAIT1: Similarly, many sockets are stuck in state FIN_WAIT1. Inspecting these, we find remote servers that establish a connection but never send any data. Our MTA eventually times out, closing the TCP connection. However, the remote host seemingly disappears. Since the remote server does not acknowledge the FIN our MTA sends on the connection, the socket stays in FIN_WAIT. §4.4.7(II) provides a detailed trace of traffic that leads to this situation.

An Example

We manually examine two emails (one spam, one ham) from our data to provide further intuition. Figure 4-32 shows our MTA's round trip time (RTT) estimation for each. Figure 4-32(a) is a valid message; the RTT estimate is low and more importantly shows little variation across the flow's duration. In contrast, Figure 4-32(b) depicts a spam mail where not only is the RTT high, it varies significantly with a minimum less than 500ms and a maximum nearly 1.5s. Such wildly varying RTT estimates are often indicative of persistent congestion and may be one indirect sign that a flow belongs to a spammer.

4.4.2 Experimental Methodology

Is it reasonable to believe that a spammer's TCP/IP traffic characteristics are sufficiently different than traffic from Mail Transport Agents (MTAs) sending legitimate mail? Clearly,

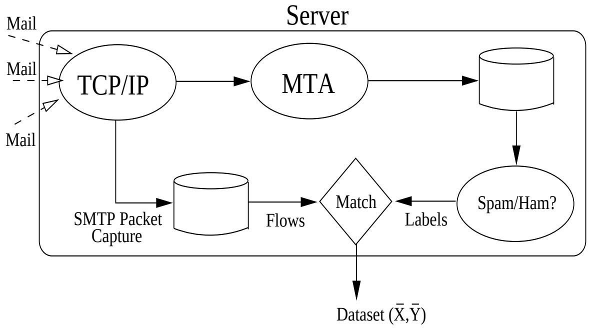


Figure 4-34: Data collection: incoming SMTP packets are captured and coalesced into flows. Each flow is matched with a binary spam or ham ground-truth label.

traffic for the connections we manually investigate is suggestive of abnormal behavior or resource exhaustion. To systematically understand large-scale behavior, we instrument an MTA to collect passive flow data for the email messages it receives.

Data Collection

Figure 4-34 depicts our collection methodology. Our server is connected to the local network via a non-congested 100Mbps Ethernet which is in turn connected to the wide area Internet via multiple diverse Gigabit-speed links. The server processes SMTP [88] sessions and writes emails to disk. In the header of each email, the server adds the remote IP and TCP port number of the remote MTA that sent the mail. Simultaneously the server passively captures and timestamps all SMTP packets. Each email is then manually labeled as spam or ham to establish ground truth.

We coalesce the captured email packets into flows. Let our server's IP address be S. Define a flow $f_{IP:port}$ as all TCP packets $(IP{:}port) \rightarrow (S{:}25)$ and $(S{:}25) \rightarrow (IP{:}port)$[7]. Each email message is matched with its corresponding SMTP flow. However, MTAs often receive many connections from the same remote host. To avoid the ambiguity in synchronizing messages in time, particularly when aggressive MTAs initiate multiple simultaneous connections, we modify our MTA (S) to include the incoming connection's source port number in each mail header. The port number is vital when receiving many, potentially simultaneous, emails from the same source.

Over the course of one week in January, 2008, we collect a total of 18,421 messages for a single mail account, 220 of which were legitimate while the remaining 18,201 were spam (98.8%). Of the ham messages, 39 were from unique mail domains. While our data set includes only one account, it reveals surprising and evocative characteristics of spam in contrast to ham.

[7] Since our server's IP and SMTP port are fixed (S:25), these fields are not included in the flow tuple.

Table 4.3: Flow properties used as classification features

Feature	**Description**
Pkts	Packets
Rxmits	Retransmissions
RSTs	Packets with RST bit set
FINs	Packets with FIN bit set
Cwnd0	Times zero window advertised
CwndMin	Minimum window advertised
MaxIdle	Maximum idle time between packets
RTT	Initial round trip time estimate
JitterVar	Variance of interpacket delay

Formulating the Learning Task

We use the collected live data set to formalize a machine learning problem. Properties of each flow (f_i) provide the learning features ($\mathbf{x}_i$). Currently we extract the features in Table 4.3. While our flows are undirected, particular features are directional, for instance received and sent packet counts, RSTs, FINs and retransmissions. Including directional features, we consider 13 features in total for each flow.

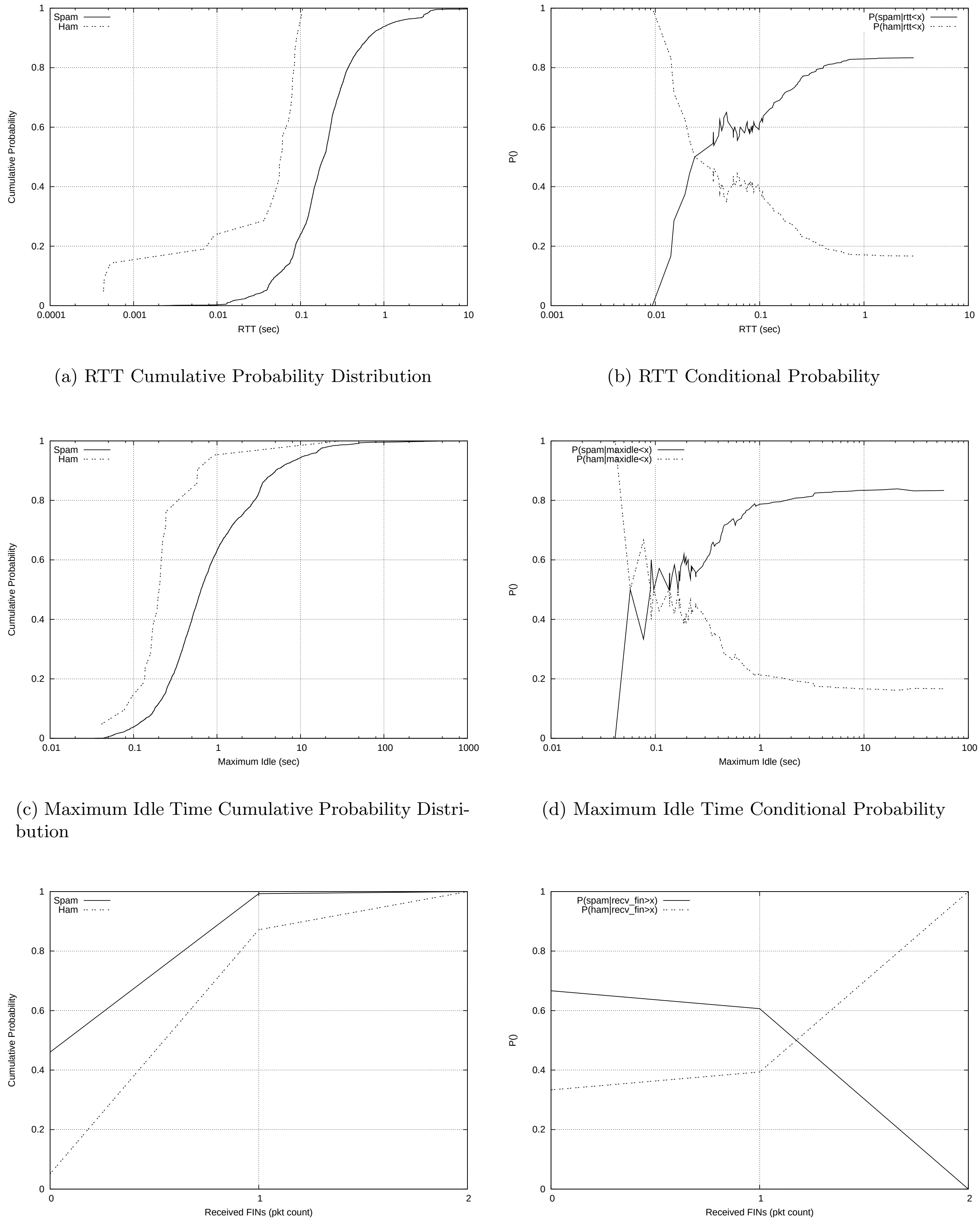
Each f_i corresponds to an email that is given a binary $y_i \in \{\pm 1\}$ label. Our data thus includes the input vector $\mathbf{x}_i \in \mathbb{R}^d, d = 13$ for flow f_i and label y_i. From these features, we wish to determine which provide the most discriminative power in picking out spam and how the number of training examples affects performance.

Transport Characteristics

Figure 4-35 compares the RTT, maximum idle time and FIN packet count between ham and spam in the entire data set. Here we define the RTT as the initial RTT estimate inferred by the three-way TCP handshake. Figure 4-35(a) shows the cumulative distribution of RTT times in our data. The difference between spam and ham is evident. While more than 20% of ham flows have an RTT less than or equal to 10ms, almost no spam flows have such a small initial RTT. The RTT of nearly all ham flows is 100ms or less. In contrast, 76% of spam flows have an RTT greater than 100ms.

A feature such as RTT can be used to provide a classifying discriminator by taking the posterior probability of a message being a spam, given that the RTT of the message (rtt) is greater than x. Bayes' rule provides a convenient way to take the causal information and form a diagnosis:

$$P(spam|rtt > x) = \frac{P(rtt > x|spam)P(spam)}{P(rtt > x)} \tag{4.24}$$



(a) RTT Cumulative Probability Distribution

(b) RTT Conditional Probability

(c) Maximum Idle Time Cumulative Probability Distribution

(d) Maximum Idle Time Conditional Probability

(e) Received FIN Count Cumulative Probability Distribution

(f) Received FIN Count Conditional Probability

Figure 4-35: Comparing spam and ham probability distributions for RTT, idle time and received FIN count (left column). The resulting conditional probability distributions (right column) serve as a discriminator.

Figure 4-35(b) shows the conditional probability of a spam message across a continuous range of RTTs. We include the probability of a ham message in the figure as well; these probabilities sum to one, hence providing mirror images of each other. With an RTT less than 10ms, the probability is strongly biased toward being a ham message. In the range $[0.02, 0.1]$s, the probability estimate is relatively neutral without a strong bias toward either category. However, after 100ms, there is a strong tendency toward the message being spam. This conditional probability distribution corresponds exactly to the data in Figure 4-35(a).

The differences in RTT raise several interesting points. For some classes of users, it is not unexpected that legitimate email originates from geographically nearby sources. Thus, it is prudent in many cases to take advantage of locality of interest. RTT may be less of a distinguishing characteristic though for users with frequent trans-continental conversations. However, approximately 50% of the spam messages have an RTT greater than 200ms, suggesting that the remote machines are quite remote, overloaded or reside on constrained links. Further, the $\sim$ 10% of flows with an RTT greater than one second cannot easily be explained by geographic distance and are more likely evidence of persistent congestion or end host behavior. We emphasize that RTT is just one potential feature. In instances where users receive legitimate email with large RTTs, the system may use a threshold strategy or simply lower the relative importance of RTT in favor of other flow features. Just as content filters are frequently customized per-user, the distinguishing flow characteristics can be unique to each user based on his or her receiving patterns.

As a second feature, consider maximum idle, the maximum time interval between two successive packets from the remote MTA. In some instances the maximum idle time directly reflects the initial RTT, but is often different. Figure 4-35(c) depicts the cumulative distribution of maximum idle times. Again, we see marked differences between the character of spam and ham flows. For instance, nearly 40% of spam flows have a maximum idle time greater than one second, events unlikely due to geographical locale. Figure 4-35(d) shows the conditional probability that the message is spam. After a maximum idle of 250ms, the probability tends strongly toward spam, as there are few legitimate messages with such a long idle time.

Finally, to emphasize that there are many potential features available in a flow, we examine TCP FIN segments. In a normal TCP session termination, each endpoint issues a finish (FIN) packet to reliably end the connection. Figure 4-35(e) shows that almost 45% of the spam email flows do not send a FIN compared to only 5% for ham. Finally, a small fraction of ham flows result in two FINs whereas only 0.7% of spam flows send more than one FIN. The resulting conditional probabilities are given in Figure 4-35(f). Section 4.4.2 details all of the features we consider.

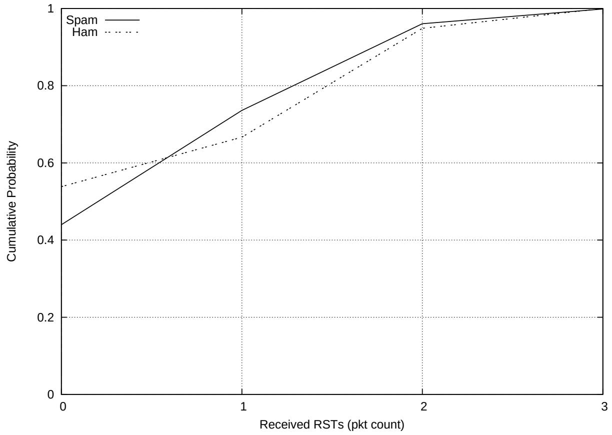


Figure 4-36: Non-features: distribution of received TCP RST count is similar for spam and ham. Surprisingly, this feature provides little discrimination.

Non-features

A strength of a statistical approach is in systematically identifying not only good features, but also poor features. Several flow properties we initially expected to be a strong indication of spam provide little differentiation. For example, one might expect ill-behaved flows to tear down the TCP connection using TCP resets (RSTs) rather than a graceful shutdown with FIN packets. However, as Figure 4-36 demonstrates, the distribution of received RSTs is very similar between spam and ham. Surprisingly, only 53% of ham flows contain no reset packets while 28% contain two RSTs.

Manual investigation of the data reveals that many MTAs, including those of popular web mail services such as Google and Yahoo, send RST packets after sending the SMTP quit command. Detailed traces of this phenomenon are provided in §4.4.7(III-IV). Frequent RSTs are not limited to a few select large servers. In addition, the source code of the popular Postfix MTA software contains the SO_LINGER socket option that causes it to perform an abortive close and generate TCP reset packets.

In all, the preceding analysis provides evidence that spam and ham flows are sufficiently different to reliably distinguish between them. The important point of note is that we examine neither the content nor origins of incoming emails. Instead our determination of an email's legitimacy is based entirely upon the incoming flow's *transport characteristics.*

4.4.3 Results

Given our data set and problem formulation as described in §4.4.2, we turn to exploiting the differences in transport characteristics. In this Section we build and train a supervised classifier and study its performance.

Building a Classifier

In this study, we use only the 39 unique ham mails so that our learning algorithm does not hone in on domain specific effects. For instance, if a majority of email arrives from Yahoo and Google MTAs, the primary features may reflect specific properties of flows from these servers. While nothing precludes learning on the basis of multiple mail flows from a single domain, we seek to understand the generality of SMTP flow characteristics. Our results will likely improve given additional training data from the same domain and MTAs.

As a result, our data set contains many more spam messages than legitimate messages. To prevent a large discrepancy in the complexion of training samples, we limit our data set to include only five times as many spam messages as valid messages. In each experiment, we select a random set of spam messages that is no more than five times larger than our ham corpus. Thus, the experiments include 39 valid emails and 195 randomly selected spam emails (234 total labeled messages and corresponding SMTP packets).

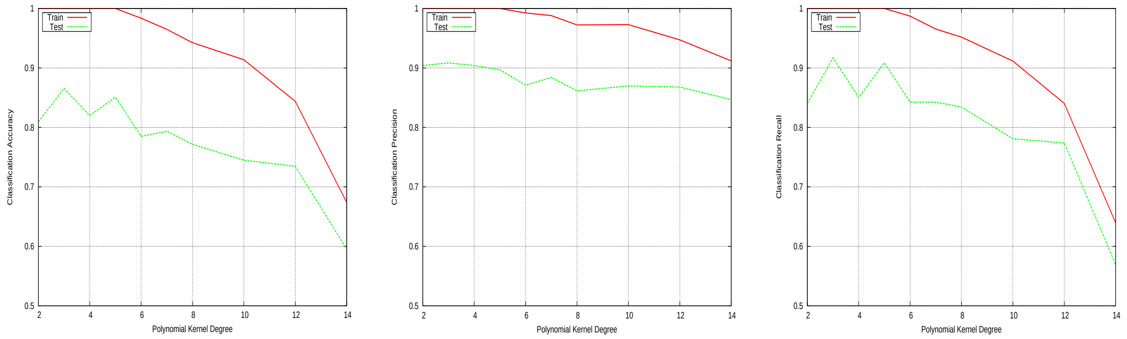
In each experiment, we separate the $(\mathbf{X}_i, Y_i)$ pairs from the feature extraction of §4.4.2 into a training and test set. We perform a horizontal concatenation of the $\mathbf{Y}$ labels and n x f feature matrix $\mathbf{X}$ to form $\mathbf{D} = [\mathbf{Y}^T : \mathbf{X}]$. To ensure generality, we randomly permute rows of $\mathbf{D}$ for each experiment and run each experiment ten times. For a permuted $\mathbf{D}$, the training data consists of the first i rows of $\mathbf{D}$ while the test set is formed from the remaining $n - i$. In this way the training and test samples are different between experiments.

We use Support Vector Machines (SVMs) for classification (§2) as maximum margin kernel methods with regularization perform well in practice on many tasks. However, we note that the general insight behind SpamFlow is independent of the exact learning algorithm. In future work, we plan to explore the use of decision trees to exploit potential correlation between feature values.

Kernel Selection

Next, we experiment with the SVM kernel parameters. Figure 4-37 shows the training and test performance using a polynomial kernel across different degrees. For a fixed training size, we examine accuracy, precision and recall (§2.3). While no overfitting is evident, the performance drops off quickly for polynomial kernels with degree six or higher.

Similar results for a radial basis kernel with varying γ are given in Figure 4-38. Here we observe training error improving with finer granularity Gaussians, but this improvement is not clearly reflected in the test error. Because the radial basis kernel provides more stable performance, and outperforms polynomial kernels in terms of recall, we elect to use an RBF kernel with $\gamma = 0.2$.

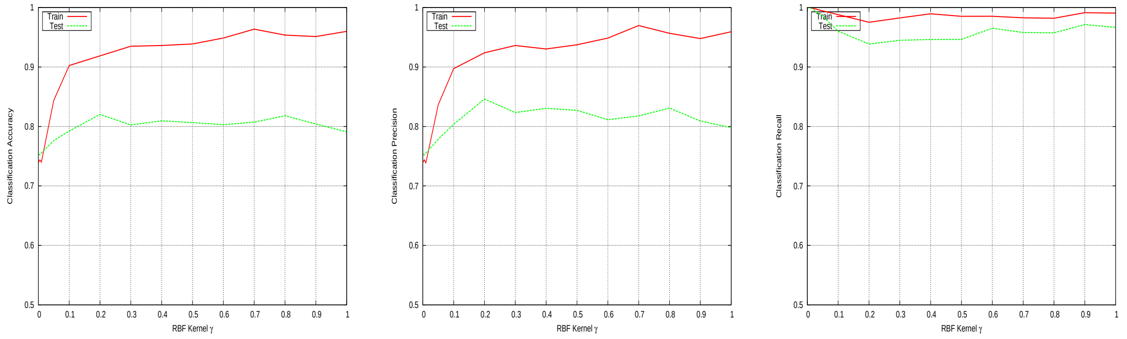


(a) Accuracy

(b) Precision

(c) Recall

Figure 4-37: Polynomial kernel selection in SpamFlow



(a) Accuracy

(b) Precision

(c) Recall

Figure 4-38: Radial basis function kernel selection in SpamFlow

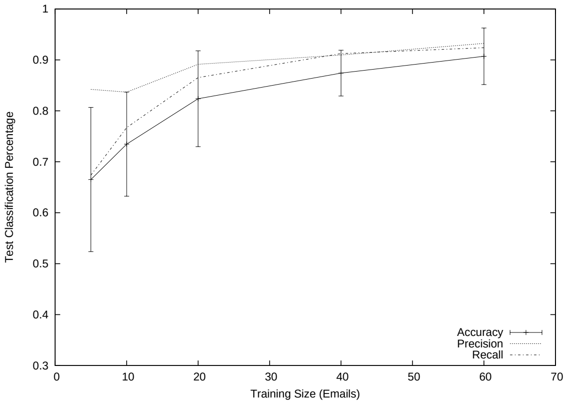


Figure 4-39: SpamFlow classification accuracy, precision and recall vs. training size

Performance

Figure 4-39 shows the classification performance, measured in terms of accuracy, precision and recall as a function of the training size. We achieve approximately 90% accuracy using 60 training emails and more than 80% accuracy with only 20. This accuracy is relatively insensitive to the size of the data set, for instance if we include only twice as many spam as valid messages. However, the standard deviation is tighter as the number of training emails increases.

Note that accuracy may be misleadingly high as the true composition of our test set includes three times as many spam messages as ham. A naïve classifier need only guess “spam” to achieve high accuracy. Thus, we also include recall, or the true positive rate and precision measures. Recall is the ratio of true positives to the number of actual positives, $recall = TP/(TP+FN)$ and is therefore a proxy for the number of false negatives. Precision is most important in this application where the majority of messages are spam. Precision is the ratio of true positives to all predicted positives, $precision = TP/(TP + FP)$, providing a metric of false positives. We see that at 40 training mails, the precision is more than 90%, corresponding to an average of two false positives per iteration.

To better understand the performance characteristics of SpamFlow, we examine the receiver operating characteristic (ROC) which plots true positive rate (recall) against false positive rate. Figure 4-40 is a ROC plot that compares the performance of 10 independent experiments for each of five different training sizes. The ideal point on the ROC plot is the upper left hand corner which corresponds to perfect true positive rate (recall) and perfect false positive rate (1-sensitivity). Thus, the more tightly clustered the points are to this upper left corner, the better.

With smaller training sizes, the ROC curve shows either an unacceptably high false positive rate, or two low of a true positive rate. The performance with larger training sizes is much better, with all experiments achieving greater than 80% true positive rate and less than 40% false positive rate.

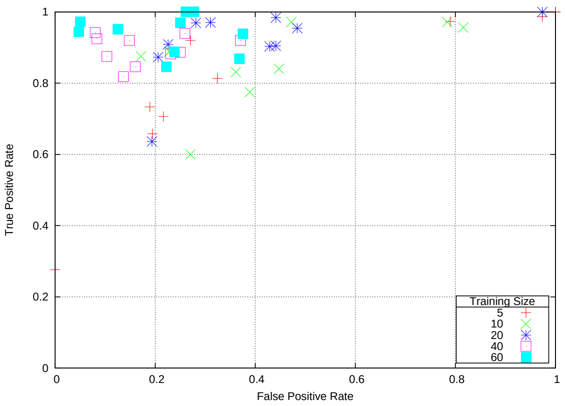


Figure 4-40: SpamFlow receiver operating characteristic

Still, the current false positive rate is higher than is ideal for our application. With further effort and a larger training set, we expect to achieve higher performance. However, we envision SpamFlow as an additional metric in an overall decision tree in just the same way modern filters use multiple tests to form an overall spam decision.

