TOPOLOGY AWARE ESTIMATION METHODS FOR INTERNET TRAFFIC CHARACTERISTICS

by

James A. Gast

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

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2003

To Anne, who gave up everything for me three times.

ACKNOWLEDGMENTS

First and foremost, this thesis would not have been possible without the patience and clear-headed thinking of Paul Barford and the healthy skepticism of Larry Landweber. They listened patiently when I questioned data that disagreed with my pre-conceptions and guided me to all the right papers and textbooks at exactly the right moments.

As with any modern program, my thesis work stands on the shoulders of countless people who wrote tools, languages, and packages that were indispensable. To name them all here would be impossible, but I want single out Dave Plonka for his dedication to tools that made it easy for me to collect and analyze traffic from Internet 2.

Over 3 decades, I have had the joy and honor of brainstorming with some of the best programmers and designers of open computer networking and none are better than the team at the Wisconsin Advanced Internet Lab. I had many important and valuable conversations with De Byrd, Joel Sommers, and Vinod Yegneswaran. I have immense gratitude to John Morgridge and the other WAIL donors for their very generous donation of equipment to WAIL and the Badger Internet Group.

The insight and all of the mathematics for the dynamic programming algorithm in the clustering part of the thesis were the work of Dr. Jin-Yi Cai. He wrote that treatment in a single amazing wonder-weekend and it did not have a single flaw.

Thomas Hangelbroek did the initial programming to determine the centroid of the global Internet and showed me matlab tricks I had never imagined.

Important and very helpful comments came from Dr. Robin Kravets. Her insights into Internet topology studies were both inspired and inspiring.

Finally, I especially want to thank Drs. David DeWitt and Jeff Naughton for their faith in me. And I want to thank the CS faculty for granting me the Anthony C. Klug fellowship in Computer Science.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Page

DISCARD THIS PAGE

LIST OF TABLES

DISCARD THIS PAGE

LIST OF FIGURES

Figure Page

Appendix
Figure | Page

TOPOLOGY AWARE ESTIMATION METHODS FOR INTERNET TRAFFIC CHARACTERISTICS

James A. Gast

Under the supervision of Assistant Professor Paul Barford

At the University of Wisconsin-Madison

Attempts to represent the global Internet in simulations and emulations have been difficult even at the most basic levels. The focus of our work is Internet topology and traffic matrix estimation to predict accurately Internet capacity, utilization, and congestion. We describe a *forest* representation of the topology of the Internet which improves on prior topologies by being more complete and accurate. We present a novel, scalable simulation environment that models the interactions of collections of flows across multi-hop networks and can accurately predict the way highly multiplexed traffic will react to congestion. We show that round trip time and *ceiling* not caused by congestion have a strong influence on the way traffic reacts to congestion. We show mechanisms that group large numbers of connections into units we call *flocks* and demonstrate that flock behavior can be seen in actual one-way delay data. Our model does not require packet-level information, but can quickly map queue depths and predict multi-hop queuing delays. Using this model, we were able to expose new phenomena that would not be apparent at lower levels of multiplexing.

The final component of this work is a traffic matrix estimation methodology that incorporates those new parameters along with the volume of traffic for each full path through the network. Ceiling and round trip time parameters were not used in earlier traffic matrix estimations because it is difficult for an Internet Service Provider to collect that data. We present a novel technique for inferring round trip times from easily gathered flow data at ISP edge nodes based on *ACK ratio*.

Paul Barford

ABSTRACT

Attempts to represent the global Internet in simulations and emulations have been difficult even at the most basic levels. The focus of our work is Internet topology and traffic matrix estimation to predict accurately Internet capacity, utilization, and congestion. We describe a *forest* representation of the topology of the Internet which improves on prior topologies by being more complete and accurate. We present a novel, scalable simulation environment that models the interactions of collections of flows across multi-hop networks and can accurately predict the way highly multiplexed traffic will react to congestion. We show that round trip time and *ceiling* not caused by congestion have a strong influence on the way traffic reacts to congestion. We show mechanisms that group large numbers of connections into units we call *flocks* and demonstrate that flock behavior can be seen in actual one-way delay data. Our model does not require packet-level information, but can quickly map queue depths and predict multi-hop queuing delays. Using this model, we were able to expose new phenomena that would not be apparent at lower levels of multiplexing.

The final component of this work is a traffic matrix estimation methodology that incorporates those new parameters along with the volume of traffic for each full path through the network. Ceiling and round trip time parameters were not used in earlier traffic matrix estimations because it is difficult for an Internet Service Provider to collect that data. We present a novel technique for inferring round trip times from easily gathered flow data at ISP edge nodes based on *ACK ratio*.

Chapter 1

Introduction

1.1 Motivation and Approach

The research community would like to answer questions that are relevant and important to the current Internet, but the task often proves difficult. The Internet is not owned, managed or maintained by any single entity, so there is no single authority that can enforce policies or provide data. How would the Internet react to catastrophes like natural disasters or intentional flooding? Will the Internet be able to continue to grow gracefully as global demand grows? Is the Internet appropriate technology for Video-On-Demand and other high-stress applications? The popularity of Peer-to-Peer protocols like Napster caused a significant shift in demand. What would happen if another new trend hit the Internet?

To address questions about the current state of the Internet, many researchers [20, 31] have called for studies of "a day in the life" of the Internet. They propose collecting information about the topology of the Internet and the traffic matrix showing which source nodes send how much data to which destination nodes. Exploring a day in the life of the Internet enables us to consider the scalability issues at a realistic level and helps us identify invariant properties that will give rise to better models, metrics, and, ultimately, global Internet service that is dependable and efficient.

Many of the simplest questions are hard to answer. Consider a link between two nodes in a heavily-interconnected network. What would happen to traffic flow if that link were broken? This simple question will help us expose some of the invariants of the global Internet and will focus our attention on two parameters often neglected in the parameter space because they are not easily discovered from current protocols and equipment. Nonetheless, Chapter 3 shows that *reaction time* and *connection bandwidth ceiling* are crucial to understanding congestion and, therefore capacity.

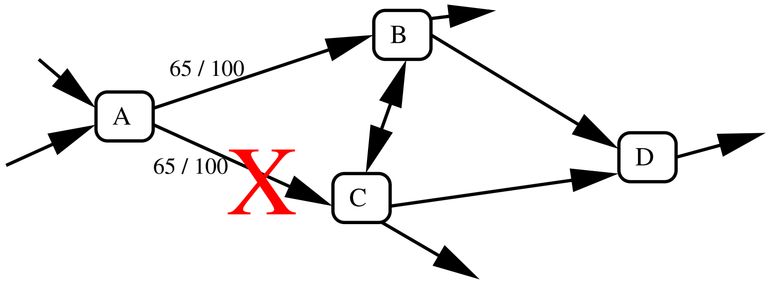

Figure 1.1 It is surprisingly difficult to predict the changes that result from a simple change in the network.

In Figure 1.1 link $A \rightarrow C$ carries 65 units of traffic out of a capacity of 100 and link $A \rightarrow B$ is similar. If link $A \rightarrow C$ were broken, where would the traffic go? Network routing would quickly discover new routes but $A \rightarrow B$ would be asked to carry 30 units of traffic more than its capacity. The result is congestion at link $A \rightarrow B$. There are many proposals for how A should react to the congestion, but the intent of all of those proposals is to ask the suppliers of data slow down. Typically, A will drop some packets. Soon after that, end-to-end congestion avoidance will reduce the future traffic. What would the resulting traffic pattern look like? If $A \rightarrow B$ becomes heavily over-subscribed, will the congestion become unacceptable? Will links like $C \rightarrow D$ actually become less congested as a result of the death of $A \rightarrow C$?

There are several unanswered research questions we will explore here.

- *What is a congestion event? Do we measure congestion in minutes or in milliseconds? How does a burst of losses relate to increases in queuing delay?*

- In a highly multiplexed world, the distressed node, A, can choose to ask only a few suppliers to slow down, or many. *How many senders will slow down? Are we discouraging too many or too few?*

- *What are the characteristics of suppliers that are important to the way they react to congestion?*

Once we know what a congestion event is,

- *Where is the congestion in the Internet?*
- *Where do we expect congestion in the future?*

1.1.1 Successful Congestion Abatement

We start by clarifying the timescales over which congestion can be studied. Zhang *et al.* [94] introduce the notion of operational stability. They consider a parameter operationally stable if it remains within bounds considered operationally equivalent. Consider a time scale of an hour. To report that an hour is mathematically steady, it would have to be described with a single time-invariant mathematical model. This is often too severe a test for operational purposes, because many mathematical non-constancies are in reality irrelevant to a particular study. They further reported that loss rate remains *operationally* stable on the time scale of an hour. We define a congestion event on the much smaller timescale of a few times the connection reaction time. That reaction time is approximately one round trip time to allow for the "please slow down" message to reach the supplier and for the packets already in transit to pass through. Thus, we visualize congestion events as discrete events with a clear start time (start of dropping or marking packets) and a clear end time (empty queue). Because wide area round trip times are typically on the order of a few milliseconds to a few hundred milliseconds, we expect reaction times to be in that range.

Each congestion event has a duration and a local intensity of packet loss. After each of those events, the suppliers have, presumably, slowed down via multiplicative decrease [40]. Assume this is enough to abate the congestion and let the loss rate drop to zero. For this discussion, assume that the suppliers are TCP (or TCP-friendly) sources. The TCP sources will then accelerate by re-growing their congestion windows. This is the additive increase mechanism TCP uses to probe for better bandwidth. The time frame to grow back to a level that causes congestion depends on the original size of the congestion window, but is typically many round trip times. During the re-growth, there may be several seconds in which aggregate offered load is less than the link capacity and no losses occur. Eventually, enough growth by enough connections will cause another congestion event and the cycle starts again. Thus, when looking at packet loss rates, we may see a sequence of congestion events. Each congestion event will be a brief burst of packet losses whose duration is driven by the predominant reaction time followed by a relatively long, lossless period. Over the course of an hour, a link may see many congestion events.

The idealization of a congestion event led us to introduce the notion of a *successful* congestion event. We define a successful congestion event as one which abates enough traffic to reduce the aggregate demand on the congested link to a level less than the capacity of that link. From the viewpoint of the queue of traffic leaving the link, this means the result of a successful congestion event is that it evokes responses from a sufficient set of suppliers to abate traffic long enough for the queue to drain. In contrast, an *unsuccessful* response to congestion would occur if the link were unable to signal enough traffic to slow down. Chronic congestion is not covered in this thesis.

1.1.2 Where Congestion Occurs

A recent study of lossy links by Padmanabhan [66] tried to discover the most likely places for losses. Not surprisingly, the links most closely watched were the links that cost money. In the commercial Internet, small Internet Service Providers (ISPs) buy service from bigger ones in an informal tiered hierarchy. Tier 1 can be thought of as a backbone. In the parlance of Border Gateway Protocol (BGP), an Internet Service Provider is analogous to an Autonomous System (AS) and is often used as the presumed border from one economic entity to another. AS's often have to pay other AS's for connection to the backbone based on total traffic and a Service Level Agreement (SLA). Padmanabhan states that:

> ...In 45% of cases, the identified lossy link crosses inter-AS boundaries and has a high latency.

He went on to conclude that only 20% of losses come from links that are neither long nor inter-AS. This gives us confidence that an Internet graph with one node per AS will still retain the important edges.

1.1.3 Gap Between Congestion Events

If, as we propose, congestion abatement happens on the time scale of round trip times (RTT), studying RTT is important. And if long-term average loss rates depend, ultimately, on the rate of re-introduction of congestion, studying window growth must also be important. Our hypothesis is that loss rates look stable on the time frame of an hour because congestion events are spread throughout the hour. The gap between those congestion events represents the amount of time TCP (or TCP-friendly) connections take to regain sufficient congestion window sizes to cause congestion. Our evidence of losses was that congestion events were much farther apart than simple window growth would predict. That led us to investigate causes for connections that do not grow beyond a bandwidth ceiling.

1.1.4 New Models with the New Parameters

Once we had identified that RTT and ceiling were crucial to understanding link capacity and fullness, we incorporated them into models that can be used to explain and explore congestion phenomenon. We hypothesized that an Autonomous System could do better traffic management and traffic engineering if it could measure these crucial parameters and use them in such models. These realizations forced us to return to the study of the graph of the Internet to look at connectivity in light of AS boundaries and round trip times.

1.1.5 Topology

Because of the massive scale of the Internet, a useful Internet traffic model should have a concise representation of the topology of the Internet. The list of nodes and links must be accurate enough to let the research community test theories and identify weaknesses, but simple enough to be tractable. Which aspects of Internet Topology are vital to understanding the functioning of the Internet and which aspects are irrelevant? Does the composition of the traffic matter? Would a model based solely on traffic quantity be fundamentally flawed?

One of our objectives was to discover the topology of the global Internet and then construct a traffic matrix that we could apply to it on a collection of backbone routers in the Wisconsin Advanced Internet Lab [49]. The experiments in Chapter 2 are designed to discover relevant aspects of the interconnections between Autonomous Systems in the Internet. The task is surprisingly difficult, since there is no single authority that knows all of the interconnections [31]. Moreover, the business relationships between Internet Service Providers are confidential.

Publicly available information about the topology of the Internet is incomplete. It is based on inter-domain routing and focuses on reachability rather than trying to enumerate all possible links. Worse yet, some of the links that are present in the public tables are unidirectional. A small number of tier-1 long-haul providers sell service to many small or local tier-n Internet Service Providers. Cost considerations often prevent small domains from providing transit to anyone outside of their autonomous system. Those small autonomous systems are logically on the periphery of the Internet. Links to them are, in that sense, unidirectional from the lower-numbered tier to the final tier. In general, autonomous systems do not provide transit from one of their providers to another of their providers.

Rather than think of the Internet as a single, large, complex graph we used a clustering method to separate out the centroid (the trans-continental and trans-oceanic backbone) component from the myriad trees of national, educational, regional, research, and local components. Then, we used an iterative method to discover a likely spanning tree for each of the latter components of the Internet. Combining the centroid with those trees makes a “forest” representation of the Internet that is very concise.

The spanning tree was a convenient form for simple algorithms. It was easy to run analyses on trees and keep the computational cost practical. Unfortunately, even the best spanning trees we could invent were hopelessly inaccurate when tested against traceroutes run through the real Internet. One of the primary reasons for this is that Internet Service Providers have multiple ways to send packets to the rest of the Internet. Some links can only be used by specific IP address pairs (e.g. in research or educational networks) and some links only carry traffic from appropriately secure IP addresses. Moreover, a wide variety of unpublished, peer-to-peer, and backup links exist (and get used) but would be very hard to discover.

Chapter 2 describes our way of testing an Internet graph by sending traceroute requests to traceroute servers scattered throughout the Internet. A machine learning algorithm allowed us to add alternate parents to nodes in the forest until a desired level of accuracy was reached. The augmented forest is accurate enough for our lab-based emulation. However, the augmentations make the tree portions of the forest no longer acyclic. This led us to question if the trade-off of extra accuracy was worth the extra cost of running less-efficient algorithms in large-scale analyses?

We developed a novel dynamic programming solution that works very quickly to come up with a provably optimal solution in the strict forest case. We then apply it to a cache placement problem, test it for speed, enlarge the algorithm to cover the extra links (needed for reasonable fidelity) and retest. Results shown in Chapter 2 show that the enlargements in the algorithm doesn't substantially change the complexity of that typical analysis.

1.1.6 Scalable Simulations

Chapter 3 explores ways to scale up simulations to levels that would be unrealistic using packet-by-packet simulation tools such as ns2 [89]. Internet2's United States backbone is Abilene. The next-generation portion has 11 nodes and 15 links, most of which run at 10.2 gigabits per second. Each link has the capacity to carry tens of thousands of simultaneous connections.

Conventional wisdom expected that the statistics of multiplexing should have made the variations in volume less pronounced as the number of independent connections, n, increases. Variance should be proportional to the square root of n. If this is true, fast links with $n > 10,000$ should have a high mean and a variance that is operationally inconsequential. Countering that is the argument that those TCP connections each react using a deterministic control system. If TCP connections resonate with each other there may be waves of congestion. Studies of various kinds of resonance collectively refer to such phenomenon as *global synchronization* [29]. We demonstrate that window synchronization, one of the forms of global synchronization, defies the independence assumption and show how connections can resonate with other connections whose RTT is similar.

Should we worry that global synchronization will cause catastrophic Internet collapse and gridlock? We found that the effect is neither severe nor persistent enough to cause such oscillations in the foreseeable future, but the ripples caused by window synchronization are valuable indicators of bottlenecks and remote congestion.

We show window synchronization in long lived flows in a traditional, small simulation environment. But that doesn't necessarily mean that this phenomenon is still significant at high levels of multiplexing. By harvesting Surveyor [45] data we found evidence that one-way delay probes see full queues far more often than queuing theory would have predicted.

That led us to the develop a scalable model that accurately predicts the queue depths over time along multi-hop paths in an environment much more complex than could be handled by a packet-by-packet simulation. The output of the model was especially sensitive to two parameters that control the way connections react to congestion: RTT and a ceiling which, at the time, we thought was a bottleneck elsewhere in that connection's sojourn. That model takes a topology description and a traffic matrix and computes the duration, intensity, and quantity of congestion events on each link. The parameter space of the model is intentionally limited to those parameters we felt were most relevant to groups of long-term TCP and TCP-friendly connections over long distances. Such special-purpose models [30] can often bring clarity and insight to particular phenomena without inappropriate complexity.

1.1.7 A Matrix of Traffic Demands

Finally, Chapter 4 uses IP flow measurements from the Abilene network along with measurements of the artifacts of congestion to construct a traffic matrix. Finding the volume of data passing from one source to one destination was easy using flow data gathered as though we were doing accounting. But discovering the RTT and the ceiling for each flow proved more elusive.

We devised a technique for inferring RTT and ceiling from the ratio of data packets to ACK packets. Connections with a high Bandwidth Delay Product (BDP) tend to use delayed ACKs. The ratio of (forward) data packets to (reverse) ACK packets is bi-modal in the data we analyzed. Connections that have a slow last-mile technology (e.g. dialup modems) are far less likely to use delayed ACKs. In fact, we found stretch ACKs responding to more than 2 data packets were very common in Abilene.

Once a connection's RTT is known, we can infer its ceiling by computing the average number of packets per RTT. A surprising number of flows had ceilings that were much lower than would have been expected from the BDP. We investigated to see if they had congestion losses to keep their throughput down, but they did not. A portion of Chapter 4 investigates instances of Receive Window Limited connections and Send Window Limited connections. We found them to be far more prevalent in Internet2 than we expected.

Because the fiber-optic backbone links are too fast for comprehensive monitoring, flow data is taken on only a 1:100 sample of the packets. Would we still be able to infer RTT and, by extension, ceiling for an Autonomous System even in a sampled environment? A portion of Chapter 4 addresses the problems associated with using sampled data.

We chose to aggregate Autonomous Systems into groups based on their attachment point to Abilene and their approximate distance from Abilene based on RTT. Any IP address in the group would have the same attachment point to Abilene and roughly the same delay. We then chose only 2 categories of delay. Thus, each group consists of an attachment point (e.g. Indianapolis) and a delay beyond Abilene (e.g. 2 milliseconds from Indianapolis to Bloomington). From Abilene's point of view, connections to or from those IP addresses would take the same paths through Abilene and see the same extra delay. Our assumption was that any IP addresses in the group could be considered equivalent for the purposes of our study.

Our ultimate traffic matrix is constructed with one row and one column for each group. The content of the cell at that intersection is the quantity of traffic (estimated from flow data). Flows are assigned an RTT (directly taken from row plus column delays, but originally estimated from AS ACK ratios and throughput) and a ceiling (based on throughput of memory limited connections to that AS).

1.2 Contributions of this Work

This thesis makes contributions in the following areas:

1. A succinct AS-level graph of the Internet that accurately reflects the routing of traffic across links and contains the links most likely to have congestive losses.
2. A method for annotating the AS-level graph based on fresh traceroutes.
3. Demonstration of the importance of RTT in congestion and congestion propagation in high speed backbones.
4. Demonstration that window synchronization scales to high multiplexing factors.
5. Demonstration that the evidence of window synchronization can be used for network engineering tasks.
6. A model for predicting the variations in queue depth (and, therefore, delay) in congested links even at high multiplexing factors.
7. A mechanism that can infer RTT from delayed and stretched ACKs.
8. Evidence that memory-limited connections are far more prevalent in high-speed long-haul backbones than previously expected.
9. Improved understanding of the way memory-limited connections reduce the ability of traffic to grow back quickly after congestion.

1.3 Thesis Outline

In Chapter 2 we develop techniques for discovering and analyzing the AS-level links in the Internet. The resulting Internet graph is both succinct and significantly more accurate than prior graphs when used to predict packet sojourn.

Chapter 3 investigates congestion and develops a model that exposes the traffic parameters that need to be captured to characterize the traffic. By simulating high speed links and high levels of multiplexing, we study congestion event onset, duration, and intensity. This model differs from prior work in that it summarizes large collections of connections into tractable flocks whose characteristics simulate connection-level traffic without the need for packet-level detail. This allows much more scalable studies of multi-hop and networked traffic with large numbers of routers and complex interconnections.

In Chapter 4 we use easily-gathered summary flow data and infer RTT to create a traffic matrix that is appropriately accurate for emulating a large, trans-continental ISP.

Finally, in Chapter 5 we review related work.

Chapter 2

Topology of the Internet

To study the way traffic flows in the Internet, we decided to construct a graph of a significant portion of the Internet and apply traffic to it. This chapter shows how we decided what form our graph would take, then how the excess links were pruned from that graph to make it more compact. To improve accuracy, links were then added whenever traceroutes showed significant new links.

The goal of this chapter is to create a graph that can be combined with a traffic matrix we will develop in Chapter 4. To motivate the study of Internet topology, we use an example of services that are geographically and topologically dispersed in the Internet. For example, a company providing real-time streaming video might want to place an affordable number of servers in carefully selected places in the Internet to minimize the number of customers whose ping time exceeds 150 milliseconds.

Routing in the Internet often requires packets to travel much farther than the shortest distance from the sender to the receiver. There are a few, obvious geographic features like major oceans that are expensive to cross, but the commercial Internet also has other long paths. In part this is the result of the business relationships between Internet Service Providers. A packet moving from an educational institution to a research facility may travel on a subsidized research network, while another packet to a commercial website might not. Section 2.6 shows why small ISPs do not provide transit services between their providers.

It is important to treat the highly-connected core of the Internet differently than the small ISPs on the edges. A few ISPs have connections to hundreds of other ISPs. This core component is so highly interconnected, that it is appropriate to model them as a clique we will call the *forest floor*. The forest floor provides extremely stable routing with professionally managed fault tolerance and very high bandwidth. This chapter builds a graph of the Internet that can be thought of as a forest – a collection of trees connected to that forest floor. Small regional, local, and leaf ISPs have much smaller out-degree, so we model clusters of them as trees. In the context of our graph, the forest floor facilitates reliable, high volume movement between the trees.

To test the utility of this graph of the Internet, we present an novel, very fast algorithm that determines the optimal locations for placing services in a strict forest. Then we augment the forest by adding links that significantly improve the accuracy of the graph with only a small impact on the performance of the algorithm. The graph is no longer a strict forest. The trees are no longer acyclic and mutually disconnected. We have not proved and we do not claim that the result of running the algorithm on the augmented graph is optimal.

2.1 The Need for a Succinct Internet Graph

Content Delivery Networks (CDNs) distribute caches in the Internet as a means for reducing load on Web servers, reducing network load for Internet Service Providers and improving performance for clients. In order to effectively deploy and manage cache and network resources, CDNs must be able to accurately identify areas of client demand. One means for doing this is by *clustering* clients that are topologically close to each other, and then placing caches in the areas where demand is typically large. This raises two immediate questions: how can clusters of clients be computed and once identified, how can caches be placed among the clusters so as to maximize their impact?

In this chapter, we address the question of client clustering by presenting a new method that generates a hierarchy of client clusters. As opposed to prior work on IP client clustering described in [47], our method uses autonomous systems as the basic cluster unit. We argue that clustering at the IP level results in cluster units which are too detailed, and too numerous and thus do not readily lend themselves to higher levels of aggregation. In contrast, clustering at the AS level provides a natural means for not only identifying clients which should experience similar performance from a given cache but also for aggregating AS's into larger groups which should experience similar performance.

We will use the problem of distributing content delivery caches as an example to motivate our the clustering method. The CDN would want to clearly understand demand to effectively distribute a finite number of caches to the most effective places in the topology. Our clustering method enables groups of AS's to be coalesced into larger groups based on best path connectivity extracted from BGP routing tables. We use best paths because these are typically the preferred route between an AS and its immediate neighbors. The difficulty is that best paths do not indicate anything about quality of a connection beyond immediate neighbors.

We address this problem by introducing notion of Hamming distance between a pair of connected AS's. Hamming distance was introduced in [84] as the minimum number of elements which must be changed to move from one set to another. For example, the Hamming distance between $\{1,3,5,7\}$ and $\{1,2,3,4\}$ is four because $\{2,4,5,7\}$ appear in one but not both of the sets. In our context, Hamming distance is applied as a measure of similarity of AS connectivity. Specifically, two nodes with a *short* Hamming distance indicate that they have many neighbors in common and are thus candidates for merging into a cluster.

The length of a connection is the Hamming distance between the neighbor sets of the AS's it connects. AS's with minimal Hamming distance are successively coalesced. By reading the BGP table entries, we construct an AS graph where each AS is a vertex and each edge represents a direct connection between those AS's. Imagine 2 nodes of the AS Graph whose edges connect to highly correlated sets of vertexes. The Hamming distance between those neighbor sets would be small. If the algorithm decides to coalesce those two vertexes, one of the vertexes will become the *exemplar* of the new cluster, and the other will become a child of that exemplar.

Our clustering algorithm removes edges from the AS graph until all that remains is a forest of trees. The benefits of making a forest are: (1) objectively identifying a small number of vertexes that can be treated as the backbone of the Internet and (2) assigning each AS to one and only one tree so that tractable algorithms can be used to predict the paths packets will take going to or coming from the backbone. It is implicitly assumed that the backbone vertexes are tightly interconnected (ideally, a clique) and that packet transfers between backbone vertexes are very fast.

Our algorithm starts by coalescing nodes whose path to the backbone is uncontested, forming small clusters of nodes whose only known path to the bulk of the Internet passes through a common parent. In the BGP tables we examined, clusters were seldom that obvious. In order to form larger clusters, the algorithm successively relaxes the Hamming distance requirements for clustering. If we relax the Hamming distance requirements too far we would eventually collapse the entire network to a tree with a single root node. Our intention, however, is to only collapse the topology to a size which readily enables evaluation of demand and facilitates our cache placement algorithms. The result of our clustering algorithm presented in this chapter is a forest of 21 root AS trees. These root AS's consist of many of the major ISPs such as BBNPlanet and AT&T, but also some smaller ISPs such as LINX due to the nature of the algorithm. The root AS's connect on average with 7.29 other root AS's indicating a high level of connectivity between these nodes. The average out-degree of the root AS's (i.e.. the number of AS with whom they peer) is 198 with a median of 97 indicating that the root AS's facilitate Internet access to a large number of other AS's.

It is also important that the forest minimizes that amount by which it overstates the path lengths between vertexes in the original graph. To test that we measured paths in terms of AS hops. In the original graph, the average number of AS hops to those 21 tree roots is 1.61. The average tree depth in our graph is 1.96. This gave us confidence that our forest does not misrepresent AS hop distance significantly. These characteristics indicate that while a forest is an idealization of the actual AS topology, it does not abstract away essential details.

To test our topology, we ran 200,000 traceroutes and quickly found that the BGP-based forest did a dismal job of predicting packet paths. Our forest had implicitly assumed that nodes with more connections toward the backbone were providers and nodes with fewer connections were their customers. Leveraging the insights of Gao, et al. [33], we endeavored to discover which links were uni-directional because they were a customer-to-provider link.

Using a simple machine learning approach, we refined the forest by adding annotations to each vertex with our guess about the tier of the node. A link from a low-tier AS to a higher-tier AS indicates the relationship of a customer (higher-tier) and a provider (lower-tier). Similarly, we tried to infer sibling and peer status. The results were still sadly inaccurate.

The breakthrough that allowed us to dramatically improve the forest was, ironically, additions that made it no longer a forest of trees. We added up to one extra link from each customer to an alternate provider based on the preponderance of the traceroutes in our training set. Now that tier-n nodes could have up to 2 parents, trees were now mini-graphs. There were links that connected mini-graphs to other mini-graphs and we had to depend on the unidirectional notation to avoid cycles. The result was a graph that correctly classified 91% of the traceroutes in the test set.

One domain in which our forest of AS's naturally lends itself is cache placement. Since our tree generation algorithm is based on best path information from BGP tables, it enables caches to be placed on AS hop paths which would actually be used in the Internet. This study assumes that placing a cache in a AS is sufficient to satisfy all demand from that AS (as well as the AS's children which are part of its cluster). We make this assumption based on the idea that most performance problems occur *across* AS boundaries and that performance *within* an AS is generally good. Our analysis of cache placement effectiveness focuses on the reduction of inter-domain traffic. There is clearly an additional benefit of improving client performance which is a simple extension of our work.

Placement of caches in trees has been treated as a dynamic programming problem by Li, et al. [52] however the means by which trees were created was not treated in that work. We address the issue of optimal cache placement by describing a dynamic programming algorithm in which each subtree calculates the optimal use for 0 to ℓ caches in its subtree. Each parent node can then discover the maximum benefit from ℓ caches by distributing all of the caches among its children or by retaining one cache for itself. We also present a greedy algorithm which iteratively chooses the AS with largest unsatisfied demand as the next site to place a cache.

We evaluate the effectiveness of these two algorithms by comparing their total cost of traffic when 0 to 50 caches are placed. We find that optimal placement of a small number of caches does measurably better than random placement, but that greedy placement performs surprisingly close to optimal when more caches are deployed.

The remainder of this chapter is organized as follows: Section 2.2 describes our process for constructing client clusters using BGP routing data; and Section 2.3 describes the results of evaluating client demand from a Web log using our clustering results. In Section 2.4 we present our algorithms for optimally placing caches based on client demand distribution. In Section 2.5 we demonstrate the effectiveness of our cache placement methods. In Section In section 2.6 we use the results of traceroutes to identify the customer-provider relationships and improve the accuracy of the graph. 2.7, we summarize our results and conclude with directions for future study. In the chapter on related work, section 5.1 discusses research related to Internet topology and clustering.

2.2 Topologically-guided Clustering

A study of sources and destinations of traffic in the Internet quickly becomes a search for a productive way to summarize large bodies of traffic into meaningful categories. Categorizations based on geography are natural, but they are an increasingly inaccurate representation of the topology of the Internet. A house in the suburbs of Buenos Aires, Argentina is 9000 kilometers away from wisc.edu, but a connection between them may have much better throughput and latency than connections that seem to travel only a hundred kilometers from an ISP in Poland to an ISP in Romania.

Our algorithm discovers the topology of the Internet by reading the **best path** data from BGP routing tables [83]. BGP tables [83] contain a great deal of information about connections beyond the next hop. This enables us to construct an AS graph without having to query every BGP router in the world.

To forward a packet, one might think a router only needs to know which of its links to use for the next hop. A subsequent router will make decisions to get the packet even closer to its destination. Fortunately for us, BGP tables [83] contain a great deal of information about connections beyond the next hop. In the early days of Internet routing the designers wanted each BGP advertisement to contain the entire path of Autonomous Systems used to deliver a packet. This gives BGP routers full disclosure of the AS path their packets will take so the packets of one company (perhaps containing trade secrets or sensitive E-mail) would not pass through arch-enemy autonomous systems. The AS path can still be used for that purpose today.

We simplify the graph of AS connectivity into a forest of trees to facilitate our analysis. We found clusters of nodes with high mutual affinity by comparing their neighbor sets. We then iteratively applied the same technique to identify clusters of clusters(super-clusters), and so on until there were only a few, very large clusters left. Our algorithm identified 21 such super-clusters. They form the first level of the forest of trees. As of 2001, a dozen of them are almost completely interconnected. Since the tree representation loses information about cross-links between branches of the tree, it is important that our algorithm minimize the impact on distance calculations using the trees.

Our work extends the IP clustering work done by Krishnamurthy and Wang [47] showing how BGP routing tables can be used to gain 99 percent accuracy in partitioning IP addresses into non-overlapping groups. All IP addresses in a group are topologically close and under common administrative control. Their client clustering paper shows other more involved techniques for gaining even higher accuracy and validating the results.

The basic unit of clustering used by our algorithm is the combination of all of the IP ranges that share a common AS number. Although clustering by AS is less specific than IP clustering, the IP addresses in our clusters share common routings. Without common routing, applications of clusters such as cache placement may not be meaningful.

Definitions

The clustering algorithm uses neighbor sets, a boolean notion of one AS being a potential parent of another AS, a distance function that acts as the length of a link and an overhang function that measures the amount by which a potential parent fails to completely dominate a child.

The following definitions are used throughout this chapter:

- AS_n is a **neighbor** of AS_m if it immediately follows or precedes AS_m in any best path. To simplify the algorithm, AS_n is always added to its own list of neighbors.

- The **set of neighbors** of AS_n is denoted by N_n. The parent of AS_n is $p(n)$, initially 0, meaning undefined.

- The **exemplar** of a cluster of AS's is the parent of all other nodes in the cluster. The neighbor set, N_e, of the cluster is maintained under AS_e, where e is the AS number of the exemplar.

- The **outdegree**, $outdegree(n)$ is the initial $|N_n|$. Although the neighbor set changes during the coalescing of clusters, it is important to note that outdegree of an AS always refers to the original outdegree, before any clustering. The outdegree of a cluster is defined to be the outdegree of its *exemplar* AS.

- AS_n is said to **dominate** AS_m if $N_n \supset N_m$. In particular,

$$dom(n, m) \equiv (N_m \setminus N_n = \emptyset) \wedge (N_n \setminus N_m \neq \emptyset)$$

- The **Hamming distance** between AS_n and AS_m is the number of neighbors exclusive to only one of them.

$$hdist(n, m) \equiv |N_n \cup N_m| - |N_n \cap N_m|$$

- The **overhang** of AS_n over AS_m is the size of the set of Neighbors of n who are not also Neighbors of m.

$$overhang(n, m) \equiv |N_n \setminus N_m|$$

- Each node has a set of **candidate parents, C_n, that is recomputed as the algorithm progresses.**

Clustering AS's using BGP routing data

To construct hierarchical trees of AS's we needed to find the best assignment of small clusters (AS's with small out-degree) to larger clusters. For this study, we extracted "best path" data from a routing table acquired from Oregon Route-views [90] dynamically on Feb. 20, 2001. BGP routers typically receive multiple paths to the same destination. The BGP best path algorithm decides which is the best path to install in the IP routing table and to use for forwarding traffic. These paths tend to use a highest-throughput lowest-latency link. Our algorithm has no other means to discover that information directly.

Our study includes only best paths, thus some feasible routes are ignored. In particular, routes that connect AS's far from the backbone to other small AS's won't be seen. We investigated using all paths and found that low-bandwidth paths for fault tolerance and historical paths with comparatively low bandwidth made the clustering results volatile. Routing tables from different sources would significantly change the computed clustering.

Clustering is performed by successive passes through the graph building large clusters by visiting small clusters and merging them into an existing larger cluster.

For each clustering pass, each node, n, without a parent (i.e. $p(n) = 0$) tries to find a suitable parent. Conceptually, the candidate parents are the nodes which dominate it, $C_n = \{m \in N_n | dom(m,n)\}$. In practice, this is too strict a requirement and we will define C_n more suitably below. Now, find the nearest among the candidate parents, $m \in C_n$. The best parent is

$$nearest(n) = \min_{m \in C_n} \{hdist(n,m)\}$$

If $C_n \neq \emptyset$, Node n is merged into the cluster of the best parent, m. Now $p(n)$ is set to m and n is removed from N_m. Note that n is not removed from other neighbor lists, since n might later be chosen as a parent by an even smaller cluster.

An interesting design decision happens in situations where $N_m = N_n$, neither neighbor list is a proper superset of the other and neither dominates. We defined domination in this way so both nodes are free to become siblings under some other parent, keeping the tree comparatively shallow. If n or m had been arbitrarily chosen as parent, the other (and its subtree) would appear to be one AS hop farther from the backbone.

It might also be meaningful to define the best parent as the farthest candidate parent. This would cause AS's to choose AS's with very high out-degree as their preferred parent. The result would have been a shallower tree that more closely matches the distance to the backbone, but it also would have lost the useful categorization of AS's into clusters with very similar sets of neighbors.

In practice, many AS's connect to more than one major provider. These AS's are not strictly dominated by any one of the nodes they have links to. To relax the domination requirement, a tolerance factor grows with each pass through the nodes without parents. The tolerance, δ, allows a node to become a child of any node with a higher out-degree if the overhang is less than the current tolerance. δ drives the speed at which the clustering completes. So the actual computation for **the set of candidate parents** is:

$$C_n = \left\{ m \in N_n \;\middle|\; \begin{array}{c} overhang(n,m) \leq \delta \ \wedge \\ outdegree(m) > outdegree(n) \end{array} \right\}$$

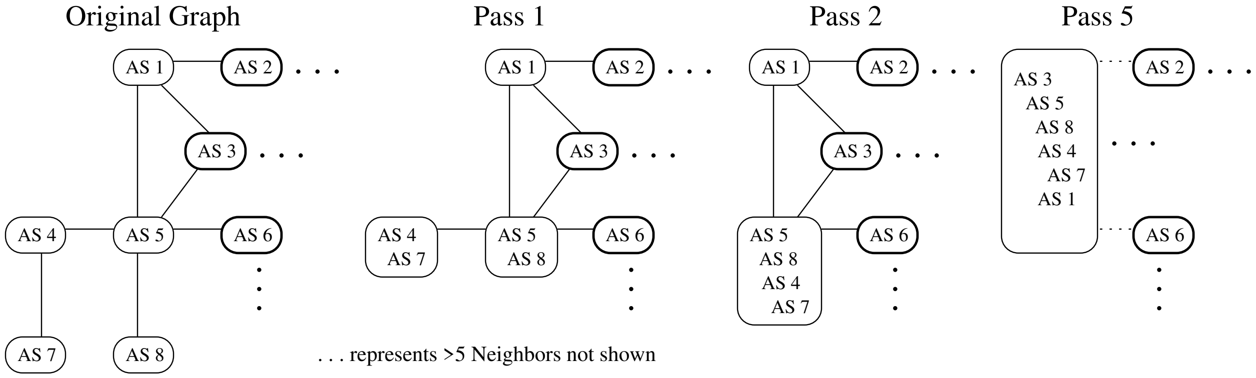

Figure 2.1 Walk-through of the clustering algorithm

Cluster generation example

A simple example demonstrates how the clustering operates in practice. In Figure 2.1, AS 2, AS 3, and AS 6 are connected to many other nodes. In this example $N_7 = \{4, 7\}$ is dominated by $N_4 = \{4, 5, 7\}$ so $dom(4, 7) = true$. For each pass, each node makes a list of candidate parents. During the first pass, AS 7 coalesces with AS 4. AS 4 is now the *exemplar* for a cluster and AS 7 is removed from N_4 reducing it to $\{4, 5\}$. The parent of AS 7, $p(7)$, is set to 4. Similarly, AS 8 is dominated by AS 5. During the second pass, AS 4 coalesces with AS 5 to form an even bigger cluster with AS 5 as the *exemplar*.

In the third pass, the algorithm has nothing to coalesce, since no node is dominated by any single neighbor. In this case $N_1 = \{1, 2, 3, 5\}$ is not dominated by AS 2, AS 3, or AS 5. Since AS 1 connects to one node (AS 2) missing from the AS 5 list, $overhang(1, 5) = 1$. Similarly, $overhang(5, 1) = 1$ because of AS 6.

In a later pass, the tolerance grows above 1.0 and the candidate parent set of AS 1 becomes $C_1 = \{3, 5\}$. The nearest of these is AS 5, so AS 1 coalesces with AS 5. During the same pass, the candidate parents of AS 5 becomes $C_5 = \{3\}$. Note that AS 1 is not a candidate parent of AS 5 because it originally had a smaller outdegree.

In the example, AS 7 would be denoted as AS3.5.4.7. The name shows the relationship that AS 7 is a child of the progressively larger super-clusters. Clients in AS 7 would benefit (albeit progressively less) from caches on the path to the backbone.

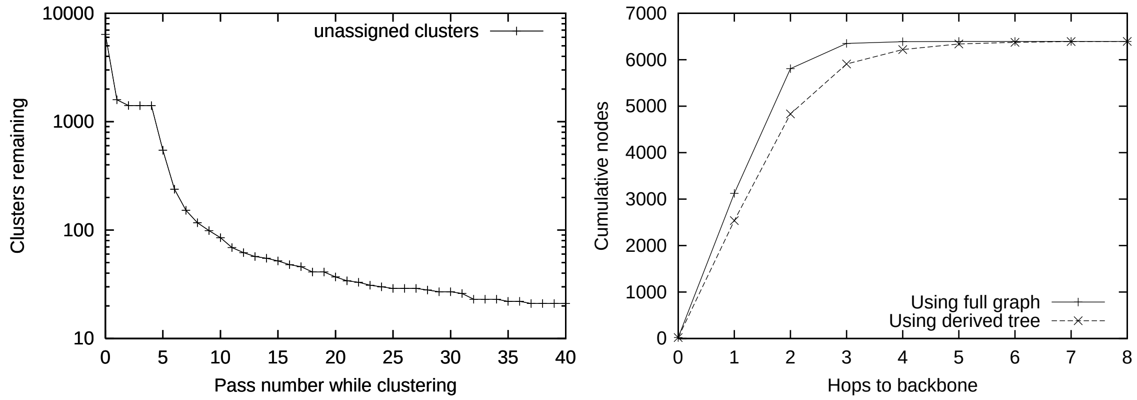

Figure 2.2 Results of AS cluster formation. The left graph shows how the number of clusters declines as clusters are coalesced. The right graph shows how the path length in the derived tree compares to the path length in the original graph of best paths.

Results of AS clustering

For this study a δ tolerance growth of 0.25 per pass was chosen. Figure 2.2 shows the number of clusters at the end of each pass through the list of AS's. The first four passes cluster all of the easily-classified AS's with small out-degree. Passes five through ten found a large number of national, government, and educational transit AS's. After the pass 37, further reduction in the number of clusters takes much longer. To avoid excess layers at the top of the tree, we stopped the algorithm at pass 40 and declared the 21 remaining *exemplars* to be the roots of the forest of 21 trees.

Figure 2.2 compares the cumulative distribution of distances to the backbone in both the original full graph and the tree left at the end of clustering. The maximum distance from the backbone was 5 in the full graph but rose to 8 in the forest. There were only 56 nodes in the forest farther than 5 hops from the backbone. This matched our goal for the backbone since over 90 percent of the 6395 nodes are within 2 hops of a backbone node in the graph and within 3 hops of a backbone node in the forest. The average node is 1.61 hops away from the 21 “backbone” nodes in the full graph, and 1.96 hops away from those same 21 nodes in the computed forest.

The resulting clustering contains 21 large trees, each headed by a particular AS. Table 2.1 shows the names of those Autonomous Systems. The list does not contain some of the AS's with

Table 2.1 Clusters Identified as Backbone by the Algorithm

Clstr #	Exemplar AS	Members	Out Degree	Peers	Depth
1	2914: Verio	150	235	13	5
2	1: BBNPlanet	171	284	12	4
3	701: Alternet	492	878	12	8
4	7018: AT&T	281	374	11	4
5	2828: Concentric	30	85	9	5
6	3549: Globalcenter	33	60	9	2
7	3561: Cable&Wireless	287	482	9	5
8	6453: Teleglobe	57	124	9	6
9	293: ESnet	41	112	8	5
10	1239: Sprint	407	645	8	5
11	2497: JNIC	45	82	8	6
12	3356: Level3	33	60	8	3
13	209: QWest	83	112	7	4
14	3300: Infonet-Europe	21	40	6	3
15	702: UUNet-Europe	56	80	5	5
16	1221: Telstra	27	61	5	1
17	1755: EBone	59	97	4	6
18	5378: INSNET	32	59	4	8
19	1849: PIPEX	26	47	3	5
20	2548: ICIX	158	189	2	3
21	5459: LINX	26	49	1	4

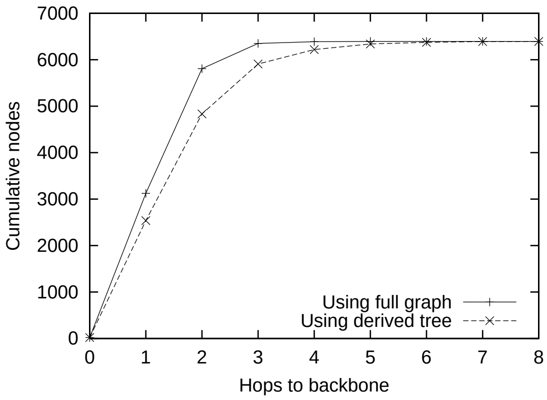

Figure 2.3 Hops to the backbone

high out-degree. Presumably, this is because they were dominated (at some small tolerance) by an AS that is on the list. Alternet had the largest number of immediate children at 492, a little over half of its out-degree (878) in the full graph. There were 2515 AS's at the second level of the tree, making the average number of children per backbone node 120. The top three levels include a total of 4833 AS's that are within 2 hops of the backbone.

AS clustering limitations

BGP routing tables don't show peering relationships that often permit packets to take shortcuts through the Internet. This is because routers will intentionally NOT advertise peers if they do not want to provide transit services for those peers. We have not studied the extent that these relationships improve global traffic statistics.

Other complications can make the AS path less accurate. In RFC 1772 [82], Route Aggregation allows an AS to advertise an aggregate route in which contiguous IP addresses can be collapsed to a single entry. The rules of BGP4 require that the aggregated route contain all of the AS numbers for any portion of the aggregation. This sometimes overstates the length of the AS path. It is also possible to use an atomic aggregate, thus effectively hiding some AS numbers from appearing in the AS path.

Our algorithm also depends on the AS path being a sequence, an ordered list of the AS numbers traversed to deliver a packet to a given IP address range. The BGP4 specification allows an AS path to be an unordered AS set, but requires that it become an AS sequence before it is passed as a advertisement to a neighboring AS. In theory, this means that any BGP4 AS path farther than 1 hop away from its ultimate destination must be an AS sequence and our algorithm assumes this to be true.

Route Views [90] is a standard source for timely, composite BGP information. It collects BGP information from routers widely distributed throughout the Internet. Nonetheless, initial investigation indicates that adding other routing tables would be unlikely to materially affect our clustering. Route Views already incorporates a sufficient number of routers near the centroid we identified.

Finally, our algorithm creates a forest that sometimes makes an AS appear farther from the backbone than it really is. This most often occurs because the cluster with the least overhang over a subject cluster is preferred when the subject cluster picks a parent. The average depth of the cluster tree was 1.961, whereas the average number of hops to the backbone in the full graph was 1.595. The right-hand graph in Figure 2.2 shows how these two metrics compare.

2.3 Client Demand Analysis

To map demand into our AS hierarchy, we needed to know the quantity and the composition of client requests that come from each leaf cluster. A simple case is a web server with a single host name. To demonstrate our cache placement techniques, we analyzed a single commercial web server log. The incoming traffic are requests to that server and *demand* is the total count of successfully answered requests and the total number of bytes delivered in replies. The number of bytes in the requests is assumed to be small. Since byte count cannot be easily captured, we will characterize the incoming requests by count rather than by size in bytes. The outgoing traffic is the replies to those requests. To simplify later analysis, we chose the set of requests that succeeded. In this way, the count of incoming requests and the count of outgoing replies were the same. It is a simple matter to total the number of bytes sent in reply to successful requests.

This process anonymizes the data so that individual IP addresses are not disclosed. It is hoped that this level of anonymity is sufficient to protect the privacy of individuals and still be able to publish useful results.

Converting IP addresses to AS numbers

The process of converting IP addresses to AS numbers is analogous to the way IP routers match the longest prefix of the IP address contained in the composite routing table obtained in the prior step. The demand summary [76] for each web server log is a compact file, suitable for sending across the network to a collection point. Each demand summary file contains one line for each AS number that had non-zero requests. The line contains the AS number, the count of successful requests, and the number of bytes in replies.

Web server log

For this study, we use a log from a commercial web server collected in February, 2001. The log contained 18 hours of requests that were globally diverse containing 402,955 requests making up 3.69 Gigabytes. There were requests from 791 different autonomous systems. The 50 AS's with

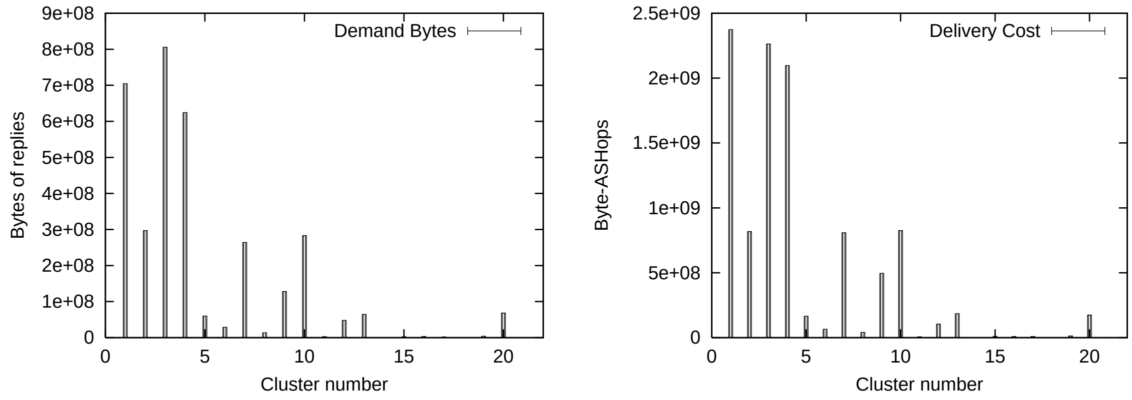

Figure 2.4 Demand aggregated to the 21 backbone nodes

the highest demand accounted for 232,991 requests and 2.19 Gb. To avoid complex error scenarios, we filtered out all of the requests except HTTP GET requests with successful result (codes 200 to 203).

Since the web server log contains result codes that indicate errors the log contains activity that we chose not to consider. In particular, result code 304 is a redirection code whose impact on our results is unclear. We will investigate the 300-series result codes in a later study. For this study, we filtered out all of the requests except HTTP GET requests with result codes 200 to 203 (various forms of success).

Demand Aggregation

Figure 2.4 shows the aggregate demand from each of the 21 major clusters in both bytes and byte-ASHops. The graphs show that the commercial web server had clients that were concentrated in certain areas of the Internet. The 3 busiest were the clusters whose exemplars were Verio, Alternet, and AT&T with 64 percent of the bytes and 63 percent of the byte-ASHops in replies. The BBNPlanet cluster was particularly interesting because it was also one of the best trees for delivering the test data in the fewest ASHops (2.752 ASHops including 1 for BBNPlanet and 1 for the root). The clusters with averages above 3.5 ASHops were those represented by ESNET, UUNET-Europe, LINX and EBONE.

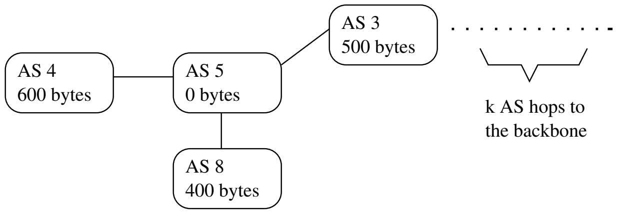

Figure 2.5 Tadpole Graph Example

2.4 Cache Placement

The result of our clustering algorithm is a forest of trees containing clusters of AS's in increasingly detailed groups. The fundamental assumption is that analysis of a load pattern against this model will yield a useful, objective measure of the value of placing caches into this forest. The problem is similar to that posed by Li, et al. [52], but we simplified it by setting the delivery cost to be the number of Autonomous Systems that the reply entered times the number of bytes in the reply.