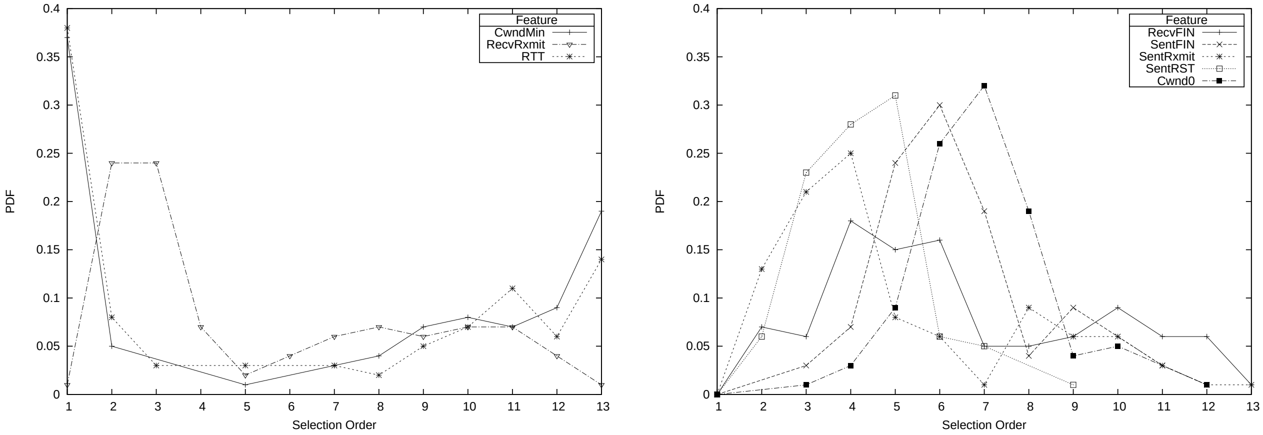
Feature Selection

In order to optimize its performance to different users and network environments, SpamFlow determines which features from Table 4.3 provide the most discrimination power. To find these, we turn to feature selection methods as discussed in §2.2.9. Forward fitting (FF) feature selection simply finds, in succession, the next single feature j that minimizes an error function $V(\cdot)$. Therefore, training error decreases monotonically with the number of features. Forward fitting effectively eliminates features that themselves are closely dependent. Often two features individually provide significant power, but the second feature provides little additional classification power. For example, the RTT and maximum idle time may be highly correlated. Forward fitting will continually seek the next best performing feature without this potential dependence.

Figure 4-41 shows the probability distribution functions of the selection order for each feature. We split the results into two plots only to improve readability. Figure 4-41(a) illustrates that both RTT and CwndMin are the most likely features to be selected first, each with approximately 40% probability. MaxIdle has around a 10% chance of being the first selected feature and the other features comprise the remaining 10%. In other words, if the learner were given the choice of one and only one feature with which to classify, the learner would choose RTT or CwndMin. RecvRxmit and SentRxmit are typically not the first or second feature, but frequently serve as the third and fourth best features.

Figure 4-41(b) gives the secondary features, those that are more likely to be chosen fifth or later in the order. These features include the RecvFIN, SentFIN, Cwnd0 and JitterVar.

To leverage the results of feature selection, we measure the prediction dependence on the



(a) Primary features: minimum congestion window and initial RTT are frequently the best feature; received retransmits are a strong second feature.

(b) Secondary features: have probability centered in the middle of the selection order indicating flow properties that distinguish ham and spam well.

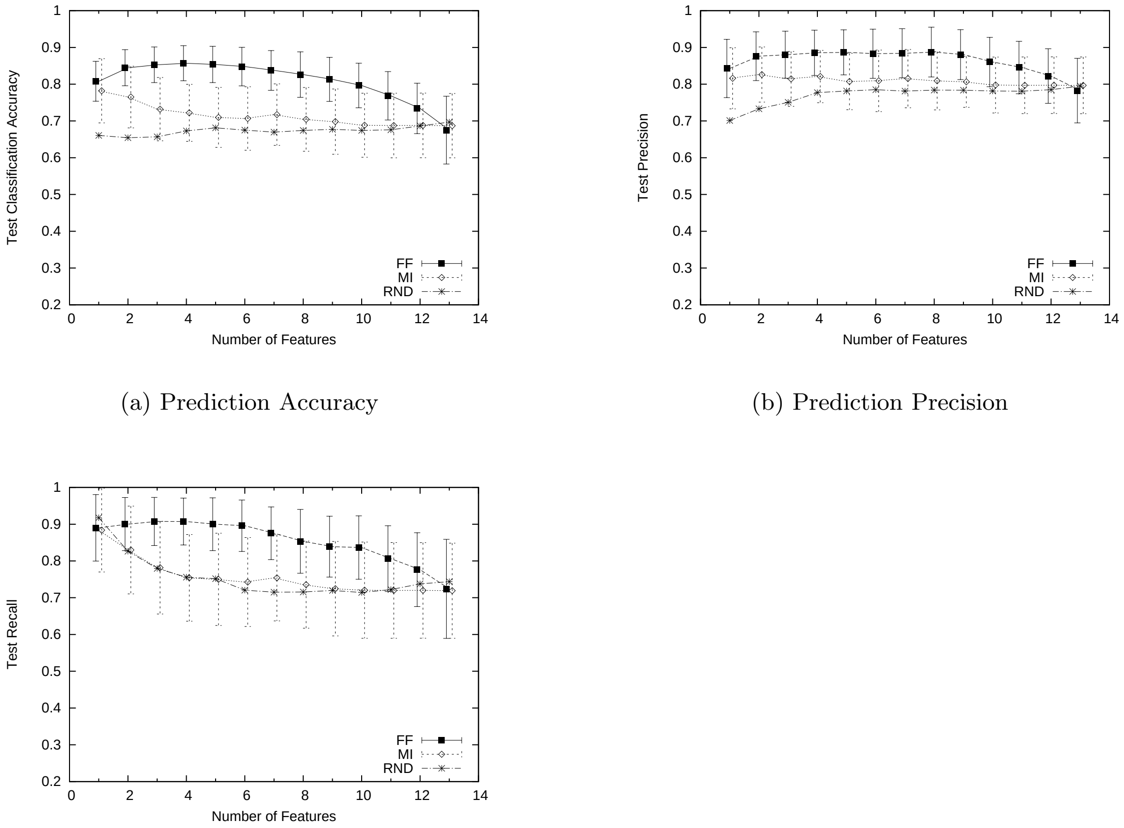
Figure 4-41: Feature selection order probability distributions demonstrate the relative discriminatory strength of different flow properties.

number of best features. Figure 4-42 gives the results of performing forward fitting, mutual information and random features in each round to select a given number of best features. We include random features to provide a useful baseline. As expected, the random features perform the worst, yet still yield 60-70% accuracy. Forward fitting achieves high accuracy, precision and recall.

4.4.4 Related Work

Current best practices for defending against spam are multi-pronged with four main techniques: content filters, collaborative filtering, reputation systems and authentication schemes. The most successful attempts thus far to combat spam have relied on fundamental weaknesses in spam messages or their senders. We review these systems as well as previous network and traffic characterization studies.

Content Filtering: A wealth of content analysis systems are used to great effect today in filtering spam. Learning methods have been effectively applied to building classifiers that determine discriminatory word features [130]. Such content analyzers exploit the fact that a spam message contains statistically different words from a user's normal mail. Even innocuous looking commercial spam, intended to subvert content filters, typically includes a link to an advertised service – thereby providing a basis for differentiation. A popular open source solution is SpamAssassin [100], although there are many competing commercial alternatives. An interesting approach from Marsono, et al. [99] also analyzes individual SMTP packets. To alleviate the burden of packet and message reassembly, their scheme proposes per-packet content analysis and filtering.



(a) Prediction Accuracy

(b) Prediction Precision

(c) Prediction Recall

Figure 4-42: Prediction performance relative to the number of features for Forward Fitting (FF), Mutual Information (MI) and Random (RND) feature selection with an SVM model. The points represent average performance across nodes in our data set, while the error bars show the standard deviation.

Our system, SpamFlow, does not perform any content analysis on the messages themselves. By providing an alternative classification mechanism, SpamFlow helps address blocking innocuous junk mail, for instance nonsense emails that are likely used to “detrain” Bayesian filters [147].

Collaborative Filtering: Spam is typically sent to many users thereby providing a signature. By aggregating the collective spam of a distributed set of users, collaborative filtering [121, 139] aims to prevent previously marked spam messages from being accepted. For example, popular web mail clients can easily provide collaborative filtering as their servers are under common administrative control and can leverage spam marked by one user to block spam to other users. Unfortunately, not all mail installations are large enough to take advantage of collaborative filtering and are unwilling to rely on vulnerable centralized repositories. Further, spammers can trivially make each spam unique in an effort to avoid collaborative techniques.

Reputation Systems: Reputation systems attempt to aggregate historical knowledge over specific MTA [88] IP addresses or mail domains. For instance, a large number of messages to unknown recipients might be recognized as a dictionary attack. MTAs that continually send undeliverable mail are given a low reputation. Often, spam honeypots are used in conjunction with reputation systems to gather additional data on spam origination. MTAs which have previously sent spam are likely to continue sending spam. Real-time databases [140, 138, 134] of these offending MTAs and IP addresses provide blacklists which MTAs can query before accepting mail. However, Ramachandran’s analysis of DNS blacklists [124] shows that as much as 35% of spam is sent from IP addresses not listed in common blacklists. Their work brings to light an important point about the dynamism of IP addresses in relation to spam. Not only are the IP addresses of botnets changing as hosts acquire new addresses, spammers are rapidly changing addresses in order to evade blacklist reputation schemes. Our SpamFlow, however, has no dependence on IP addresses making it particularly attractive in defending against botnet spam.

Authentication Schemes: Authentication schemes attempt to verify the sender or sender’s domain to prevent spoofing-based attacks. Sender Policy Framework [150] limits IP addresses to sourcing mail only for authorized domains. Domain keys [6] uses public keys to associate each email with an entity.

Characterization Studies: Casado et al. perform passive flow analysis on approximately 25,000 spam messages to determine bottleneck bandwidths [33]. Their study finds significant modes at modem, Ethernet and OC-12 speeds, suggesting that spammers employ both farms of low-speed as well as high speed servers. In contrast, we perform a detailed passive flow analysis in order to find relevant features for forming classification decisions.

Brodsky’s trinity system identifies botnets by counting email volumes, thereby identifying spam without content analysis. Similarly, the spamHINTS project [46] leverages the sending patterns of spammers to identify the sources of spam. In addition to analyzing server logs, spamHINTS proposes to examine sampled flow data from a network exchange point to obtain a large cross section of email traffic patterns and volumes. For instance, hosts that source email continually or have particular patterns can be identified through a set of heuristics. In contrast, our work analyzes the individual packets of SMTP transactions to obtain much more detailed flow information, e.g. congestion windows and round trip times. Further, SpamFlow relies on machine learning techniques rather than heuristics to build a classification system.

Our work is in a similar spirit to [123] which attempts to characterize the network properties of spammers, for instance the IP blocks to which they belong. Instead, by taking a step down the protocol stack and examining the transport level properties of spam, we hope to take advantage of previously unexploited information.

4.4.5 Conclusions and Future Work

Our results are promising, demonstrating that even rough metrics of a flow's character can aid in differentiating incoming emails. By providing a method that does not rely on either content or reputation analysis, SpamFlow is a potentially useful tool in mitigating spam. Whereas reputation systems are vulnerable to IP address dynamics, SpamFlow has no reliance on addresses. While content analysis is easy to game, SpamFlow attempts to exploit the fundamental character of spam traffic. We plan to gather a significantly larger data set that includes more valid messages and additional features.

Can spammers adapt and avoid a transport-based classification scheme? By utilizing one of the fundamental weaknesses of spammers, their need to send large volumes of spam on bandwidth constrained links, we believe SpamFlow is difficult for spammers to evade. A spammer might send spam at a lower rate or upgrade their infrastructure in order to remove any congestion effects from appearing in their flows. However, either strategy is likely to impose monetary and time costs on the spammer.

The initial RTT is the strongest indication of spam for our data set. A spammer might attempt to artificially lower the inferred RTT by optimistically acknowledging packets that have not yet been received. However, an adversary cannot reliably know the remote host's initial sequence number for the TCP connection and therefore cannot easily fake the initial RTT. Such attempts to hack TCP to disguise the effects we observe are likely to expose other features, for instance retransmits and duplicate packets.

While RTT is the strongest discriminator on our data, other mail users may have different email interactions with geographically dispersed MTAs. Further, the observed spam RTT may vary for MTAs in countries other than ours. However, such differences demonstrate the strength of a statistical approach. Just as content based filtering is personalized for individual users, the particular features for transport based filtering can be tailored to the end recipients.

Because SpamFlow performs neither content nor reputation analysis, its functionality could be pushed deeper into the core of the network without compromising privacy concerns. SpamFlow is unique in this regard. In addition, with a wider cross-sectional view the performance of SpamFlow would likely improve.

Utilizing available flow information may aid not only in preventing spam, but also other types of attacks that originate from botnets and compromised machines. For instance, denial of service attacks similarly rely on sending large quantities of data over constrained links. We wish to gather data to better understand the broader applicability of our approach.

4.4.6 Frequently Asked Questions

In presenting SpamFlow and related research, many of the same questions are commonly raised. To allay some of these concerns, we provide this frequently asked questions subsection.

- ***Q:*** *You're disenfranchising distant servers!* In our dataset, RTT is indeed one of the strongest indicators of a remote spam source. In many cases however, this may be desirable; we envision SpamFlow being customized on a per-user, per-network basis in the same way current content filters are tailored. A North American user who rarely receives email from China may in fact wish to bias against that email, particularly if e.g. content analysis flags the message as spam. Thus, SpamFlow need not be the sole determinant of mail validity.

  Further, RTT is but one feature. In our dataset, the minimum congestion window also figures strongly into the discriminant function. A user who typically receives valid email from remote sources will leverage properties of the TCP flow other than RTT for differentiation.

- ***Q:*** *Can SpamFlow be more conservative in using RTT?* Note that approximately 5% of the spam flows we examine have an initial RTT greater than a full second – longer than even the expected latency from a satellite link or trans-oceanic crossing. Even a highly conservative filter can still leverage RTT to eliminate these extremely large RTT spam flows.

- ***Q:*** *Doesn't SpamFlow privilege well-connected senders?* Insofar as SpamFlow will detect poorly connected servers attempting to send large volumes of mail. Personal, home or small business servers do not have the same volume requirement as spammers and thus are unlikely to induce the same TCP congestion effects we observe.

  In reality, there is a value judgment that makes SpamFlow practical and reasonable. Specifically, users who wish to ensure that their emails are delivered typically invest in suitable infrastructure, contract with an outside provider or use their service provider's email systems. Companies are not sourcing large amounts of crucial email from hosts attached by consumer-grade connections. The vast majority of home users utilize their provider's email infrastructure or employ popular web-based services. Thus, SpamFlow only discriminates against sources that are *both* poorly connected *and* injecting large volumes of mail.

- ***Q:*** *What about email lists?* In contrast to spam, which must be sent continually, email list traffic can be scheduled in order to not cause local congestion. For instance, even a 64kbps link (very slow in current terms) can support hundreds of serial recipients every five minutes for 10KB sized messages.

- ***Q:*** *Isn't your false positive rate too high?* Our current results exhibit a higher than desired false positive rate, largely due to the disproportionate number of spam mails in our training set. In future research, we plan to obtain a much larger quantity of legitimate mails in order to even the training complexion and better train the machinery. However, the existing system is still highly usable as a component, or vote, in an overall system that also utilizes other mail evaluation techniques such as content filters.

- ***Q:*** *How do you anticipate spammers will react to SpamFlow?* We believe SpamFlow addresses spam at a different layer of abstraction than existing solutions, one where the spammers cannot easily defend. To reduce congestion or other tell-tale signs within their traffic stream, spammers either must reduce their sending rate, distribute their sources more widely or obtain better-connected hosts. All three potential solutions are expensive in real economic terms that matter to spammers. Even if spammers perform scheduling to ensure that their flows do not self-interfere and cause resource contention, the reduction in spam volume is beneficial. Therefore, our hope is that SpamFlow proves to be difficult to subvert.

4.4.7 Supporting `tcpdump` traces

In this subsection, we provide several `tcpdump` traces which are both illuminating and substantiate earlier discussions.

I.) We observe many instances where our server's TCP socket remains stuck in the LAST_ACK TCP state. After establishing a connection, the remote MTA sends the SMTP HELO and MAIL FROM commands (lines 1 and 4). Our server's TCP acknowledges these packets and responds with SMTP 250 status codes indicating that the command is accepted. Over the next four retransmissions (6-9), our server's response is never acknowledged. Finally, 43 seconds later, the remote MTA sends a FIN (10). Our server sends an ACK (11) but does not shut down the connection because it still has data to transmit. After 20 seconds, the remote MTA sends another FIN (12) and then stops responding.

```
  09:51:52.883259 IP spammer1 > srv:  P 1:11(10) ack 31 win 65505
2 09:51:52.883894 IP srv > spammer1:  P 31:55(24) ack 11 win 32120
```

```
   09:51:53.278033 IP spammer1 > srv:  .  ack 55 win 65481
 4 09:52:14.499220 IP spammer1 > srv:  P 11:42(31) ack 55 win 65481
   09:52:14.499429 IP srv > spammer1:  P 55:63(8) ack 42 win 32120
 6 09:52:16.555063 IP srv > spammer1:  P 55:63(8) ack 42 win 32120
   09:52:20.675041 IP srv > spammer1:  P 55:63(8) ack 42 win 32120
 8 09:52:28.915089 IP srv > spammer1:  P 55:63(8) ack 42 win 32120
   09:52:45.395040 IP srv > spammer1:  P 55:63(8) ack 42 win 32120
10 09:52:57.468021 IP spammer1 > srv:  F 42:42(0) ack 55 win 65481
   09:52:57.468118 IP srv > spammer1:  .  ack 43 win 32120
12 09:53:16.969113 IP spammer1 > srv:  FP 11:42(31) ack 55 win 65481
   09:53:18.355043 IP srv > spammer1:  P 55:63(8) ack 43 win 32120
14 09:54:24.275059 IP srv > spammer1:  P 55:63(8) ack 43 win 32120
   ...
```

II.) Often we see a remote MTA connect, acknowledge our server's response but never send any data. After establishing the TCP connection (lines 1-4), our server's MTA identifies itself (5). Without a response, our server retransmits three seconds later (6). The remote MTA then acknowledges the packet (7), but sends no data of its own. Nearly 20 minutes later, our MTA times out and sends a status code 451 SMTP timeout (8) and then a TCP FIN (9). Our server remains in TCP state FIN_WAIT1 as the remote MTA never sends an acknowledgment and has seemingly disappeared.

```
   13:40:20.257512 IP spammer2 > srv:  S 776725085:776725085(0) win 24000
 2 13:40:20.257755 IP srv > spammer2:  S 4032400311:4032400311(0) ack 776725086 win 32696
   13:40:23.585053 IP srv > spammer2:  S 4032400311:4032400311(0) ack 776725086 win 32696
 4 13:40:25.077725 IP spammer2 > srv:  .  ack 1 win 24000
   13:40:25.099313 IP srv > spammer2:  P 1:31(30) ack 1 win 32696
 6 13:40:28.095065 IP srv > spammer2:  P 1:31(30) ack 1 win 32696
   13:40:29.633161 IP spammer2 > srv:  .  ack 31 win 24000
 8 14:00:25.095141 IP srv > spammer2:  P 31:53(22) ack 1 win 32696
   14:00:25.095421 IP srv > spammer2:  F 53:53(0) ack 1 win 32696
10 14:00:31.095079 IP srv > spammer2:  FP 31:53(22) ack 1 win 32696
   14:00:43.095043 IP srv > spammer2:  FP 31:53(22) ack 1 win 32696
12 14:01:07.095062 IP srv > spammer2:  FP 31:53(22) ack 1 win 32696
   ...
```

III.) Google MTAs consistently send TCP RST packets. In this example, the Google MTA sends an SMTP quit command (line 1) and then initiates an active close. The active close induces the FIN packet (2) from Google which our server acknowledges (3). Our server passively closes the connection in the other direction by sending a FIN (5). The Google MTA does not acknowledge this FIN and instead sends a RST (6).

```
   11:55:57.807504 googl > srv:  P 187089:187095(6) ack 143 win 5720
 2 11:55:57.807510 googl > srv:  F 187095:187095(0) ack 143 win 5720
   11:55:57.807628 srv > googl:  .  ack 187096 win 32614
```

```
11:55:57.807863 srv > googl:  P 143:167(24) ack 187096 win 32614
11:55:57.808181 srv > googl:  F 167:167(0) ack 187096 win 32614
11:55:57.834759 googl > srv:  R 46149836:46149836(0) win 0
```

IV.) Similarly, flows from Yahoo MTAs exhibit TCP resets. This example shows our server responding to an SMTP quit command and then performing an active close on the socket (2). The Yahoo server simultaneously performs a close (3). Our server acknowledges Yahoo's FIN (4). Instead of sending an acknowledgment, the Yahoo MTA sends three resets (5-7).

```
  11:20:35.023406 srv > yahoo:  P 113:137(24) ack 1426 win 32120
2 11:20:35.023782 srv > yahoo:  F 137:137(0) ack 1426 win 32120
  11:20:35.023983 yahoo > srv:  F 1426:1426(0) ack 113 win 33304
4 11:20:35.024073 srv > yahoo:  .  ack 1427 win 32120
  11:20:35.076591 yahoo > srv:  R 776208340:776208340(0) win 0
6 11:20:35.076969 yahoo > srv:  R 776208340:776208340(0) win 0
  11:20:35.077381 yahoo > srv:  R 776208341:776208341(0) win 0
```

4.5 Summary

While a complete view of the network may be best harnessed by a distributed learning network plane, we find significant value to end-nodes acting as autonomous learning agents. Although end-nodes already contain a variety of intelligent functionality, we show that there is ample room for these nodes to gather and use available data in non-traditional ways to their advantage. End-nodes can improve their performance by becoming intelligent participants in the network through learning, prediction and classification. Because end-node intelligence is compatible with the current Internet architecture, such schemes have the added advantage of immediate deployment without implementation hurdles.