To do this, we assign a weight to each leaf node equal to the number of bytes given to it in successful replies. Parent clusters of that leaf are responsible for finding the optimal use of m proxy caches for each value of ℓ up to the total number of proxy caches we can afford to place. Each node can choose to distribute those ℓ caches in any amounts among its children and can choose to keep one for itself. We visualize this as pebbles placed onto the tree wherever a proxy cache is indicated. Our cache placement study assumes that any proxy cache will completely satisfy all requests sent to it. We assume that all requests are sent to web servers on the backbone. The cost of each reply is the number of AS's that see the reply (including the originating AS) multiplied by the size in bytes of the reply. The cost of the requests is ignored.

Figure 2.5 shows a subtree near the bottom of a large tree. In the absence of caches, the 600 bytes of replies for AS 4 would be seen by $k + 3$ systems as they traveled from the backbone. Placing a pebble at AS 4 will satisfy its 600 byte demand locally. If that were the only pebble placed, the other 900 bytes of demand would escape and their cost would be $(500(k+1)+400(k+3))$. So, the total cost of the AS 3 subtree given only a single pebble (and placing it at AS 4) is $600 + (1700 + 900k)$.

For any vertex v of the tree T, denote the subtree rooted at v by T_v. For $k \geq 0$ we consider a *tadpole graph* (T, k) defined as T appended by a single path extending upwards from the root of T with k extra vertices. Traffic is said to escape if the request and reply need to traverse the k vertices in the tail. The cost of a tadpole graph (T, k) is the cost of the subtree traffic plus k times the cost of the traffic that escapes.

From the point of view of AS 3, the cost of the traffic will be different depending on how many pebbles are used. We will use ℓ to represent the number of pebbles available. If $\ell = 0$, AS 3 can place 0 pebbles and its cost is $0 + (3500 + 1500k)$. If AS 3 can place $\ell = 4$ pebbles, the cost of its subtree is 1500, although in this case, the pebble placed at AS 5 is not useful.

An interesting problem lies in comparing the options AS 3 has if offered only 1 pebble. At $k = 0, \ell = 1$, AS 3 should place the pebble at AS 4 for a total cost of 2300. But at $k = 100$, AS 3 would choose to put the only pebble on AS 3 for a total cost of 3500. Clearly, cost is not a simple function of k.

The reader may want to test his understanding by optimizing the cost of AS 3's subtree at $k = 0$ if we offer him 2 pebbles. The node can choose to keep one for himself and let his children use one, or he can choose to let his children use both.

Simultaneous placement algorithm

We are given a rooted tree with n vertices. Every leaf v is associated with a non-negative weight $w[v]$. There are m pebbles, where m is at most the number of leaves. Consider any placement of up to m pebbles on any vertex of the tree. A placement of pebbles is called *feasible* if every leaf with a non-zero weight $w[v] > 0$ has an ancestor which has a pebble on it. Here the ancestor relation is the reflexive and transitive closure of the parent relation, in particular every vertex is an ancestor of itself. The cost of any feasible placement P is defined as follows:

$$c(P) = \sum_{v} c(v),$$

where the sum is over all leaves v, and the cost associated with the leaf v, denoted by $c(v)$, is $(\lambda + 1) \cdot w[v]$, where λ is the distance from v to the closest pebbled ancestor of v. Here the distance between two vertices of the tree is the number of edges on the unique shortest path between them. For technical reasons we define the cost of an infeasible placement to be ∞.

The goal is to find a feasible placement P with at most m pebbles such that $c(P)$ is minimized.

Binary tree case

We first consider the case of binary trees, where every vertex has at most two children. Of course a leaf has no children. Thus for non-leaves, either there is a unique child, or there are two children, in which case we order them as left and right arbitrarily.

For any vertex v of the tree T, denote the subtree rooted at v by T_v. Generically, if v has a unique child then we denote that child by v_1, and if there are two children then we denote them v_1 and v_2 respectively. For $k \geq 0$ we consider a *tadpole graph* (T, k) defined as T appended by a single path extending upwards from the root of T with k extra vertices. Note that $(T, 0) = T$.

For $\ell \geq 0$, We will consider the optimal placement of at most ℓ pebbles in T_v, and denote the minimal cost by $f_v(0, \ell)$. More generally, for $k > 0$ and $\ell \geq 0$, we will consider the optimal placement of one pebble at the tip of the sperm graph (T_v, k) which has distance k from the root v of T_v, and at most ℓ pebbles within T_v. We denote by $f_v(k, \ell)$ the minimal cost $c(P)$ of all feasible pebbling P of (T, k) with at most ℓ pebbles in T_v, and where if $k > 0$ we stipulate that one additional pebble is placed at the tip of the external path from v. If $k = 0$ and $\ell = 0$ then we have a feasible pebbling if and only if all weights in T_v are zero, in which case $f_v(0, 0) = 0$. Note that for any $k, \ell \geq 0$ and $k + \ell \geq 1$, a feasible pebbling exists. For $k = \ell = 0$, and if some non-zero weights exist in T_v, and thus no feasible pebbling exists, we denote $f_v(0, 0) = \infty$.

We will compute $f_v(k, \ell)$ for all $k, \ell \geq 0$, inductively for v according to the height of the subtree T_v, starting with leaves v.

More formally, let L_v be the number of leaves in T_v. Let $d_v = d_v(T)$ be the depth of v in T, i.e., the distance from the root of T to v (by our definition of distance, the depth of the root is 0). Let $h(T_v)$ be the height of the tree T_v, which is the maximum depth of all leaves in T_v, i.e., $h(T_v) = \max_u d_u(T_v)$, where u ranges over all leaves in T_v. A tree with a singleton vertex has height 0. Inductively for $0 \leq h \leq h(T)$, starting with $h = 0$, we compute $f_v(k, \ell)$, for all $v \in T$ such that the subtree T_v has $h(T_v) = h$, and for all $0 \leq k \leq d_v$, and for all $0 \leq \ell \leq L_v$.

Base Case $h = 0$:

In the base case $h = 0$ and we are dealing with a singleton leaf, together with an extension of a path of length k if $k > 0$, and no extensions if $k = 0$.

Thus, for $k = 0$,

$$f_v(0,0) = \begin{cases} 0 & \text{if } w[v] = 0 \\ \infty & \text{otherwise,} \end{cases}$$

and for $\ell = 1$, (note that $h(T_v) = h = 0$ implies that $L_v = 1$),

$$f_v(0,1) = w[v].$$

Now for $k \geq 1$,

$$f_v(k,0) = (k+1) \cdot w[v],$$

and for $\ell = 1$,

$$f_v(k,1) = w[v].$$

Inductive Case $h > 0$:

For the inductive case $h > 0$, we have some v with $h(T_v) = h$, and we assume we have computed all $f_{v'}(k, \ell)$ for children v' of v. There are two cases, v has either one or two children. First we consider v has a unique child v_1. For either $k = 0$ or $k > 0$, we can consider either placing a pebble at v or not placing it there. But we claim that without loss of generality we don't need to place it there. Because v has only one child, if an optimal pebbling places a pebble at v, we can obtain at least as good a pebbling by moving the pebble from v to v_1, and if v_1 is already pebbled we can remove one pebble. Thus, we have an optimal pebbling of (T_v, k) using at most ℓ pebbles in T_v without a pebble at v. Hence,

$$f_v(0, \ell) = f_{v'}(0, \ell),$$

and for $k > 0$,

$$f_v(k, \ell) = f_{v'}(k+1, \ell).$$

Suppose now v has two children v_1 and v_2. Basically we must decide how to distribute ℓ pebbles in the subtrees T_{v_1} and T_{v_2} with ℓ_1 and ℓ_2 pebbles each. There is a slight complication as to whether to place a pebble at v, the root of T_v, which affects how many pebbles there are to be distributed, either $\ell_1 + \ell_2 = \ell$ or $\ell - 1$.

First $k = 0$. If we place a pebble at v, (which of course presupposes $\ell > 0$), then there are $\ell_1 + \ell_2 = \ell - 1$ pebbles to be distributed in T_{v_1} and T_{v_2}, but with respect to these two subtrees the "k" values are both 1, i.e., we have $f_{v_1}(1, \ell_1) + f_{v_2}(1, \ell_2)$, minimized over all pairs $\ell_1 + \ell_2 = \ell - 1$. (To be precise, all pairs (ℓ_1, ℓ_2), such that $0 \le \ell_1 \le L_{v_1}$, $0 \le \ell_2 \le L_{v_2}$ and $\ell_1 + \ell_2 = \ell - 1$; but we will not specify this range explicitly in the following.)

If we don't place a pebble at v, then there are $\ell_1 + \ell_2 = \ell$ pebbles to be distributed in T_{v_1} and T_{v_2}, and since $k = 0$ for T_v, with respect to these two subtrees we still have the "k" values 0. So we have $f_{v_1}(0, \ell_1) + f_{v_2}(0, \ell_2)$, minimized over all pairs $\ell_1 + \ell_2 = \ell$.

The optimal cost $f_v(0, \ell)$ is the minimum of these two minimizations, i.e.,

$$f_v(0, \ell) = \min \left\{ \begin{array}{c} \min\limits_{\ell_1 + \ell_2 = \ell - 1} \{f_{v_1}(1, \ell_1) + f_{v_2}(1, \ell_2)\}, \\ \min\limits_{\ell_1 + \ell_2 = \ell} \{f_{v_1}(0, \ell_1) + f_{v_2}(0, \ell_2)\} \end{array} \right\}.$$

(It is understood that in case $\ell = 0$, the first minimization is vacuous and should be omitted. This is the standard convention, a minimization over an empty set (no non-negative ℓ_i sum to -1) is ∞. Also the second minimization is merely $f_{v_1}(0, 0) + f_{v_2}(0, 0)$ which is typically ∞ unless all weights in T_v are zero, in which case it is 0.)

We consider the case $k \ge 1$ next. For $\ell = 0$ we have

$$f_v(k, 0) = f_{v_1}(k + 1, 0) + f_{v_2}(k + 1, 0).$$

Suppose $\ell > 0$. Again we have the possibilities of placing a pebble at v or not. Thus,

$$f_v(k, \ell) = \min \left\{ \begin{array}{c} \min\limits_{\ell_1 + \ell_2 = \ell - 1} \{f_{v_1}(1, \ell_1) + f_{v_2}(1, \ell_2)\}, \\ \min\limits_{\ell_1 + \ell_2 = \ell} \{f_{v_1}(k + 1, \ell_1) + f_{v_2}(k + 1, \ell_2)\} \end{array} \right\}$$

This completes the description of the computations of $f_v(k, \ell)$. The final answer is $f_r(0, m)$, where r is the root of T and m is the number of pebbles. If m is given, (typically much smaller than the number of leaves), in the above computations one never needs to compute for ℓ, the number of pebbles allowed, beyond m, i.e., all $\ell \leq m$.

We estimate the complexity of the algorithm. Let $H = h(T)$ be the height of the tree. Typically $H \approx O(\log n)$. For leaves, the algorithm spends $O(d_v) = O(H)$ time per leaf. For each vertex with one child the time is $O(d_v \min\{L_v, m\}) = O(Hm)$. For each vertex with two children it is $O(d_v \min\{L_v, m\}^2) = O(Hm^2)$. Hence the total running time is at most $O(nHm^2)$, which is only $O(nm^2 \log n)$ with $H \approx O(\log n)$.

It is also clear that the above algorithm can be easily modified to compute the actual optimal algorithm in addition to the optimal cost.

General trees

We now generalize the above algorithm to an arbitrary tree. First, for a leaf node v, we define $f_v(k, \ell)$ to be the minimal cost $c(P)$ of all feasible pebbling P of (T_v, k) with at most ℓ pebbles in T_v, and where if $k > 0$ we stipulate that one additional pebble is placed at the tip of the external path from v. Note that in the case of leaf node, T_v is a singleton, and if $k > 0$ then (T_v, k) is a single path of length k. Also $0 \leq \ell \leq L_v = 1$, and $0 \leq k \leq d_v$.

Thus, the computation for the leaves are identical to that in the binary tree. If $k = 0$, then

$$f_v(0, 0) = \begin{cases} 0 & \text{if } w[v] = 0 \\ \infty & \text{otherwise,} \end{cases}$$

and for $\ell = 1$,

$$f_v(0, 1) = w[v].$$

For $k \geq 1$,

$$f_v(k, 0) = (k + 1) \cdot w[v],$$

and for $\ell = 1$,

$$f_v(k, 1) = w[v].$$

We now consider non-leaf nodes v. Let Δ be the number of children of v, let $v_1, v_2, \ldots, v_\Delta$ be its children from left to right, and let the subtrees rooted at the children of v be $T_{v,1}, T_{v,2}, \ldots, T_{v,\Delta}$

respectively. Denote by $T_{v,[d]}$ the subtree of T_v induced by the vertex set of $\{v\} \cup \bigcup_{i=1}^{d} T_{v,i}$, for $1 \leq d \leq \Delta$. Denote by $L_{v,d}$ the total number of leaves in $T_{v,[d]}$.

Define $f_{v,d}^{b}(k,\ell)$, where $b = 0$ or 1, $1 \leq d \leq \Delta$, $0 \leq \ell \leq L_{v,d}$, and $0 \leq k \leq d_v$, as follows. First let $k = 0$. If $b = 0$, $f_{v,d}^{0}(0,\ell)$ is the minimal cost of a pebbling placement of the subtree $T_{v,[d]}$, where we use at most ℓ pebbles in $T_{v,[d]}$, and no pebble is placed on v. (When no feasible pebbling placement exists with this constraint we have $f_{v,d}^{0}(0,\ell) = \infty$.) If $b = 1$, $f_{v,d}^{1}(0,\ell)$ is the same as above except v is placed with a pebble out of ℓ pebbles.

This definition is generalized for $k \geq 0$. For $f_{v,d}^{b}(k,\ell)$, we consider $(T_{v,[d]}, k)$ in place of $T_{v,[d]}$ and for $k > 0$ we stipulate that one additional pebble is placed at the tip of the external path from v of distance k from v. As before this additional pebble is not counted in ℓ.

We then define

$$f_{v}^{b}(k,\ell) = f_{v,\Delta}^{b}(k,\ell),$$

and

$$f_v(k,\ell) = \min\{f_v^0(k,\ell), f_v^1(k,\ell)\}.$$

Again we will compute $f_v(k,\ell)$ for all $k,\ell \geq 0$, inductively for v according to the height of the subtree T_v, starting with leaves v. The base case $h = 0$ having already been taken care of, we assume $h > 0$ and $h(T_v) = h$.

First we consider the left most subtree $(T_{v,1}$ with $d = 1$, i.e., we compute $f_{v,1}^{b}(k,\ell)$ for $(T_{v,[1]}, k)$.

If $k = 0$ and $b = 0$, then

$$f_{v,1}^{0}(0,\ell) = f_{v_1}(0,\ell).$$

Note that $h(T_{v_1}) < h$ and thus inductively $f_{v_1}(k,\ell)$ have been all computed already.

Similarly for $k = 0$ and $b = 1$, then

$$f_{v,1}^{1}(0,\ell) = \begin{cases} \infty & \text{if } \ell = 0 \\ f_{v_1}(1,\ell-1) & \text{if } \ell \geq 1. \end{cases}$$

Note that in the last equation the "k" value in f_{v_1} is 1 due to the stipulation that by $b = 1$ we placed a pebble on v.

Now we consider $k \geq 1$. Again if $b = 0$,

$$f^0_{v,1}(k,\ell) = f_{v_1}(k+1,\ell).$$

Similarly for $k \geq 1$ and $b = 1$,

$$f^1_{v,1}(0,\ell) = \begin{cases} \infty & \text{if } \ell = 0 \\ f_{v_1}(1,\ell-1) & \text{if } \ell \geq 1. \end{cases}$$

We proceed to the case of $1 < d \leq \Delta$. This time we inductively assume that we have already computed not only all $f_{v'}(k,\ell)$ with $h(T_{v'}) < h$, but also the relevant quantities for $(T_{v,[d-1]}, k)$.

Thus, for $k = 0$ and $b = 0$,

$$f^0_{v,d}(0,\ell) = \min_{\ell'+\ell''=\ell}\{f^0_{v,d-1}(0,\ell') + f_{v_d}(0,\ell'')\}.$$

To be precise the minimization is over all pairs (ℓ', ℓ''), such that $0 \leq \ell' \leq L_{v,d-1}$, $0 \leq \ell'' \leq L_{v_d}$ and $\ell_1 + \ell_2 = \ell \leq L_{v,d}$.

For $k = 0$ and $b = 1$,

$$f^1_{v,d}(0,\ell) = \min_{\ell'+\ell''=\ell}\{f^1_{v,d-1}(0,\ell') + f_{v_d}(1,\ell'')\}.$$

Note that in f_{v_d} we had the "k" value 1 since by $b = 1$ we have stipulated that a pebble is placed on v. The range of (ℓ', ℓ'') is the same as before except in fact ℓ' must be ≥ 1, otherwise the value ∞ will appear. (In particular, for $\ell = 0$ the minimization is ∞.)

Finally we consider the case $d > 1$ and $1 \leq k \leq d_v$. For $k \geq 1$ and $b = 0$, we have

$$f^0_{v,d}(k,\ell) = \min_{\ell'+\ell''=\ell}\{f^0_{v,d-1}(k,\ell') + f_{v_d}(k+1,\ell'')\}.$$

And for $k \geq 1$ and $b = 1$, we have

$$f^1_{v,d}(k,\ell) = \min_{\ell'+\ell''=\ell}\{f^1_{v,d-1}(k,\ell') + f_{v_d}(1,\ell'')\}.$$

Note that in the last equation, in fact the minimization is over all pairs (ℓ', ℓ'') with $\ell' \geq 1$, as well as $\ell' \leq L_{v,d-1}$, $0 \leq \ell'' \leq L_{v_d}$ and $\ell_1 + \ell_2 = \ell \leq L_{v,d}$. But we do not need to explicitly state that

$\ell' \geq 1$, since for $\ell' = 0$, $f^1_{v,d-1}(k, 0) = \infty$ can be shown by an easy induction. Also note that the "k" value in f_{v_d} is 1, due to the stipulation by $b = 1$ that v is pebbled by one of the ℓ' pebbles.

We have completed the description of the algorithm. The final answer is $f_r(0, m)$, where r is the root of T and m is the number of pebbles. Again, there is no need to compute for any value $\ell > m$, if m is the total number of pebbles given.

The complexity of the algorithm can be easily estimated as before. For leaves, the algorithm spends $O(d_v) = O(H)$ time per leaf. Thus the total work spent on leaves is at most $O(nH)$. For any non-leaf v, suppose the degree of v is Δ_v, then the computation work spent for v is $O(\Delta_v H m^2)$. Thus the total amount of work spent for non-leaves is $O(\sum_v \Delta_v H m^2) = O(nHm^2)$. Hence the total running time is at most $O(nHm^2)$, which is again only $O(nm^2 \log n)$ with $H \approx O(\log n)$.

This is a polynomial time algorithm that computes the optimal pebbling placement as well as the optimal cost of the pebbling placement. The running time is $O(nHm^2)$, for any rooted tree of n vertices, height H, and m pebbles.

Implementation of the Simultaneous Placement Algorithm

Our simultaneous placement algorithm is a *dynamic programming* algorithm that visits each node exactly once to determine the best use of m caches in its subtree. The algorithm discovers the optimal placement of all values of ℓ caches from 0 to m so as to minimize the total cost of traffic.

The result of running the evaluation on any node v is a $k \times m$ matrix $f_v(k, \ell)$ containing the total costs of the subtree where $0 \leq \ell \leq m$ is the number of caches and k is the distance to the nearest source of the data. For each element of the matrix, the node must choose how many pebbles to give to each of its children and whether or not to keep a pebble for itself.

We define $f^0_v(k, \ell)$ to be the cost if a pebble is not used at v and $f^1_v(k, \ell)$ to be the cost if v distributes $\ell - 1$ pebbles to its daughters and keeps one pebble for itself.

Leaf nodes can compute their cost matrix $f_v(k, \ell)$ easily. If they are given one or more pebbles, their cost is simply the number of bytes of replies needed by that AS. Assume t_v is that number of local traffic bytes at node v. If a leaf is given zero pebbles, its cost is $k * t_v$. In the implementation, we used a matrix that is 15 rows high, representing values of k from 0 to 14. In our study, the maximum number of pebbles, m, is set to 50 but could be increased at the cost of running time and memory consumed by the algorithm.

Define $f^b_{v,d}(k, \ell)$ to be the cost of a subtree of T_v, where $1 \leq d \leq \Delta$ is one of the Δ daughters of node v.

Each row k of $f^0_v(k, \ell)$ is computed using row $k+1$ from the daughters. Start with the first daughter's $k+1$ row intact. Then for each subsequent daughter, test all distributions of $\ell' + \ell'' = \ell$ pebbles in which ℓ' pebbles are given to the prior daughters and ℓ'' pebbles are given to the new child.

$$f^0_{v,d}(k, \ell) = \min_{\ell'+\ell''=\ell} \{f^0_{v,d-1}(k, \ell') + f_{v_d}(k+1, \ell'')\}.$$

When all Δ children have been combined, the resulting $f^0_{v,\Delta}(k, \ell)$ matrix is $f^0_v(k, \ell)$.

Now we construct $f^1_v(k, \ell)$. The first element, $f^1_v(0, 0)$ is ∞, because no pebble is available. To find the rest of $f^1_v(k, \ell)$, take the first row of $f^0_v(k, \ell)$ and shift it down by 1 pebble because the children will only have $\ell - 1$ pebbles to distribute. Note that all other rows of the matrix are copies of row 0.

$$f^1_v(k, \ell) = f^0_v(0, \ell - 1)$$

Finally, each element $f_v(k, \ell)$ is the minimum of $f^1_v(k, \ell)$ and $f^0_v(k, \ell)$.

To compute the best placement for the whole tree, we compute the cost matrix of the root, $f_{root}(k, \ell)$. The row $k = 0$ contains the minimum cost for the whole tree for values of $0 \leq \ell \leq m$.

Practical computational cost

Let i be the number of interior (non-leaf) nodes in the tree (1594 in our study). Let H be the height of the tree, the maximum number of AS-hops for any path (15 in our study). Let m be the maximum number of proxy caches placed (50 in our study).

Each AS is visited exactly once to compute its cost matrix. The total number of cost matrices computed is i.

Each cost matrix has K rows. The total number of cost rows computed is $i * K$.

Each of those rows is a combination of the contributions from all of the children of the node. Let δ be the number of children of node v. As previously noted, there will be K rows at node v.

Each of those rows will have $m + 1$ items representing values from 0 to m pebbles. The initial local cost matrix of the parent will be combined δ times with other matrices (once for each child).

After several simple optimizations, our test run with a tree of 21 backbone nodes totaling 6395 nodes had 69,486 row combinations in the 6395 matrix combinations.

The complexity of the algorithm can be estimated for a more general case. For leaves, the algorithm spends $O(d_v) = O(H)$ time per leaf. Thus the total work spent on leaves is at most $O(nH)$. For any non-leaf v, suppose the degree of v is Δ_v, then the computation work spent for v is $O(\Delta_v H m^2)$. Thus the total amount of work spent for non-leaves is $O(\sum_v \Delta_v H m^2) = O(nHm^2)$. Hence the total running time is at most $O(nHm^2)$, which is again only $O(nm^2 \log n)$ with $H \approx O(\log n)$.

Theorem 1 There is a polynomial time algorithm that computes the optimal pebbling placement as well as the optimal cost of the pebbling placement. The running time is $O(nHm^2)$, for any rooted tree of n vertices, height H, and m pebbles. The proof follows from the above discussion.

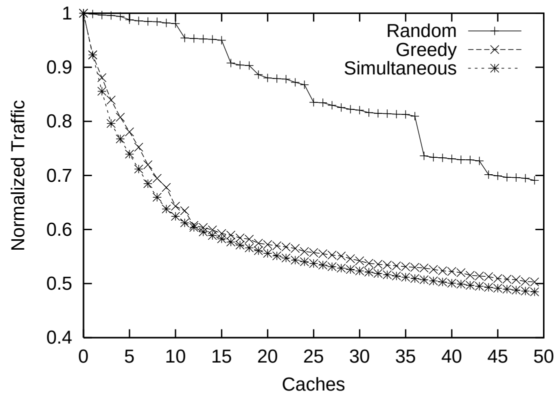

Figure 2.6 Performance versus random and greedy placement

2.5 Evaluation of Cache Placement Impact

To measure the benefit of each new cache added to the tree, we compute the total traffic seen by the sample web server log. Figure 2.6 shows the total traffic normalized to the traffic that would result if 0 caches are used. In our test data, 3.41 Gigabytes of replies came from 790 of the 6395 clusters. Using the tree produced by the clustering algorithm, on average traffic touched 3.07 AS's including the AS at the backbone and the originating AS. The total cost of traffic in this test data was 10.46 Gigabyte-ASHops.

Random Placement

For comparison, we compute costs for a placement algorithm that more closely matches the way caches might be placed opportunistically in a practical case. We randomly chose 50 locations out of the top 200 demand sites. The results in Figure 2.6 show that an occasional good guess causes a noticeable decrease in traffic. In a graph that shows all 200 demand sites (not shown here), the random algorithm took 193 caches to reduce the normalized traffic below 0.62, a level that is a slight knee in the curves for other algorithms. Averaging a number of random runs would smooth the curve, but would be unlikely to lower it.

Greedy Placement

Figure 2.6 also shows the results of a greedy placement algorithm that incrementally places each cache at the hottest remaining site in the forest. Two greedy algorithms were attempted with very similar results. Assume p caches have already been placed. Incremental placement is accomplished for the $(p+1)$ cache by pre-defining locations for the prior p pebbles. The algorithm is then run with only one pebble allocated to the entire Internet. In fact, Figure 2.6 shows an even simpler algorithm to determine the placement of a single, new cache. It chooses the uncached AS with the highest local demand. We were surprised to see how well the greedy algorithms performed and how closely their performance matched each other. The greedy algorithm reduced the total traffic below 0.62 (normalized) by using the 10 AS's with the highest local demand. In fact, the first 11 locations chosen by the greedy algorithm matched the first 11 locations chosen by the simultaneous placement algorithm (albeit in a different order).