In this Chapter, we create and test two operational end-host systems that make use of ignored data. We observe that Internet addressing is hierarchical, but discontinuous and fragmented, suggesting that learning can discover additional structure beyond what is available in routing tables or registries. We develop a network-specific clustering algorithm to find common partitions across the entire Internet address space. Using this clustering method, we endow agents with the ability to predict round-trip latencies to random Internet destinations without any network or coordinated support (§4.2). Further, we adapt our algorithm to accommodate structural and temporal dynamics.

In §4.4, we create a packet flow classification technique which detects traffic originating from remote, resource constrained hosts. This method provides the basis for "SpamFlow," a novel spam detection tool that relies on neither content nor reputation analysis. In addition to providing high spam classification accuracy, we detect the majority of false negatives missed by traditional content filters. By using learning to exploit a fundamental weaknesses in sourcing spam, we show that SpamFlow is adaptable and not easily subvertible. Our hope is that SpamFlow serves as a step forward in providing a means to combat spam and impose a greater cost on parties sourcing spam.

A central point of debate is whether security architecture requires coordination and cooperation to succeed, or if individual nodes can operate in an intelligent, autonomous way to protect themselves. Can agents achieve high predictive performance without explicit help from the network? Autonomy enables incremental deployment, protection where deployed and ability to compose in the form of chaining. In the next Chapter, we more carefully examine these issues by including learning and intelligence within the network core.

No one else could make electronic sounds so lusciously melodic, by sheer contrast with all the rattling and plicking that had gone on before.

- Economist Obituary on Karlheinz Stockhausen

Chapter 5

Learning within the Network Core

5.1 Introduction

Thus far, we have considered the application of learning to enable a variety of functionality in end-hosts and end-applications. As such, our approach has followed a traditional end-to-end notion of where in the network to place learning and intelligence. The end-to-end arguments espouse placing functionality at the correct layer so as not to burden all applications with functionality that is needed only by a few. However, as we point out throughout this thesis, the assumptions surrounding the Internet are changing. For example, in an Internet where all applications require security, different design choices may still remain consistent with the end-to-end argument.

In this Chapter, we examining adding intelligence into the core of the network. Pushing learning into the core affords a more complete view of the network, permits cooperation against adversaries and promises mediation within the system. To this end, we consider two distinct problems that require a distributed learning solution within the core:

1. IP Source Address Validation (§5.2): IP source address forgery, i.e. "spoofing" continues to afford attackers a fertile vector for varied exploits despite historical precedent. Best common filtering practices suffer from an incentive problem; networks ensure they source no spoofed traffic, but cannot prevent its reception. Thus, a central impediment to existing mitigation techniques is their reliance on global participation. In contrast, we consider the task of identifying spoofed source packets at points topologically distant from the true source. We employ learning to leverage the predominance of legitimate traffic[1] in order to identify spoofed traffic. Internet-scale simulations reveal that as few as three central autonomous systems suffice to block more than two-thirds of spoofed traffic while interfering with less than 1% of legitimate traffic. To demonstrate real-world applicability, we implement our learning approach as a

[1] We define "legitimate" in this context as non-spoofed traffic. In contrast to email where the majority of messages are spam, the majority of Internet IP packets are legitimate.

Linux firewall module capable of processing >50,000pps.

2. Intelligent Routing Plane (§5.3): Routing decisions on the modern, commercial Internet are a function of not only path length but also policy and economics. Present interdomain routing protocols afford only blunt mechanisms to effect policy and cause the network to be complex, error prone and inefficient. As the Internet increasingly becomes critical infrastructure, providers struggle to guarantee reliability while optimizing along economic dimensions. Anticipated and unanticipated events such as faults, attacks, peering and pricing changes all currently necessitate manual intervention. We argue for learning in routing, where the network can react rationally to unknown situations. In contrast to overlays, we advocate additional intelligence *within the routing system* to optimize the multi-dimensional problem and mediate potentially conflicting needs of both providers and end users. We propose a notion of quality-of-service (QoS) where a network uses best effort traffic to explore its available routing paths and exploits this knowledge to increase the performance of high priority users and applications. Using live packet traces to drive simulations on Internet-like topologies, we demonstrate how learning allows an autonomous system (AS) to optimize its available inter-AS capacity.

5.2 Problem 1: Distributed Learning for IP Source Validation

The classic design tenets of the Internet architecture produced a network capable of remarkable scalability and interoperability, but relegated security to the end hosts [41]. One result of this design is that the network includes no explicit notion of IP source authenticity and will forward packets with forged headers. Malicious users and compromised hosts capitalize on the ability to "spoof" source IP addresses for a wide variety of attacks including anonymity, indirection and reflection.

As good Internet citizens, many operational networks implement source address filtering best common practices [58]. These ingress filters allow only packets with source addresses that the networks owns or advertises. Unicast reverse path forwarding (uRPF) [10] similarly checks whether the route toward the source of an incoming packet corresponds to the interface on which that packet arrived in order to prevent spoofing. These filtering techniques are quite effective, but may be limited by multi-homing, route asymmetry, ad hoc filter list maintenance and router design. For instance, uRPF is implemented as a check against a router's forwarding table (FIB) rather than its routing table (RIB). As the FIB is populated by the best route as calculated on the RIB, the FIB does not contain all feasible paths. Thus, networks may opt to deploy uRPF in loose mode where source addresses are not checked against a particular interface, but instead are verified to exist anywhere within the FIB. Loose uRPF therefore still permits many attacks and is for instance useless against reflector attacks.

More importantly however, current anti-spoofing filtering techniques are hindered by an incentive problem. Networks that perform source address filtering ensure they source no spoofed traffic, but *cannot prevent its reception.*

A second fundamental problem with existing anti-spoofing methods is their dependence on *global participation.* Providers that follow all anti-spoofing best common practices still receive anonymous, malicious traffic via other networks that do not properly filter. As a result, both previous research [108, 116] and recent attacks [113] demonstrate that source address spoofing remains a viable attack vector. Arbor network's survey of 55 large network operators in 2007 [114] reveals that only 53% of the respondents employ uRPF filtering at the customer edge and only 45% use uRPF at peering edges. Despite two-decade old exploits [14], new source spoofing based attacks continue to emerge; we review three in §5.2.2.

Our own measurements as part of the ANA Spoofer Project [21, 22] include more than three years worth of data on the extent, nature, evolution and geographic variation of Internet source address validation and filtering. We find 20% of autonomous systems and 17% of netblocks permit spoofing while 17% of filtering policies may be circumvented with careful source selection [20]. Further, using our "`tracefilter`" tool, we determine that 80% of existing filters are employed within two hops of the source hosts [23]. Thus, a single unfiltered ingress point provides a means to circumvent global spoofing protection mechanisms. Once a packet with spoofed source information makes it to the core of the network, it is virtually guaranteed to be delivered to the victim.

The Internet's architectural inability to prevent spoofing implies we cannot reliably anticipate or defend against the next exploit or shift in attack patterns that leverage source address spoofing. Hence, network operators are forced to rely upon defensive point solutions to mitigate known spoofing attacks. Any practical solution to the aforementioned problems must protect parties who implement the solution from receiving spoofed traffic without relying on large-scale distributed coordination or cooperation.

5.2.1 A Learning-Based Solution

This work considers the task of identifying spoofed source packets at points topologically distant from the true source. We present Robust IP Source Address Verification with Kernel Oriented Learning (Raskol). Raskol is a machine learning agent that may be placed anywhere in the network, e.g. in a router or end host. We employ learning to leverage the predominance of legitimate traffic in order to identify spoofed traffic. After an initial supervised training phase, Raskol classifies incoming packets as either spoofed or valid source. To perform this classification Raskol relies on a simple, basic property of the current Internet: attackers have no control over the path their packets take through the network. By examining the source address in conjunction with the received time-to-live (TTL), a rough indicator of the path length the packet traversed, Raskol can identify spoofed packets with surprisingly high accuracy.

Our research analyzes Raskol in two modes: i) deployed at victim end-nodes, i.e. the eventual recipients of the spoofed traffic; and ii) deployed sparsely throughout the Internet routing substrate. Raskol at end-nodes provides immediate benefit and does not require the participation of that network's provider(s). While end-host filtering is quite effective, our research shows that end-hosts lack enough network observability to prevent false negatives. Hence, end-host or end-network filtering is best suited to situations where the host is under current attack and blocking some legitimate traffic is acceptable in return for relieving the duress.

In contrast, routers within the core afford a broader view of the network and offer increased filtering potential by being on the critical path of more traffic. We thus examine a distributed version of Raskol. We show how learning within the Internet core provides much higher accuracy while blocking traffic closer to its true source. False negatives, instances where non-spoofed packets are marked or filtered, carry a large burden. To minimize incorrect blocking of legitimate traffic, we show how one may use an ensemble of weak classifiers within the routing substrate. Finally, we provide a realistic method to perform distributed classification. By propagating prior decisions within the packet's TTL, upstream routers can weigh into the classification of the remaining routers along the path without changing the existing Internet semantics. Our method thus provides high-levels of protection without relying on large-scale distributed coordination or cooperation.

The primary contributions of this research are:

1. A machine learning-based IP source spoofing prevention that operates on the eventual victim recipient. Our mechanism discriminates real versus spoofed traffic with over 87% accuracy.

2. Internet-scale simulations showing that as few as three central autonomous systems suffice to block more than two-thirds of spoofed traffic while interfering with less than 1% of legitimate traffic.

3. A Linux firewall module implementation capable of processing more than 50,000 packets per second, demonstrating real-world applicability.

5.2.2 Understanding the Threat

Despite being first exploited over two-decades ago [14, 109], IP source spoofing is a persistent problem and a continued threat as demonstrated by the Spoofer project [20] and analysis of backscatter [108, 116]. Both distributed and centralized attacks use spoofing for anonymization or reflection [118]. The anonymity afforded by spoofing greatly complicates the job of network operators defending their networks.

Spoofing is no longer limited to simple Denial of Service (DoS) attacks, but is being used in a multitude of ways. Arguably, the rise of large-scale zombie farms and NAT deployment

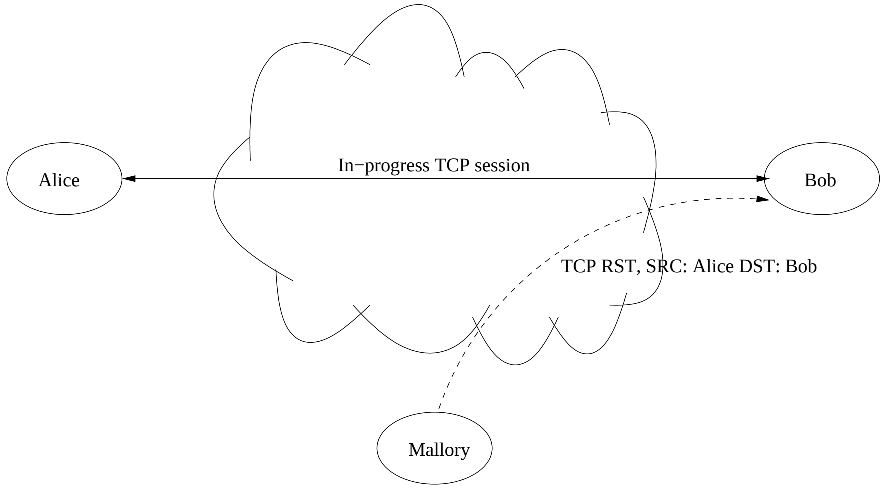


Figure 5-1: In-Window TCP Reset Attack: 1) Alice and Bob have a TCP session in progress with a large congestion window. 2) Mallory sends TCP reset packets that spoof Alice's source address to Bob. 3) If the reset sequence number is received in-window, the third party successfully terminates Alice and Bob's connection.

may negate the need for spoofing in certain classes of attack. However, in this Section we illuminate three recent attacks with very different purposes that rely on the ability to spoof source addresses. The diversity of new exploits attest to both the continued threat of spoofing-based attacks as well as the ability to spoof on the Internet.

In-Window TCP Reset Attack

A recent, non-bandwidth attack uses spoofed source TCP reset packets [145]. A TCP stack that receives a reset with a sequence number within its current window terminates the connection. Typically sessions are protected from third-party resets through port obfuscation, short duration and small window size. However, high-speed links with large bandwidth-delay products yield a situation where an attacker can find an in-window sequence number for sufficiently well-known and long-lived connections. Figure 5-1 illustrates a TCP reset attack. Such attacks can disrupt persistent tunnels and even IP routing.

For example, Border Gateway Protocol (BGP) sessions [128] are established over TCP and are long-lived with a well-known port. An attacker able to spoof the source of a core router can break a BGP session causing route withdrawals and inflicting instability in the global routing system, an effect unforeseen by the original protocol designers. While the operational community protects against third-party BGP resets with MD5 authentication [69], countless other TCP reset exploits of a similar character exist.

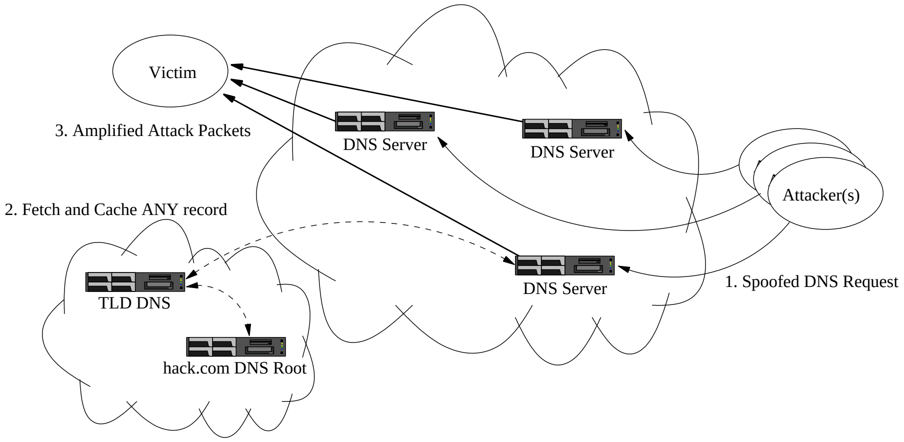


Figure 5-2: DNS Amplifier Attack: 1) Attacker spoofs DNS request with victim's source for large TXT record 2) Third-party DNS servers fetch and cache record. 3) Server sends victim large query result. Attacker's small DNS query packets are amplified and victim cannot identify true attack source.

DNS Amplifier Bandwidth Attack

Figure 5-2 illustrates a bandwidth-based DoS attack that relies on spoofing for reflection [118], amplification and anonymity. We assume the attacker finds or places a large TXT record in the Domain Name Service (DNS) system. The attacker sends spoofed DNS queries with the victim's source address to many public DNS servers. Each request queries for the large TXT record. The public DNS servers retrieve the record and send it to the victim. Thus, the attacker's small DNS query packets are amplified and the victim cannot identify the true source. Because the results are cached, the attacker can continually query the servers and generate a DoS attack from innocent third parties.

This attack is difficult to defend against since the third-party DNS servers cannot distinguish legitimate requests from spoofed ones and traffic filtering is impractical against DNS traffic. Further, the attacker can find or create many different domains dynamically to evade detection or circumvent any policy that might be placed on DNS servers. Such DNS attacks have been seen in the wild by the operational community [112].

Spam Filter Circumvention Attack

Due to the widespread prevalence of worms, viruses and bot-farms used to send unsolicited commercial email, providers often prevent hosts on their network from establishing random SMTP (TCP port 25) connections [88] and force user authentication with their own Mail Transfer Agent (MTA). The final new attack we mention does not disrupt the network but is instead a clever means to circumvent provider-based spam filtering. Known as “fantasy mail,” this attack has been repeatedly observed on production networks [111].

In Figure 5-3, we assume the spammer controls a zombie. Because the zombie's provider

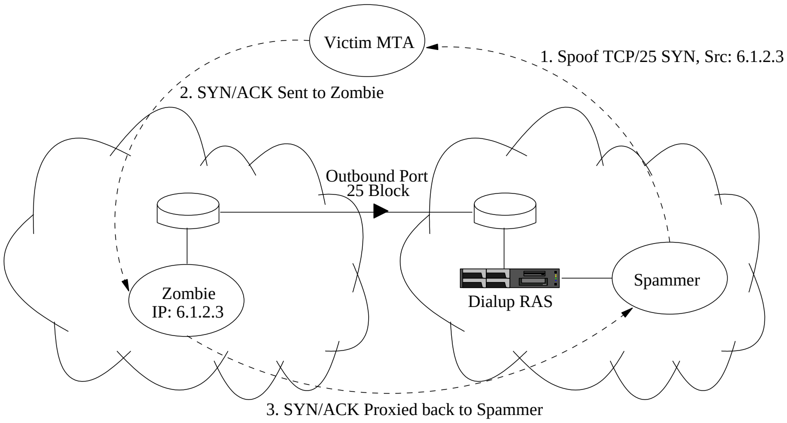

Victim MTA
1. Spoof TCP/25 SYN, Src: 6.1.2.3
2. SYN/ACK Sent to Zombie
Outbound Port
25 Block
Zombie
IP: 6.1.2.3
Dialup RAS
Spammer
3. SYN/ACK Proxied back to Spammer

Figure 5-3: Circumventing Provider Filters: spammer controls zombie on a network that filters outbound TCP port 25 SYNs. 1) Spammer connects via dialup and spoofs a TCP port 25 SYN to an MTA with the zombie's source. 2) MTA replies to zombie with a SYN/ACK. 3) Zombie forwards the MTA's packets back to the spammer. Spam appears to originate from the zombie, making it untraceable.

filters outbound SMTP SYN traffic, control of the zombie provides no additional advantage to the spammer. Instead, the spammer finds a second network that permits spoofing; recent attacks use dialup. Using the dialup account, the spammer sends a TCP SYN to port 25 of a public MTA with the zombie's source address[2]. The zombie forwards the SYN-ACK from the MTA to the spammer over a direct TCP connection or IRC. The spammer can then send the correct spoofed packet to complete the TCP three-way handshake. By repeating for all packets, the spammer circumvents the provider's filter and spam appears to originate from the zombie. In this case the filters confuse network operators who cannot determine how the zombie sent the spam. This example highlights the fact that seemingly straight forward filtering rules are exploitable and require even more complex rules. Such a cycle is creating an environment that is operationally difficult to maintain and debug – a factor that undermines well-meaning attempts at security. We cannot reliably anticipate or defend against the next exploit or shift in attack patterns that leverage spoofing.

5.2.3 Measurements

In support of our research, we perform an Internet-wide active measurement study to determine the extent, nature and evolution of provider source address validation and filtering. Known as the the "Spoofer Project," we collect more than three years of data to test filtering policy, locality and specificity.

Our "spoofer tester" allows willing users to test their Internet provider. An invocation of the application runs several tests including sending packets from three source addresses

[2] Because the TCP socket API does not provide mechanisms for falsifying source addresses or modifying normal TCP behavior, spammers implement such attacks using raw sockets.

Table 5.1: Observed, relative and extrapolated IP source address spoofing coverage

Metric	Observed Spoofable	Believed Unspoofable	Extrapolated to Internet
Netblocks	467 (16.5±1.8%)	2369	41.3k
IPs	34.3M (11.2±1.1%)	272.1M	256.5M
ASs	225 (19.5±3.0%)	928	3796

intended to infer common filtering policies. The first source is as yet unallocated by IANA [73]. This address should not appear in routing tables since it is not delegated. The second source is within a private [129] netblock. Private IPs allow for site-local addressing, but are not routeable on the Internet. Some networks employ filters that block traffic from these "martian" regions of unallocated and private address space. Next, the tester spoofs a valid, allocated source that is globally routeable. This address tests a common filtering policy of permitting only packets with source IP addresses present in the routing table. Such filters [10] are simple to implement and do not require periodic maintenance.

Our testing includes inferences on the location and specificity of provider filtering in the network. See [20] for a complete exposition of the methodology.

We advertised availability of the spoofer application in multiple forums and received coverage in Slashdot [21]. Over the 26 month period between February 2005 and April 2007, we received 5,870 client reports, 4,659 of which are unique[3]. With routing information from RouteViews [105], we map each client to a netblock and AS. From the size of the netblock, we approximate the number of IP addresses the report represents. Without direct evidence of the ability to spoof we classify netblocks as "believed unspoofable." Table 5.1 gives the number of netblocks, addresses and ASs that are spoofable as observed in our data set along with 99% confidence intervals.

In aggregate, 2172 unique clients can spoof none of our three primary classes of spoofing (unallocated, private and valid), while 433 (17%) can spoof at least one of the three. Many hosts experience inconsistent filtering where a subset of the spoofed packets arrive at our measurement station. Approximately 16% of the observed netblocks, corresponding to 21% of the observed autonomous systems, allow some form of spoofing. At the time of writing our view of the BGP table includes 2.3B IP address, 18,700 autonomous systems and 215,000 netblocks. Assuming a uniform distribution of testing, projecting these numbers to the entire Internet yields over 381M spoofable addresses and 4,400 spoofable ASs.

Filtering presents a conundrum for network operators. Conventional wisdom dictates that ingress filtering is performed near the edges of the network rather than the core. However the edge of the network contains the largest number of devices and interfaces. Thus

[3] Notably, we received no reports of alarms or abuse, illustrating both the difficulty and apathy of preventing spoofing.

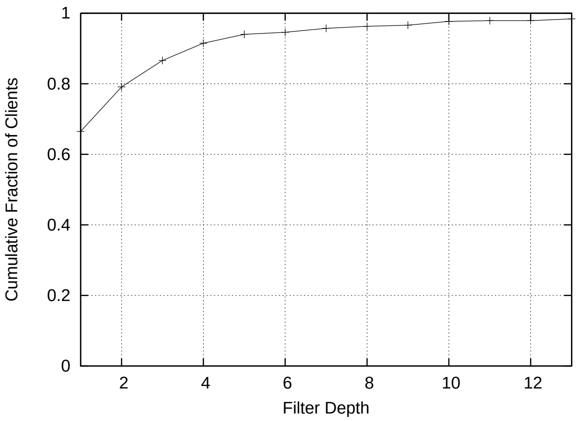


Figure 5-4: Tracefilter Localization: Cumulative distribution of inferred source validation filter depth (hops from sender)

appropriately deploying, managing and maintaining these filters is operationally challenging, particularly when preventing valid address spoofing.

Figure 5-4 shows the cumulative distribution of provider filter depth, measured in IP hops from the client, as inferred by the tracefilter mechanism described in [23]. 80% of the clients are filtered at either the first or second hop router. Thus, networks today generally rely upon the edges to properly validate source information. If spoofed packets make it through the first few hops into the network, a spoofed packet is likely to travel unimpeded to the destination.

Our findings suggest that concerted spoofing attacks remain a serious concern. Given the ample evidence of attacks and measurements of network vulnerability, we map this data to a notion of different attack models.

5.2.4 Attack Models

To better understand the efficacy of our filtering technique, we evaluate it against various real-world attack models in addition to the recent attacks previous detailed. Table 5.2 lists the attack models we consider. Many of the attacks are similar but differ in subtle yet important ways.

For instance, some attacks are effective with only a single attacking host, while others employ many, possibly coordinated, attackers. Our first dimension of distinction among attacks is whether the attackers are distributed. Next, we divide among attacks that target a single fixed victim or a distributed set of victims. Thus, the most basic attack is a single attacker sending spoofed packets to a single victim, for example in an in-window TCP reset attack. In contrast, attacks with many attackers against one or more victims are generally classified as Distributed Denial of Service (DDoS) attacks. Typically these attacks originate from large clusters of compromised machines or zombie "botnets" [49] that are under the

Table 5.2: Attack models based on source-IP address spoofing

Attack Mode	**Distributed Attackers**	**Source Addresses**	**Victims**	**Example**
Basic	No	Fixed	Single Fixed	Rogue TCP Reset, MS-SQL Ping Storm[107]
Basic Random	No	Random	Single Fixed	DoS SYN Flood, Blaster[36]
Basic Smart	No	Random Valid	Single Fixed	Smart DoS SYN Flood
Reflector	No	Fixed	Distributed	DNS reflector attack[118]
Worm	Yes	Random	Distributed	MS-SQL Slammer/Sapphire[35, 16]
DDoS	Yes	Random	Single Fixed	DDoS attack[34]
DDoS Fixed	Yes	Fixed	Distributed	Stacheldraht[54]
DDoS Smart	Yes	Random Valid	Single Fixed	Smart DDoS attack

common control of an attacker.

Many worms are distributed among both attackers and victims. For instance, the MS-SQL slammer [35, 16] and blaster [36] worms exploit vulnerabilities on victim hosts and then attempt to propagate. These worms use spoofed source IP addresses and randomly scan the IP address space to find new machines to infect.

Spoofing affords attackers a third dimension along which to tailor an attack. The source addresses attackers insert into their packets may be fixed, random or intelligently chosen. For example, the Stacheldraht[54] worm uses a fixed `3.3.3.3` spoofed source IP address. As [22] demonstrates, many of the deployed filters at service providers are circumventable with careful source selection. For example, by using only valid addresses, i.e. those appearing in the global routing tables, an attacker improves their chances of sourcing the spoofed packets and remaining unidentifiable. Valid source selection also prevents many of the backscatter effects described in [108].

In our analysis and simulations, we evaluate Raskol's filtering performance against a variety of the attack models described here in Table 5.2; we will refer to each by the mode name in the remainder of this discussion.

5.2.5 Spoofing Prediction

Our spoofing prediction methodology relies on a simple, basic property of the current Internet architecture: attackers have no control over the path their packets take through the network. Packets with source address a from a node s toward destination t will follow a path p determined by the network, irrespective of a:

$$s \overset{p}{\rightsquigarrow} t \; \forall a \qquad (5.1)$$

In normal operation, a host sends packets with its true source address, whereas an attacker uses $a \neq s$. The adversary is assumed to have full control over a, but cannot influence the choice of p. Given a complete map M of all Internet paths, t can verify whether a packet with address a that traversed a path p_{recv} is valid: $M(a) = p_{recv}$. M need not be a router-level or AS-level map. While the path itself is not encoded into packets, packet carry a TTL value. Intended to prevent infinite routing loops, the TTL field of a packet is decremented by each router along the path, providing an implicit path length signal.

Our intent is that every autonomous agent in the network build an independent view of the network. How individual agents, either network elements or end nodes, determine the map $M(\cdot)$ is the central learning problem we tackle. Because the space of possible addresses is roughly 2^{32}, an exhaustive search is impossible. In addition, even if the agent had access to a routing table, the information is too coarse-grained due to aggregation and will not reflect the internal structure of remote networks.

Path Length

Routers decrement the TTL field of each packet. In order to determine the forward path length, a receiver must infer the packet's originating TTL value. Fortunately, operating systems set the TTL of outgoing packets in powers of two. We use use the well-known heuristic of selecting the next highest power of two as the originating TTL. If the observed TTL is greater than 128 we infer an original TTL of 255 and if less than 32 we infer 32. Thus, for a packet with an observed TTL of ttl_o, the path length $|p|$ is:

$$|p| = \left(2^{\lceil \log_2(ttl_o) \rceil}\right) - ttl_o \tag{5.2}$$

Training

Raskol must first train its classifier. One possible training set is simply traffic normally received by the agent. Because TCP connections are more difficult to spoof since the remote end must actively participate in the three-way handshake, TCP traffic could be used for training. For instance, a web server continually receives and processes TCP connections as part of its normal operation, providing an ample training set.

A key premise of the training phase is that the number of legitimate users is much larger than the number of malicious users. Therefore it is possible to leverage legitimate nodes and users to drive out misbehaving traffic.

Dataset

To collect a data corpus for our experiments, we use a simple active measurement procedure. We select unsigned 32-bit integers at random until one is found as a valid IP address in a public global routing table. If the randomly selected destination responds to ICMP echo

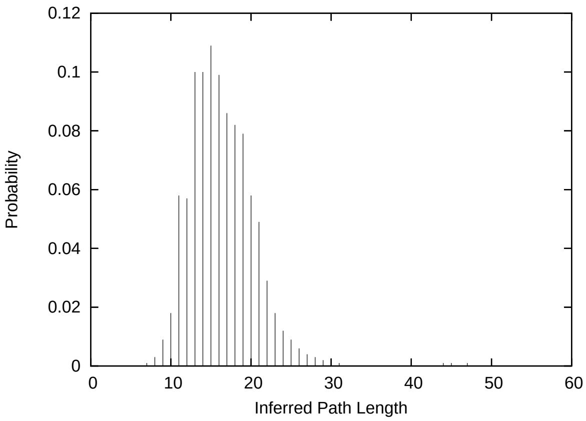


Figure 5-5: Probability mass function of 30,000 node path lengths in data set.

requests, i.e. "ping," we record the received TTL. Because our measurement agent issued the ping request, the responses are valid with high probability. Our data set consists of approximately 30,000 randomly selected (ip, ttl) pairs. The data is publicly available from: `http://ana.csail.mit.edu/ttl`. Figure 5-5 displays the probability mass function of path lengths in the dataset.

A critical concern is the ability of the adversary to mimic this distribution and evade any hop-based TTL or counting mechanism. We consider such attacks on our classification scheme in §5.2.11.

Let the set of legitimate IPs and their corresponding TTLs from the aforementioned data set be $\{L\} = (ip, ttl)$. We generate an identically sized set of spoofed pairs $\{S\}$ by assigning a random IP address to each TTL. i.e. $S_i = (\{0,1\}^{32}, L_i(ttl))$. Next, we take l training points from each of $\{L\}$ and $\{S\}$ at random. Let $T_L \in L$ be the subset of l randomly selected legitimate training points and T_S be l randomly selected spoofed training points. The remaining non-training points $L - T_L$ are used as test points. Let T be the set of training points and R test samples. Formally:

$$T_L \overset{l}{\leftarrow} L; T_S \overset{l}{\leftarrow} S \tag{5.3}$$

$$T = T_L + T_S; R = (L - T_L) + (S - T_S) \tag{5.4}$$

5.2.6 End-Host Packet Classification

We employ Support Vector Machines (SVM) to allow agents to classify incoming packets as either spoofed or non-spoofed. SVMs [146] work well in many learning situations because they generalize to unseen data: the machine is defined by only a subset of the training points, or support vectors. For classification, SVMs find a hyperplane that provides a maximal separation between classes. This optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes in some dimensional space. The support

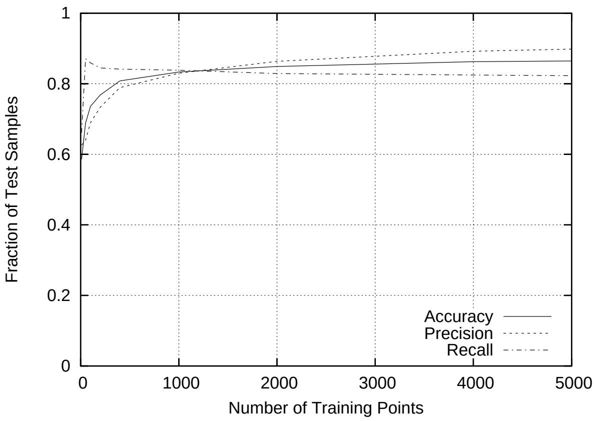


Figure 5-6: Raskol classification accuracy, precision and recall vs. training size

vectors are exactly those data points which define this shortest line. Additional data points do not affect the final solution unless they redefine the margin. For this reason, SVMs are amenable to continuous, adaptive on-line learning, a desirable property in network environments.

Source Address Spoofing Prediction

We use SVM classification to identify IP packets with spoofed source addresses. IP addresses are simply unsigned 32-bit integers. We transform the IP addresses into a 32 dimension input space where each bit of the address corresponds to a dimension. The input to the SVM is an IP address bit vector $\mathbf{x}[0..31] \in \{0, 1\}$ and integer feature $\mathbf{x}[32] \in \mathbb{N}$ with labels $y \in \pm 1$ where y indicates spoofed or non-spoofed traffic.

To reduce possible dependence on the choice of training set, all results we present are the average of five independent experiments. We randomly permute the order of the data set so that, after splitting, the training and test samples are different between experiments. In this way, we ensure generality in the learning algorithm.

Learning algorithms require optimization along several dimensions. We begin by analyzing the tradeoff between training size and model generality. Our objective is to find the number training points required to perform well when predicting future points without over fitting. Figure 5-6 plots the classification accuracy, precision and recall of predictions in the test set versus training set size.

Tunability

An important property of Raskol is the ability to tune its performance to meet the requirements of the network or application it is protecting. In situations where spoofed packets are never allowed, Raskol can be tuned to reject spoofed packets with high probability at the expense of marking many legitimate packets incorrectly. Conversely, the filter can be tuned to always pass legitimate packets, but will often pass spoofed packets. Thus, Raskol will always have a percentage of false positives and false negatives.

In our simulations, we select a middle ground where legitimate packets pass the filter with around 90% probability, but permit as much as 20% of spoofed packets through. Marking approximately 10% of the legitimate packets as spoofed will cause normal traffic to be affected even when not in the presence of attack traffic. Therefore, we envision Raskol being used in a defensive mode: once a node or network is under attack, Raskol's protection is switched on. Although Raskol impacts some small percentage of legitimate traffic, doing so is preferable to providing no service while under duress. In addition, some client applications are robust to losses from a server amongst a set, for instance DNS. If the client receives no response from the first DNS server, it tries alternate servers. For this reason, one might consider employing Raskol's protection on some subset of the root or gTLD name servers continually.

Problem Dimension

The selection of training complexion is crucial to creating a machine that generalizes well and operates efficiently. We examine the informational content of each bit, or "feature," in the IP address. Let $\boldsymbol{\theta}$ be a feature vector where $\theta_i \in \mathbf{x}$. Intuitively, the most significant bits correspond to large networks and should provide the most discriminatory power. Here "most-significant features" correspond directly to BGP prefix masks, i.e. `192.160.0.0/12`. We run Raskol against our data set using 4000 points for training while varying the number of input IP features and maintaining the TTL feature. For example, the first 12 features of IP address `192.168.1.1` is the bit vector $\boldsymbol{\theta} =$`110000001010`.

We plot Raskol's prediction accuracy as a function of the dimensionality of the input space in Figure 5-7. The optimal number of most significant features is between 8 and 12, after which test error begins to increase. This increase in test error is symptomatic of the noise introduced by the least significant bits of the IP address which add no discretionary strength.

5.2.7 Internet-Wide Simulation

In this Section, we turn our attention to performing distributed classification within the core of the network. While we are able to experiment with sourcing and filtering spoofed traffic in a few limited network ingress points, we have no control over the core Internet infrastructure. We therefore turn to simulation. We develop a custom simulator to understand the performance of our spoofed traffic filtering algorithm when deployed within the core of the Internet.

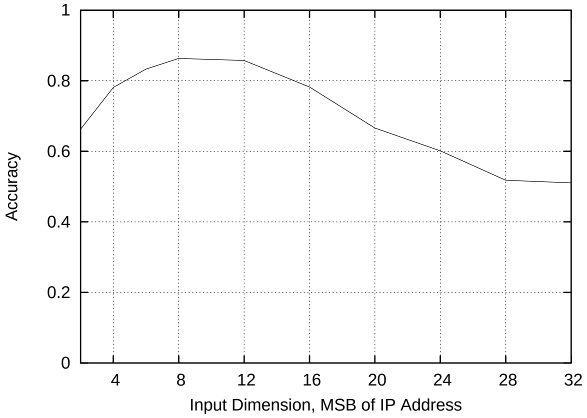


Figure 5-7: Prediction accuracy vs. number of IP address most significant bits.

Including Internet Structure

Because of the nature of commercial AS relationships in the Internet, the route between arbitrary hosts is not necessarily the shortest hop path. Modeling the customer, provider and peering structure of the Internet has been a subject of frequent study. Gao first proposed a “valley-free” algorithm [61] that assigns each AS a degree corresponding to the number of links to other ASes. The route follows the path from the source up to successively higher degree ASes until it reaches a “top-tier” AS. The remainder of the route is defined by following successively lower degree ASes. Roughly speaking, this procedure follows the intuition customer (low degree AS) traffic moving up to providers until the traffic reaches the top-level provider. Traffic then flows down along provider to customer links to reach the destination.

Naturally, there are many more subtleties, particularly surrounding settlement-free peering relationships. He, et al. [68] recently proposed an ensemble of techniques to find and model many of these missing peering links. In our simulator, we use the AS relationships as inferred by their “Lord of Links” algorithm.

Our simulator must also reasonably model not just the AS graph and routing between ASes, but also the IP-level structure of the network. To model IP routing, we use the routeviews [105] service which collects full routing tables, i.e. IP prefix information, from multiple providers. Each prefix in the routeviews table is assigned to its announcing AS. In instances of multi-homing, where the same prefix is announced by two or more ASes, we deterministically select one of the ASes as the owner.

Running Simulations

The simulator allows us to model a representative approximation of the Internet at the granularity of autonomous systems (ASes) and IP routing. The model thus gives insight into how our filtering scheme would perform if we had the ability to deploy it on network

Table 5.3: Formalisms for spoofed source attacks and simulations

Autonomous Systems	Let x be the number of autonomous systems. Let as_i be the i'th autonomous system. Let AS be the set of all autonomous systems, $AS = as_1 \ldots as_x$. Let $deg(as_i)$ be the degree of as_i.
Prefixes	Let p_i^j be the i'th prefix belonging to autonomous system j. Let p^j be all prefixes belonging to autonomous system j, $p^j = \bigcup_i p_i^j$ Let P be the set of all network prefixes, $P = \bigcup_j p^j$.
Filters	Let f_i be the i'th filtering autonomous system. Let F be the set of filtering autonomous systems, $F = \bigcup_i f_i$.
Attackers	Let n be the number of hosts sending spoofed packets. Let a_i be the true IP address of the i'th attacker sending spoofed packets. Let s_i^j be the source address host i selects in the j'th round of an experiment. Let A be the set attacking hosts, $A = a_1 \ldots a_n$.
Victims	Let m be the number of unique destinations that receive spoofed packets from members of S. Let v_i be the IP address of the i'th victim receiving spoofed packets. Let V be the set of victims, $V = v_1 \ldots v_m$. Let send(a_i, s_j, v_k) send a packet from attacker a_i with source s_j to victim v_k using valley-free routing.
Statistics	Let α_i^j be the number of spoofed packets received at autonomous system i in round j. Let β_i^j be the number of non-spoofed packets received at autonomous system i in round j.

core routers today.

As there are many variables among the attack models described in §5.2.4, we introduce some formal notation in Table 5.3 to describe experiments on the simulator.

The simulator provides a send function which allows any AS to source traffic to any other AS with an arbitrary source address. If the AS is sending legitimate traffic, the source is selected randomly from within the set of prefixes owned by that AS. Similarly, the destination address is taken at random from prefixes announced by the destination AS. However, the simulator allows us to also send packets with any source address to mimic an array of spoofing behaviors.

To include background traffic, an AS will send legitimate traffic to other ASes. This is an essential feature of the simulation, otherwise the learning task is much easier. By including random traffic to random destinations, the learned model must be more complex. The simulator finds a random prefix within the topology among prefixes that do not belong to the originating AS. It then selects a random address within that prefix. The packet is sent through the topology using a valid source address chosen among the AS's advertised prefixes.

Following the attack models we wish to investigate, the spoofed source addresses may be fixed, e.g. `192.168.1.100`, chosen randomly from $\{0,1\}^{32}$, or taken intelligently from the set of all valid addresses. By this, we mean the set of addresses that are announced in the global routing tables and excluding bogons, reserved [129] and martians [73]. By using these "valid" or legitimate addresses, the attacker increases their chances of evading existing filters while remaining anonymous or assuming the identity of a third-party. To generate packets with valid source addresses, we assume the attacker has complete knowledge of all valid addresses and select the source at random from this set.

Algorithm 5.1 gives the basic form of the basic attack simulations; minor variations enable us to model all of the basic attacks given in §5.2.4.

Algorithm 5.1 Basic Attack Simulation Experiment

```
$n = m = 1$
**for** $i = 1$ to 10 **do**
    $AS_{sorted}$ =sort($AS, deg(AS)$)
    $f_i = AS_{sorted}[i]$
    $F = F + f_i$
    **for** $r = 1$ to $R$ **do**
        $as_1 \leftarrow AS \forall as_k \in AS, deg(as_k) = 1$
        $as_2 \leftarrow AS \forall as_k \in AS, deg(as_k) = 1, as_k \neq as_1$
        $p_1 \leftarrow P^{as_1}$
        $p_2 \leftarrow P^{as_1}$
        **for** $j = 1$ to $S$ **do**
            $a_1 \leftarrow p_1$
            $v_1 \leftarrow p_2$
            send($a_1, a_1, v_1$)
        **for** $j = 1$ to $S$ **do**
            $a_1 \leftarrow p_1$
            $v_1 \leftarrow p_2$
            $s_1^r \leftarrow 2^{32}$
            send($a_1, s_1^r, v_1$)
```

Our DDoS simulations follow a similar procedure, but assign a single edge victim at random and use increasing numbers of attackers distributed at random from the network. Algorithm 5.2 gives the pseudo-code for simulating DDoS attacks.

Ensembles of Weak Classifiers

In addition to providing a realistic IP routing substrate to experiment upon, the simulator provides an implementation of Raskol, our anti-spoofing filtering mechanism, that may be applied to one or more ASes. During a simulation, statistics are maintained at each AS for

Algorithm 5.2 DDoS Attack Simulation Experiment

```
$m = 1$
for $i = 1$ to $10$ do
  $AS_{sorted}$ =sort$(AS, deg(AS))$
  $f_i = AS_{sorted}[i]$
  $F = F + f_i$
  for $r = 1$ to $R$ do
    $as_{victim} \leftarrow AS \forall as_k \in AS, deg(as_k) = 1$
    for $n = 1$ to $N$ do
      $as_n \leftarrow AS \forall as_k \in AS, deg(as_k) = 1, as_k \neq as_{victim}$
      $p_{victim} \leftarrow P^{as_{victim}}$
      $p_n \leftarrow P^{as_n}$
      for $j = 1$ to $S$ do
        $a_n \leftarrow p_n$
        $v_{victim} \leftarrow p_{victim}$
        send$(a_n, a_n, v_{victim})$
      for $j = 1$ to $S$ do
        $a_n \leftarrow p_n$
        $v_{victim} \leftarrow p_{victim}$
        $s_1^r \leftarrow 2^{32}$
        send$(a_n, s_1^r, v_{victim})$
```

analysis.

When multiple ASes serve as classifiers, we use several methods to aggregate the decisions of each into a final filtering decision. In the most basic mode, the classifier for each AS acts autonomously. If any AS along the path a packet traverses determines that the packet contains a spoofed source address, the packet is dropped. We show in our results that while this approach is very effective in preventing spoofed packets, the false positive rate may be unacceptably high.

We make several important observations. First, the placement of our filtering algorithm is critical to the achieved performance. Without any filtering AS on the path, none of the spoofed traffic can be filtered. Second, the total number of filters a packet encounters contributes to the overall performance. Each AS has a different observable view of the network and thus can contribute in different ways. Given the ability to "tag" packets with classification decisions made by each AS on the path, the overall performance increases.

Therefore, the second mode of filtering is to make each AS a weak classifier and form a final decision based on the ensemble of all classifiers along the packet's path. Before considering how ASes can communicate decisions amongst themselves, we use an oracle view to "mark" each packet with the number of positive or negative classification votes it receives. The recipient of the packet then drops the packet if and only if the packet received at least one vote and the number of positive votes exceeds the number of negative votes. We term this oracle mode of operation as the "composite vote."

Naturally, the network cannot operate in the oracle mode. To propagate classification decisions within the packet to the next classifying AS along the packet's path, we again turn to the TTL field. Since the TTL field is one of the primary features each classifier uses to determine whether the packet is spoofed, one classifier can influence the next by artificially decrementing or incrementing the TTL field. Further, by only changing the TTL field, as is naturally done by each router to begin with, we do not change the normal semantics of the Internet in any way. We term this distributed mode of operation as the "weak ensemble."

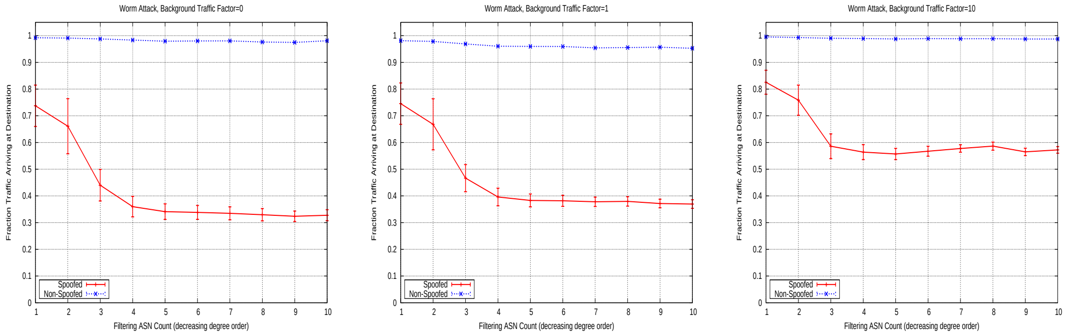
5.2.8 Core Filtering Results

Given our filtering schemes and simulator, we turn now to modeling spoofed-source attacks. Figure 5-8 presents the results from the worm attack experiment. In each iteration, a single source AS is taken as the attacker. The attacker uses random source addresses and sends packets to edge destination ASes. The simulator maintains, at each AS, the number of legitimate and spoofed packets received and filtered. In each graph, we vary the number of core ASes that implement our algorithm and perform filtering. We order the core ASes by degree and add each successive filtering AS by decreasing degree. In this fashion, we consider the effect of deploying our algorithm across a small number of the very largest ASes. Deploying among these core, high-profile ASes represents a much smaller hurdle than universal deployment or global cooperation.