Moreover, these incremental placement algorithms (random and greedy) more closely model the financial reality that moving a cache from one location to another is typically not economic.

Simultaneous Placement

Running the dynamic programming algorithm discovered ways to cut the total traffic gigabyte hops by half using 42 caches. This is 10 fewer caches than a greedy placement and it is also a point at which extra caches give little benefit. With 200 caches, the simultaneous placement algorithm was able to reduce the traffic to 4 Gigabyte-ASHops.

Perhaps the greatest benefit of the simultaneous placement algorithm is the shape of the graph. Figure 2.6 clearly shows diminishing returns beyond placing 11 caches. By running the algorithm once, an analyst can see what the optimal result is for the entire range of 0 to m caches and compare the benefits to the cost per cache.

2.6 Incorporating Knowledge of AS Relationships

To validate that our AS forest was accurate, we ran a series of empirical traceroutes. Our hope was that packets traveling between widely separated AS's would hop from AS to AS according to the links in our AS forest. To do that we constructed a utility to send traceroute requests to route servers that were widely dispersed in our AS forest topology. Each traceroute request specifies a destination that is randomly chosen from the entire periphery of our AS forest. We denote the set of route servers as R and the set of destinations as D. A traceroute request sent to route server $r \in R$ specifying destination $d \in D$ would have the resulting path of hops $H_{r,d}$. An element of $H_{r,d}$ is a hop h with a hop number, the IP address of the router reporting the hop, and the round trip time from r to the reporting router. In a subsequent step, we add in the AS number associated with that IP address. The hop numbers on the hops in $H_{r,d}$ increase by one each time the traceroute gets closer to the destination. If the traceroute is a success, the last hop will have the IP address of the intended destination, d.

Converting a traceroute with IP addresses into a traceroute with AS numbers is an imperfect process. For each hop, h, of each traceroute, we translated the router link IP address to an AS number using the centralized BGP table. Our results sometimes skip over an AS because packets are lost, we got no response from the router, or because the router's interface had an IP address that belongs to the AS at the other end of the link. Because of route aggregation and other practical limitations of BGP, our translation from IP address to AS could be wrong as well. Finally, ISP's need not use globally-routable IP addresses for links inside their own domain. If we miss seeing the ingress into the AS, we might completely miss seeing the AS. Thus, our translated AS path might understate the length of the true AS path.

An example traceroute, $H_{r,d}$, is shown in Table 2.2, where r is a route server in Switzerland in AS_{8493} and d is an IP address in Wisconsin inside AS_{59}. The first hop goes to $195.202.193.6$, presumably a border router connecting AS_{8493} to AS_{8404}. Each row of the traceroute is successively closer to the destination, $d = 128.105.2.10$. This traceroute shows that AS_{8493} is able to pass a packet directly to AS_{8404}, even though they differ in depth by two. This is a common occurrence

Table 2.2 Sample AS traceroute

Hop	IP Address	ASN	AS depth	RTT
1	195.202.193.6	AS8493	3	0
2	62.2.154.81	AS8404	1	1
3	62.2.4.222	AS8404	1	4
4	213.242.67.1	AS3356	0	5
5	212.187.128.61	AS3356	0	6
6	212.187.128.138	AS3356	0	6
7	64.159.1.69	AS3356	0	27
8	4.24.164.102	AS1	0	118
9	140.189.8.1	AS2381	1	128
10	146.151.164.50	AS59	2	129
11	128.105.2.10	AS59	2	130

in our forest, and is probably the result of clustering AS_{8493} to a parent that has a higher out-degree and also has a link to AS_{8404}. Note also that the route starts out far from the centroid (hop one is at depth three), travels toward the centroid, reaches the forest floor, and then travels outbound to it's final destination.

Choosing traceroute starting points

For the traceroute starting points, we chose from the list of looking glass sites, traceroute servers and route servers listed at www.traceroute.org. Many of those hosts provide a simple interface that responds to an HTTP GET. The result is often plain text or trivially encapsulated text inside HTML. The results were then parsed by a simple java program at our data collection site. From the www.traceroute.org list of 882 servers we chose a list, R, of 135 servers, each in a different AS, two or more hops from the centroid, that respond to an HTTP GET request with easily parsed HTML.

Choosing traceroute destinations

To construct the traceroute destination set, D, we probed IP addresses to find one representative IP address in each AS. Consider a representative IP address, d. If a local traceroute to that address failed, the last hop in $H_{\ell,d}$ will not be to IP address d. Even when the last hop fails to reach a working IP address, if a prior hop already shows the desired AS, it is a usable AS trace and d can be added into D. We tried 10 more IP addresses by incrementing d in an attempt to find an IP address that would include at least one hop in our desired destination AS. If an AS had more than one net-block of IP addresses, the other net-blocks were also probed. In our case, we were not able to find a suitable IP address in 11% of the AS's.

Over the week of March 11, 2002, we performed 200K traceroutes. Although this number is comparable to other studies [71, 37] and much smaller than one study [9], our study did not need repetitions of the same routes. When we had the traceroute collection fully automated we were careful not to overload any single host with more than one traceroute request per minute. We are grateful to the user community for maintaining traceroute servers and we do not want to abuse their hospitality.

Noting the relationship between AS's

We now improve on the forest constructed in Section 2.2 by annotating hop constraints and by discovering new links that were not present in BGP tables. The annotations we add to each hop along a path let us avoid using links for transit traffic if the ISP paying for the link would be unlikely to allow transit between one of its providers and another of its providers.

The pattern we expected to see in each traceroute was the one identified by Gao [33]: each packet should flow uphill customer to provider, $c \to p$, (or laterally, sibling to sibling, $s \leftrightarrow s$) until it reaches the highest point needed to reach an AS (or a sibling or peer of an AS) upstream of the destination. Then the packet should flow only downhill provider to customer, $p \to c$, until it reaches the destination.

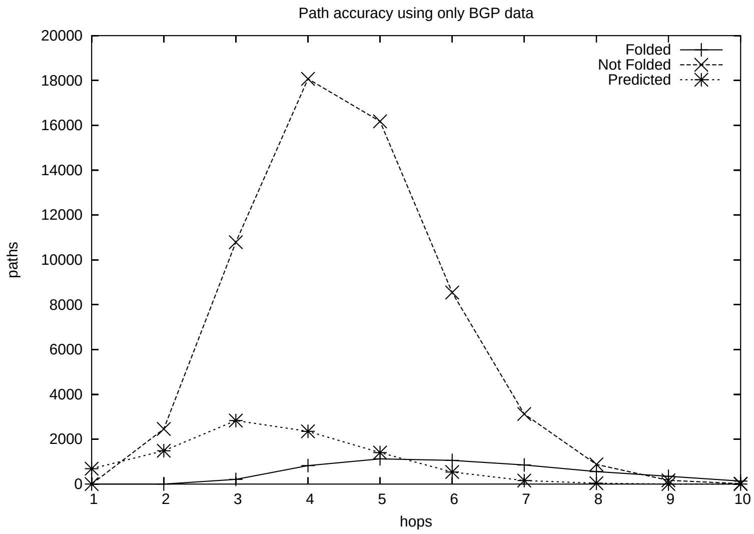

Figure 2.7 Early forest predicted only a tiny portion of the non-folded routes seen by traceroute.

As other researchers previously noted [17, 33], a significant number of AS connections are hidden from most BGP tables. Figure 2.7 shows the results of the 74,963 unique complete traceroutes when applied to the AS forest derived solely from BGP information. The majority of the paths were from 3 to 6 AS hops long. A small number of paths were as long as 12 AS hops and a small number of IP addresses found routing loops at the inter-AS level.

The *folded traces* are the AS paths that appeared to flow uphill after having taken a downhill hop. At that point, our AS forest had only provisional labels to categorize each link as a customer-provider link or a sibling link. The *not folded traces* are the paths that did not violate the uphill-to-downhill laws but contained links not in our AS forest. For a hop from AS_m to AS_n we compare $Depth_m$ to $Depth_n$ in cases where the AS forest did not have a link at (m, n). Finally, the *predicted traces* are paths that only contained AS hops in the AS forest.

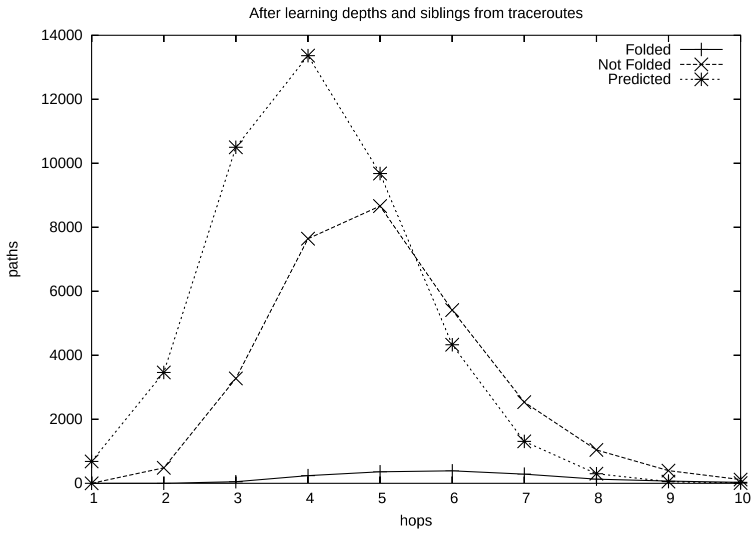

Figure 2.8 Adjusting the annotations in the graph reduced the number of folded (implausible) paths and improved prediction.

Figure 2.8 shows the same paths after the $Depth_n$ values have been refined. In this case, we pause for learning each time a traceroute shows an uphill hop after the packet had already reached a pinnacle. We used a *Current Best Hypothesis* algorithm [59] to test each hop of the traceroute. Imagine a trace $(k, l, ..., m, n)$ in which l was thought to be downhill from k, but n was thought to be uphill from m. This folded trace violates one or more of the annotations we have made. At least one of the links between k and m was annotated $A(k, l)$ as a $p \rightarrow c$ link. Choose k and l to be the closest instance of a $p \rightarrow c$ link. On the evidence of this traceroute, that could be a false positive. Alternatively, $A(m, n)$ was $c \rightarrow p$, preventing us from using it on the downhill side (a false negative). A special case where $l = m$ is easily handled.

To choose the appropriate generalization or specialization, we select the link most refuted by the evidence. That is, we track the failure count $F(m, n)$ and success count $S(m, n)$ of each annotation. If the total evidence $E = F(k, l) + S(k, l) + F(m, n) + S(m, n)$ exceeds a a learning rate threshold, α, we assume that we have seen enough cases to render a judgment. Each link, (k, l), has an error proportion $Err(k, l) = F(k, l)/(F(k, l) + S(k, l))$. If $Err(k, l) > Err(m, n)$ we change (m, n) to $s \leftrightarrow s$ by setting $Depth_m = Depth_n$. Alternatively, if the downhill link was more probably incorrect, we set $Depth_n = Depth_m$. Since we have changed the depth of an AS, we correct all of the annotations of the links to that AS.

The algorithm found exchange points like the Russian Universities Federal Network (AS3267) quickly. $Depth_{3267}$ went from 9 hops from the backbone to 1. Others like the Milan Interconnection Point (AS16004) rose 4 times. Whenever a $Depth_n$ changes, other links become $c \rightarrow p$ or $p \rightarrow c$.

Figure 2.8 shows the results of learning depths. Bars show the average of 10 runs over the same traceroutes using 10-fold cross-validation with $\alpha = 6$. Higher values of α would require a larger data set.

Since this fixed many of our mistakenly labeled customer-provider paths, previously folded paths were now non-folded. Our algorithm had reversed some customer-provider pairs. Also, there were improvements when unidirectional customer-provider links were upgraded to bidirectional sibling links.

Adding learned relatives

In many cases, the traced routes showed links that were not present in our BGP-based AS forest or even the BGP-based AS graph. We decided to add the most recent *alternate parent* to each AS whenever a trace showed an unexpected uphill hop from that AS. We limited the learning to identifying a single alternate parent for each AS. If we saved all of the alternate parents, the program would eventually have learned all of the routes seen, but the number of “correct” paths from one AS to another would grow too fast. This would have made our subsequent service placement algorithm ineffective. We placed no limit on the number of learned siblings at the same $Depth_n$.

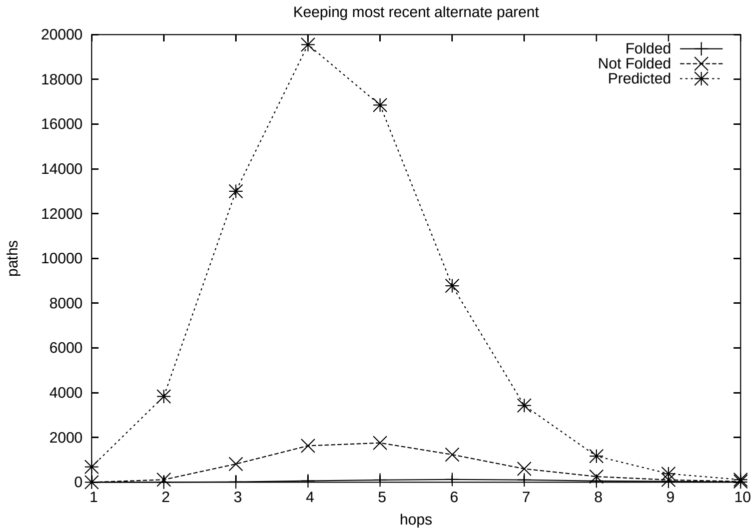

Figure 2.9 Results with final AS forest

Figure 2.9 shows the results of allowing each node in the AS forest a list of siblings and a single, alternate uphill link. We considered more sophisticated techniques for discovering the best of the discovered links, but were satisfied that the simplest technique (saving the most recent) was effective and reacted well dynamically. Again, the results are the average of 10-fold cross validation with training sets of 67,467 traces and test sets of 7,496 traces. Over 91% of the test set traces correctly followed the uphill-then-downhill pattern and were composed only of links contained in our AS graph. Links with 5 or more AS hops had noticeably higher error rates.

Now that the AS forest can credibly predict the path of traceroutes, we return to the service placement problem to see how the addition of alternate parents affects the dynamic programming problem.

2.7 Clustering Study Summary

In this chapter we have described methods for creating AS clusters based on BGP routing data. The algorithm for creating a forest of AS numbers objectively discovers the AS's that form a highly interconnected backbone for the Internet. The resulting forest slightly overstates the average number of hops from any point in the Internet to a common backbone, but is close enough to allow the study of client demand and cache placement.

We have also presented a new, optimal method for placing caches in the AS hierarchy generated by our clustering method. We compared the effectiveness of our algorithm to two incremental techniques using a commercial Web log. We found that greedy placement of caches worked nearly as well as the sophisticated, optimal technique when the number of caches was small or large.

Finally, this chapter presented a new methodology for annotating the inter-AS links to identify customer-to-provider links and treat them appropriately when predicting packet travel. An important discovery was the need to allow for one alternate parent for each AS to achieve acceptable accuracy. This makes the AS-level graph more complex, but still much more succinct than the full graph with little loss of accuracy.

Future Clustering Work

An important improvement in the topology would be annotations indicating the capacity and propagation delay of each link. The current topology considers an entire AS to be a single node. This is inaccurate when there are a large number of geographically dispersed routers in a single AS. A trip across a particular AS might be arbitrarily short or it may be trans-continental or trans-oceanic. The traceroutes used to validate the topology could also be harvested to determine which links are long. An algorithm could be developed to separate each large AS into as many smaller units as can be realistically differentiated. This approach requires that IP net blocks be used as sources and destinations rather than AS numbers. The result would be a topology that would contain long links as well as inter-AS links, and therefore, contain 80% of the links on which losses occur. More research would be needed to assess the typical delay, jitter, and loss rate for each link.

Moreover, the nodes could then be associated with an interior buffering capacity (adding to jitter). The resulting topology would be useful for capacity planning and quality of service studies.

The clustering algorithm could be made more general by varying the size of the centroid used as the forest floor. The current choice to make the centroid very small (the 21 roots of the trees in section 2.2) was done to accommodate visualizations. We believe that other studies (e.g. losses, route stability, or jitter) would be better served by a much larger centroid containing the bulk of the professionally-managed tiers of the global Internet.

Chapter 3

Large Scale Simulation of Congested Behaviors

In Chapter 2 we developed a concise, accurate graph of the Internet that naturally lends itself to analysis. In this chapter, we investigate traffic congestion with such a graph. Most of the traffic has to travel across multiple hops. Many of those connections have long round trip times. Our approach is to aggregate large numbers of connections into just a few equivalence classes so that we can analyze traffic patterns at a macroscopic level. This poses a problem. What parameters need to be captured to characterize a collection of flows? In this chapter, we show that volume alone is not enough to characterize the way connections (and, ultimately, collections of connections) react to congestion.

This chapter chronicles a succession of simulations that led to the formation of a concise model of congestion events. The model will be shown to accurately predict the proportion of time a heavily congested link actually presents no queuing delay at all. Graphs produced by packet-level simulations are compared to model output for validation. Our conclusion is that two new parameters, RTT and ceiling, are important inputs to the function that determines how collections of connections react to congestion. These are similar to parameters identified by the end-to-end community to model the effect of a multi-hop interior on the individual flows.

The collection of connections with a common reaction will be referred to as a *flock*. We investigate aspects of flock formation and behavior. A discussion shows how connections with similar RTT and a shared bottleneck can fall into resonant cadence. In this case the resonance is referred to as window synchronization, and it helps us measure the extent to which congestion events are successful. The notion of using RTT and a ceiling to characterize an individual connection was well documented by Padhye [65] along with a closed form for the end-to-end case. We investigate it hop-by-hop. Moreover, we extend our analysis of window synchronization to include a collection of many connections with similar RTT.

In Chapter 4 we will try to infer the values of these important parameters from measurements that can be taken at the edges of an ISP. Unlike traditional traffic matrix estimation, our traffic matrix will incorporate these extra parameters for each flock.

3.1 Simulating Congestion and the Effect on Traffic

Much of the research in network congestion control has been focused on the ways in which transport protocols react to packet losses. Prior analyses were frequently conducted in simulation environments with small numbers of competing flows, and along paths that have a single low bandwidth bottleneck. In contrast, modern routers deployed in the Internet easily handle thousands of simultaneous connections along hops with capacities above a billion bits per second.

Packet dropping (seen by the intended recipient as a packet loss) is a simple mechanism for signaling congestion. As each packet travels through consecutive links toward its final destination, it may be competing with many other packets for space on links. If, in the aggregate, p_i packets arrive during a interval in which the capacity of the link is smaller, the excess packets are enqueued in buffers on the ingress router. If the queue continues to grow in subsequent intervals, it may get backlogged enough that the router decides to ask connections to slow down. In the simplest case, *drop tail*, if the queue is full at the moment a packet arrives, the packet is dropped. If the packet loss is detected by the anticipated recipient, a flow control indication can be sent to the connection's sender to tell it to slow down. The seminal work on congestion avoidance is Jacobson's Congestion Avoidance and Control [40]. It tells the story of how a link from LBL to UC-Berkeley plummeted from 32 Kbps to a mere 40 bps during an episode of congestive collapse. The problem was that senders responded to a packet loss by flooding the network with another copy of that and all subsequent packets in a transmission window. Jacobson goes on to outline a set of principles for *conservation of packets* in which a new packet is not put into the network until an old packet leaves. The goal is to discover a sending rate, λ, that will match the bandwidth delay product of the path. Each ACK packet received by the sender *clocks out* a new data packet. For a mature connection that has already discovered a bandwidth delay product, TCP occasionally probes to see if it could increase λ. It does this by adding one more packet once per RTT, effectively performing additive increase on λ. When λ grows too large for this connection's share of a bottleneck link, the router feeding that link will drop one or more packets. When the sender fails to receive an acknowledgment of that packet within a reasonable time-frame (based on an estimate of the RTT), the sender reduces λ to $\lambda/2$, multiplicative decrease.

Since packet loss is still the major mechanism for communicating congestion from the interior of the network, characteristics of losses and bursts of losses remain important. Poisson models of traffic initiation were tried and rejected [73, 31]. Fractals or Self-Similarity [51, 25, 26] have been exploited for their ability to explain Internet traffic statistics. These models show that large timescale traffic variability can arise from exogenous forces (the composition of the network traffic that arrives) rather than just endogenous forces (reaction of the senders to feedback given to them from the interior).

Traffic engineering tradition has been to size links to accommodate mean load plus a factor for large variability. The problem comes in estimating the large variability. Cao *et al.* [13] provides ways to estimate this variability and suggests that old models do not scale well when the number-of-active-connections (NAC) is large. As NAC increases, packet inter-arrival times will tend toward independence. In particular, that study divides time up into equal-length, consecutive intervals and watches p_i, the packet counts in interval i. In that study, the coefficient of variation (standard deviation divided by the mean) of p_i goes to zero like $\frac{1}{\sqrt{NAC}}$. The Long Range Dependence (LRD) of the p_i is unchanging in the sense that the autocorrelation is unchanging, but as NAC increases, the variability of p_i becomes much smaller relative to the mean. In practical terms, links utilization of 50% to 60% average measured over a 15 to 60 minute period is considered appropriate [12] for links with average NAC equal to 32. Cao's datasets include a link at OC-12 (622 Mbps) with average NAC above 8,000. Clearly, traffic engineering models that implicitly assume NAC values below 32 are inappropriate for fast links.

The model presented in this chapter is a purely endogenous view. For simplicity, it only explores oscillations caused by the reactions of sources to packet marking or dropping. Each time a packet is dropped (or marked), the sender of that packet cuts his sending rate (congestion window, cWnd) using multiplicative decrease. Because there is an inherent delay while the feedback is in transit, a congested link may have to give drops (or marks) to many senders. If the congestion was successfully eliminated, connections are likely to enjoy a long loss-free period and will grow their cWnd using additive increase. If the connections grow and shrink their cWnd in synchrony, the global synchronization is referred to as window synchronization [93].

The most significant effort to reduce oscillations caused by synchronization is Random Early Detection (RED) [28]. RED tries to break the deterministic cycle by detecting incipient congestion and dropping (or marking) packets probabilistically. On slow links, this effectively eliminates global synchronization [56]. But a comprehensive study of window synchronization on fast links has not been made.

Key to understanding window synchronization is an understanding of the congestion events themselves. One objective in this chapter is to develop a mechanism for investigating the duration, intensity and periodicity of congestion events. Our model is based on identifying distinct portions of a congestion event, predicting the shape of congestion events and the gap between them. Our congestion model is developed from the perspective of queue sizes during congestion events that have a shape we call a “shark fin”. Packets that try to pass through a congested link during a packet dropping episode are either dropped or placed at the end of an (almost) full queue. While this shape is familiar in both analytical and simulation studies of congestion, its characteristics in measurement studies have not been reported.

The validation of these effects required highly accurate one-way delay measurements taken during a four month test period with a wide geographic scope. We use data collected with the Surveyor infrastructure [77] to show evidence that shark fins exist in the Internet. There are distinct spikes at very specific queue delay values that only appear on paths that pass through particular links.

Next, we explored the implications of regular spacing between congestion events. Connections shrink their congestion windows (cWnd) in cadence with the congestion events. The cWnd’s slowly grow back between events. In effect, the well-known saw-tooth graphs of cWnd [87] for the individual long-lived connections are brought into phase with each other, forming a "flock", a set of connections whose windows are synchronized. Window synchronization has been studied, but we document flocks that span a larger range of round trip times than previously reported [29].

From the viewpoint of a neighboring link, a flock will offer an aggregate load that rises together. When it reaches a ceiling (at the original hop) the entire flock will lower its cWnd together. We believe flocking can be used to explain synchronization of much larger collections of connections than any prior study of synchronization phenomena.

The simulations in this chapter use infinitely long-lived TCP connections. Actual traffic includes a mixture of short and long-lived connections along with other traffic that is not controlled by any congestion avoidance. Non-responsive connections do not slow down in response to losses. There are also constant bit-rate sources (like Internet radio or video conferencing) that neither speed up nor slow down in the presence of losses. We chose to avoid this complexity on the presumption that traffic can be divided into connections that remember the prior congestion event versus *uncontrolled* traffic that does not. We depend on the independence assumption to assert that the uncontrolled traffic adds to the mean but that it's contribution to the variance of p_i becomes very small relative to the mean at values of NAC found in gigabit links. Our findings would still apply after subtracting the effect of uncontrolled traffic.

Explicit Congestion Notification (ECN) [81] promises to significantly reduce the delay caused by congestion feedback. We will assume that marking a packet is equivalent to dropping that packet. In either case, the sender of that packet will (should) respond by slowing down. Whenever we refer to dropping a packet, marking a packet would be preferable because it does not require retransmission and does not disrupt the steady pacing of packets arriving and generating ACKs to clock out new data packets.

We investigate a spectrum of congestion issues related to our model in a series of ns2 [89] simulations. We explore the accuracy of our model over a broad range of offered loads, mixtures of RTT's, and multiplexing factors. Congestion event statistics from simulation are compared to the output of the model and demonstrate an improved understanding of the duration of congestion events.

The strength of this model is that it easily scales to paths with multiple congested hops and the interactions between traffic that comes from distinct congestion areas. Extending the model to large networks promises to give better answers to a variety of traffic engineering problems in capacity planning, performance analysis and latency tuning.

The rest of this chapter is organized as follows. In Section 3.2, we present the Surveyor data that enabled our empirical evaluation of queue behavior. Section 3.3 introduces the notion of an aggregate window for a group of connections and shows how the aggregate reacts to a single congestion event. Section 3.4 presents ns2 simulations that show how window synchronization can bond many connections into flocks. Each flock then behaves as an aggregate and can be modeled as a single entity. In Section 3.5, we present our model that accurately predicts the interactions of multiple flocks across a congested link. Outputs include the queue delays, congestion intensities and congestion durations. Sample applications in traffic engineering are enumerated. Section 3.6 presents our conclusions and suggests future work in this topic. In the chapter on related work, Section 5.2 discusses related work relevant to this chapter.