Figure 5-8(a) shows the fraction of spoofed and legitimate traffic received across all ASes in the Internet versus the number of core ASes that are performing filtering. In this first case, we do not include any background traffic, but simply consider the case where the attacking AS sends both legitimate and spoofed traffic. From the figure, we see that with only one filtering AS, approximately 25% of the spoofed packets are filtered while nearly all of the legitimate packets pass unimpeded. As we increase the number of core filters, the amount of spoofed traffic decreases significantly to around 35% with five filtering points.

Next, Figure 5-8(b) depicts the same experiment, but includes background traffic at a factor of one. By this we mean that the simulator chooses random source and destination AS pairs exclusive of the attacker and victim and sends traffic between them. The simulator chooses $(bgfactor)(num_{training})$ of these random sources and sinks. Thus, a background traffic factor of 10 induces 10 times the amount of background traffic as training traffic in the experiment. In Figure 5-8(b), we see that the additional background traffic decreases the efficacy of the filtering, both by increasing the amount of spoofed traffic received and decreasing the amount of legitimate traffic received. The situation is even worse in Figure 5-8(c) where the background traffic factor is ten. Here, greater than 50% of the spoofed traffic passes the filters, even with ten core filtering ASes.

Figure 5-9 performs a similar experiment, observing the fraction of each traffic class received versus the number of core filters, for an attack where the source address is spoofed, but fixed. We see that there are nearly no false positives in this scenario, and more than

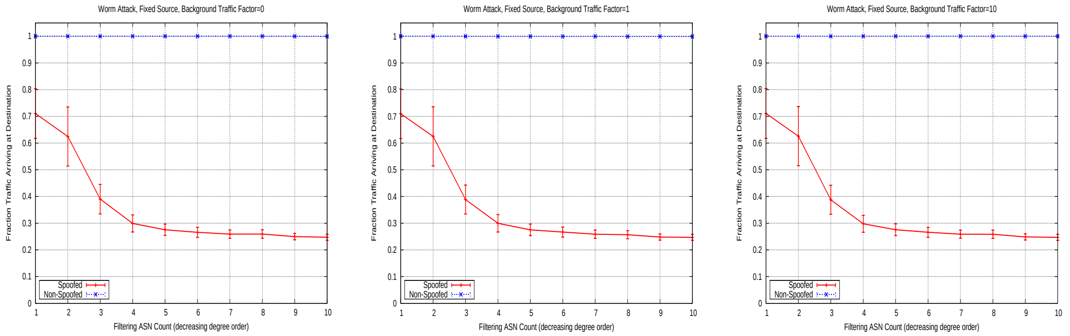


(a) No Background Traffic

(b) Background Traffic Factor = 1

(c) Background Traffic Factor = 10

Figure 5-8: Simulation of worm attack, single source, random source addresses

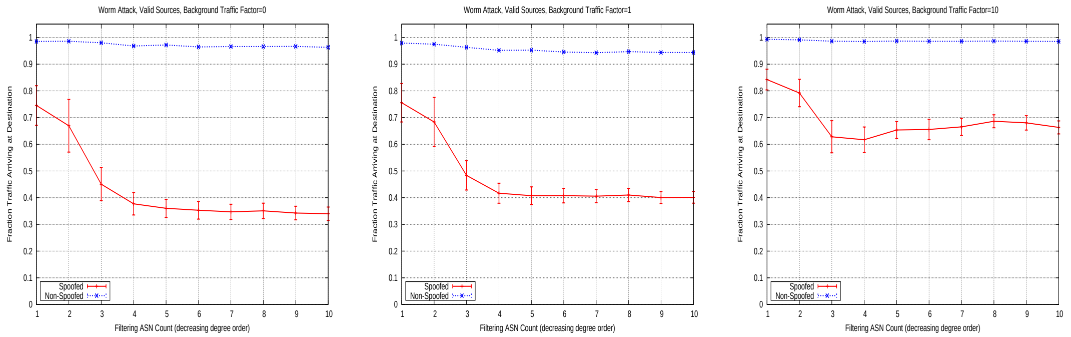


(a) No Background Traffic

(b) Background Traffic Factor = 1

(c) Background Traffic Factor = 10

Figure 5-9: Simulation of worm attack, single source, fixed source address



(a) No Background Traffic

(b) Background Traffic Factor = 1

(c) Background Traffic Factor = 10

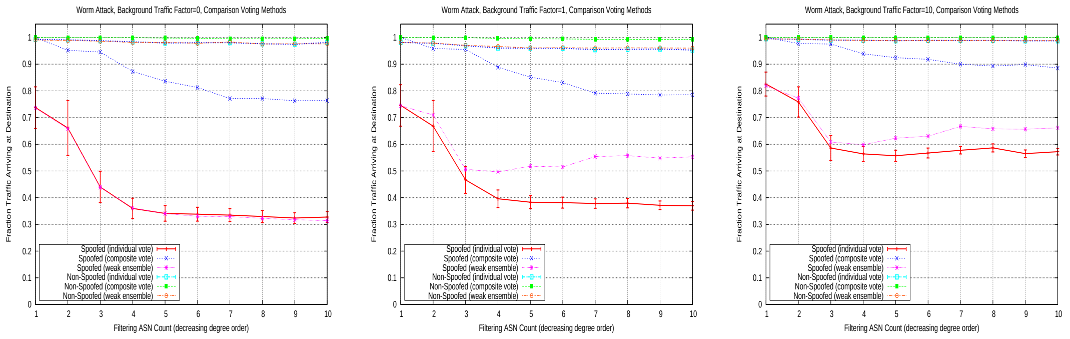
Figure 5-10: Simulation of worm attack, single source, valid source address

70% of the spoofed traffic is blocked, even in the presence of significant background traffic. This result meets our intuitive notion of the complexity the classifier must deal with; the spoofed traffic that is received is almost all a function of packets that traverse paths where no filters are in place. Thus, this experiment demonstrates the upper bound performance of the algorithm when considering a fixed number of core ASes.

Next, we consider a “smart” attacker, that selects source addresses from the set of all globally routed, and thus valid, addresses rather than at random. Figure 5-10 shows that such an attacker gains an advantage over a random attacker as compared to Figure 5-8. In all instances, more spoofed attack traffic received with less legitimate traffic received. With a background traffic factor of ten, more than 70% of the spoofed traffic passes the filters.

Our primary concern is in providing a defense with minimal impact on legitimate traffic, i.e. minimize false positives. Figure 5-11 considers the various means of performing distributed classification in our system and their relative performance against attack. In the composite vote instance, less than 0.01% of the legitimate traffic is falsely dropped at the cost of including a much larger fraction of the attack traffic. However, by implementing our method of a distributed weak ensemble of classifiers, we achieve *comparable performance to individual filters while maintaining near zero false positives.* Figures 5-11(b) and 5-11(c) show that our scheme is capable of maintaining this low false positive rate even in the presence of large amounts of background traffic.

Finally, we examine the DDoS attack case. While DDoS is one of the most important cases to consider, it is a subcase of the worm experiments given above since the victim is a single host. Figure 5-12 shows the performance across experiments that vary the number of attackers from 1 to 100 and include different amounts of background traffic. In Figure 5-12(a), we see the effect of different background loads in the experiment. Here, the effect is less pronounced as the large number of attacking hosts mimics the effect of background traffic seen in the worm experiments. With a background traffic factor of ten, we block at

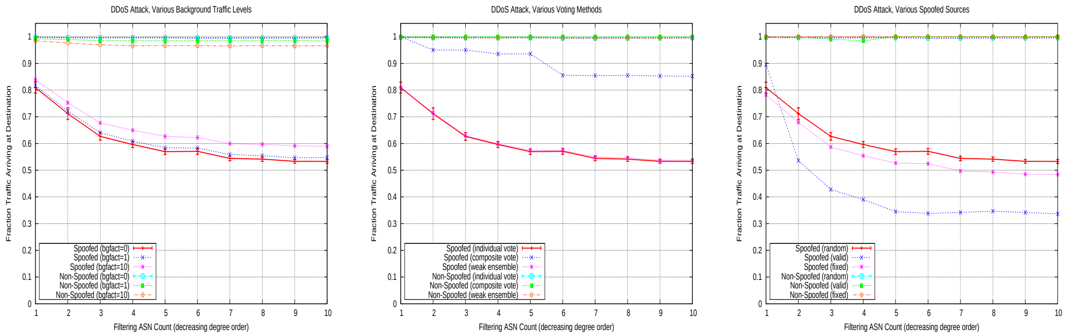


(a) No Background Traffic

(b) Background Traffic Factor = 1

(c) Background Traffic Factor = 10

Figure 5-11: Simulation of worm attack, various voting methods



(a) Various Background Loads

(b) Various Voting Methods

(c) Various Spoofed Sources

Figure 5-12: Simulation of DDoS attack

most 20% of the spoofed traffic and unintentionally drop approximately 4% of legitimate traffic.

In Figure 5-12(b) we see the effect of the different aggregate voting, or classification, schemes. The primary take away from this graph is that our distributed classification scheme with an ensemble of weak votes performs identically to the individual filtering situation, but without the corresponding increase in false-positives. This result holds even in the DDoS situation. Lastly, Figure 5-12(c) compares the filtering efficacy for DDoS attacks that employ different methods of source selection.

5.2.9 Implementation

We implement Raskol as a Linux kernel module to illustrate its speed and efficacy in a real system.

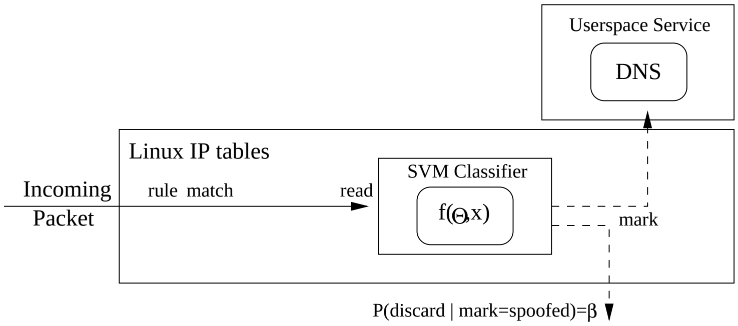


Figure 5-13: A Linux IP tables implementation. Packets matching a filter rule are sent to our classifier. Raskol classifies each packet and either discards or passes them to userspace.

IP Tables Module

The Linux IP Tables mechanism provides a powerful way for us to place Raskol's functionality not within any particular application, but in Kernel space such that it may protect any arbitrary application. An IP tables rule consists of a match criteria and a corresponding action criteria. The match criteria allows a great deal of flexibility; an example of a simple rule would be to match all UDP packets to port 53 originating from a particular network prefix.

The match action may be simple, for instance to simply drop the packet, or may be more complex, by dropping the packet with some probability. We implement Raskol as an IP tables module as shown in Figure 5-13. Incoming packets which match a filter rule are sent to the Raskol kernel module. Using our SVM classification methodology, Raskol makes a prediction of the validity of the source address. Based on that decision, the packet is either dropped or accepted and sent up to the userspace application.

Predictions must be fast. If Raskol cannot keep up with a high packet per second and bit per second rate, Raskol itself becomes the attack target. Since we apply known SVM methods in a novel arena, we present empirical performance results rather than computational complexity measures. Note that Joachims [80] contains an analysis of several techniques to reduce the time and space complexity of the quadratic programming required in SVMs.

On a 2.4GHz 32-bit Intel x86 architecture running Raskol, we measured the ratio of packets sent from a test host to packets processed by Raskol. Figure 5-14 shows the speed of Raskol as a function of Kpps since the classification is packet per second rather than bandwidth limited. Raskol handles more than 20Kpps and gracefully degrades until approximately 80Kpps at which point it processes approximately 80% of the packets. We note that these results are meant to demonstrate the feasibility of Raskol and better performance numbers are possible with additional programming and tuning. For example, Raskol engines can be chained. A higher performance architecture could accept all incoming packets. If the classification engine is busy, the packet is simply accepted and sent along to the next classifier in the chain. Thus, with multiple processors and dedicate hardware and

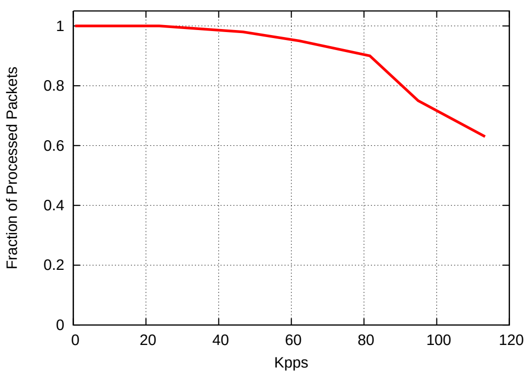


Figure 5-14: Raskol Linux implementation classification speed

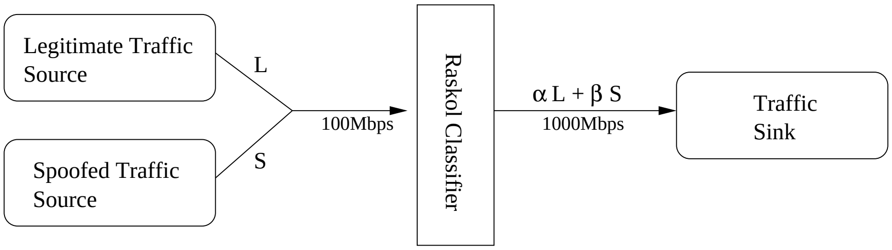


Figure 5-15: Attack Simulation Setup

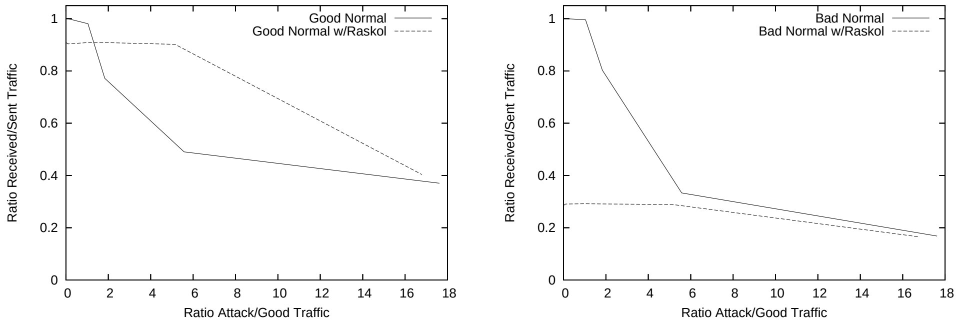
tuning, Raskol can defend against high traffic attacks.

Simulating the Attack

We build the simple test bed as shown in Figure 5-15 to simulate an attack. Two hosts compete for bandwidth on a 100Mbps link to a machine running Raskol. The classifier machine then has a 1000Mbps link to a traffic sink. The legitimate traffic source sends traffic at rate L while the spoofed traffic source sends traffic with random source IP addresses at rate S. We add an identifier into the payload of the spoofed traffic so that it may be differentiated from the legitimate traffic when we count the received packets at the sink. The sink receives a fraction α of L and β of S. Our goal is to ensure effective protection, i.e. $\alpha > \beta$.

Using this simulation setup, we vary the ratio $\frac{S}{L}$ and measure α and β with and without Raskol's classification engine active. Figure 5-16(a) plots αL with and without Raskol. We see that even with six times more spoofed than legitimate traffic, 90% of the legitimate traffic is received by the sink whereas less than 50% is received without Raskol active. Past an order of magnitude greater spoofed traffic than legitimate traffic, the link is saturating and hence Raskol's performance gracefully reaches the same level as without Raskol.

Figure 5-16(b) shows the fraction of spoofed traffic received as a function of the traffic composition ratio (βS vs. $\frac{S}{L}$). With Raskol turned on, we see the fraction of spoofed packets received is less than 30%.



(a) Fraction legitimate traffic received as a function of traffic composition ratio (αL vs. $\frac{S}{L}$)

(b) Fraction spoofed traffic received as a function of traffic composition ratio (βS vs. $\frac{S}{L}$)

Figure 5-16: Raskol protection performance against a random attack model

5.2.10 Related Work

Several proposed packet marking [15] and traceback [31, 137] mechanisms provide a means to trace spoofed packets to their origin, removing the advantage of anonymity. Yaar gives a substantially improved traceback scheme in [152]. Unfortunately, network support for such schemes is minimal and network administrators bombarded by attacks prefer proactive rather than reactive measures.

Previous research investigates various means of victim-centered DoS attack mitigation methods. These methods generally fall into either address or path-based approaches. Address-based techniques [89, 81], profile traffic received during normal periods to create a model against which to judge incoming addresses while under attack.

In contrast, Jin et al. give a path-based scheme to block spoofed packets based on implicit hop count [78] while the Pi path identification scheme [151] explicitly encodes path information into each packet. In a similar vein, Duan et al. detail a filtering mechanism based on feasible path construction [55]. An extensive simulation of these victim-centered schemes [47] reveals surprisingly high false positive rates and require a substantial learning period. Our scheme, given its tunability and high accuracy is a step toward addressing many of the short comings of previous approaches.

Despite these research efforts, finding and preventing spoofed traffic remains a difficult problem for network operators [65].

5.2.11 Next Steps

A natural question is what probability the adversary has of guessing a TTL that will allow a packet with a spoofed address a to pass the classifier, particularly if the adversary can test all possible TTL values in order to determine the path length from s to t.

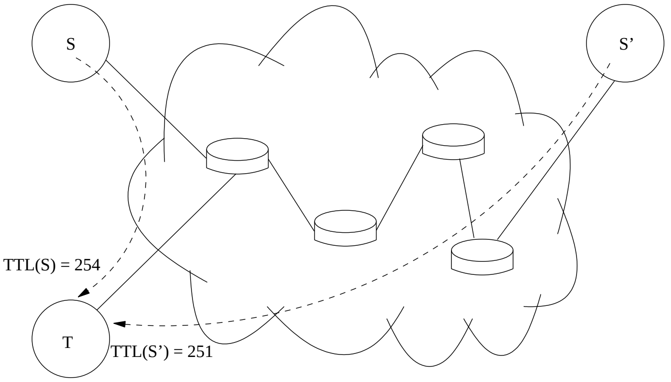


Figure 5-17: An adversary S' is unable to spoof S for any initial TTL value given the imposed topology assuming S sends all traffic with a TTL=255.

In order to defeat such scanning attacks, the agent may react by blocking all traffic originating from a that has an incorrect TTL for some period of time. In addition, if all operating systems default to setting the initial 8-bit TTL value to 255, a malicious node s which is a distance of $|p(s \rightsquigarrow t)| > |p(a \rightsquigarrow t)|$ will not be able to find any acceptable path length to spoof. For example, assume the topology given in Figure 5-17. Legitimate traffic from node s to t arrives with a TTL of 254. An adversary s' will not be able to spoof s for any initial TTL due to the imposed topology. Such a change requires unifying operating system behavior with respect to the initial TTL value, but has no functional impact on either existing applications or infrastructure. An equivalent formulation is to change the semantics of the TTL field such that hosts always source the packet with an initial value of zero while routers increment the TTL. While an adversary can then make themselves appears arbitrarily distant from the source, it is impossible for them to fake fewer hops than their actual hop distance to the destination.

5.3 Problem 2: An Intelligent Routing Plane

The "optimal" route in today's Internet is rarely as simple as the shortest path length. Rather, optimal paths may be a complex combination of hops, latency, available bandwidth, economic cost and other factors. Further, the Internet is comprised of interconnected autonomous systems that both cooperate and compete. Therefore, nodes and routers may have different notions of the value and importance placed on these variables.

Existing Internet routing protocols do not directly accommodate high-level objectives and policy primitives, e.g. maximizing revenue or minimizing loss, and have no means to evaluate their performance in achieving such objectives. The result is a routing system which is brittle and often inefficient, particularly when reacting to non-standard situations such as attacks and faults.

In this section, we propose adding additional sophistication to the routing plane in the form of learning. An intelligent routing plane is a step toward designing a system that naturally accommodates various “tussles” [45] between actors and allows mediation *within the system* as part of its natural design. In examining learning within the routing system, we make the following contributions:

1. Separation of traffic into classes in order to tackle the exploration versus exploitation tradeoff and prevent convergence to local minima in this distributed learning environment. Best effort traffic, for instance peer-to-peer “bulk” traffic is used to explore available paths, while interactive, or “high” priority, traffic receives the benefit of optimization.

2. Use of the IP clustering method given in §4.3 to address issues of memory consumption. There is no fixed memory requirement, rather routers learn to optimize their available memory.

3. Simulations that model current Internet inefficiencies and demonstrate the power of an intelligent routing plane to overcome certain types of sub-optimality.

4. Identification of learning as a general framework for additional classes of routing problems.

Our hope is to make different design choices better suited to a modern Internet while remaining consistent with end-to-end design arguments. The approaches we consider aim to provide many of the design criteria first brought forth in [44] including increased transparency, accommodation of change and mediation along tussle boundaries.

5.3.1 Internet Routing Background

Routing protocols address a core problem in networks: implementing a distributed computation whereby nodes examine available edges and determine best paths between sources and destinations; see [50] for an algorithmic overview. Conventional routing protocols optimize over variables that are relatively static over short time-scales such as path length or edge weights. However, the Internet’s transition from an academic to commercial Internet has challenged notions of “optimal” routing. For instance, an optimal Internet path may be a complex combination of path length, latency, loss, available bandwidth and economic cost. In addition, each network along an end-to-end path may have different or conflicting notions of optimality.

To implement policy and effect commercial relationships, operators rely on the blunt mechanisms available to them from the Internet’s distributed routing system, Border Gateway Protocol (BGP) [128]. BGP organizes entities under common administrative control into autonomous systems (ASs). BGP is well-studied, yet several persistent core problems in Internet routing, which we detail in §5.3.3, continue to drive active research. For example, recent work examines the extent which BGP plays in affecting performance [148].

This research adds additional intelligence to the routing plane. Because of the dynamic, complex and multi-party nature of the Internet, routing is naturally a learning task. This intelligence must thus include a strong notion of learning. Indeed, the Internet is characterized by many entities with conflicting interests and strategies and hence incomplete knowledge that requires exploring various state spaces. Learning enables the network to react rationally to unknown situations and environments. For instance, anticipated and unanticipated events such as faults, attacks, peering and pricing changes all currently necessitate manual intervention. We therefore contrast our approach with pure algorithmic optimization problems with complete knowledge.

Providers optimize their internal network given historic load, expected failure scenarios, contracts, etc. Recent work shows a means to perform dynamic on-line intradomain traffic engineering [84]. However, Hu et al. find that the primary source of bottleneck links is between ASs [70]. Because the Internet is a system of interconnected commercial entities, end-to-end paths often travel through multiple providers. Thus, nodes may experience connection failures and congestion even though their own provider has a fully optimized internal network. Therefore, to tackle a concrete learning problem with tangible benefit, we examine the following hypothesis: *end-nodes suffer communication failures due to provider's inability to optimize their available inter-AS capacity subject to real-world constraints.*

5.3.2 The Role of Overlays

As critical infrastructure, Internet end-to-end connectivity failures and congestion are increasingly problematic. For example, applications such as voice and video are especially sensitive to network disruptions.

Commercial overlay [5] and routing indirection services [76] attempt to compensate for some of the Internet's shortcomings. By providing indirection points, these overlays can often achieve superior performance on select paths, but at the cost of significant measurement burden. Yet, even with buffering and overlays, applications cannot mask persistent soft and hard internetwork failures.

Further, giving end-nodes routing choice via an exogenous overlay system addresses the same problem as placing such functionality within the network. Optimization within the network is appealing from a learning perspective because more information is available. Routers aggregate the data, information and policies of many networks and nodes. In contrast, end-nodes have a limited view and only perform local optimization or optimization among collaborating nodes.

In contrast to overlays, we advocate additional intelligence *within the routing system* as part of its natural design (§5.3.5). As the Internet increasingly becomes critical infrastructure, the network must be capable of providing the same benefits as overlays realize for a few without limitations of view, scope or scale. The network should be capable of optimizing the multi-dimensional problem of potentially conflicting provider and user needs.

5.3.3 The Problem

Internet routing is a classic example of cooperative competition and a current "tussle" space [45]. Service providers are commercial entities that compete, yet must cooperate to facilitate global connectivity. Border Gateway Protocol (BGP) [128], the Internet's interdomain routing protocol, logically groups networks with a common administrative policy into autonomous systems (ASes). The natural marketplace that has emerged from BGP's design is one where ASes negotiate connections to other ASes on the basis of their relative sizes, traffic volumes and traffic balances, etc. Depending on these variables, an AS may pay a provider AS for connectivity or arrange a "settlement-free" peering arrangement where no money is exchanged. Peering relationships are established when two ASes realize that a fraction of traffic they exchange via an upstream provider can instead be off-loaded in a mutually beneficial connection.

Thus, ASes must implement routing policy based on business relationships. Configuring routers today is a highly manual, technical, error-prone and detailed process akin to programming in assembly language. Simultaneously, users connected to the network are players in the tussle that seek service, contribute load and may even send malicious traffic.

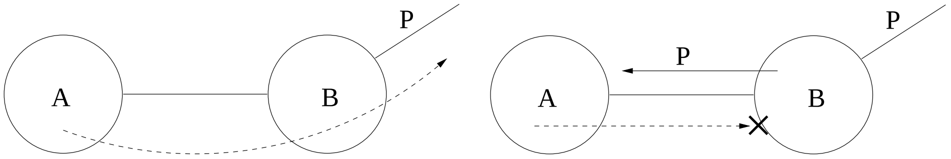
The Internet was designed to provide transparent, best-effort core transport. In fact, making minimal assumptions about the functionality the network provides has been identified as one of the enablers of the Internet's success [41].

Along many metrics the Internet is a phenomenal success. What then are the problems with the current routing architecture? A large body of research is devoted to exposing both theoretical and empirical routing failures on the Internet. Recently, Wang et al. demonstrate the adverse impact of certain BGP routing events on end-to-end traffic behavior [148].

Rather than incremental solutions to specific shortcomings in the routing system, we posit that several fundamental problems exist which are insurmountable without an architectural shift. These interrelated issues include:

- **Coupling between reachability and policy:** ASes will route traffic to prefer customers over peers and peers over providers [61]. To enforce policy constraints which are driven by economic goals, operational networks typically attempt to drive behavior with the blunt mechanisms available in BGP. For instance, operators will artificially modify reachability information or employ BGP path-prepending and local preference values to influence routing. These BGP mechanisms are often utilized simply as a means to lower transit cost. For example, a network wishing to selectively carry traffic might not announce an available, reachable prefix to a peer. The coupling between policy and reachability both complicates the job of network operators and leads to non-obvious faults and failure modes [57].

- **Security:** Not only do users not necessarily trust one another, they may have different or adverse interests [44]. Users may wish to be protected from particular classes of



(a) Network A has no route to P from peer B, yet can still send traffic to P.

(b) Network A has a route to P, but B silently discards A's traffic to P.

Figure 5-18: Failures of policy enforcement in BGP.

traffic as defined by the network, remote entity or application. The lack of security and attack prevention mechanisms in the architecture are responsible in large part for continual denial of service (DoS) attacks and other exploits.

- **Policy Enforcement:** Even if network A has no route to a prefix P from a peer B, nothing prevents A from forwarding a packet destined for P to B (Figure 5-18(a)). Conversely, network B might advertise prefix P to peer A and yet silently discard all packets destined to P (Figure 5-18(b)). Thus, not only is reachability information a coarse and ill-suited tool for traffic engineering, it alone does not suffice in enforcing policy [133].

- **User Control:** Users have no control over the path their packets take in the current routing system. As a result, overlay and content distribution networks have emerged as an indirect means to influence path selection.

- **Inefficiency:** Optimizing the complex multivariate nature of policy-compliant routing is difficult [141]. In addition to routing loop pathologies, path inflation, hot-potato routing and path asymmetry all arise from policy requirements. Previous research has shown that the widespread manual tuning by service providers makes the entire network less efficient.

- **Incentives:** Current routing has no way to directly accommodate the incentives of various players in a highly commercialized Internet. There is a large divide between today's routing protocols and the major objectives of service providers, e.g. maximizing profit, providing differentiated services, enabling reliability, etc [3].

We propose *learning* as a natural solution to many of these problems in the current routing architecture. Rather than point solutions, we advocate a fundamental design shift which elevates learning to a first-class component of future architectures.

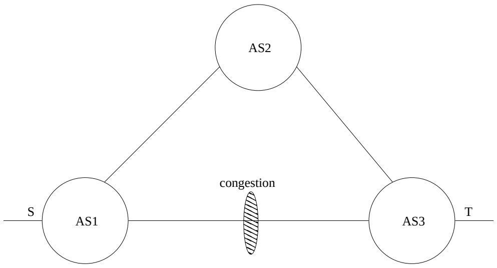


Figure 5-19: Sub-optimality: A feasible non-congested path exists from AS1 to AS3 but communication between s and t still experiences congestion.

5.3.4 Sub-optimality Example

The detour measurement study is one of earliest results demonstrating that the Internet's path selection can be suboptimal in terms of packet loss, availability and latency [132]. The resilient overlay network (RON) work from Andersen et al. [7] recognizes the potential inefficiencies of interdomain routing and demonstrates a more efficient, albeit less scalable, path selection methodology. In particular, RON shows that a single level of indirection, as opined by the detour project, suffices to improve the loss probability. Yang and Wetherall show how hosts could potentially use deflections to influence route selection [155]. Commercial CDN and overlay services [5, 76] employ similar techniques whereby they continually measure potential paths and direct traffic over high-performing paths.

As an illustrative example, consider Figure 5-19 with three ASs. Source s and sink t are customers of AS1 and AS3 respectively. With simple shortest AS path routing traffic between s and t will be preferentially routed over the link between AS1 and AS3. Take the case where the AS1 to AS3 link is congested. We realistically assume point-to-point links where BGP messages receive priority; therefore while a link may be congested, the BGP session remains up, often termed a "brownout." AS1 has a feasible path through AS2, but will not use it because AS1 has an existing available best path. If the path through AS2 has sufficient capacity to support the s to t traffic flow, AS1 might elect to use this alternate path. We call this situation Pareto suboptimal: an alternate allocation of traffic between members of the system can make at least one AS better off without adversely affecting any other AS.

Of course, ASes have complex business relationships with their customers and other ASes. The example in Figure 5-19 holds if we assume that AS1 and AS3 have a peering relationship, while AS1 and AS3 are customers of AS2. In this case, AS1 may prefer to send traffic via AS2 given the AS1 to AS3 link is congested, even though this alternate path is potentially more economically expensive to both AS1 and AS3. But AS1 has no way to enact such a policy. In other words, the economics may dictate sending traffic over a more expensive path when it benefits the customer of the AS.

5.3.5 Learning in the Routing Plane

To accommodate the aforementioned tussles, a new routing architecture needs a high-level notion of how well it is performing. This performance can modeled as a multi-dimensional optimization problem in the sense of a utility function or reward. Reward is both immediate, based on local knowledge, and delayed in the form of feedback from prior decisions. Routing then becomes a distributed optimization to maximizes an individual AS's notion of utility.

Routers have both fixed information and dynamic information. Fixed information includes the set of links, routers, negotiated contracts, users, etc. Dynamic information includes e.g. traffic levels, congestion, and internal and external packet loss. Because of the non-deterministic, non-stationary nature of Internet routing, the optimization problem necessarily involves learning: exploring routing paths and predicting forwarding decisions expected to maximize long-term utility. We refines these notions of utility and horizon in subsequent sections.

Rather than complicated error-prone router configuration, consider a highly expressive language stating policy as a higher level abstraction [66]. Such a global policy could be distilled into a configuration for the routing plane to implement. With different policies, the network could easily accommodate many objectives, including automatically routing around faults or minimizing packet loss. In this way, the network is automatically adaptable, efficient, reconfigurable and resilient in a way that is not currently possible.

For many network architects, the core of the network must remain "dumb." Including intelligence anywhere other than at the very edge of the network is an anathema. In reality, however, the core of the network already contains a large amount of intelligence. Routers maintain hundreds of thousands of routes and often many thousands of tunnel circuits. Indirectly, networks are highly traffic engineered in order to maximize resilience and efficiency.

A useful analogy is with the prevailing opinion a decade ago that routers could never forward variable length IP packets, with variable length network masks, at line rate. Subsequent research proved this common wisdom false [117] and commercial routers now forward at line rate.

To demonstrate one potential advantage of such learning, we propose a different notion of quality-of-service (QoS) as an example of our larger framework. We divide a network's customers into classes and use traffic from best effort customers and services to explore available routing paths. By probing alternate paths with low priority traffic, the network can substantiate decisions that optimize performance for high priority customers (§5.3.11).

We explore only the ability of an AS to learn a policy that improves the performance of high-priority flows. This problem provides a concrete example to explore learning within the routing plane. However, we believe that the utility of learning within a routing context is much more general; learning provides one framework for *separating desired goals and constraints from their implementation.*

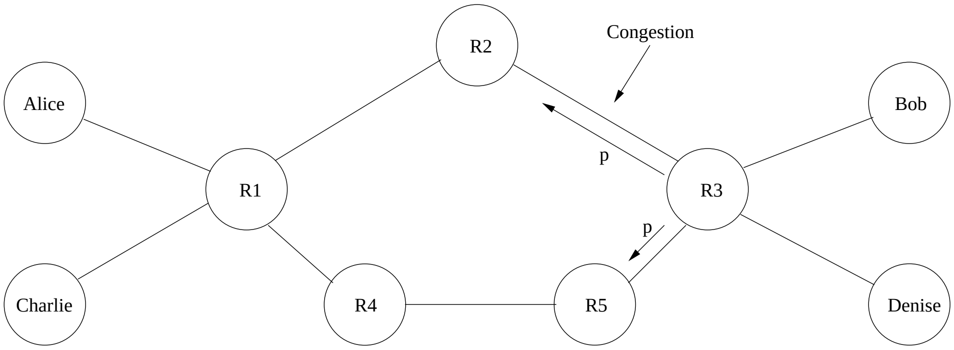


Figure 5-20: $R1$ has no visibility into the persistent congestion between $R2$ and $R3$. Customers of $R1$ are best suited to evaluate their performance, yet are unable to influence their route and avoid poor paths. Learning provides a means to optimize routing and mediate between users and providers.

5.3.6 Designing for User Control

As a basis by which to reason about learning within the routing plane, consider the simple example in Figure 5-20. Alice and Charlie are customers of a common provider and are connected to router $R1$. Similarly, Bob and Denise are connected to $R3$. Let Bob and Denise belong to a BGP prefix p advertised by $R3$ and received by $R1$. $R1$ thus has two paths for p in its routing table and will select one deterministically based on policy. Say that the shortest path via $R2$ is the preferred route for $R1$ to p. Further assume that congestion and packet loss occurs on the $R2$ to $R3$ link. We make the following observations:

- With traditional routing algorithms as used today in the Internet, persistent congestion or packet loss will not cause traffic to reroute. Traffic from Alice to Bob is affected by the congestion, yet Alice and Bob have no way to influence the route their traffic takes.

- Congestion may be outside the local autonomous system where it is difficult if not impossible to detect in a scalable manner. For example, while explicit path probing is viable for maintaining the integrity of intradomain performance, probing all end to end paths is infeasible.

- $R1$ has no visibility into the performance of downstream paths. The entity best able to assess the performance of an end-to-end path is an end-node.

Suppose the connection between Alice and Bob is poor due to the congestion on link $R2$ to $R3$. The application which Alice is running may be insensitive to the congestion and Alice is well served by the network. On the other hand, Alice may elect to signal her local router that the connection is poor.

Overlays attempt to circumvent such path problems by routing traffic within the overlay. Rather than making this process exogenous to the routing plane, we examine the fundamental problem within the routing layer through distributed knowledge.

For the sake of developing intuition, we consider how $R1$ might reason and learn in its environment. If Alice signals negative reward to $R1$ for her traffic's performance in reaching Bob, $R1$ could make the following decision: "Alice is telling me that the connection to Bob is poor. I know that Bob is part of a larger aggregate with a common policy, namely the BGP advertisement p. I have an alternate path to p. To maximize my long term reward, I will move Alice's traffic to p to the alternate path. Further, for a new connection from one of my customers to any other host in p, I will predict that the best path is the alternate path." In this way, Charlie's traffic to Denise is similarly optimized, demonstrating how the IP clustering methods (§4.3) can be utilized in a routing context.

Observation 1: Hosts signaling information to routers is a valuable design with end-to-end design [131] merits. Protocols such as XCP and ECN [125] allow routers to signal congestion information to end hosts. In contrast to explicit congestion notification proposals, hosts may usefully signal information to routers. End hosts are best-suited to to signaling routers as they are the only entity with a complete end-to-end view of the path.

Observation 2: The Internet was designed to be best effort. Implementing heavy weight mechanisms to ensure packet delivery, latency, lack of congestion, etc. are counter to the end-to-end arguments. Similarly, our architecture endows hosts with the ability to signal the network as to how well it is performing relative to the expectations of applications. For instance, in the case of packet loss, an application that can tolerate the loss may be unwilling to pay extra for better packet delivery probability. Our design maintains end host choice.

Observation 3: User directed routing provides demonstrable benefit, but ISPs are reluctant to cede control. We mediate this tussle by adding learning into the routing substrate.

5.3.7 Designing for Efficiency

As a second motivating example, consider the classic example of "hot potato" routing in Figure 5-21. AS1 and AS2 peer in two geographically distinct locations. When Alice sends traffic to Bob, router $R1$ has two possible internal forwarding paths. However, $R1$ will typically send traffic destined to AS2 via $R3$ rather than carrying the burden of transiting the traffic across its own network and exchanging packets at the $R2 \rightarrow R4$ interface. Since AS2 has similar economic motivation, Bob's return traffic to Alice is asymmetric.

Consider congestion in AS2 along the $R3 \rightarrow R4$ path. This congestion is not visible to AS1. If AS1 is attempting to provide a premium service to Alice, it might instead opt to carry her traffic internally in order to route around problems in AS2, had AS1 had complete knowledge. Note that we are not arguing that congestion in particular should be made visible to the routing system – rather end-nodes can usefully send feedback to the routing system to achieve policy objectives.

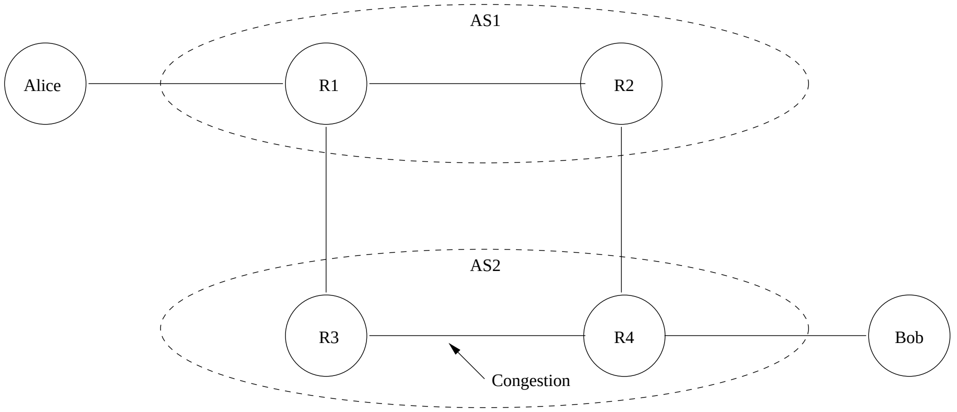


Figure 5-21: Hot potato routing sub-optimality effects: because of AS1's limited visibility into the routing system, AS1 may make inefficient internal routing decisions. The result is potentially provide poor, yet avoidable, service to its own customers.

Observation 4: Autonomous Systems are part of a global system. Hence, an AS can provide a high-bandwidth, non-congested network and yet customers of that AS can still experience *avoidable* suboptimal performance. This performance degradation is suboptimal relative to an alternate path available to the AS.

Observation 5: ISPs are eager for ways to differentiate themselves amid rapid commoditization of IP transport. Service differentiation gives ISPs a means to achieve value pricing whereby they use performance as a means to separate business customers from the casual web surfer.

Observation 6: Both explicit user feedback and service levels allow ISPs to provide service differentiation. In this manner, ISPs can achieve value pricing whereby they use performance as a means to separate business customers from the casual web surfer.

5.3.8 Designing for Incentives: Monetizing the Tussle

As Afergan notes [3, 1], service providers are primarily concerned with maximizing profit, yet are forced to use the inexact and blunt mechanisms available in BGP to effect commercial realities. An alternate view of routing in a highly commercialized Internet is that policy alone suffices as the sole determinant of packet forwarding. Maximizing profit may involve selecting the lowest-cost route among a set of feasible forwarding paths, or providing differentiated services to select customers, thereby allowing the provider to accrue subsequently higher monetary benefit.

A new routing architecture should make explicit recognition of incentives in the system. Given that providers have a rich set of business goals, an intelligent routing substrate that implements policy is appealing. A routing infrastructure based on a learned policy naturally accommodates a language that providers really want to use.

For instance, a router's policy might be as simple as minimizing cost while ensuring

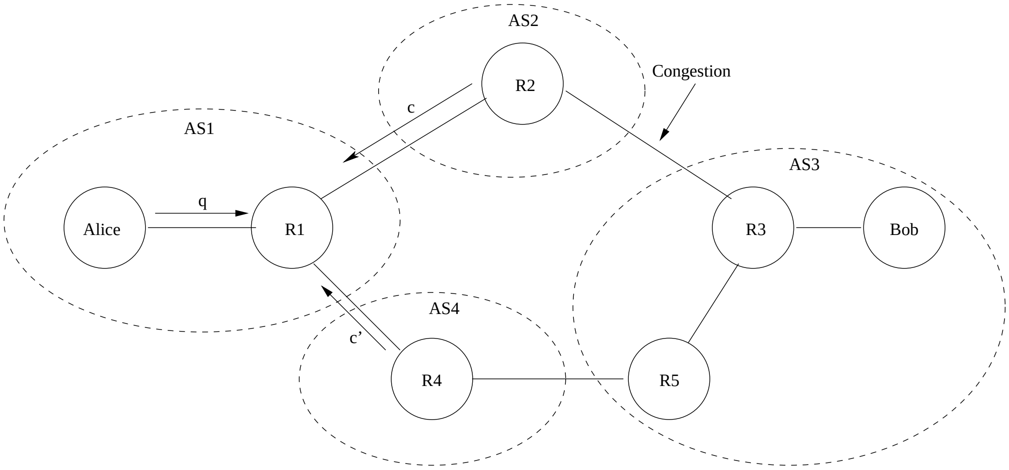


Figure 5-22: Monetizing the routing tussle implies strategic games and suggests learning as a means for providers to operate within the induced game.

successful delivery. This very simple policy, corresponding well to the business objectives of many present day ISPs, is nearly impossible to implement with today's technology. Consider instead an intelligent routing plane. A router might understand the desired policy as specified by the network operators, experiment with different forwarding paths and learn from the results.

A router could forward a packet with destination t to a congested, but lowest-cost peer P. Assume for the moment that the router has oracle knowledge and finds that the packet is dropped. Because the packet is dropped, the router receives a negative reward. Upon continued negative reinforcement, the router will instead switch to a higher-cost provider that does not drop (any fraction of) packets destined to t. The router receives positive reward, less than when using the lowest-cost provider, but yet better than the negative reward. Periodically, the router will experiment with alternate paths. When the lower-cost provider P is no longer congested and does not drop packets, the router will switch back to using P for destinations t to further increase its cumulative reward.

Figure 5-22 re-examines the previous example (Figure 5-20) of avoidable congestion, but in the context of monetary relationships. This simple, but concrete example, immediately reveals many interesting problems in monetizing the routing tussle. Alice is in AS1 and connected to $R1$. AS3 advertises a prefix p where Bob is a member of p. AS2 advertises p to AS1 with cost of c while AS4 advertises p to AS1 with a cost of c'.

Let Alice and AS1 negotiate a variable rate contract where for every transaction[4], Alice agrees to pay some amount q. If on finding that her connection to Bob is poor due to the congestion between AS2 and AS3, Alice could change her price to $q' < q$. If $q' > c' - c$, $R1$ may choose a policy to move Alice's traffic to $R4$. However, $R1$ might also reason that Alice is willing to pay more for the better service if no competing service exists: $q' > q$. Thus a

[4] A transaction is loosely defined here, it might be a packet, flow or other measureable metric.

monetized routing scheme implies a game, an observation that has been made previously [3]. Monetizing the tussle preserves the interests of the majority of stakeholders. Our insight is that routers can learn in order to operate and intelligently make predictions within the confines of such a game.