3.2 Surveyor Data: Looking for Characteristics of Queuing

Empirical data for this study was collected using the Surveyor [77] infrastructure. Surveyor consists of 60 nodes placed around the world in support of the work of the IETF IP Performance Metrics Working Group [39]. The data we used is a set of active one-way delay measurements taken during the period from 3-June-2000 to 19-Sept-2000. Each of the 60 Surveyor nodes maintains a measurement session to each other node. A session consists of an initial handshake to agree on parameters followed by an long stream of 40 byte probes at random intervals with a Poisson distribution and a mean interval between packets of 500 milliseconds. The packets themselves are Type-P UDP packets of 40 bytes. The sender emits packets containing the GPS-derived timestamp along with a sequence number. See RFC 2679 [4]. The destination node also has a GPS and records the one-way delay and the time the packet was sent.

Each probe's time of day is reported precise to 100 microseconds and each probe's delay is accurate to $\pm$ 50 microseconds. Data is gathered in sessions that last no longer than 24 hours.

The delay data are supplemented by traceroute data using a separate mechanism. Traceroutes are taken in the full mesh approximately every 10 minutes. For this study, the traceroute data was used to find the sequence of at least 100 days that had the fewest route changes.

Deriving Propagation Delay

The Surveyor database contains the entire delay seen by probes. Before we can begin to compare delay times between two paths we must subtract propagation delay fundamental to each path. For each session, we assume that the smallest delay seen by that session is the propagation delay between source and destination along that route. Any remaining delay is assumed to be queuing delay. Sessions were discarded if traceroutes changed or if any set of 500 contiguous samples had a local minimum that was more than 0.4 ms larger than the propagation delay. The presumption here is that the minimum one-way delay for any set of 500 contiguous samples will be the propagation delay. If the minimum changed, then the propagation delay probably changed. Since the granularity of traceroutes (one per 10 minutes) was so much larger than the spacing between packets

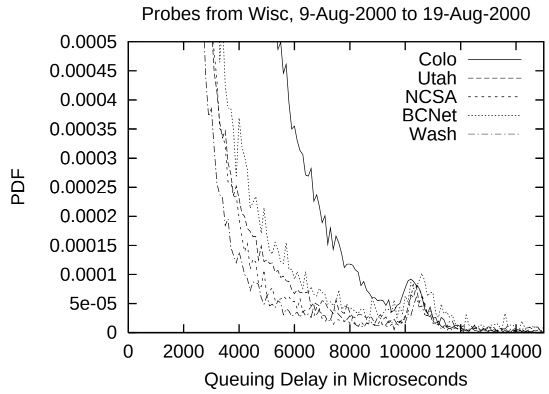

Figure 3.1 Probability density of queuing delays of 5 paths

(500 milliseconds average), we felt we needed to track changes in propagation delay to accurately discard any packets near a route change.

Peaks in the Queuing Delay Distribution

Figure 3.1 shows the PDF of a variety of paths with a common source. They all share one OC-3 interface (155 Mbps) at the beginning and have little in common after that. The Y-axis of this graph represents the number of probes that experienced the same one-way delay value (adjusted for propagation delay). Counts are normalized so that the size of the curves can be easily compared. Each histogram bin is 100 microseconds of delay wide.

Our conjecture was that a full queue in the out-bound link leaving that site was 10.3 milliseconds long, and that probes were likely to see almost empty queues (outside of congestion events) and almost full queues (during congestion events).

Figure 3.2 is included here to put the PDF in context. The cumulative distribution function (CDF) shows that the heads of these distributions differ somewhat. The paths travel through different numbers of queues and those routers have different average queue depths and link speeds. But 99% of the queue delay values are below 5 ms. From the CDF alone, we would not have suspected that the PDF showed peaks far out on the tail that were similar width and height.

Figure 3.3 shows that a distinctive peak in the PDF tail is a phenomenon that is neither unique nor rare. These paths traverse many congested hops, so there is more than one peak in their queuing

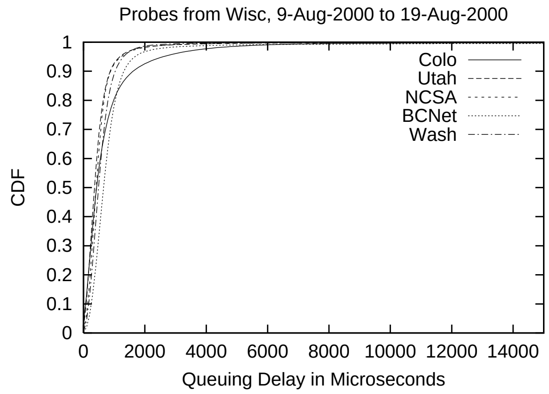

Figure 3.2 Cumulative distribution of queuing delays experienced along the 5 paths.

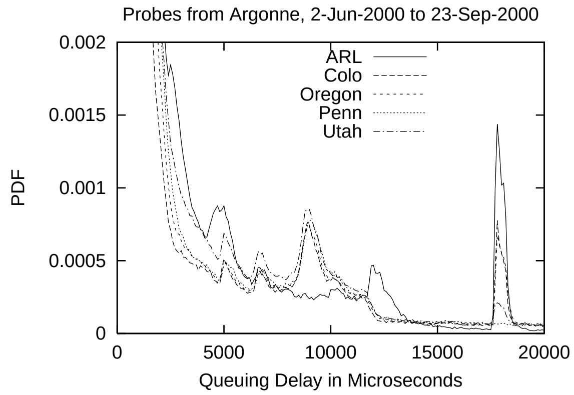

Figure 3.3 Probability density of queuing delays on 5 paths that share a long prefix with each other.

delay distribution. The path from Argonne to ARL clearly shows that it diverges from the other paths and does pass through the congested link whose signature lies at 9.2 ms. Note that these paths from Argonne do not show any evidence of the peak shown in figure 3.1, presumably because they do not share the congested hop that has that characteristic signature.

Other Potential Causes Of Peaks

Peaks in the PDF might be caused by measurement anomalies other than the congestion events proposed in this chapter. Hidden (non-queuing) changes could come from the source, the destination, or along the path. Path hidden changes could be caused by load balancing at layer 2. If the load-balancing paths have different propagation delays, the difference will look like a peak. ISPs could be introducing intentional delays for rate limiting or traffic shaping. There could be delays involved when link cards are busy with some other task (e.g. routing table updates, called the coffee break effect [68]). Our data does not rule out the possibility that we might be measuring some phenomenon other than queuing delay, but our intuition is that those phenomenon would manifest themselves as slopes or plateaus in the delay distribution rather than peaks.

Hidden source or destination changes could be caused by other user level processes or by sudden changes in the GPS reported time. For example, the time it takes to write a record to disk could be several milliseconds by itself. The Surveyor software is designed to use non-blocking mechanisms for all long delays, but occasionally the processes still see out-of-range delays. The Surveyor infrastructure contains several safeguards that discard packets that are likely to have hidden delay. For more information see [44].

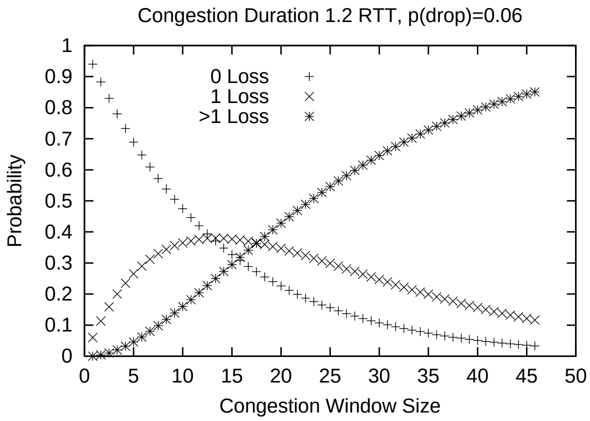

Figure 3.4 Showing the probability of losing 0, exactly 1, or more than one packet in a single congestion event as a function of cWnd.

3.3 Window Size Model

We construct a cWnd feedback model that predicts the reaction of a group of connections to a congestion event. This model simplifies an aggregate of many connections into a single flock and predicts the reaction of the aggregate when it passes through congestion.

Assume that a packet is dropped at time t_0. The sender will be unaware of the loss until one reaction time, R, later. Let C be the capacity of the link. Before the sender can react to the losses, (C / R) packets will depart. During that period, packets are arriving at a rate that consistently exceeds the departure rate. It is important to note that the arrival rate has been trained by prior congestion events. If the arrival rate grew slowly, it has reached a level only slightly higher than the departure rate. For each packet dropped, many subsequent packets will see a queue that has enough room to hold one packet. This condition persists until the difference between the arrival rate and the departure rate causes another drop.

Figure 3.4 shows the probability that a given connection will see ℓ losses from a single congestion event. This example graph shows the probabilities when passing packets through a congestion event with 0.06 loss rate, L. Here R is assumed to be 1.2 RTT. Each connection with a congestion window, W, will try to send W packets per RTT through the congestion event. We now compute the post-event congestion window, W'.

With probability $p(NoLoss)$, a connection will lose no packets at all. Its packets will have seen increasing delays during queue buildup and stable delays during the congestion event. Their ending W' will be $W + R/RTT$. This observation contrasts with analytic models of queuing that assume all packets are lost when a queue is "full".

With probability $p(OneLoss)$ a connection will experience exactly 1 loss and will back off. The typical deceleration makes W' be $W/2$.

With probability $p(Many)$, a connection will see more than one loss. In this example, a connection with $cWnd$ 40 is 80% likely to see more than one loss. Some connections react with simple multiplicative decrease (halving their congestion window). TCP Reno connections might think the losses were in separate round trip times and cut their volume to one fourth. Many connections (especially connections still in slow start) completely stop sending until a coarse timeout. For this model, we simply assume W' is $W/2$.

If an aggregate of many connections could be characterized with a single cWnd, W, a reaction time, R, and a single RTT, the aggregate would emerge from the congestion event with cWnd W'.

$$W' = p(NoLoss) * (W + \frac{R}{RTT}) + (p(OneLoss) + p(Many)) * \frac{W}{2}$$

This change in cWnd predicts the new value after the senders learn that congestion has occurred. In section 6, we will incorporate a simple heuristic to include a factor that represents the quiet period if the losses were heavy enough to cause coarse timeouts.

3.4 Congestion Events and Flock Formation

We use a series of $ns2$ simulations to understand congestion behavior details. The simulations use infinite sources constantly providing data using TCP New Reno for flow control.

One Hop Simulation

We begin with a simulation of the widely used dumbbell topology to highlight the basic features of our model. All of the relevant queuing delay occurs at a single hop. There are 155 connections competing for a 155 Mbps link. We use infinitely long FTP sessions with packet size 1420 bytes and a dedicated 2 Mbps link to give them a ceiling of 2 Mbps each. To avoid initial synchronization, we stagger the FTP starting times among the first 10 ms. End-to-end propagation delay is set to 50 ms. The queue being monitored is a 500 packet drop-tail queue feeding the dumbbell link.

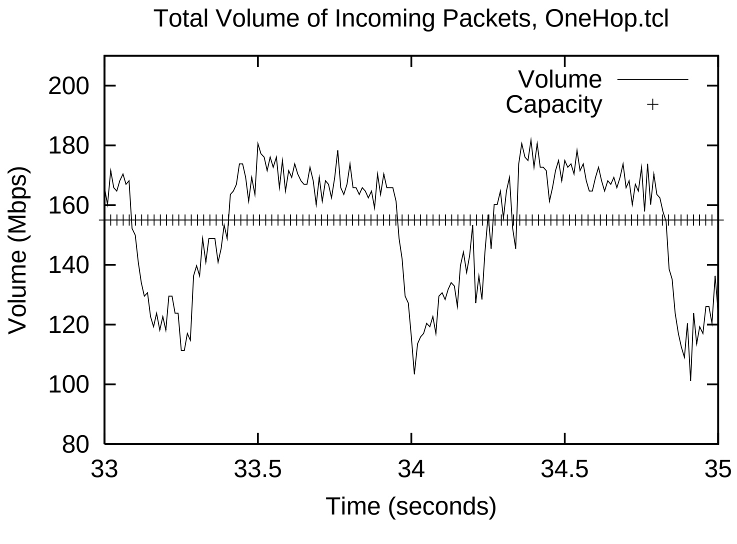

Figure 3.5 Ingress Traffic in One Hop Simulation

Portions of the Shark Fin

Figure 3.5 shows two and a half complete cycles that look like shark fins. Our model is based on the distinct sections of that fin:

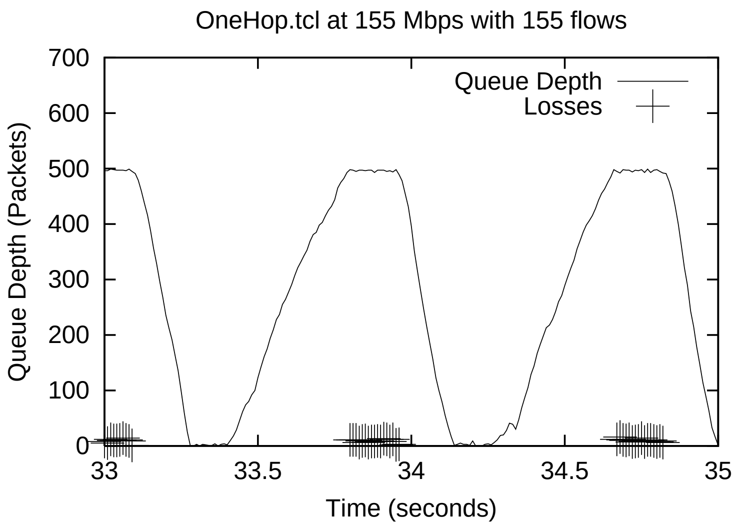

Figure 3.6 Queue Rise and Fall in One Hop Simulation

- **Clear**: While the incoming volume is lower than the capacity of the link, Figure 3.6 shows a cleared queue with small queuing delays. Because the graph here looks like grass compared to the delays associated with congestion, we refer to the queuing delays as "grassy". This situation persists until the total of the incoming volumes along all paths reaches the outbound link's capacity.

- **Rising**: Clients experience increasing queuing delays during the "rising" portion of Figure 3.6. The shape of this portion of the curve is close to a straight line (assuming acceleration is small compared to volume). The "rising" portion of the graph has a slope that depends on the acceleration and a height that depends on the queue size and queue management policy of the router.

- **Congested**: Drop-tail routers will only drop packets during the congested state. This portion of Figure 3.6 has a duration heavily influenced by the average reaction time of the flows.

  Because the congested state is long, many connections had time to receive negative feedback (packet dropping). Because the congested state is of relatively constant duration, the amount of negative feedback any particular connection receives is relatively independent of the multiplexing factor, outbound link speed, and queue depth. The major factor determining the number of packets a connection will lose is its congestion window size.

- **Falling**: After senders react, the queue drains. If an aggregate flow contains many connections in their initial slow start phase, those connections will, in the aggregate, show a quiet period after a congestion event. During this quiet period, many connections have slowed down and a significant number of connections have gone completely silent waiting for a timeout.

PDF of Queuing Delay for One Hop Simulation

Figure 3.7 shows the PDF of queue depths during the One Hop simulation. This graph shows distinct sections for the grassy portion (d_0 to approximately d_{50}), the sum of the rising and falling

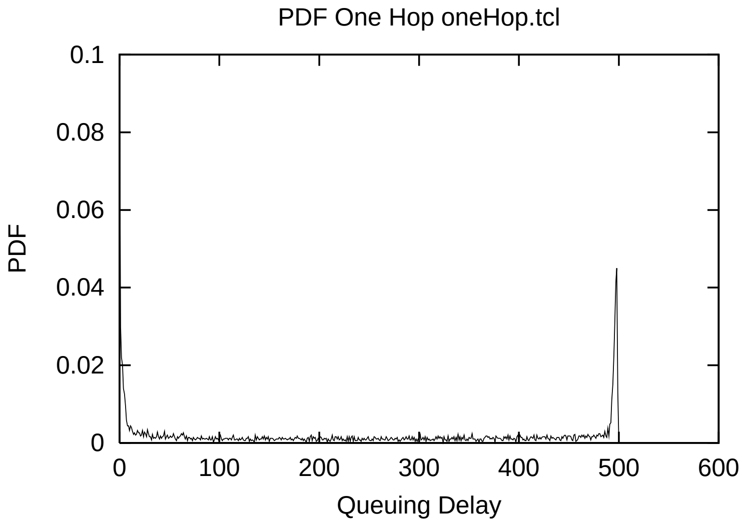

Figure 3.7 Probability of a Given Queuing Delay in the One Hop Simulation

portions (histogram bars for the equi-probable values from d_{50} to d_{480}) and the point mass at 36.65 ms when the delay was the full 500 packets at 1420 bytes per packet feeding a 155 Mbps link.

By adding or subtracting connections, changing the ceiling for some of the traffic or introducing short-term connections, we can change the length of the period between shark fins, the slope of the line rising toward the congestion event, or the slope of the falling line as the queue empties. But the basic shape of the shark fin remains over a surprisingly large range of values and the duration of intense packet dropping (the congestion event) remains most heavily influenced by the average round trip time of the traffic.

Two Hop Simulation

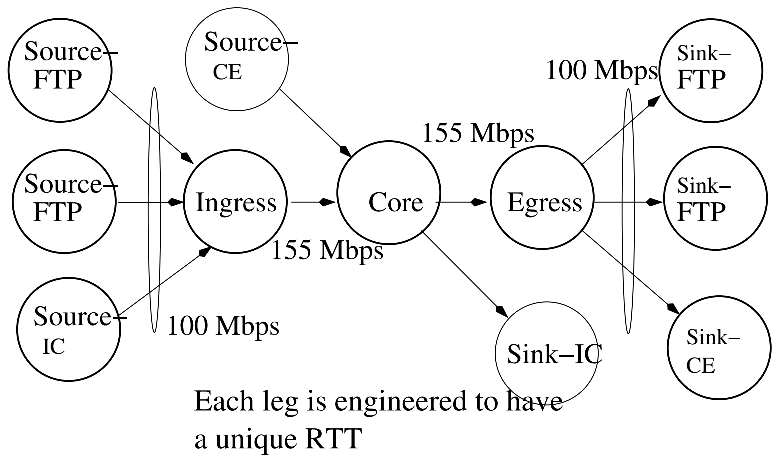

Figure 3.8 Simulation layout for two-hop traffic

To further refine our model and to understand our empirical data in detail, we extend our simulation environment to include an additional core router between the ingress and the egress as shown in figure 3.8. Both links are 155 Mbps and both queues are drop-tail. To make it easy to distinguish between the shark fins, the queue from ingress to core holds 100 packets but the queue from core to egress holds 200 packets. Test traffic was set to be 15 long-term connections passing through ingress to egress. We also added cross traffic composed of both web traffic and longer connections. The web traffic is simulated with NS2's PagePool application WebTraf. The cross traffic introduced at any link exits immediately after that link.

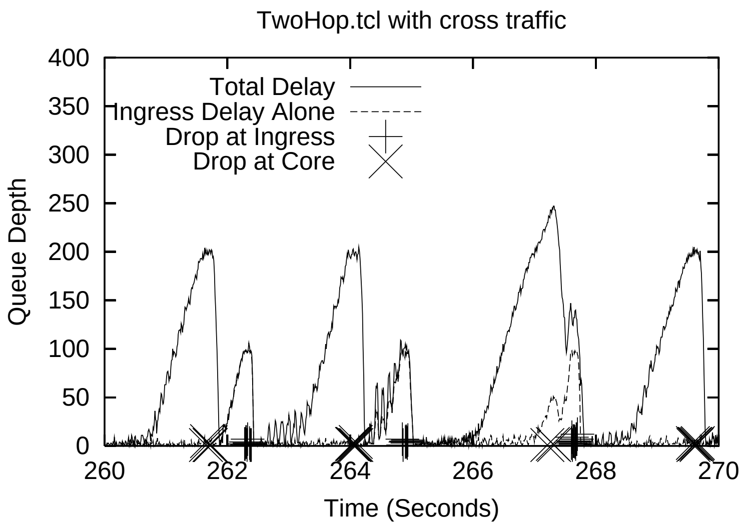

Figure 3.9 Both signatures appear when queues of size 100 and 200 are used in a 2-hop path.

Figure 3.9 shows the sum of the two queue depths as the solid line. Shark fins are still clearly present and it is easy to pick out the fins related to congestion at the core router at queue depth 200 as distinct from the fins that reach a plateau at queue depth 100.

The stars along the bottom of the graph are dropped packets. Although the drops come from different sources, each congestion event maintains a duration strongly related to the reaction time of the flows. In this example, one fin (at time t_{267}) occurred when both the ingress and core routers were in a rising delay regime. Here the dashed mid-delay line shows the queue depth at the ingress router. At most other places, the mid-delay is either very nearly zero or very nearly the same as the sum of the ingress and core delays.

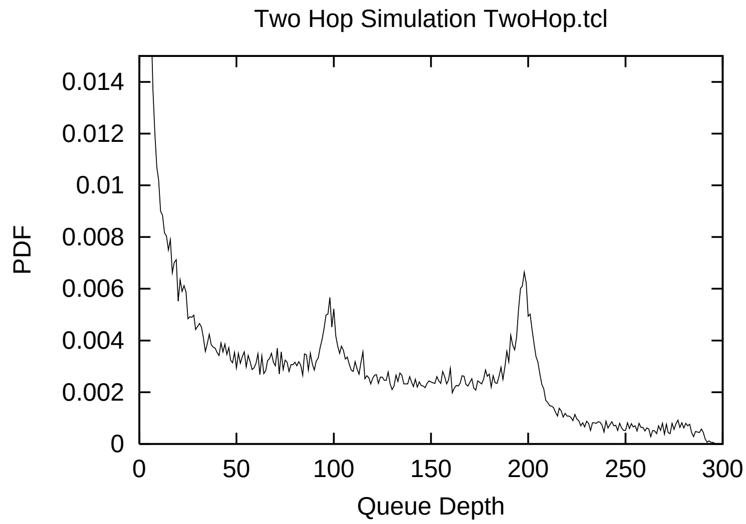

Figure 3.10 The distinctive signature of each queue shows up as a peak in the PDF.

Figure 3.10 shows the PDF of queue delays. Peaks are present at a queue depths of 100 packets and 200 packets. This diagram also shows a much higher incidence of delays in the range 0 to 100 packet times due to the cross traffic and the effect of adding a second hop. In terms of our model, this portion of the PDF is almost completely dictated by the packets that saw grassy behavior at both routers. The short, flat section around 150 includes influences from both rising regime at the ingress and rising regime at the core. The falling edges of shark fins were so sharp in this example that their influence is negligible. The peak around queue size=100 is 7.6 ms. It is not as sharp as the One Hop simulation in part because its falling edge includes clear delays from the core router.

For example, a 7.6 ms delay might have come from 7.3 ms spent in the ingress router plus 0.3 ms spent in the core. The next flat area from 120 to 180 is primarily packets that saw a rising regime at the core router. A significant number of packets (those with a delay of 250 packet times, for example) were unlucky enough to see rising regime at the ingress and congestion at the core or a rising regime at the core and congestion at the egress.

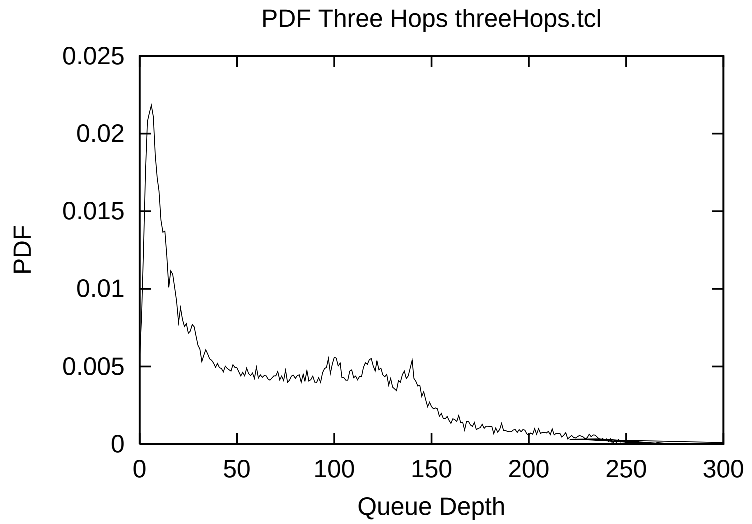

Figure 3.11 Three hop simulation shows three distinct peaks

Flocking

In the absence of congestion at a shared link, individual connections would each have had their own saw-tooth graph for cWnd. A connection's cWnd (in combination with it's RTT) will dictate the amount of load it offers at each link along it's path. Each of those saw-tooth graphs has a ceiling, a floor, and a period. Assuming a mixture of RTT's, the periods will be mixed. Assuming independence, each connection will be in a different phase of its saw-tooth at any given moment. If N connections meet at an uncongested link, the N saw-tooth graphs will sum to a comparatively flat graph. As N gets larger (assuming the N connections are independent) the sum will get progressively flatter.

During a congestion event, many of the connections that pass through the link receive negative feedback at essentially the same time. If (as is suggested in this chapter) congestion events are periodic, that entire group of connections will tend to reset to their lower cWnd in cadence with the periodic congestion events. Connections with saw-tooth graphs that resonate with the congestion events will be drawn into phase with it and with each other.

Contrast this with another form of global synchronization reported by Keshav, et al. [80] in which all connections passing through a common congestion point regardless of RTT synchronize. The Keshav study depends on the buffer (plus any packets resident in the link itself) being large enough to hold 3 packets per connection. In that form, increasing the number of connections would eliminate the synchronization. Window synchronization theory does not depend on large buffers or slow links, but rather it depends on a mixture of RTTs that are close enough to be compatible.