Observation 7: Learning allows routers to operate intelligently within an economic routing game.

An important point to note is that reachability need not be used to define policy in an intelligent routing architecture: a provider can advertise all paths according to the economic burden of carrying each traffic class.

Observation 8: Routers need not concern themselves with the details of policy enforcement. If end hosts signal their degree of satisfaction with the current policy, the routers indirectly learn of non-compliant domains.

5.3.9 Designing for Service Differentiation

ISPs are increasingly purveyors of a commodity service: transporting bits across their network. While content providers benefit from continual innovation, ISPs are forced to become more efficient in order to remain competitive. An alternate response that many ISPs have pursued is offering bundled services such as security protection, Virtual Private Networks (VPNs) and other premium services.

An innovation that other industries have made, that ISPs have yet to fully exploit, is fine-grained price discrimination. For years the research community has proposed service differentiation in the Internet through mechanisms such as IntServ [28] and DiffServ [26] whereby different classes of traffic receive different treatment, e.g. priority, in the network. While these quality-of-service mechanisms are present in many enterprise networks, they are mostly non-existent within the Internet.

Consider an ISP that uses learning to route. As with all learning problems there is a natural balance between exploration and exploitation. To find the best paths, the network must explore different possible routes. For how long should the network explore alternate paths before using (exploiting) a path? Because network conditions are continually changing, the learning must periodically probe secondary paths to determine if a secondary path provides better performance that a current primary path.

Many strategies exist for probing alternate paths. For example, RON overlays [7] continually send probe packets to assess the state of paths within the overlay. In contrast, our insight is that ISPs can separate users into two (or more) classes, for instance a “gold” and “bronze” class. Traffic from gold customers is always sent over the best path. Traffic from bronze customers, however, may be sent over alternate secondary paths in order to provide probing and feedback for the learning algorithm. Customers that always want to receive premium service will be willing to pay more, whereas bronze customers pay less but may have their traffic sent over sub-optimal paths.

The distinction between bronze and gold may also be made on the basis of application. For instance, providers may discriminate between bulk and interactive traffic. Bulk traffic may be in the form of peer-to-peer file sharing where providers may wish to shift the burden of such applications to their benefit. In such bulk traffic applications, there often exists a level of degraded performance that users are willing accept, or alternatively, are unwilling to pay to improve. In contrast, gold traffic might include interactive applications such as web traffic which has a high-impact on user satisfaction. We do not attempt to provide any classification scheme for users or applications, but note the flexibility of traffic classes to accommodate various useful policies.

Observation 9: A routing plane can provide service differentiation using low priority traffic to learn path performance. The learning thus provides a natural means for price discrimination within the network.

5.3.10 The Learning Problem

The notions of non-deterministic reward and delayed feedback for learning policy and making predictions corresponds directly to ideas from the reinforcement learning community [82]. Routing becomes a learning problem, where the agent begins learning by probabilistically choosing an egress interface for an incoming packet. The agent maintains state and receives feedback about the success of each routing decision, in the form of reward, in order to substantiate better future decisions. To better formalize the learning task, we add the following notions:

1. ***Action Space:*** Routers are agents with a set of links to other routers. Routers service incoming packets f_i with destination addresses d_i and must forward the packet out an available interface.

2. ***High-level Configuration:*** Routers are configured with high-level objectives, e.g. maximize profit subject to constraints C.

3. ***Reward:*** The notion of reward becomes a primary objective function. Reward is the sum of immediate and delayed reward for taking an action a on a packet f_i: $r(a, f_i)$.

4. ***Signaling Feedback:*** Routers receive feedback on the success of prior decisions to substantiate future policy. This feedback may be in the form of user control with end-nodes signaling local routers with performance feedback as presented previously. Alternatively, the system might infer a TCP flow's performance based on the SYN and ACK stream. Such flow monitoring is possible with e.g. netflow [40].

5. ***Predictions:*** Reward is dynamic, therefore some actions will be a prediction of the optimal action with delayed or incomplete feedback.

6. ***Learning:*** Routers learn an on-line policy Π which maximizes the expected long-term reward, i.e. Π is an estimation of the optimal means to achieve the high-level configuration. A policy Π defines the optimal action to take given the agent's learned history, operating horizon and current forwarding task. One determinant is a policy which maximizes the infinite horizon reward: $\text{argmax}_{\Pi} \sum_{i=0}^{\infty} r()$.

Learning Problem: Given non-deterministic, delayed feedback from multiple sources, how can a router learn a policy that best estimates the high-level configuration objectives.

Reinforcement learning to determine an optimal policy has many potential benefits such as a natural multi-path routing, the ability to automatically load balance links and route around multiple failures. We further postulate that such a routing infrastructure would facilitate tenable solutions for handling attacks on the network. For example, DoS attacks, when recognized, could be squelched at their source given a policy that provides a recursive negative reward.

Moving the routing infrastructure to a model of learning is also a move toward a probabilistic world. The computational complexity of machine learning problems have limited their applicability in other problem domains. The sheer volume of traffic in even a single provider's network provides a large and ample amount of data on which to make informed decisions. The limiting research problems then are memory, communication cost, convergence, stability and incremental deployment.

Thus, an important component of this research is in decomposing problems of learning in a distributed environment in ways that make a statistical approach is tractable. Earlier work presented in this thesis provides some of the building blocks for achieving these goals, for instance the IP clustering algorithm and feature selection techniques.

To investigate learning within the routing plane, we begin by augmenting current inter and intra-domain routing protocols. While future work may relax this assumption, we note that leveraging available routing information in the current network provides a stable basis on which to improve and guides the search space in learning. Such model-specific information can substantially improve convergence time. Specifically we wish to understand how additional intelligence can aid the routing task without sacrificing existing structural information.

5.3.11 Experimental Results

To better understand the potential benefit of our proposed learning, we implement reinforcement learning using multiple traffic classes and clustering in a custom simulator. Our simulator models a set of ASes connected by a topology and tasked with a traffic load. The objective of the learning AS in this experiment is to minimize the number of its high priority flows that experience congestion. The AS explores potential alternate paths with lower priority traffic flows.

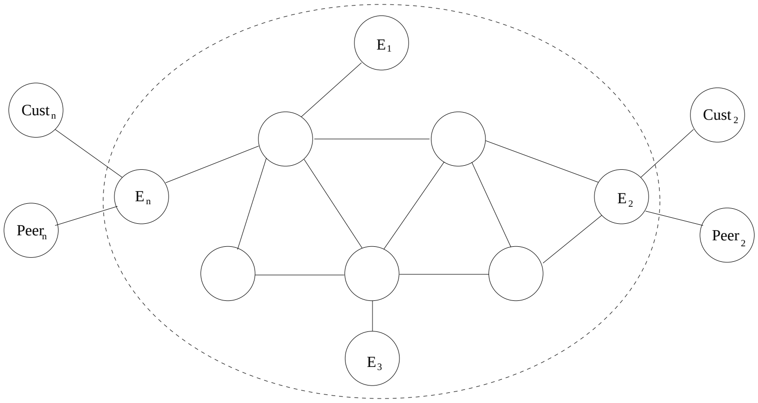


Figure 5-23: Experimental transit network model

Our simulations make two main simplifying assumptions which we justify here. First, the AS must be able to gauge the effectiveness of prior decisions. We assume that the AS has the ability to detect congested flows, for instance by observing TCP retransmissions or timeouts. Detecting and maintaining such information is well within the technically feasible bounds and implemented in commercial routers [39] as part of netflow version 9 [40].

Second, while an AS is comprised of many individual routers, we model it as a single entity and assume perfect communication between all routers within the AS. Because all routers within the AS are under a common administrative control, this second assumption is well founded.

Figure 5-23 provides a depiction of the transit network model we evaluate. We consider a single AS with an internal core as well as n external customer and peer facing routers $E_1 \dots E_n$. For traffic entering E_i with destination IP address d, the network chooses an egress E_j and the best $E_i \rightsquigarrow E_j$ tunnel. We say that a network is Pareto inefficient if a path p exists between any two users which is better than the current path $p' \neq p$ subject to some set of constraints C.

We use live traffic traces [115] to drive simulation on Internet-like topologies generated by BRITE [101]. The topology uses a Waxman connection strategy with uniformly distributed link bandwidths. We emphasize that our simulation is designed to demonstrate the potential for learning rather than exactly modeling the Internet. As earlier work shows, routing on alternate paths can yield performance gains [7]. Therefore we use the simulator to verify the ability to learn a policy which can achieve our goal. In the future, we plan to employ the simulator to implement more complex goals with additional constraints. The simulator also allows us to compare the performance and efficiency of policy learning against other schemes such as overlay routing.

In the simulation, one AS implements our learning algorithm and attempts to optimize its customer's traffic within the constraints of the topology. We deterministically assign each destination in the trace to a random AS in the topology. Similarly, we assign one-third

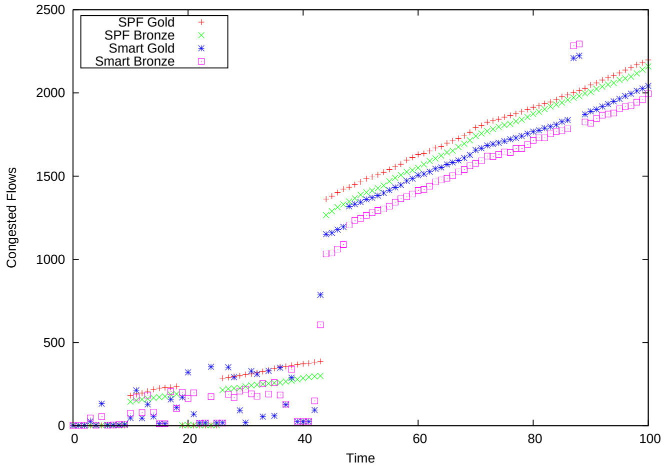


Figure 5-24: Simulation of live trace data on Internet-like topologies for shortest path and learning routing strategies.

of the sources as "gold" customers of the designated learning AS and the remaining as best effort traffic. We discretize the traffic trace into 0.1sec time slices.

Figure 5-24 shows our initial simulator results by comparing the number of congested flows where the designated AS uses shortest AS path routing versus learning routing.

After an initial period of exploration while the learner finds a policy, the learning AS consistently outperforms the static policy as anticipated. In addition, the overall number of congested flows is lower even when placing bronze flows on potentially poor paths.

Observation 10: Even using bronze flows for exploration, the number of congested bronze flows is lower in the intelligent architecture.

5.3.12 Open Questions

Our research suggests several potential benefits in including learning within the routing plane, however many open questions remain. For example, we assume that the behaviors of bronze customers will be sufficiently similar to those of gold customers in order for the bronze traffic to benefit decisions made on gold traffic. Additional measurement studies and empirical evaluate are necessary in order to validate this assumption.

In contrast to path probing techniques used in overlays, our system explores potential paths with real traffic; this is a distinct difference in approach. A useful experiment we plan to implement is to correlate various properties such as path latencies with the performance of a file transfer. The correlation between e.g. latency and path performance is very interesting in understanding how to operate with limited information. By making the choice of optimal path based upon collaborative learning, we believe that real traffic is the only way to obtain statistically significant data for formulating decisions.

Finally, we are very interested in the ability and effectiveness of clustering within the routing and learning domain. There is a widely held belief that there exists substantial temporal and spatial locality within network traffic. Whereas other domains exploit this locality, for instance caching on microprocessors, the networking world has dismissed caching. However, given the explosion of routing table sizes, increasing complexity and limited router memory, particularly fast memory on the forwarding path, caching and clustering may be very useful. Using our IP clustering algorithm, we believe that a router can optimize its available memory toward achieving high reward. Thus, regardless of the amount of installed memory, routers could use learning to determine how to cluster rewards. Memory then becomes just another parameter of the constrained optimization problem.

5.3.13 Relation to Prior Work

QoS remains an elusive target, without widespread deployment on the Internet after many years. Recent work examines the potential for interprovider QoS [48]. However, measuring and optimizing network performance will continue to require learning.

Routing is a well-explored research topic with many deterministic and randomized algorithms. Some of the first efforts to apply machine learning to routing are from Littman and Boyan's Q-routing algorithm [95] based on the Q-learning approach from reinforcement learning (RL).

Reinforcement learning is deceptively appealing: with only a reward function, an agent can learn to act optimally in any environment. However, RL's major drawback is the tension between exploration and exploitation. The algorithm must determine when to stop searching alternatives before using the knowledge it has learned thus far to its advantage. Random exploration can prove intractable given too large a state space without any prior domain-specific guidance. How long should the algorithm search alternatives before using the knowledge it has learned thus far to its advantage? Random exploration can prove intractable given too large a state space without any prior domain-specific guidance. Internet routing tables are very large; can a learning algorithm bootstrap using existing routing tables and optimize for some small set of routes?

Boyan's Q-routing scheme is a distributed distance vector routing protocol which uses packet delivery latency as the reward metric. While Q-routing is demonstrably superior on small, simulated networks, it suffers from several drawbacks. First, the per-packet feedback requires a large amount of communication overhead. Q-routing requires per-interface, per-destination state $O(n)$ with a large constant factor. A significant difficulty in the full distributed, autonomously acting agents in Q-routing is lack of convergence and stability. Without specific coordination, the network can exhibit oscillatory behavior, particularly in a dynamic network as witnessed in the early ARPAnet [87].

Further, networks are intrinsically non-stationary and Q-routing cannot find a better alternate path unless the current best path becomes congested. Can a learner do better by periodically exploring random alternatives if the environment is changing? Boltzman simulated annealing techniques can be used to settle values over time, but learning must be continual in a dynamic network. Choi and Yeung present predictive Q-routing to allow for such policy exploration [38]. By retaining information on prior best paths, the network can return to a previous best path after a congestion event passes while standard Q-routing would settle into a new local minimum.

In contrast to Q-routing which uses delay as the utility metric, we take an economic view of reward that more closely models the true business goals of networks. Monetary reward changes on time scales an order of magnitude larger than network congestion events, yielding stability to our routing approach. Rather than a pure exploratory approach, we propose a layer of indirection. By starting with a set of existing nexthops, we can obtain feedback on the particular performance of routing different prefixes on these nexthops. Using clustering algorithms, we can optimize for a small set of routes that can provide a majority of the realizable benefit.

The feedback based routing scheme of Zhu et al. [158] focuses on interdomain availability and first proposed separating structural and dynamic information. Feedback routing constructs dynamic information by observing round trip TCP times at access routers and performing explicit path probing. From the structural information, each access router builds two maximally disjoint paths to each network destination prefix. Based on current dynamic information, the access routers specify the best path. In contrast to our approach, feedback routing only attempts to improve availability without considering incentives. By giving the access routers the ability to choose a complete source-based path, feedback routing does not accommodate the tussle between users and providers.

While other proposals contain a large disconnect between the need for line rate (at the speed of the router's interfaces) and probabilistic routing, our scheme remains fast. Our algorithm runs over the routing information database (RIB) and the forwarding information database (FIB) remains unchanged until a better route is computed.

The recent 4D architecture [153] proposes a clean-slate approach to network control. 4D centrally discovers network devices and combines objectives into decisions. These decisions are pushed via a dissemination layer which implements the data forwarding layer on individual network routers. The high-level motivation and goals of 4D are similar to our own, however our approach advocates using learning to resolve the inherent decision complexity and conflicts to achieve objectives.

5.4 Conclusions

Some classes of problem are best solved by adding intelligence to the network core. The core of the network affords a broader and more complete view; intelligence can be placed on collections of users, traffic, etc. to better optimize global and local performance or maintain security. Core intelligence is contentious; this Chapter intends to be provocative by motivating when such designs are sound. Many of the security and optimization problems the Internet faces require this sort of collaboration and global scope, thus motivating our exploration into core intelligence.

As the Internet continues to grow in scale and scope, it penetrates more deeply and broadly, across users and societies. The problems the Internet faces are global problems and are therefore not well-solved by humans with only local perspective. Automated intelligence can be especially useful in situations where it is not possible to wait for human intervention, for example attacks, faults, worm propagation, etc.

In this Chapter, we examine distributed learning within the network. To ground our discussion in a practical and current threat, we apply learning to the IP source address validation problem. Current mitigation techniques are hampered by incentive issues; new attacks based on IP source forgery appear continually. Our work exploits learning to allow the eventual recipient to form a classification decision on incoming packets, thereby removing the incentive problem. Whereas existing techniques require global participation, we demonstrate that the network core has a sufficiently broad view to filter the majority of forged traffic with minimal collateral impact.

Section 5.3 takes routing as a final example of intelligence in the core. Because of the lack of separation between reachability and policy in Internet routing, operational networks are forced to drive behavior with low-level configuration. The resulting network is complex, fragile and prone to pathologies and non-obvious failures. Given that providers have a rich set of business goals, we propose an intelligent routing substrate that implements policy.

Our approach separates information collection from use. We show that end-nodes are well-suited to understanding path properties and providing information for the core of the network. The core of the network can then use this information to aggregate across many sources and destinations. Through simulation, we examine the role of learning in realizing such future routing plane architectures. While other routing protocols, techniques and architectures exist for accomplishing many of the same tasks, we posit that learning naturally accommodates an inherently complex and multivariate problem.

A routing infrastructure based on a learned policy naturally accommodates a language that providers really want to use. For instance, a router's policy might be as simple as minimizing cost while ensuring successful delivery. This very simple policy, corresponding well to the business objectives of many present day ISPs, is nearly impossible to implement with today's technology. Monetary reward changes on time scales an order of magnitude larger than network congestion events, yielding additional stability to our routing approach.

Learning provides one framework for separating desired goals and constraints from their implementation. Many of the root problems we outline in §5.3.3, but do not consider here, may also be addressed through learning. Given that providers have a rich set of business goals, an intelligent routing substrate that implements policy is appealing. For instance, new routing architectures may include explicit recognition of economics. By adopting learning as a component of the routing infrastructure, our hope is to provide a an architecture that optimizes an inherently multi-dimensional problem and mediates the potentially conflicting requirements of both providers and end users.

The major differences were that it was thrillingly more expensive, and involved a huge amount of sophisticated measuring and regulating equipment which was far better at knowing, moment by moment, what people wanted than mere people did.

- Douglas Adams

Chapter 6

Discussion and Future Work

The Internet has become a ubiquitous substrate for communication in all parts of society. However, many assumptions underlying the original network architecture are changing or have already changed. Amid problems of scale, trust, complexity and security, the modern Internet accommodates increasingly critical services. The network's ability to configure itself, recover from and defend against attacks, and provide new services all may require intelligence in the architecture. We argue that both end-nodes and core infrastructure must become more intelligent in an evolving Internet. In motivating alternate designs, we appeal to the end-to-end arguments to guide our decision of *where* in the network, if at all, intelligent functionality should be placed.

While learning is often viewed as an ad-hoc method of tackling problems that are poorly understood, we show that learning affords the system designer the ability to solve highly complex problems when the basic problem structure is well understood. As the Internet continues to evolve, we believe learning will take on larger and more important roles.

6.1 Key Contributions

The key contribution of this thesis is presenting learning as a fundamental component at different levels within network architecture. Not only is learning useful at the application level, it is a valuable tool in the system designer's toolbox. In demonstrating this utility, we take a bottom-up approach and examine the practical application of learning to several real-world Internet problems.

Chapter 3 shows how learning enables peer-to-peer nodes to locate and attach to peers likely to answer future queries, while Chapter 4 argues for increased intelligence in end-nodes. We develop an on-line, non-stationary IP clustering algorithm to accommodate natural structural and temporal Internet dynamics in §4.3. Using our clustering method, we endow agents with predictive capability.

Learning provides a principled method to capitalize on the wealth of available, but under-utilized information at different levels in the network. For example, in §4.4 we create a packet flow classification technique which detects traffic originating from remote, resource constrained hosts. This method provides the basis for “SpamFlow,” a novel spam detection tool. By using learning to exploit a fundamental weaknesses in sourcing spam, SpamFlow is both adaptable and not easily subvertible. Finally, in Chapter 5, we examine distributed learning within the core of the network. Not only do we illuminate many benefits to learning within the network, we take steps toward the practical application of such learning.

This thesis serves first to validate the potential for using learning methods to address several distinct problems on the Internet and second to illuminate design principles in building such intelligent systems in network architecture. Beyond addressing current stressors and enabling new functionality, our hope is that this work advances the role of learning within networks.

6.2 Lessons for Practitioners

While machine learning is a valuable tool for systems architects, it is most valuably applied when the problem is well-understood but complex. Our research therefore strives to give system designers guidelines as to when to apply learning. Several design principles emerge throughout the thesis which we highlight here:

- **Tractability:** finding means to limit and decompose network problems is crucial to the success of a statistical approach. If the problem space is not well-understood, or contains no structure, no amount of learning will be useful.

- **Exploit structure:** our research demonstrates that the Internet’s IP address space allocation gives structure upon which agents can learn. While we use this structure for tasks such as latency prediction, other networking problems are likely to benefit.

- **Kernel functions:** we frequently turn to kernel functions to transform the input spaces, e.g. IP addresses, into a feature space amenable to support vector learning. Kernel functions are well-suited to dealing with the non-linearity common in complex problems found on the Internet.

- **Tunability:** predictive mechanisms must be tunable for different applications and environments. This implies understanding parametric learning models. For instance, designers may balance speed versus accuracy or false positives versus false negatives.

- **Feature Selection:** feature selection can be used as a means of not only reducing the computational complexity of a problem, but also reducing the communication complexity in a distributed environment.

- **Probability:** machine learning affords agents a powerful metric in the form of probability. In the face of uncertainty, the expert system designer may wish to provide bounds or thresholds based on reasonable beliefs. Learning is thus useful in providing agents a most likely prediction with a degree of belief.

- **Dual-mode operation:** while under duress, networks may operate in an emergency mode where dropping legitimate traffic, users, etc. is acceptable. Human indirection and the ability to turn a defense mechanism on for survivability is important.

- **Exploit weaknesses:** the predominance of legitimate users can be leveraged. Learning may be used to harness the power of many to exploit attack weaknesses while preventing evasion.

- **Ensembles of weak agents:** in distributed environments, using ensembles of weak classifiers is a powerful construct. By combining and synthesizing multiple weak votes, agents can form more accurate validity assertions.