Flock Formation

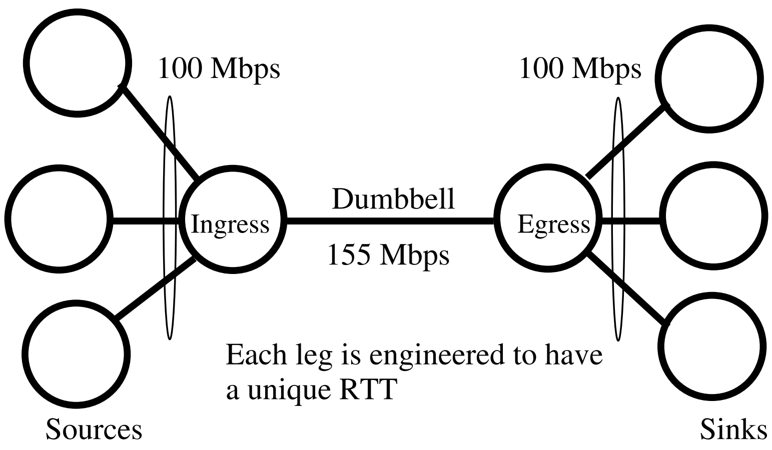

Figure 3.12 Simulation environment to foster window synchronization.

To demonstrate a common situation in which cWnd sawtooth graphs fall into phase with each other, we construct the dumbbell environment shown in Figure 3.12. Each of the legs feeding the dumbbell runs at 100 Mbps, while the dumbbell itself is a 155 Mbps link.

We give each leg entering the ingress router a particular propagation delay so that all traffic going through the first leg has a Round Trip Time of 41 ms. The second leg has traffic with

RTT 47 ms, and the final leg has traffic at 74 ms RTT. We wanted to use a range of values that represented regional round trip times that had no simple common factor.

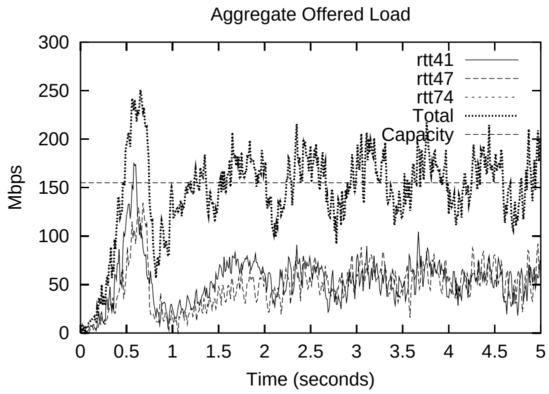

Figure 3.13 Connections started at random times synchronize cWnd decline and buildup after 2 seconds.

Figure 3.13 shows the number of packets coming out of the legs and the total number of packets arriving at the ingress. Congestion events happen at 0.6 sec, 1.3 sec and 1.8 sec. As a result of those congestion events, almost all of the connections, regardless of their RTT, are starting with a low cWnd at 2.1 seconds. After that, the dumbbell has a congestion event every 760 milliseconds, and the traffic it presents to subsequent links rises and falls at that cadence.

Not shown is the way in which packets in excess of 155 Mbps are spread (delayed by queuing) as they pass through. The flock at RTT 74 ms is slow to join the flock, but soon falls into cadence at time $t_2.1$. Effectively, the load the dumbbell passes on to subsequent links is a flock, but with many more connections and a broader range of RTT's.

Range of RTT Values in a Flock

Next we investigate how RTT values affect flocking. We use the same experimental layout shown in Figure 3.12 except that a fourth leg has been added that has an RTT too long to participate in the flock formed at the dumbbell. Losses from the dumbbell come far out of phase with the range that can be accommodated by a connection with a 93 millisecond RTT.

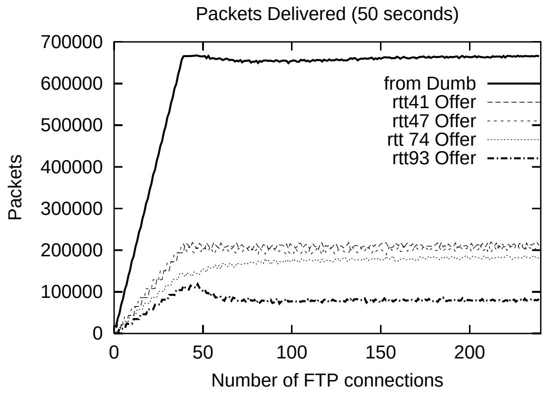

Figure 3.14 Connections with RTT slightly too long to join flock.

Figure 3.14 shows the result of 240 simulation experiments. Each run added one connection in round-robin fashion to the various legs. When there is no contention at the dumbbell, each connection gets goodput limited only by the 100 Mbps leg. The graph plots the total goodput and the goodput for each value of RTT.

The result is that the number of packets delivered per second by the 93 ms RTT connections is only about half that of the 74 ms group. In some definitions of fair distribution of bandwidth, each connection would have delivered the same number of packets per second, regardless of RTT.

This phenomenon is similar to the TCP bias against connections with long RTT reported by Floyd, et al. [27], but encompasses an entire flock of connections.

It should also be noted that turbulence at an aggregation point (like the dumbbell in this example) causes incoming links to be more or less busy based on the extent to which the traffic in the leg harmonizes with the flock formed by the dumbbell. In the example in Figure 3.14, the link carrying 93 millisecond RTT traffic had a capacity of 100 Mbps. In the experiments with 20 connections per leg (80 connections total), this link only achieved 21 Mbps. Increasing the number of connections did nothing to increase that leg's share of the dumbbell's capacity.

Formation of Congestion Events

The nature of congestion events can be most easily seen by watching the amount of time spent in each of the queuing regimes at the dumbbell. We next examine the proportion of time spent in each portion of the shark fin using the same simulation configuration as in the prior section.

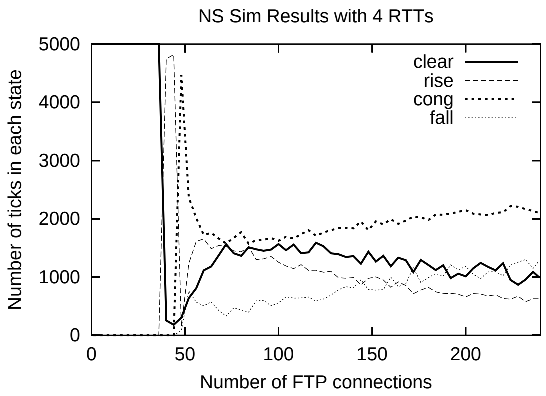

Figure 3.15 Proportion of time spent in each queue regime.

Figure 3.15 shows the proportion of time spent in the clear (no significant queuing delay), rising (increasing queue and queuing delay), congested (queuing delay essentially equal to a full queue), and falling (decreasing queue and queuing delay).

When there are fewer than 40 connections, the aggregate offered load reaching the dumbbell is less than 155 Mbps, and no packets need to be queued. There is a fascinating anomaly from 40 to 45 connections that happens in the simulations, but is likely to be transient in the wild. In this situation, the offered load coming to the dumbbell is reduced (one reaction time later) by an amount that exactly matches the acceleration of the TCP window growth during the reaction time. This results in a state of continuous congestion with a low loss rate. We believe this anomaly is the result of the rigidly controlled experimental layout. Further study would be appropriate.

As the number of connections builds up, the queue at the bottleneck oscillates between congested and grassy. In this experiment, the dumbbell spent a significant portion of the time (20%) in the grassy area of the shark fin, even though there were 240 connections vying for it's bandwidth.

Duration of a Congestion Event

Figure 3.16 shows the duration of a congestion event in the dumbbell. As the number of connections increases, the shark fins become increasingly uniform, with the average duration of a congestion event comparatively stable at approximately 280 milliseconds.

Chronic Congestion

Throughout this discussion, TCP kept connections running smoothly in spite of changes in demand that spanned 2 to 200 connections. We searched for the point at which TCP has well-known difficulties when a connection's congestion window drops below 4. At this point, a single dropped packet cannot be discovered by a triple duplicate ACK and the sender will wait for a timeout before re-transmitting.

Figure 3.17 shows what happened when we increased the number of connections to 760 and spread out the RTT values. When the average congestion window on a particular leg dropped below 4, the other legs were able to quickly absorb the bandwidth released. In this example, the legs with 74 millisecond RTT and 209 millisecond RTT stayed above cWnd 4 and were able to gain a much higher proportion of the total dumbbell bandwidth. When each leg had 90 connections (total 720), the 209 ms leg had an average cWnd of 14.1, compared to 1.9 for it's nearest competitor, RTT 74. Subsequent runs with other values always had the 209 ms leg winning and the 74 ms leg coming in second.

In this case, TCP connections with 209 ms RTT actually fared better than many connections with shorter RTT. This is directly contradicts the old adage, "TCP hates long RTT". We speculate that the sawtooth graph for cWnd for those connections is long (slow) enough so that the loss risk is low for two consecutive congestion events. Perhaps the new adage should be "TCP hates cWnd below 4".

Short-Lived Flows and Non-Responsive Flows

Next we considered simulations that added a variety of short-lived connections. It is common for the majority of connections seen at an Internet link to be short-lived, while the majority of packets are in long-lived flows. For our purposes, we consider a connection short-lived if its lifetime is shorter than the period between congestion events. The short-lived connections have no memory of any prior congestion event. They neither add to nor subtract from the long-range variance in traffic. As the number of active short-lived connections increases, more bandwidth is added to the mean traffic. At high bandwidth (and therefore a high number of active connections), both short-lived flows and non-responsive flows (typically a sub-class of UDP flows that do not slow down in response to drops) simply add to mean traffic.

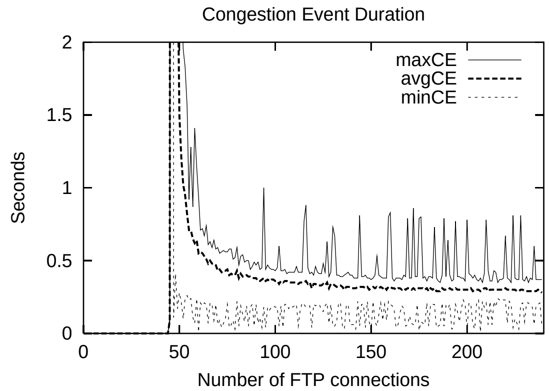

Figure 3.16 Congestion Event Duration approaches reaction time.

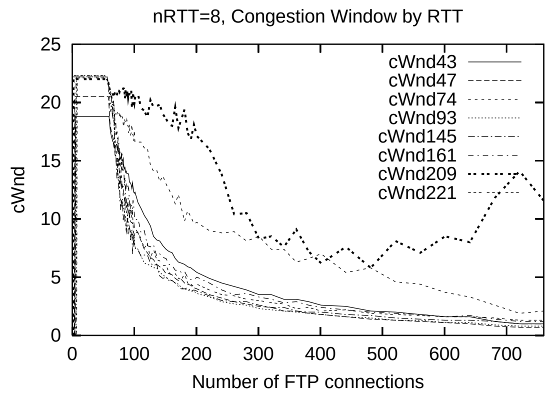

Figure 3.17 As flocks at each RTT drop below cWnd 4, they lose much of their share of bandwidth.

3.5 Congestion Model

The simulation experiments in Section 5 provide the foundation for modeling queue behavior at a backbone router. In this section we present our model and initial validation experiment.

Input Parameters

For a fixed size time tick, t, let C be the capacity of the dumbbell link in packets per tick and Q be the maximum depth the link's output queue can hold. The set of flocks, F, has members, f, each with a round trip time in ticks, RTT_f, a number of connections, N_f, a ceiling, $Ceiling_f$, and a floor, $Floor_f$. The values of $Ceiling_f$ and $Floor_f$ are measured in packets per tick and chosen to represent the bandwidth flock f will achieve if it is unconstrained at the dumbbell and only reacts to it's worst bottleneck elsewhere.

Operational Parameters

Let B_t be the number of packets buffered in the queue at tick t. Let D_t be the number packets dropped in tick t, and L_t be the loss ratio. Let $V_{f,t}$ be the volume in packets per tick being offered to the link at time, t. Reaction Time, R_f, is the average time lag for the flock to react to feedback. Let $A_{f,t}$ be the acceleration rate in packets per tick per tick at which a flow increases its volume in the absence of any negative feedback. Let $W_{f,t}$ be the average congestion window.

Initially,

$$V_{f,0} = Floor_f$$

$$W_{f,0} = \frac{V_{f,0} * RTT_f}{N_f}$$

$$R_f = RTT_f * 1.2$$

$$A_{f,0} = ComputeAccel(W_{f,0})$$

$$B_0 = 0$$

For each tick,

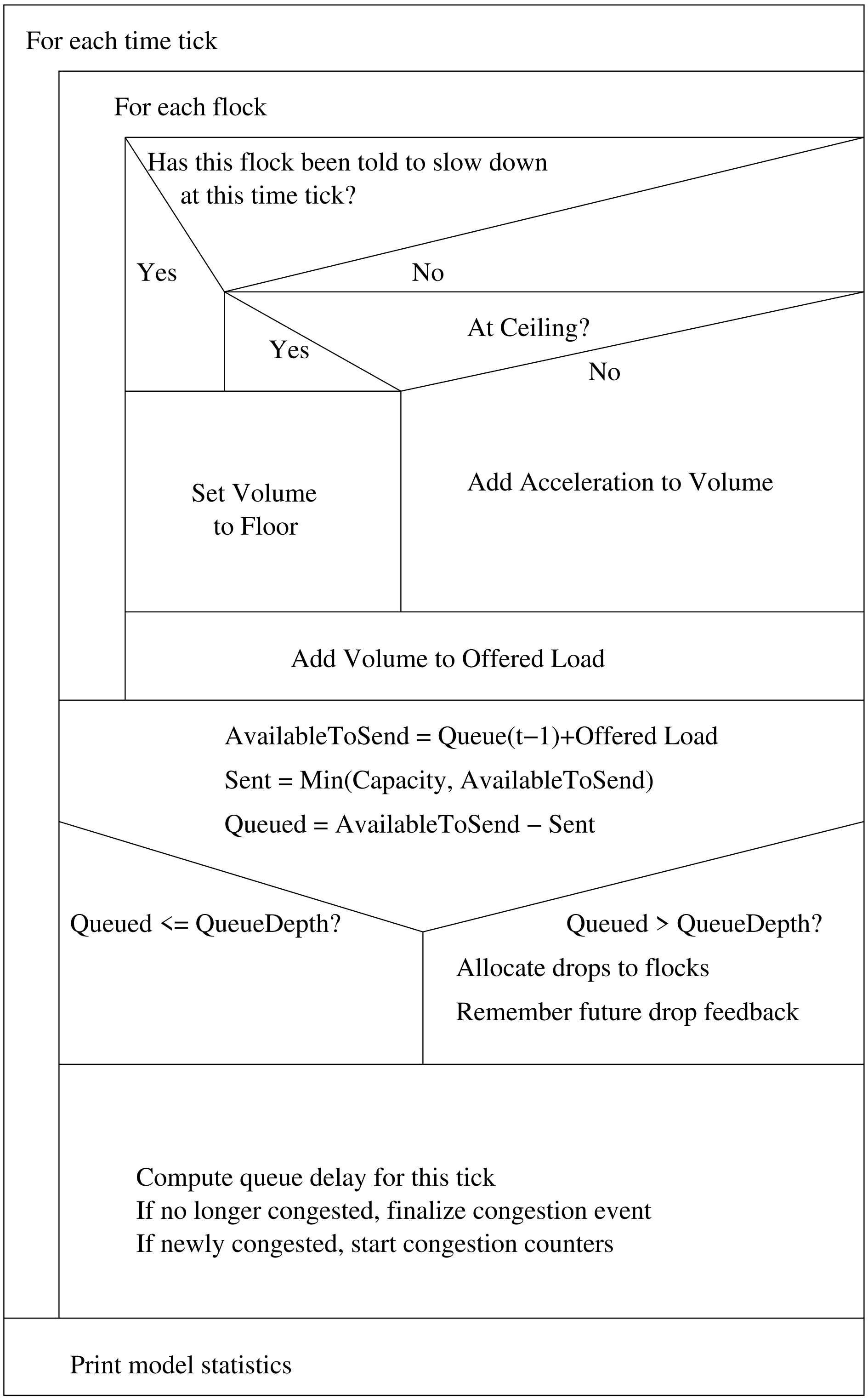

Figure 3.18 Scalable Model Logic

$$AvailableToSend_t = B_t + \sum_{f=0}^{F} V_{f,t}$$

$$Sent_t = min(C, AvailableToSend_t)$$

$$Unsent_t = AvailableToSend_t - Sent_t$$

$$B_{t+1} = min(Q, Unsent_t)$$

$$D_t = Unsent_t - B_{t+1}$$

$$L_t = \frac{D_t}{D_t + Sent_t}$$

$$RememberFutureLoss(L_t)$$

For each flock, prepare for the next tick:

$$W_{f,t+1} = ReactToPastLosses(f, L, R_f, W_{f,t})$$

$$A_{f,t} = ComputeAccel(f, W_{f,t})$$

$$V_{f,t+1} = V_{f,t} + A_{f,t}$$

RememberFutureLoss retains old loss rates for future flock adjustments.

ReactToPastLosses looks at the loss rate that occurred at time $t - R_f$ and adjusts the congestion window accordingly. If the loss rate is 0.00, $W_{f,t}$ is increased by $1.0/RTT_f$, representing normal additive increase window growth. If the loss rate is between 0.00 and 0.01, $W_{f,t}$ is unchanged, modeling an equilibrium state where window growth in some connections is offset by window shrinkage in others. The factor 0.01 is somewhat arbitrarily chosen. Future work should either justify the constant or replace it with a better formula. If the loss rate is higher than 0.01, $W_{f,t}$ is decreased by $W_{f,t}/(2.0 * RTT_f)$. If $Ceiling_f$ has been reached, $W_{f,t}$ is adjusted so $V_{f,t+1}$ will be $Floor_f$. To represent limited receive window, $W_{f,t}$ is limited to $min(46, W_{f,t})$. The constant here is 46 because 46 packets of 1500 bytes each fill a 64K Byte receive window. Early implementations of LINUX actually used 32K Byte receive windows, but memory became cheap. Without using window scaling, receive windows are limited to 64K Bytes.

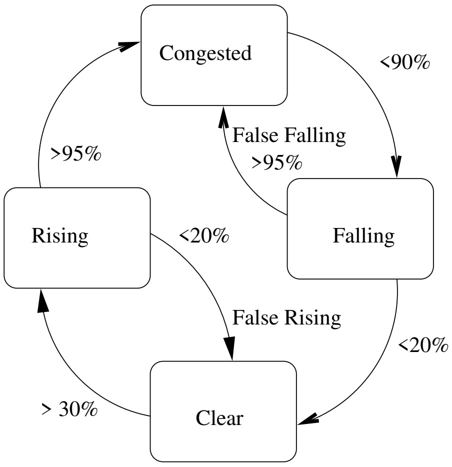

Figure 3.19 Finite State Machine for tracking the duration of congestion based on queue occupancy.

In ComputeAccel, if $W_{f,t}$ is below 4.0, acceleration is set to N_f packets per second per second (adjusted to ticks per second). Otherwise ComputeAccel returns N_f/RTT_f. Notice that the computation of the acceleration and $W_{f,t+1}$ given differs from the formula for W' given in section 3. The compromise was deployed when the model failed to accurately predict the quiet time after a congestion event. The quiet time is primarily caused by connections that suffer a coarse timeout.

The Reaction Time, R_f, should actually depend on one RTT_f plus a triple, duplicate ACK. We use the simplification here of taking $1.2 \times RTT_f$ because we did not want to model the complexities of ACK compression and the corresponding effect on clocking out new packets from the sender.

Outputs of the model

The model totals the number of ticks spent in each of the queue regimes: Clear, Rising, Congested, or Falling. The Finite State Machine is shown in Figure 3.19. The queue is in Clear until it rises above 30%, Rising until it reaches 95%, then Congested, then Falling when it drops to 90%, and Clear again at 20%. False rising leads to Clear if, while Rising, the queue drops to 20%. False falling leads to Congested if, while Falling, the queue grows to 95%.

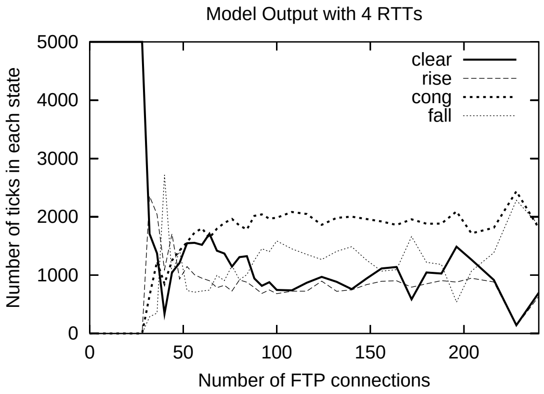

Figure 3.20 Queue regimes predicted by the congestion model

Calibration

Figure 3.20 shows what the model predicts for the simulation in Figure 3.15. Improvements will be needed in the model to more accurately predict the onset of flocking, but the results for moderate cWnd sizes are appropriate for traffic engineering models. The model correctly approximated the mixture of congested and clear ticks through a broad range of connection loads. Even though the model has simple algorithms for the aggregate reaction to losses, it is able to shed light on the way in which large flocks interact based on their unique RTTs.

Extending the Model to Multi-Hop Networks

The ultimate value of the model is its ability to scale to traffic engineering tasks that would typically be found in an Internet Service Provider. Extending the model to a network involves associating with each flock, f, a sequence of h hops, $hop_{h,f} \in Links$. Each link, $link \in Links$, has a capacity, C_{link}, and a buffer queue length, Q_{link}.

Headroom Analysis

The model predicts the number, duration, and intensity of loss events at each link, $link$. It is easy to iteratively reduce the modeled capacity of a link, C_{link}, until the number of loss events increases. The ratio of the actual capacity to the needed capacity indicates the link's ability to accommodate more traffic.

Capacity Planning

By increasing the modeled capacity of individual links or by adding links (and adjusting the appropriate hop sequences, $hop_{h,f}$), traffic engineers can measure the improvement expectation. Similarly, by adding flocks to model anticipated growth in demand, traffic engineers can monitor the need for increased capacity on a per-link basis. It is important to note that some links with relatively high utilization can actually have very little stress in the sense that increasing their capacity would have minimal impact on network capacity.

Latency Control

Because the model realistically reflects the impact of finite queue depths at each hop, it can be used in sensitivity analyses. A link with a physical queue of Q_{link} can be configured using RED to act exactly like a smaller queue. The model can be used to predict the benefits (shorter latency and lower jitter) of smaller queues at strategic points in the network.

Backup Sizing

After gathering statistics on a normal baseline of flocks, the model can be run in a variety of failure simulations with backup routes. To test a particular backup route, all flocks passing through the failed route need to be assigned a new set of hops, $hop_{h,f}$. Automatic rerouting is beyond the scope of the current model, but would be possible to add if multiple outages needed to be modeled.

3.6 Simulation Summary

The study of congestion events is crucial to an understanding of packet loss and delay in a multi-hop Internet with fast interior links and high multiplexing. We propose a model based on flocking as an improved means for explaining periodic traffic variations.

A primary conclusion of this work is that congestion event are either successful or unsuccessful. A successful congestion event discourages enough future traffic to drain the queue to the congested link. Throughout their evolution, transport protocols have sought to make end-to-end connections more efficient. *Fast Retransmit* in RFC 1122 [22] allows senders to recognize the loss of a packet when they see a triple duplicate ACK from a receiver (caused by receiving the 3 packets after the missing packet). From the viewpoint of the link queue, this made congestion events more likely to be successful. With fast retransmit, senders are reacting sooner and the delay is independent of the window size (assuming cWnd larger than four). This widens the portion of the design space in which congestion events are successful. The protocols work well across long RTTs, a broad range of link capacities and at multiplexing factors of thousands of connections. The result is that a larger fraction of the congestion events in the Internet last for one reaction time and then quickly abate enough to allow the queue to drain. Depending on the intensity of the traffic and the traffic's ability to remember the prior congestion, the next congestion event will be sooner or later.

The shape of a congestion event tells us two crucial parameters of the link being served: the maximum buffer it can supply and the RTT of the traffic present compared to our own. We hope this study helps ISPs engineer their links to maximize the success of congestion events. The identification of 4 named regimes surrounding a congestion event may lead to improvements in active queue management that address the impact local congestion events have on neighbors. The result could be a significant improvement in the fairness and productivity of bandwidth achieved by flocks.

When the model is applied to multi-hop networks, it can be used for capacity planning, backup sizing and headroom analysis. We expect that networks in which every link is configured for 50% to 60% utilization may be grossly over-engineered when treated as a multi-hop network. It is clear that utilization on certain links can be high even though the link is not a significant bottleneck for any flows. Such links would get no appreciable benefit from increased bandwidth because the flocks going through them are constrained elsewhere.

Future Work

Further validation of the model's scalability and accuracy would be important and interesting. The model predicts the proportion of rising, falling, grassy and congested ticks even on heavily loaded links with small window sizes. This should be validated by accurately measuring one-way delays in a measurement infrastructure like Surveyor. Congestion event duration and the gap between congestion events should be validated in an emulation environment with appropriately large number of connections (at least thousands). Measurement equipment would need to record losses on a much finer time scale (on the order of 1 ms granularity) than is currently available using SNMP.

We plan to extend the model to cover the portion of the design space where congestion events are unsuccessful. By exploring the limits of multiplexing, RTT mixtures, and window sizes with our model we should be able to find the regimes where active queue management or transport protocols can be improved. We also need to expand the model so it more accurately predicts the onset of chronic congestion.

We don't know if small buffers in routers are better than large buffers. Routers with a very small number of buffers send very informative losses to senders rather than building up large queues that add jitter. Intuitively, this gives timely feedback to TCP senders and trains the TCP senders to stay within their share of the bandwidth. The model needs to be exercised with an appropriate topology and an appropriate traffic matrix of responsive and unresponsive traffic to compare congestion using a small vs. a large amount of buffer space in routers.

The traffic engineering applications for the model are particularly interesting. Improvements are needed to automate the gathering of baseline statistics (as input to the model) and to script commonly used traffic engineering tasks so the outputs of the model could be displayed in near real time.

Chapter 4

Traffic Matrix Estimation

In the Chapter 3 we established the importance of RTT and Ceiling in determining how a flock of traffic would react to congestion. In this chapter we will construct a traffic matrix for an ISP. Using only information that can be readily collected and updated, can we construct a traffic matrix that will be appropriately accurate for traffic engineering analyses? Clearly, our traffic matrix will have to contain information about not just the volume of traffic from each source to each destination, but also information about the RTT. Although we have identified Ceiling as an important parameter, we were unable to invent a reliable mechanism for measuring Ceiling at the edge of an ISP.

There are many reasons to build a traffic matrix. From a traffic engineering point of view, the traffic matrix is used for capacity planning, performance analysis and backup assessment tasks. It helps assess bottlenecks accurately, test proposed upgrades, and identify critical links that would cause the most traumatic routing changes if they died.

The central challenge overcome in this chapter is the difficulty of determining window sizes and RTTs from packets passing through. An ISP does not have the luxury of seeing the entire connection end-to-end, nor do the packets carry any information that would immediately show the current window sizes. So it became necessary to infer the congestion avoidance parameters from data that could be economically gathered.

We noticed a feature of TCP that occurs in environments with a high bandwidth delay product. TCP allows recipients to ACK every second packet using a mechanism called delayed ACKs [88]. We will show in section 4.3 that ACKs clocking out more than one new sender packet are common in high bandwidth delay product connections but much less common in any other situation. Our hypothesis is that high bandwidth delay product flows are likely to be memory limited – a tendency we will leverage to infer the likely RTT of a flow. An example will clarify the inference. Consider a connection with a 32 KByte rWnd, 320 KBytes per second throughput, and a bottleneck bandwidth more than 10 Mbps. The receive window limit is 32 KBytes per window times 10 windows per second times 8 bits per byte, making 2.56 Mbps. Since this is less than the bottleneck bandwidth, the connection will not be able to deliver any more than its 32 KBytes per window memory limit. Assuming that we can identify this flow in the flow records (our hypothesis is that this flow will have a high incidence of delayed ACKs), we can directly read the duration and the total number of bytes from the flow record. Dividing the 32 KBytes per window by 320 KBytes per second, we infer that each window is approximately 0.100 seconds.

The delayed ACK mechanism allows recipients the option of acknowledging only every second packet, provided that the ACK is not withheld for more than a configurable timeout. Used properly, delayed ACKs can reduce the protocol processing overhead in the sending and receiving hosts and reduce the number of packets needed sent across the reverse path. Use of the term *delayed ACK* strongly implies an ACK that increases the sender's left window edge by exactly 2 packets.

While studying delayed ACKs we also noticed a substantial amount of stretch ACK behavior. This is a regime in which on average and over a long period of time, each ACK packet releases even more than 2 new, source data packets. Stretch ACKs are referred to in RFCs as early as RFC 1122 [22]. We offer some suggestions for possible causes, but offer no proof. For the purpose of this thesis, we define an ACK that moves the sender's left window edge by more than 2 MSS packets as a *stretch ACK*. We treat stretch ACKs as an indicator of high BDP no different than delayed ACKs.

4.1 Capturing and Simplifying Abilene Traffic

This chapter delves into problems associated with measuring and reproducing real-life customer demands to place on the topology we developed in Chapter 2. The goal is to produce a traffic matrix that is appropriately accurate for our chosen topology. We wanted a topology we could implement in emulation in the Wisconsin Advanced Internet Lab [49]. Each row in the

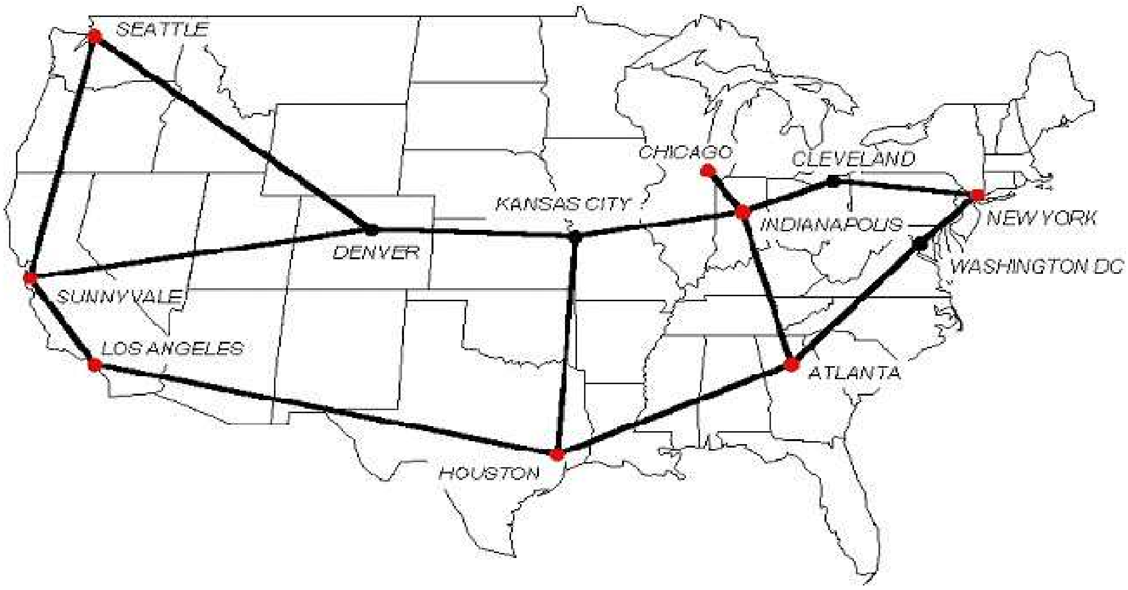

Figure 4.1 Abilene Network Backbone, February 2003

matrix represents the demands from one source and each column represents one destination. The entries in each cell in the matrix are parameters important to a particular study. For example, each cell might contain an array of connections with each connection having a round trip time, a protocol, and parameters to characterize the on / off times for the connection.

A particular traffic matrix is a single moment in time. Once we have a matrix that represents the Internet of today, we want to structure it so we can assess the Internet of many possible futures. If we choose parameters wisely, the traffic matrix will be useful in hypothetical scenarios such as scaling the volume to reflect an increased number of connections or growing the ceiling to reflect faster last-mile technology connecting users to a particular network.

We chose the Abilene [1] topology because we had access to flow data for each 5 minute segment of an entire day at all of the routers in that network. The geographic layout of Abilene is shown in figure 4.1. Abilene also exposes a wide array of router statistics and design data that makes it an excellent environment for future extensions to this research.

By contract, the Abilene backbone only carries traffic from Internet2 sites to Internet2 sites. This makes the routing straightforward. Another aspect of Abilene that makes our study simpler

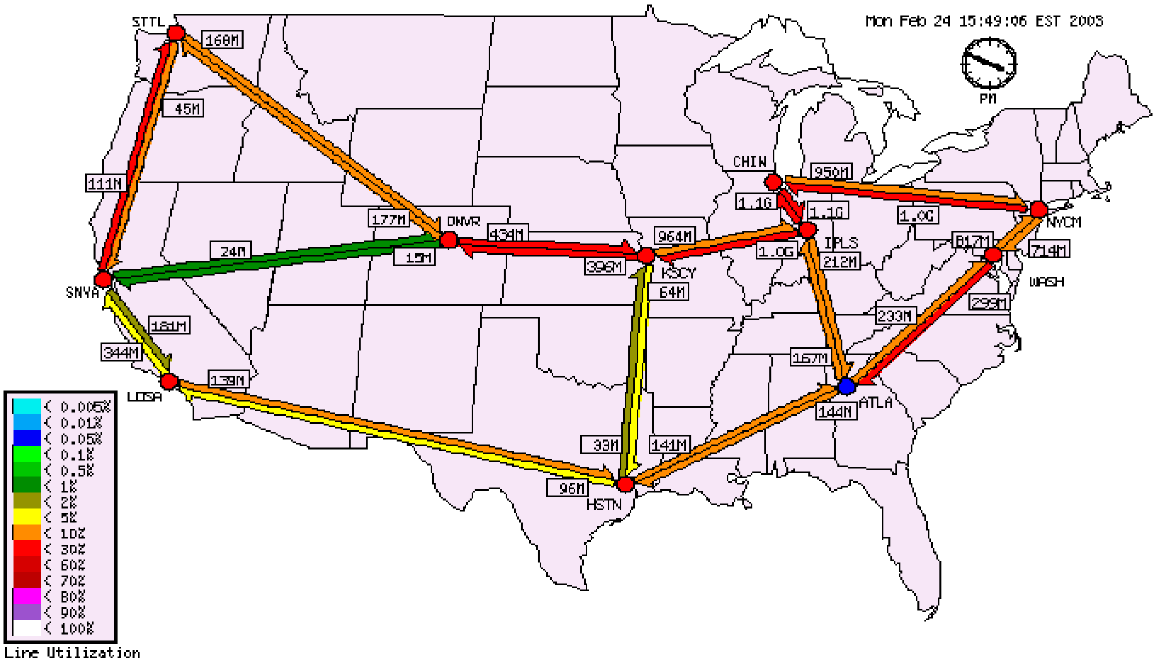

Figure 4.2 Weather map of Abilene shows bits per second for each link averaged over 5 minutes

is that most of the Autonomous Systems that connect to Abilene only connect at a single point. Multiple points of interconnect are more common between commercial Internet Service Providers.

Parameterizing the Model

Internet traffic can be characterized by many parameters. Some parameters are closely related. For example, volume (bits per second), and packet count (packets per second) are clearly related. Other parameters like composition (ports used) and protocol give hints about the way the traffic will react to congestion and the urgency of the traffic.

At the simplest level, the traffic volume in Abilene can be seen in the weather map [64] illustrated in figure 4.2. The link utilization shows the number of bits per second averaged over the preceding 5 minutes along each link. The link at 714 Mbps from New York City Manhattan (NYCM) to Washington (WASH) represents all connections that feed into New York City from other Abilene nodes or from links that enter Abilene at NYCM and head to Washington. Here link color tells us the link is currently carrying between 5% and 10% of its capacity.

Since it is a readily-available and easily-understood metric, link utilization is the most commonly used tool on the traffic engineer's tool-belt. Link utilization easily identifies links that are grossly under-utilized and can alert the engineers to problems if it plummets or skyrockets unexpectedly.

But link utilizations above 95% are normal and appropriate for long-haul links in the commercial Internet. As we saw in Chapter 3, high link utilization is not, by itself, a cause for concern. Connections with congestion windows of 8 or more packets per RTT are well within the region where TCP and TCP-friendly regimes are efficient and reliable. Moreover, link utilization does not tell us the ultimate destination of packets, making it useless for analyses that predict traffic in the event of a link failure. If a particular link went down, how much of its traffic would have to be re-routed and which links would it impact? Other traffic engineering questions also depend on the original sources and the ultimate destinations of traffic. How would congestion be affected if we added new links? If a link is upgraded, will it cause other links to become bottlenecks?

Chapter 3 emphasized that simple link utilization alone is not sufficient to predict the way traffic will react to congestion or to predict the way neighboring congestion will affect future traffic at the link being analyzed. To get more detail than simple link utilization, we used volume (a number analogous to the number of simultaneous TCP-style connections) and Round Trip Time (RTT). Later, we added a notion of a ceiling (a bottleneck before or after our backbone or a memory limit at either the sender or receiver).

As we discussed in Section 3.5, RTT is a crucial parameter in the achievable window size of a connection. In this chapter, we augment that by showing how receiver and sender memory limitations cause connections to reach ceilings before they reach their bandwidth delay product. These limitations will be common when optical or gigabit connections to ISPs become more common to the extent that the bottleneck bandwidth lies there or outside that boundary.

Measuring Demand

A complete set of packet headers with accurate timestamps from a network like Abilene would provide a unique and important starting point for measuring demand. A library of protocol characterizations could be developed that would let us label each flow with accurate information about the way it reacts to congestion.

Unfortunately, fast backbones handle far more packets than we can reasonably capture or analyze. In this chapter, we use flow profiling [7]. Each flow is a unidirectional series of IP packets of a given protocol, traveling between a source and a destination within a certain period of time. The source and destination are defined as an IP address and port. A single flow record is considerably smaller than the packet headers for the flow. A complete TCP connection is two or more unidirectional flows recorded by a router as an accounting record. The flow record shows the source IP and port, destination IP and port, start time, duration, protocol, and other information not needed here.

Abilene routers cannot afford to dedicate excessive resources to gathering and transmitting flow data. After all, their primary function is routing data packets. Abilene routers are set to sample uniformly one packet out of every 100 and build flow records only from the packets sampled. For summary statistics, this gives appropriate accuracy. Ramifications of the 1% sampling are discussed in Section 4.2. Capturing an entire day for all 11 routers in Figure 4.2 consumed about 13 Gigabytes of flow records. Flow records were then analyzed using FlowScan [74] to collect together the volume of data from each source to each destination.

We expected little statistical difference between flows to and from the same autonomous system. As a useful simplification and to improve anonymization, we aggregated all flows based on source AS and destination AS. Over 90 percent of those AS's had a unique attachment point to Abilene. To determine attachment points to Abilene, we used only the destination AS number for each flow. The source AS for flows has to be considered unreliable, since some IP address spoofing slips through Abilene ingress filters. In Section 4.2, we show that 52 percent of the traffic on our test day could have its entire path through Abilene described solely by knowing its source AS and destination AS. The other 48% had either a source or destination that was an AS with more than one attachment point.

Round Trip Time Estimate

Flow data gives no obvious clue to the RTT for the flow. Each flow record shows start time, end time, byte count and packet count. Two flows with radically different RTT could have identical flow records if they had different window sizes. RTT is a crucial parameter for understanding everything from congestion reaction time to jitter in queue depth [78].

The quest for clues to the RTT of a flow led us to an interesting discover. Connections with a high bandwidth delay product have fewer ACKs per data packet than connections with a lower BDP. Even the shortest Abilene backbone hop (NYCM to WASH) guarantees at least a 3 millisecond RTT due to the speed of light propagation delay. In Section 4.3 we discuss a technique using ACK ratios to identify AS's in places like New Zealand or Israel that have a long delay after leaving Abilene. Using this technique, we separated the AS's into those whose propagation delay was dominated by their distance from Abilene versus those whose external propagation delay was negligible.

The actual traffic matrix generated does not need to differentiate between AS's. All AS's near a particular Abilene node are lumped together as equivalent. Other, more distant AS's are named for their egress point and a digit specifying the category of extra propagation delay. In practice, we found adequate results using only 2 categories of extra propagation delay.

The remainder of this chapter is organized as follows: Section 4.2 describes how the data was gathered to compute demand by AS. Section 4.3 describes the technique for using ACK and data streams in flow data to estimate RTT. Section 4.4 describes the process of aggregating traffic based on ingress, egress and external delay. Section 4.5 summarizes the results and concludes with future directions. In the chapter on related work, Section 5.3 discusses work related to traffic matrix estimation and the phenomenon of delayed and stretch ACKs.

4.2 Populating the Traffic Matrix

The Abilene project [1] provides a wide range of performance and design data about the United States backbone for Internet2. Dynamic websites show such things as the current utilization of the major backbone links [64] and the recent graphs of traffic on every major feed into Abilene. Router statistics show how many packets were dropped and how many were forwarded. Flow data shows traffic broken down by such things as protocol or port.

To predict the number of clear, rising, congested and falling ticks at each link in Abilene, we take a model traffic matrix and run the model at each hop of each flock for each tick. All links can be run in parallel, but the results of one tick affect the window sizes from each flow for the next tick.

Each flock is characterized by a 3-tuple (ingress point, egress point, and exterior delay) along with a multiplexing factor and a ceiling. Flocks that share the same 3-tuple can be combined into a single flock with the total of the ceilings and the total of the multiplexing factors.

The results of the model include a detailed measure of the composition of the congestion at each link. In addition, the model measures the resulting overall throughput of each flock. The graph of achievable congestion window sizes shows how each end-to-end path is affected by global Abilene congestion.

Minimal Window Size

There is enough information readily available in Abilene to model the traffic volume, understand the traffic routing, and compute throughput. It is somewhat harder to measure customer satisfaction. For the sake of this thesis, we will define explicitly that a customer is unhappy if congestion in Abilene causes his congestion window to fall below 4 and stay below 4 until his retransmission timeout (RTO) reaches more than 10 times RTT. The numbers are not as arbitrarily chosen as they might seem. TCP depends on the triple-duplicate ACK mechanism to recover from losses without falling back to a coarse timeout. TCP connections get roughly linear performance as their window size decreases to 4. But TCP performance drops dramatically when it depends on coarse timeouts. As more and more timeouts are needed, the exponential backoff algorithm causes throughput to drop to frustrating and unacceptable levels. The abandonment rate is the rate at which customers give up on TCP connections that are in progress. We assert that the abandonment rate will be higher in environments with large numbers of coarse timeouts than in environments with no coarse timeouts.

Service-level agreements (SLA's) often specify a maximum acceptable loss rate (perhaps because it is easily measured). Managers assume that packet losses are bad and that the only way to avoid customer complaints is to over-engineer capacity. But packet losses are the most important feedback to TCP connections to tell them what bandwidth they should appropriately pace for. In fact, many customers would get almost exactly the same total throughput even if they received substantially fewer losses from the core of the network. To investigate abandonment rate, we modeled the range of congestion window sizes seen across the day.

Backbone Interfaces

In order to predict the path packets take through Abilene, we needed to construct a graph that would map a flow with source AS, AS_s, and a destination AS, AS_d, onto the links that the flow would traverse.

Table 4.1 Sample Link Tuples

From	To	Mbps	Queue Depth	Delay
SNVA	DNVR	10200	100	10
SNVA	LOSA	10200	100	3
STTL	SNVA	600	100	8
SNVA	KSCY	10200	100	12
DNVR	KSCY	2400	100	4
STTL	DNVR	2400	100	10
...	...			

Table 4.1 shows data that was gathered or inferred for each link in Abilene. Each link is unidirectional. For example, the path from Sunnyvale (SNVA) to Los Angeles (LOSA) tells the capacity of the queue, a queue depth indicating the ability of the queue to buffer traffic headed to LOSA, and the delay in milliseconds it contributes to RTT. Another tuple for LOSA to SNVA will show the link from the point of view of LOSA and LOSA's router's queue.

Link delay was averaged and rounded from traceroute differences for connections that cross those links. The delays listed are double the one-way delay to simplify the way connection RTT is accumulated from hops.

Volume of Traffic

Abilene flow data was used to discover the volume of traffic going from each source to each destination. A typical flow record shows the detail available for each flow. The data received from Abilene has been anonymized by zeroing out the low-order 12 bits of each IP address. To further protect the privacy of customer data, each of the IP addresses was anonymized by scrambling the top 20 bits. In tables shown in this thesis, IP addresses have been simplified to small, fictitious numbers. All other data in Table 4.2 came from actual flow records.

Table 4.2 shows a few typical flows to illustrate the features and problems. The first two records show a flow and it's reverse flow between source IP 1.0.0.0 and IP 2.0.0.0 on ports 2490 and 2424. Note that each flow record is one direction of the round trip. In Abilene, we are fortunate that the reverse path travels along the same links. In the commercial Internet asymmetric routing is more typical [71]. In the case of asymmetric routing, one of these records might be visible but the reverse path may be handled by a different ISP.

The records at time 16:03 in Table 4.2 are, presumably, the ACK packets and the data packets for a single connection. The connection from 2.0.0.0 to 3.0.0.0 shows 16 data packets, but only 13 ACK packets. It was very common for the number of ACK packets to be substantially smaller than the number of data packets. Notice also the records for the connection between 2.0.0.0 and 6.0.0.0. Since the packets are sampled at 1:100, the record for a data flow is often far away from the record

Table 4.2 Sample Flow Data Records

Date	Time	Source	Destination	Packets	Bytes
2003/04/24	16:03:47	1.0.0.0.2490	2.0.0.0.2424	3	120
2003/04/24	16:03:56	2.0.0.0.2424	1.0.0.0.2490	15	22500
...					
2003/04/24	16:04:01	2.0.0.0.3273	3.0.0.0.4458	16	24000
2003/04/24	16:04:02	3.0.0.0.4458	2.0.0.0.3273	13	520
...					
2003/04/24	16:04:16	2.0.0.0.1073	6.0.0.0.3592	7	280
...					
2003/04/24	16:04:15	4.0.0.0.3597	2.0.0.0.1073	9	13500
2003/04/24	16:04:16	2.0.0.0.1073	4.0.0.0.3597	7	280
...					
2003/04/24	16:04:18	6.0.0.0.3592	2.0.0.0.1073	17	25500
...					
2003/04/24	16:04:37	5.0.0.0.4377	2.0.0.0.2920	1	40
2003/04/24	16:04:39	5.0.0.0.4377	2.0.0.0.2920	1	40
2003/04/24	16:04:39	2.0.0.0.2920	5.0.0.0.4377	5	7500

for the corresponding ACK flow. In fact, the connection between 5.0.0.0 and 2.0.0.0 show how a single connection can often look like several flows in each direction.

Moreover, a flow may or may not show up at a prior or subsequent hop. Care must be taken to avoid counting the same flow as though it were N flows if it passes through N nodes.

Table 4.3 Traffic Matrix Flow Tuple

Ingress	Egress	Exterior Delay	Volume	Ceiling
ATLA	LOSA	20	15	12
ATLA	LOSA	2	7	46
HSTN	IPLS	2	3	21
LOSA	KSCY	200	8	7
KSCY	IPLS	2	25	44
IPLS	LOSA	20	16	24

The actual tuples used in the model need three parameters for each modeled flow. Table 4.3 gives examples. The volume is assumed to be 100 times the number of data packets captured. The exterior delay is estimated into broad categories based on the AS of the source and the AS of the destination using an algorithm described in section 4.4. The Ceiling is estimated by taking the volume of the flow and dividing by the duration.

Table 4.4 shows the final traffic matrix derived from the flow data. The total volume has been normalized so that the unambiguous traffic adds up to 1000 units. Four Abilene nodes are shown, broken into their near and far attached Autonomous System equivalence groups. The other 7 near and 7 far groups are lumped into the category “other” solely for the presentation in this paper. The actual model uses all 22 AS equivalence groups. The column and row for “unamig” is the total of the data used in the model for that column or row.

Data whose source or destination is ambiguous is not factored into the model. It is included in Table 4.4 to show which fraction of the traffic is ignored.

Table 4.4 Excerpt from Observed Traffic Matrix. Each entry is the volume of that flock in units normalized to a total volume of 1000 unambiguous connections

	Dest ambig	chinF	chinN	iplsF	iplsN	losaF	losaN	snvaF	snvaN	Other	unambig	tot
ambig	117.3	6.4	18.0	3.1	18.5	14.2	7.3	7.5	2.3	168.4	245.8	363.1
chinF	5.3	0.1	0.1	0.3	0.9	4.9	1.1	0.7	0.5	16.6	25.3	30.6
chinN	15.7	0.2	0.6	0.1	7.0	8.5	9.5	5.7	4.1	57.9	93.7	109.4
iplsF	4.5	0.7	0.9	0.2	0.2	0.8	0.3	0.4	0.1	9.2	12.7	17.1
iplsN	16.5	2.7	4.4	0.3	1.1	4.2	9.2	1.3	1.8	68.5	93.5	110.0
losaF	30.0	3.7	11.9	1.1	6.4	0.1	0.0	0.6	0.0	60.1	83.8	113.8
losaN	12.0	1.5	11.1	0.3	1.6	0.0	0.0	0.2	0.0	26.9	41.7	53.7
snvaF	36.3	1.2	3.2	0.7	1.9	0.4	0.1	0.1	0.0	36.6	44.3	80.5
snvaN	4.2	1.0	0.4	0.0	0.9	0.0	0.0	0.1	0.0	22.7	25.1	29.4
Other	132.8	16.9	58.1	9.3	46.4	29.6	27.4	17.1	16.5	467.0	579.9	712.8
Unambig	257.4	28.0	90.9	12.2	66.2	48.4	47.6	26.3	23.2	657.2	1000.0	1257.4
Total	374.7	34.4	108.9	15.3	84.7	62.6	54.9	33.8	25.5	825.6	1245.8	1620.5

4.3 Ramifications of Sender and Receiver Memory Settings

We developed a method to infer the Round Trip Times of connections from a flow data at the AS level even if the flow data is sampled by as little as 1:100. Before we can discuss evidence of memory limited flows, we briefly discuss TCP receive window, TCP send window and the effect it has on congestion reaction.

Up to this point, we have assumed that flows will speed up sending more packets per RTT window until they reach a limit based on their congestion window. Those flows are cWnd-limited and will react to a congestion event (if they see it) by multiplicative decrease in their volume. But what about flows that are incapable of supplying data fast enough to reach a congestion limit?

TCP receive window

The TCP receive window (rWnd) is specified by the receiver at initial connection. It is a promise from the receiver to devote at least rWnd memory to this connection. Even if the user-level process receiving the data is far behind, the kernel promises to accept delivery of rWnd bytes of data. Typical values range from 16K Bytes to 64K Bytes. Use of values above 64K Bytes would consume more than the 16-bit field for window size. An additional negotiated option, window scaling, allows rWnd values larger than 64K Bytes. Window scaling is growing in popularity, but actual rWnd values above 64K Bytes are still rare in the Internet.

Connections with a high bandwidth delay product often reach their memory limit before reaching the cWnd that would have been their fair share. An example will clarify this. Suppose a connection has a bottleneck bandwidth, $BW_c = 100Mbps$, and $RTT = 250ms$. This connection can get 4 windows per second and would need to supply 25 Mbits per window to fill its bottleneck's available bandwidth. Assuming 8 bits per byte, this translates to over 3 megabytes per window.