6.3 The Future of Learning and Networks

Our use of learning in applications and end-nodes is often a powerful construct simply because there is no easy way to re-architect the network. It is the network's basic properties, for instance incomplete information, that necessitate learning and allow learning agents an advantage. However, we must not be short-sighted and understand when learning is fundamental and when learning is simply a by-product of circumstance. For instance, a clean-slate network architecture could redefine the way addresses are delegated, removing the need for learning as a means to understand the underlying address structure. Our research suggests that future architectures should be much more well-instrumented, not only to facilitate better learning for the problems we consider, but also for new and unanticipated services. Instrumentation and measurement must be central considerations of any redesign.

While learning is often viewed as a contentious solution by network architects, on closer inspection it is apparent that services such as search engines are already using machine learning to provide an essential and extensively used element of the current Internet. While machine learning is used for Internet services and at the network application layer, a key question this thesis explores is the benefit of pushing increased intelligence down into the network architecture itself. How deep and how prevalent learning becomes remains to be seen.

As learning becomes a more widely accepted tool in the system designer's arsenal, there is potential for dueling algorithms. For instance, learning agents may be locked in a battle to obtain a relative advantage. In non-malicious instances, such learning is likely to converge to more efficient policies. However, to mitigate the learning power of attackers, effective defense will certainly require not only individual nodes learning, but also learning facilitated by the network among cooperative legitimate nodes. Concepts such as trusted beacon traffic among infrastructure nodes may facilitate truthful learning in the face of adversaries attempting to influence the learning system.

6.4 Open Questions and Future Research

Our research advocates learning as an attractive approach to address a variety of architectural strains on the Internet. Learning has significant potential in the realm of incentive compatibility. Many of the game theoretic protocols from Afergan [1] accommodate incentives, but require complete information. Learning thus provides important glue between theoretical algorithms and their practical application. We take this position a step further by postulating that the way to best accommodate incentives in future architectures is by adding intelligence to the network. We plan to explore future research that examines incorporating learning into the architecture to aid incentive mechanisms.

Concepts from Chapter 4 suggest more general ways to combat a variety of attacks. While we focus on the ability to recognize traffic originating from resource constrained portions of the network to single out botnets and thus identify potential spam, the same methods could likely be used to combat worms and other malicious attacks.

Chapter 5 advocates learning within the core of the network. We recognize that such distributed learning presents many practical difficulties, including communication cost and convergence amid network dynamics. However, we believe that the problems we consider are best solved via learning. Current computation difficulties therefore do not necessarily invalidate the approach. While we attempt to mitigate the practical challenges of core learning, we hope that our work serves to illuminate interesting issues within learning theory.

Bibliography

[1] Mike Afergan. Using repeated games to design incentive-based routing systems. In *Proceedings of IEEE INFOCOM*, April 2006.

[2] Mike Afergan and Robert Beverly. The state of the email address. *ACM SIGCOMM Computer Communications Review, Measuring the Internet's Vital Statistics*, 35(1):29–36, January 2005.

[3] Mike Afergan and John Wroclawski. On the Benefits and Feasibility of Incentive Based Routing Infrastructure. In *Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems*, pages 197–204. ACM Press, 2004.

[4] Tarem Ahmed, Boris Oreshkin, and Mark Coates. Machine learning approaches to network anomaly detection. In *Proceedings of the 2nd Tackling Computer Systems Problems with Machine Learning Techniques Workshop*, April 2007.

[5] Akamai. Akamai content distribution network, 2007. `http://www.akamai.com`.

[6] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. DomainKeys Identified Mail (DKIM) Signatures. RFC 4871 (Proposed Standard), May 2007.

[7] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient overlay networks. In *Symposium on Operating System Principles*, October 2001.

[8] Martin Arlitt, Balachander Krishnamurthy, and Jeffrey C. Mogul. Predicting short-transfer latency from TCP arcana: A trace-based validation. In *Proceedings of Internet Measurement Conference*, 2005.

[9] A. Asvanund, S. Bagla, M. Kapadia, R. Krishnan, M. D. Smith, and R. Telang. Intelligent club management in peer-to-peer networks. In *Proceedings of First Workshop on Economics of P2P*, 2003.

[10] F. Baker and P. Savola. Ingress Filtering for Multihomed Networks. RFC 3704 (Best Current Practice), March 2004.

[11] M. Basseville and I. Nikiforov. *Detection of abrupt changes: theory and application.* Prentice Hall, 1993.

[12] Steven Bauer, Peyman Faratin, and Robert Beverly. Assessing the assumptions underlying mechanism design for the internet. In *Economics of Networked Systems*, June 2006.

[13] S. Bellovin. The Security Flag in the IPv4 Header. RFC 3514 (Informational), April 2003.

[14] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communications Review*, 19:2:32–48, 1989.

[15] S. M. Bellovin. ICMP traceback messages. IETF Internet Draft, September 2000. `http://www.cs.columbia.edu/~smb/papers/draft-bellovin-itrace-00.txt`.

[16] Robert Beverly. MS-SQL slammer/sapphire traffic analysis, 2003. `http://ana.csail.mit.edu/slammer/`.

[17] Robert Beverly. A Robust Classifier for Passive TCP/IP Fingerprinting. In *Proceedings of the 5th Passive and Active Measurement (PAM) Workshop*, pages 158–167, April 2004.

[18] Robert Beverly. Reorganization in Network Regions for Optimality and Fairness. Master's thesis, MIT, August 2004.

[19] Robert Beverly and Mike Afergan. Machine learning for efficient neighbor selection in unstructured P2P networks. In *Proceedings of the 2nd Tackling Computer Systems Problems with Machine Learning Techniques Workshop*, April 2007.

[20] Robert Beverly and Steven Bauer. The Spoofer Project: Inferring the extent of source address filtering on the Internet. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI) Workshop*, pages 53–59, July 2005.

[21] Robert Beverly and Steven Bauer. Can you spoof IP addresses? *Slashdot*, May 2006. `http://it.slashdot.org/article.pl?sid=06/05/02/1729257`.

[22] Robert Beverly and Steven Bauer. The ANA Spoofer Project, 2006. `http://spoofer.csail.mit.edu/`.

[23] Robert Beverly and Steven Bauer. Tracefilter: A tool for locating network source address validation filters. In *USENIX Security (Poster)*, August 2007.

[24] Robert Beverly and Karen Sollins. Exploiting transport-level characteristics of spam. Technical Report MIT-CSAIL-TR-2008-008, MIT, 2008.

[25] Robert Beverly, Karen Sollins, and Arthur Berger. SVM learning of IP address structure for latency prediction. In *SIGCOMM Workshop on Mining Network Data*, pages 299 – 304, September 2006.

[26] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475 (Informational), December 1998. Updated by RFC 3260.

[27] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the Internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1), August 2001.

[28] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994.

[29] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[30] T. Bu, L. Gao, and D. Towsley. On characterizing BGP routing table growth. *Computer Networks*, 45(1):45–54, 2004.

[31] Hal Burch and Bill Cheswick. Tracing anonymous packets to their approximate source. In *Proceedings of the 14th Systems Administration Conference (LISA)*, pages 319–327, 2000.

[32] B. Carpenter. Architectural Principles of the Internet. RFC 1958 (Informational), June 1996. Updated by RFC 3439.

[33] Martin Casado, Tal Garfinkel, Weidong Cui, Vern Paxson, and Stefan Savage. Opportunistic measurement: Extracting insight from spurious traffic. In *Proceedings of the ACM HotNets Workshop*, November 2005.

[34] CERT. TCP SYN Flooding and IP Spoofing Attacks CA-1996-21, 1996. `http://www.cert.org/advisories/CA-1996-21.html`.

[35] CERT. MS-SQL Server Worm Advisory CA-2003-04, 2003. `http://www.cert.org/advisories/CA-2003-04.html`.

[36] CERT. W32/Blaster Worm Advisory CA-2003-20, 2003. `http://www.cert.org/advisories/CA-2003-20.html`.

[37] Y. Chawathe, N. Lanham, S. Ratnasamy, S. Shenker, and L. Breslau. Making gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM*, 2003.

[38] Samuel P. M. Choi and Dit-Yan Yeung. Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *Advances in Neural Information Processing Systems*, volume 8, pages 945–951. The MIT Press, 1996.

[39] Cisco. Optimized edge routing, 2007. `http://www.cisco.com/go/oer`.

[40] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004.

[41] David D. Clark. The design philosophy of the DARPA internet protocols. In *Proceedings of ACM SIGCOMM*, pages 102–111, 1988.

[42] David D. Clark, Craig Partridge, Robert T. Braden, Bruce Davie, Sally Floyd, Van Jacobson, Dina Katabi, Greg Minshall, K. K. Ramakrishnan, Timothy Roscoe, Ion Stoica, John Wroclawski, and Lixia Zhang. Making the world (of communications) a different place. *SIGCOMM Comput. Commun. Rev.*, 35(3):91–96, 2005.

[43] David D. Clark, Craig Partridge, Chris Ramming, and John Wroclawski. A knowledge plane for the internet. In *Proceedings of ACM SIGCOMM*, August 2003.

[44] David D. Clark, Karen R. Sollins, John Wroclawski, and Theodore Faber. Addressing reality: an architectural response to real-world demands on the evolving internet. *Computer Communications Review*, 33(4):247–257, 2003.

[45] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, 2005.

[46] Richard Clayton. Using early results from the spamHINTS project to estimate an ISP abuse team's task. In *Third Conference on Email and Anti-Spam*, July 2006.

[47] M. Collins and M. Reiter. An empirical analysis of target-resident DoS filters. In *IEEE Symposium on Security and Privacy*, pages 103–114, May 2004.

[48] Communications Futures Program. QoS for inter-provider VPNs, September 2006. `http://cfp.mit.edu/groups/internet/qos.html`.

[49] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI) Workshop*, July 2005.

[50] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[51] Koby Crammer and Yoram Singer. Pranking with ranking. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2001.

[52] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings of SIGCOMM*, August 2004.

[53] Tom Dietterich and Pat Langley. *Machine learning for cognitive networks: Technology assessment and research challenges.* John Wiley, 2007.

[54] D. Dittrich. The stacheldraht distributed denial of service attack tool, 2000. `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt`.

[55] Zhenhai Duan, Xin Yuan, and Jaideep Chandrashekar. Constructing inter-domain packet filters to control IP spoofing based on BGP updates. In *Proceedings of IEEE INFOCOM*, 2006.

[56] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286, New York, NY, USA, 2006. ACM Press.

[57] Nick Feamster. Practical verification techniques for wide-area routing. In *ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[58] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[59] Michael Freedman, Mythili Vutukuru, Nick Feamster, and Hari Balakrishnan. Geographic locality of IP prefixes. In *Proc. ACM Internet Measurement Conference*, October 2005.

[60] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632 (Best Current Practice), August 2006.

[61] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.

[62] Gnutella. Gnutella: Distributed information sharing, 2000. http://wiki.limewire.org/index.php?title=GDF.

[63] Shen Tat Goh, Panos Kalnis, Spiridon Bakiras, and Kian-Lee Tan. Real datasets for file-sharing peer-to-peer systems. In *Proceedings of 10th International Conference of Database Systems for Advanced Applications*, 2005.

[64] William Sealy Gosset. The probable error of a mean. *Biometrika*, 6(1), 1908.

[65] B. R. Greene, C. Morrow, and B. W. Gemberling. ISP security: Real world techniques. NANOG 23, October 2001. http://www.nanog.org/mtg-0110/greene.html.

[66] Timothy G. Griffin and Joao L. Sobrinho. Metarouting. In *Proceedings of ACM SIGCOMM*, 2005.

[67] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Internet Measurement Workshop*, November 2002.

[68] Yihua He, Georgos Siganos, Michalis Faloutsos, and Srikanth Krishnamurthy. A systematic framework for unearthing the missing links: Measurements and impact. In *Proceedings of USENIX NSDI*, April 2007.

[69] A. Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385 (Proposed Standard), August 1998.

[70] Ningning Hu, Li (Erran) Li, Zhuoqing Morley Mao, Peter Steenkiste, and Jia Wang. Locating internet bottlenecks: algorithms, measurements, and implications. In *Proceedings of ACM SIGCOMM*, pages 41–54, New York, NY, USA, 2004. ACM Press.

[71] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel. Internet Registry IP Allocation Guidelines. RFC 2050 (Best Current Practice), November 1996.

[72] Geoff Huston. BGP reports. http://bgp.potaroo.net.

[73] IANA. Special-Use IPv4 Addresses. RFC 3330 (Informational), September 2002.

[74] IANA. Well-known port numbers, 2006. http://www.iana.org/assignments/port-numbers.

[75] Carla Inclan and George C. Tiao. Use of cumulative sums of squares for retrospective detection of changes of variance. *Journal of the American Statistical Association*, 89(427):913–923, September 1994.

[76] Internap, 2007. http://www.internap.com.

[77] IronPort. Spammers continue innovation: Ironport study shows image-based spam, hit & run, and increased volumes latest threat to your inbox, June 2006. http://www.ironport.com/company/ironport_pr_2006-06-28.html.

[78] C. Jin, H. Wang, and K. Shin. Hop-count filtering: An effective defense against spoofed DoS traffic. In *Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS)*, pages 30–41, October 2003.

[79] Yu Jin, Gyrgy Simon, Kuai Xu, Zhi-Li Zhang, and Vipin Kumar. Gray's anatomy: Dissecting scanning activities using IP gray space analysis. In *Proceedings of the 2nd Tackling Computer Systems Problems with Machine Learning Techniques Workshop*, April 2007.

[80] Thorsten Joachims. Making large-scale SVM learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[81] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *11th International WWW Conference*, May 2002.

[82] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[83] B. Kahin. Commercialization of the Internet summary report. RFC 1192 (Informational), November 1990.

[84] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: responsive yet stable traffic engineering. In *ACM SIGCOMM*, pages 253–264, New York, NY, USA, 2005. ACM Press.

[85] Thomas Karagiannis, Andre Broido, Nevil Brownlee, K claffy, and Michalis Faloutsos. Is P2P dying or just hiding? In *Globecom*, November 2004.

[86] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and K claffy. Transport layer identification of P2P traffic. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, October 2004.

[87] A. Khanna and J. Zinky. The revised ARPANET routing metric. *SIGCOMM Comput. Commun. Rev.*, 19(4):45–56, 1989.

[88] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001.

[89] Balachandar Krishnamurthy and Jia Wang. On network-aware clustering of web clients. In *ACM SIGCOMM*, pages 97–110, 2000.

[90] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *Proceedings of ACM SIGCOMM*, 2005.

[91] Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In *National Conference on Artificial Intelligence*, pages 223–228, 1992.

[92] George Lee. CAPRI: A common architecture for autonomous, distributed diagnosis of internet faults using probabilistic relational models. In *Proceedings of the First Workshop on Hot Topics in Autonomic Computing*, 2006.

[93] George Lee, Peyman Faratin, Steven Bauer, and John Wroclawski. A user-guided cognitive agent for network service selection in pervasive computing environments. In *IEEE International Conference on Pervasive Computing and Communications*, 2004.

[94] George Lee and Lindsey Poole. Diagnosis of TCP overlay connection failures using bayesian networks. In *Proceedings of the SIGCOMM Workshop on Mining Network Data*, 2006.

[95] Michael Littman and Justin Boyan. A distributed reinforcement learning scheme for network routing. Technical Report CS-93-165, Carnegie Mellon University, 1993.

[96] Michael Littman, Nishkam Ravi, Eitan Fenson, and Rich Howard. Reinforcement learning for autonomic network repair. In *International Conference on Autonomic Computing*, pages 284–285, 2004.

[97] Jun Liu, Ibrahim Matta, and Mark Crovella. End-to-end inference of loss nature in a hybrid wired/wireless environment. In *Proceedings of WiOpt: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.

[98] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An information plane for distributed services. In *Proceedings of USENIX OSDI*, November 2006.

[99] Muhammad N. Marsono, M. Watheq El-Kharashi, Fayez Gebali, and Sudhakar Ganti. Distributed layer-3 e-mail classification for spam control. In *Canadian Conference on Electrical and Computer Engineering*, May 2006.

[100] Justin Mason. Filtering spam with spamassassin. In *Proceedings of SAGE-IE*, October 2002.

[101] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An approach to universal topology generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, August 2001.

[102] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang. IPv4 address allocation and the BGP routing table evolution. *ACM SIGCOMM CCR*, 35(1):71–80, 2005.

[103] Messaging Anti-Abuse Working Group. Email metrics report, 2007. `http://www.maawg.org/about/EMR`.

[104] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes - which naive bayes? In *Third Conference on Email and Anti-Spam (CEAS)*, pages 27–28, July 2006.

[105] David Meyer. University of Oregon RouteViews, 2007. `http://www.routeviews.org`.

[106] Mariyam Mirza, Joel Sommers, Paul Barford, and Xiaojin Zhu. A machine learning approach to tcp throughput prediction. In *International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS)*, 2007.

[107] MITRE. MS-SQL Ping DoS Storm CVE-2002-0650, 2002. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0650`.

[108] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, 2006.

[109] Robert Morris. A Weakness in the 4.2BSD Unix TCP/IP Software. Technical Report 117, AT&T Bell Laboratories, 1985.

[110] Donald R. Morrison. PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric. *J. ACM*, 15(4):514–534, 1968.

[111] Christopher Morrow. BLS FastAccess internal tech needed, 2006. `http://www.merit.edu/mail.archives/nanog/2006-01/msg00220.html`.

[112] NANOG. DoS attack against DNS?, 2006. `http://www.merit.edu/mail.archives/nanog/2006-01/msg00279.html`.

[113] NANOG. BCP38 business case document, 2007. `http://www.merit.edu/mail.archives/nanog/2007-04/msg00692.html`.

[114] Arbor Networks. Worldwide infrastructure security report, 2008. `http://www.arbornetworks.com/report`.

[115] NLANR. Passive measurement and analysis traces, 2006. `http://pma.nlanr.net/`.

[116] R. Pang, V. Yegneswaran, P. Barford, and V. Paxson. Characteristics of Internet Background Radiation. In *Proceedings of ACM Internet Measurement Conference*, October 2004.

[117] Craig Partridge, Philip P. Carvey, Ed Burgess, Isidro Castineyra, Tom Clarke, Lise Graham, Michael Hathaway, Phil Herman, Allen King, Steve Kohalmi, Tracy Ma, John Mcallen, Trevor Mendez, Walter C. Milliken, Ronald Pettyjohn, John Rokosz, Joshua Seeger, Michael Sollins, Steve Storch, Benjamin Tober, and Gregory D. Troxel. A 50-gb/s ip router. *IEEE/ACM Trans. Netw.*, 6(3):237–248, 1998.

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

[119] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.

[120] J. Postel. Domain Name System Structure and Delegation. RFC 1591 (Informational), March 1994.

[121] Vipul Ved Prakash. Vipul's razor, August 2007. `http://razor.sourceforge.net/`.

[122] Y. Qiao, J. Skicewicz, and P. Dinda. An empirical study of the multiscale predictability of network traffic. In *Proceedings of IEEE HPDC*, 2003.

[123] Anirudh Ramachandran and Nick Feamster. Understanding the network-level behavior of spammers. In *Proceedings of ACM SIGCOMM*, September 2006.

[124] Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of ACM Conference on Computer and Communications Security*, October 2007.

[125] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.

[126] Amir H. Rasti, Daniel Stutzbach, and Reza Rejaie. On the long-term evolution of the two-tier gnutella overlay. In *IEEE Global Internet*, 2006.

[127] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM*, June 2002.

[128] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.

[129] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996.

[130] M. Sahami, S. Dumais, D. Heckerman, and E Horvitz. A bayesian approach to filtering junk e-mail. In *AAAI Workshop on Learning for Text Categorization*, July 1998.

[131] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[132] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of internet path selection. In *Proceedings of ACM SIGCOMM*, pages 289–299, 1999.

[133] Tom Scholl, Aman Shaikh, Nathan Patrick, and Richard Steenbergen. Peering dragnet: Examining BGP routes received from peers. NANOG 38, 2006. `http://www.nanog.org/mtg-0610/scholl-shaikh.html`.

[134] Secure Computing. Ironmail, 2007. `http://www.securecomputing.com`.

[135] Richard Segal. Combining global and personal anti-spam filtering. In *Fourth Conference on Email and Anti-Spam (CEAS)*, 2007.

[136] Keith Sklower. A tree-based routing table for berkeley UNIX. In *Proceedings of the USENIX Technical Conference*, pages 93–104, 1991.

[137] Alex C. Snoeren, Craig Partridge, Luis A. Sancheq, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer. Hash-based IP traceback. In *Proceedings of ACM SIGCOMM*, 2001.

[138] SORBS. Spam and open-relay blocking system (SORBS), 2007. `http://www.sorbs.net`.

[139] SpamCop. Spamcop, 2007. `http://www.spamcop.net`.

[140] Spamhaus, 2007. `http://www.spamhaus.org/sbl/`.

[141] N. Spring, R Mahajan, and T Anderson. Quantifying the causes of path inflation. In *Proceedings of ACM SIGCOMM*, 2003.

[142] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of INFOCOM*, 2003.

[143] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, October 2005.

[144] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM*, 2003.

[145] J. Touch. Defending TCP Against Spoofing Attacks. RFC 4953 (Informational), July 2007.

[146] V. N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, 1995.

[147] Jessica Vascellaro. Empty spam feasts on in-boxes, August 2006. `http://online.wsj.com/article_email/SB115448102123224125-lMyQjAxMDE2NTA0MjQwODIxWj.html`.

[148] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet path performance. In *Proceedings of ACM SIGCOMM*, pages 375–386, 2006.

[149] Bernard Wong, Aleksandrs Slivkins, and Emin Gn Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proceedings of SIGCOMM*, August 2005.

[150] M. Wong and W. Schlitt. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408 (Experimental), April 2006.

[151] Avi Yaar, Adrian Perrig, and Dawn Song. Pi: A path identification mechanism to defend against DDoS attacks. In *IEEE Symposium on Security and Privacy*, May 2003.

[152] Avi Yaar, Adrian Perrig, and Dawn Song. FIT: Fast Internet traceback. In *Proceedings of IEEE Infocom*, March 2005.

[153] Hong Yan, David A. Maltz, T. S. Eugene Ng, Hemant Gogineni, Hui Zhang, and Zheng Cai. Tesseract: A 4D network control plane. In *Proceedings of USENIX NSDI*, April 2007.

[154] Xiaowei Yang. NIRA: A new internet routing architecture. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, 2003.

[155] Xiaowei Yang and David Wetherall. Source selectable path diversity via routing deflections. In *Proceedings of ACM SIGCOMM*, September 2006.

[156] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

[157] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, 1997.

[158] Dapeng Zhu, Mark Gritter, and David R. Cheriton. Feedback based routing. In *ACM HotNets*, October 2002.