TCP send window

TCP senders are not required to send data just because the receiver is willing to receive it. In fact the burden of actually keeping unacknowledged data lies with the sender. The sender keeps a safety copy of every unacknowledged TCP packet in case it has to be retransmitted. This retransmission buffer takes up memory, in the normal uncongested case, for one RTT. The TCP send window, *sWnd*, is not mentioned in any TCP protocol interaction because there is no need to inform the recipient.

Mathis [53] maintains a web page to help configure systems for high performance data transfers. He reports that typical Unix systems include a default TCP send window of 32 KBytes to 61 KBytes. The default maximum values for TCP send window are between 128 KBytes and 1 MB. Note that Windows NT 4.0 had no support for window scaling and could not accommodate TCP send windows above 64 KBytes.

Consider a cWnd-limited connection, C, limited by cWnd=30,000B competing with a memory-limited connection, M, characterized by sWnd=22,500B and cWnd=30,000B. For simplicity, we assume the same RTT for both connections. During a 1 RTT congestion event with a loss rate, $L = 0.06$, C will send 20 packets of 1,500B each and has a $p(NoLoss) = 0.29$ chance that it will be unaware of the congestion event. The expected resulting window size for connection C will be $31,500 * 0.29 + 15,000 * 0.71 = 19,787$. This reflects the 29% chance the window will grow to 31,500B and the 71% chance it will shrink. In the aggregate, this was a drop of 10,213 Bytes, or 34%. The memory-limited connection will fare much better with 15 packets passing through the event. The $p(NoLoss) = 0.40$ causes an expected result of $22,500*0.40+15,000*0.60 = 17,965$. Note that 40% of the time this connection will not see any losses so it will neither shrink nor grow. Connection C abated 10,213 Bytes of traffic per window, but connection M abated only 4,535 Bytes of traffic per window.

The cumulative effect of a succession of congestion events is that the message to “please slow down” tempers flows with high congestion windows far more than their memory-limited competitors. The cWnd-limited connections react more strongly to the connection event and take longer (in the aggregate) to come back up to the ceiling (if any) that limits their growth elsewhere in their path. The memory-limited connections have no bandwidth bottleneck elsewhere in their path (or they would not have been memory-limited). And, they grow back to their limit quickly before leveling out. To the extent that a large collection of connections is memory-limited, it will abate less in response to a congestion event and will grow back faster.

Delayed ACK Mechanism

So far, we have seen that memory limits significantly change the way a connection reacts to congestion, but we have not shown any mechanism for differentiating bandwidth-limited connections from memory-limited connections. We will discuss the delayed ACK mechanism in TCP when packets arrive in rapid succession. Later, we will use a measure of the prevalence of delayed ACKs to distinguish between memory-limited and congestion-limited connections.

TCP tends to space the packets evenly across the window. The clear intent of the designers of TCP was that almost all of the packets sent by a TCP connection are an immediate response to a received ACK. But, the penalty for not acknowledging a single packet is very small. Imagine, as in the example above, a TCP sliding window allows 20 unacknowledged packets in flight. If the recipient skips sending half of the ACK packets the sender will receive ACKs only for packets 2, 4, 6, 8, . . . 20. The sender reacts to ACK 2 by sending out packets 21 and 22. The connection still easily fills the available window with data packets. In this example the odd numbered ACKs would have had very little value though they cost CPU time, network time, and interrupts. If the additive increase is triggered by the number of ACKs rather than the movement of the left edge of the sender's window, cWnd increase will only grow every other RTT.

Even in the early days of TCP, designers recognized that there could be several data packets queued up inside the receiver. It would be wasteful to send an ACK while processing every packet. A cumulative ACK could be generated when the queue becomes empty. The notion of a delayed ACK (one for every K^{th} segment) was already in use when RFC 1122 [22] suggested that a TCP implementation *SHOULD* limit K to 2.

RFC 1122 also states that a TCP implementation *MUST* set the maximum delayed ACK timeout to 500 milliseconds. Later, RFC 3449 [21] states that, in practice, the delayed ACK timeout is typically less than 200 milliseconds.

ACK Ratio

In the rest of this chapter, we will refer to the ACK ratio of a connection based on the average number of data packets per ACK packet. An ACK ratio can easily be obtained from flow data by taking the total number of ACKs for a connection and dividing it by the total number of data packets. A ratio of 2:1 would mean, on average, each ACK packet acknowledges 2 data packets, moves the senders left window edge by the size of 2 data packets and allows the sender to release those 2 data packets. Note that this is the average over the entire life of the connection, including ACKs that are received in the final round after data has finished.

We hypothesize that the ACK ratio will be close to 1:1 for connections which have a bottleneck, but will be higher if delayed ACKs can be used. Consider a connection whose packets travel through a slow bottleneck. For example, at 56 kbps, each 1500 byte packet takes 214 milliseconds of transmission time. The delayed ACK timer is likely to be smaller than 214 ms, so every data packet will be acknowledged. On the other hand, a connection whose slowest hop is 100 Mbps can have 8 such packets arrive in 960 microseconds.

Connections should not turn on the delayed ACK mechanism until after they exit slow start. During slow start it is important to inform the sender of round trip time and also set the release of data packets to a widely dispersed pattern. On exiting slow start, delayed ACK may be enabled. Later, the recipient will turn off the delayed ACK mechanism if he sees a gap in the sequence numbers of incoming packets. The gap probably signals a lost packet and the recipient wants to start the fast-retransmit regime quickly. Recipients continue to emit one ACK (without delay) for every incoming packet until the missing packet is received. This also tends to keep the packet pacing well clocked. Congestion-limited connections should have lower overall ACK ratios because the 1:1 fast retransmit regime lasts for one entire RTT after each gap in packet sequence numbering. To the extent that memory-limited connections are lossless, they have no need to ever turn off delayed ACKing.

Figure 4.3 shows the number of bytes in flight as a function of time. The data comes from a tcpdump of a portion of a long FTP over a 70 ms RTT connection through Abilene from Wisconsin to Colorado. The tcpdump was taken on the sender side so that the flight size could be directly

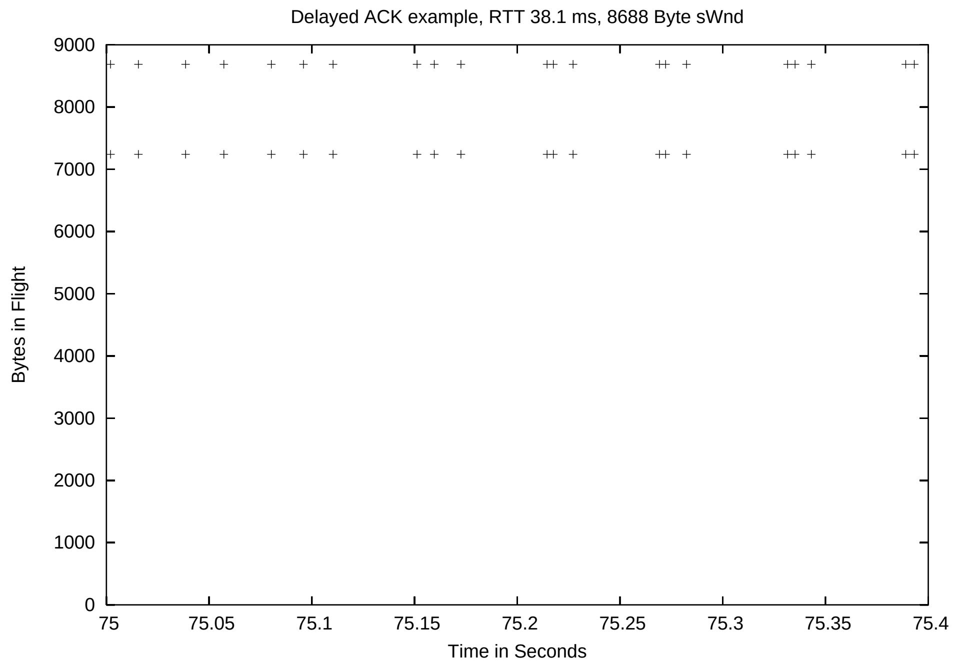

Figure 4.3 Flight size graph shows one plus for each packet emitted by the sender. The 6 packets in each round are not evenly spaced.

computed from captured packets. Flight size is the number of bytes sent but not yet acknowledged. The connection in this example uses 1,448 Byte packets and is send-window limited to 8,688 Bytes (6 packets) unacknowledged. This example shows a connection after slow start that uses delayed ACKs in a high BDP environment. Each time an ACK arrives, it acknowledges two old packets (in this case 4 packets ago). The graph then shows a column of two data packets released in rapid succession. The first packets at 75.002 seconds is at 7,240 bytes in flight (presumably because there were 5 prior packets that are still unacknowledged). But the next packet leaves only slightly later at 75.003 and shows up at 8,688 bytes in flight. In all cases, bytes in flight includes the bytes in the packet being plotted (in these cases, 1,488 bytes). Those two packets are so close together in time that they seem to be on the same vertical line. Time between columns of data packet departures is idle time for the connection, waiting for the number of bytes in retransmission buffers (the bytes in flight) to drop below the sWnd of 8,688.

Jitter in the departure times may be caused by uncertainty in the amount of time it takes to dispatch the user-level process. As time progresses, the variations in the amount of time needed to dispatch the user-level processes at both ends of the connection contribute to a compression of the gap between ACKs. This can be seen around $t = 75.3$, where the idle gap is no longer being controlled and packets are released in pairs that are haphazardly spaced.

Notice particularly that, although there were no losses, the connection in Figure 4.3 did not accelerate because it is memory-limited on send window. The connection gets a throughput of 8,688 bytes per RTT even though the receiver would have permitted more throughput and the congestion control conventions would have allowed the sender to try to send faster.

Stretch ACK Mechanism

TCP implementations *SHOULD* emit an ACK packet for every second data packet or more frequently. But the Abilene flow data indicates that many TCP implementations have ACK ratios that are significantly higher than 2:1. This could happen because of several flaws identified in RFC 2923 [50] and RFC 2525 [23], but this effect is too prevalent to be explained by those defects.

These RFCs use the term “Stretch ACK” to refer to a TCP receiver which generates an ACK less often than every second full-sized segment.

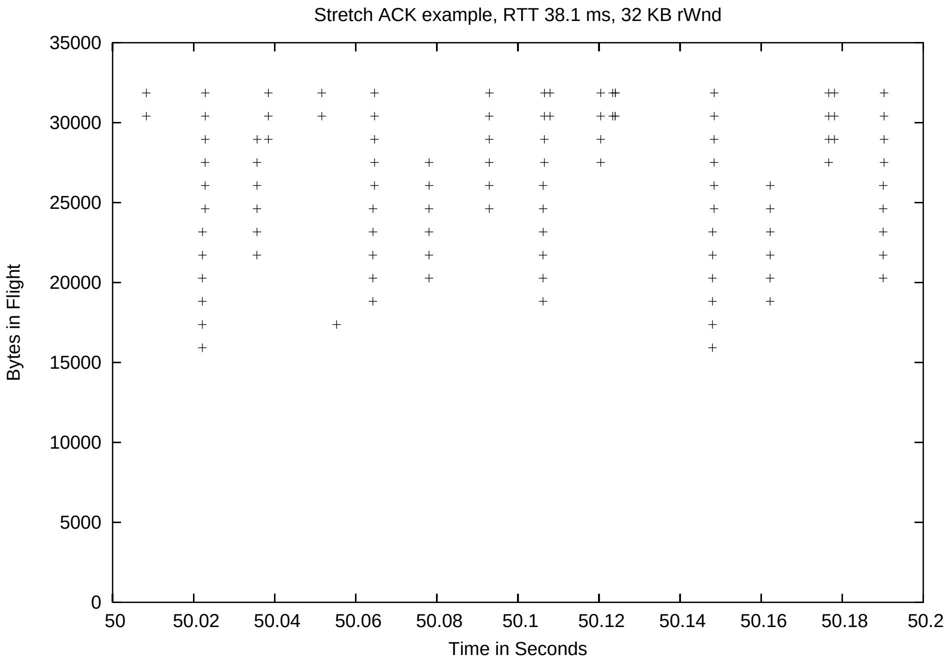

Figure 4.4 Typical Stretch ACK Connection

A typical “stretched ACK” connection is shown in Figure 4.4. In this example, both the send window, sWnd, and the receive window, rWnd, are set to 32K Bytes. The graph shows that the number of bytes in flight varies from a low of 16,000 to a high of 32,000, but that the packets are, again, clumped into vertical bursts. Idle stretches, like the 20 millisecond gap at time 50.14, appear when the sender is waiting for an ACK. This 20 millisecond gap is over 52% of the 38 millisecond RTT. Not shown in the graph is the fact that the ACK that arrived at 50.148 released 6 packets and an ACK slightly later at 50.149 released 6 more.

Stretch ACKs have not been widely studied in the literature because they do not appear in low BDP environments and, even in high BDP environments they do not, in themselves, present a problem. Since the entire path from sender to receiver consists only of high-speed connections, it is likely that the routers in the path have enough buffering to handle the burstiness.

From reading the LINUX 2.4.18 source, we propose that the stretch ACKs seen in Abilene could be caused by timer management and by granularity in dispatching the user-level processes that consume the packets. When the recipient's kernel receives a data packet, the kernel chooses not to send an ACK if the queue to the user-level process is not empty. This obviates the need for an additional timeout (and the overhead associated with adding a timeout to the sorted list of timeouts only to delete it later when the cumulative ACK is sent). The ACK is, instead, generated when the queue to the user-level recipient process becomes empty.

Timer management in LINUX became a major performance issue when LINUX became a popular platform for web servers and proxy caches. Although it is quick to maintain the timers for a dozen simultaneous TCP connections, the overhead of maintaining the myriad TCP timers became a serious scalability limit if hundreds or thousands of simultaneous connection were active.

RFC 3449 [21] describes various techniques to create stretch ACKs as a means of controlling ACK congestion. These techniques have been proposed in environments like cable modems, where the upstream path is significantly narrower than the downstream path and the end user has only a few, limited opportunities to send ACKs. If these techniques are in common practice the model in this thesis will become much less accurate.

Fraction of Achievable Bandwidth

Any memory-limited connection may consume only a fraction of the BDP along its path. An easy way to characterize the intensity of the connection is to compare λ, the bandwidth it is using, with the available bandwidth. For example, a connection using 2 Mbps in a 100 Mbps path is using 2% of the achievable bandwidth.

If a several flows follow the same path through Abilene and have the high ACK ratio associated with a memory-limited λ, then any difference between them has to be explained based on their memory-limit or their RTT. We will assume that stretch ACKs happen in lossless connections. Each connection grows λ until it reaches its memory limit. Then the fraction of time it spends non-idle is the Fraction of Achievable Bandwidth (FAB). Since each flow record has a duration associated with it, we can compute the throughput for that flow record. We further assume that the connection with the highest throughput for that full path through Abilene is the achievable bandwidth.

Evidence of Delayed and Stretched ACKs in Abilene

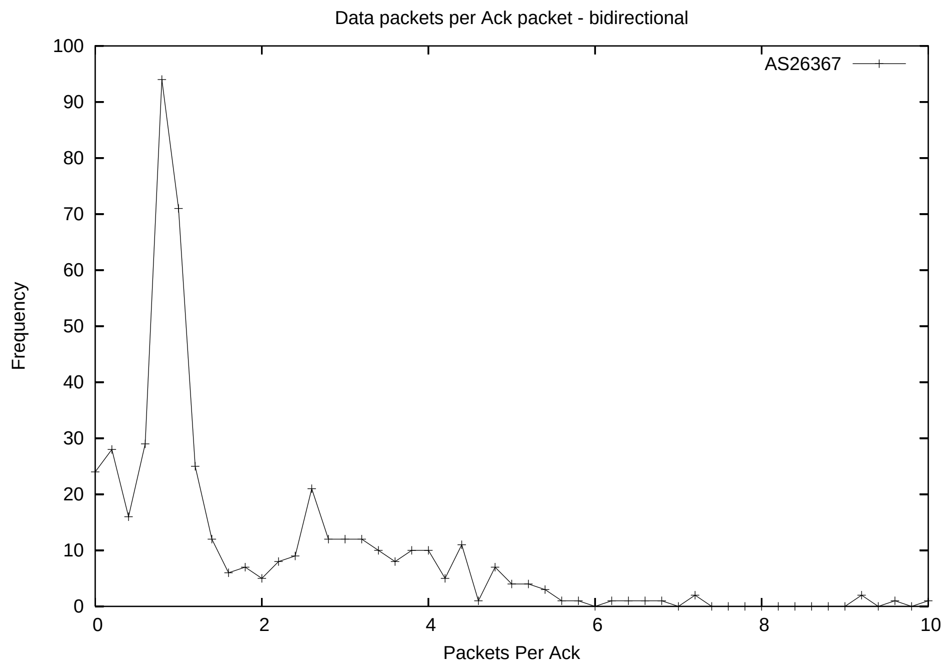

Figure 4.5 Typical Delayed ACK Connection

Figure 4.5 shows the ACK ratios derived from our Abilene flow data for connections from Bradley University. Each flow was matched to its reverse flow by IP address and port. Only flows whose data packets averaged more than 1,200 bytes each, whose ACK packets averaged less than 45 bytes each and with at least 6 data packets and at least 4 ACK packets were considered.

The graph shows a fairly clear bimodal distribution with a large number of connections having an ACK ratio of 1:1, but another set of connections that have ACK ratios between 2:1 and 5:1. Our supposition is that the former are connections on the dialup network that are limited by their congestion windows and the latter are connections that are memory-limited and do not grow their window large enough to cause congestion at any hop in their entire path.

Note that these flows were monitored at Abilene and only contain connections that took at least one Abilene hop. As a result, none of these connections have RTT less than the time it takes to get from Bradley to Abilene's Chicago router and from there to at least one other Abilene router. The minimum RTT for those connections is 4 milliseconds.

Evidence of Memory-Limited Flows from Wisconsin

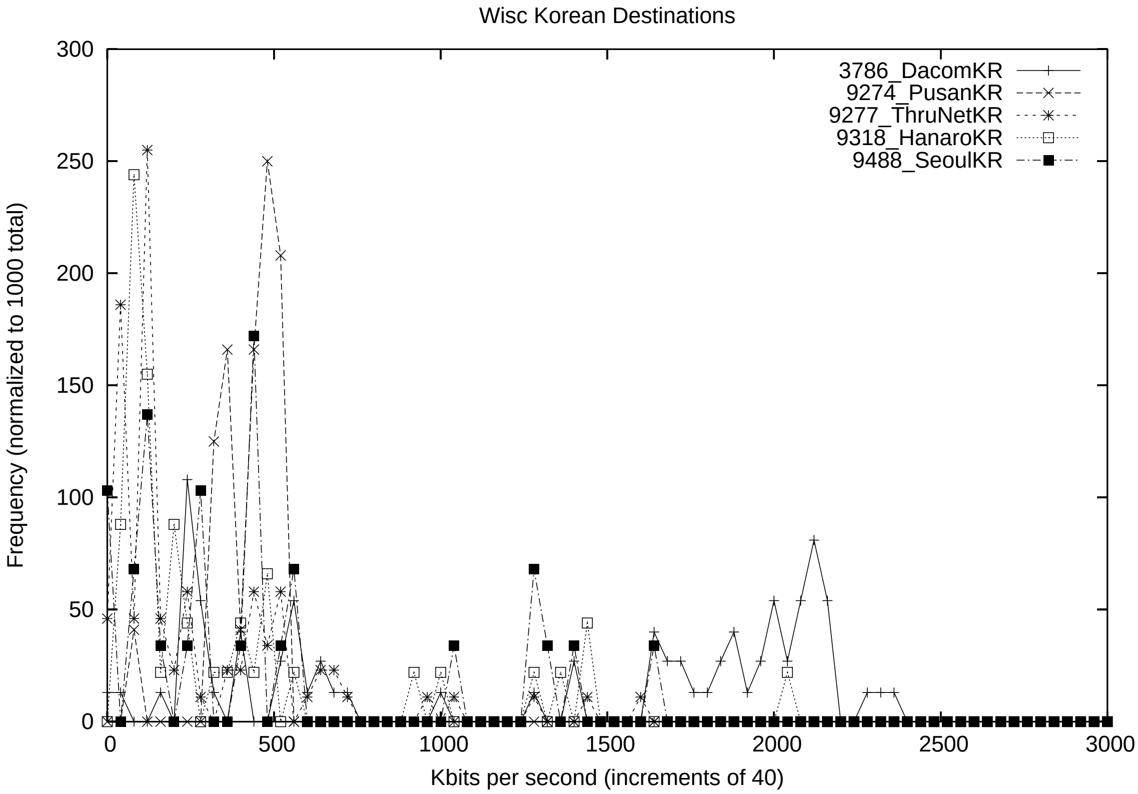

Figure 4.6 Throughput to Selected Korean Destinations from Wisconsin

Graphs 4.7 and 4.6 were gathered from non-sampled flow data at the University of Wisconsin's border router on June 22, 2003. Five Korean domains and six European domains were monitored for one full day. The graphs show the proportion of flows with each throughput in Mbits per second.

Memory-limited flows would consistently reach a throughput inversely proportional to the RTT. If the RTT is relatively stable, the graph of throughput should have tall peaks at each of the popular

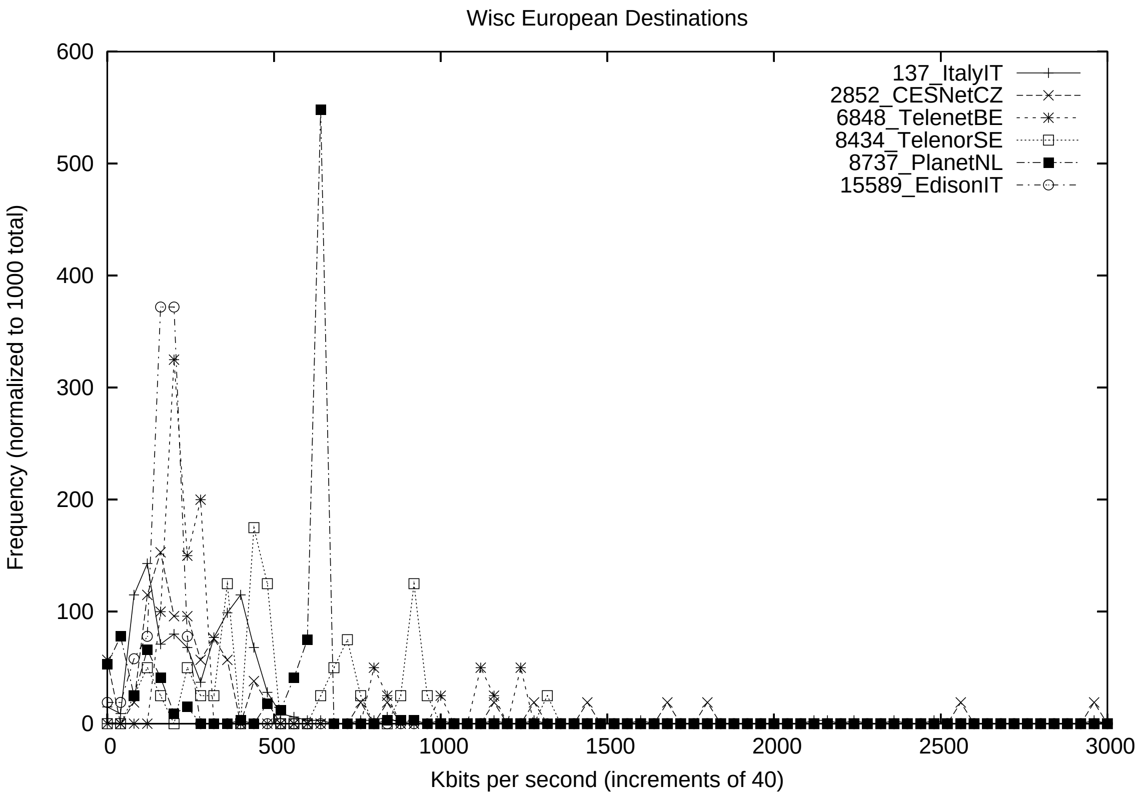

Figure 4.7 Throughput to Selected European Destinations from Wisconsin

window sizes. Simple traceroutes were used to determine actual RTT. The Korean sites ranged from 194 ms RTT to 233 ms RTT except for ThruNet (528 ms). The peaks in the graph at 490 Mbps represent a memory limit at 16 KBytes. This is likely to be the sWnd of the popular mirror.cs.wisc.edu, the most heavily used IP address in our flow data. Other peaks could be the result of other memory limits.

The European destinations show a similar peak at 690 Mbps. This is where it would be expected given the 148 ms RTT to those destinations.

4.4 Coalescing Traffic into Minimal Unique Set

In this section we aggregate flows that are equivalent from the viewpoint of Abilene backbone congestion. Flows that share the same ingress, egress and RTT can be aggregated simply by adding their volume and ceiling.

AS exit points

The autonomous system is a convenient aggregation level for flows. There were 545 AS's mentioned as destinations in the flows captured from Abilene on April 24, 2003. Of those, 470 had a unique exit interface. Even at routers several hops away from their exit, we can be confident that we can predict the entire remaining path of that flow through Abilene.

It would also be possible to classify flows based on the interface they used to enter Abilene. Unfortunately, some flows have spoofed IP addresses and Abilene doesn't have completely accurate ingress filtering. To avoid complications caused by IP address spoofing, we consider an AS to have a unique attachment to Abilene if it has a unique exit point. We assume that the entry point for any AS is the same as the exit point for that AS.

When each flow was aggregated to the AS level, 61.1% of the flows entered Abilene at a known interface and exited Abilene at a known interface.

Table 4.5 shows all autonomous systems that were the destination of more than one percent of Abilene traffic on March 19, 2003. In addition, 2.3% of the bytes passing through Abilene went to routable IP addresses that we were not able to translate to an AS number. Notice that NCSA accounted for a large number of bytes but a very small number of flows and very short duration. We believe that this might have been UDP traffic for a video teleconference. It is interesting to notice that AS's with high byte counts don't necessarily have high flow counts.

Exterior Delay Estimation

We use the estimation from stretch ACKs shown in Section 4.3 to arrange the list of autonomous systems into sorted order. Only TCP flows with more than six packets and more than

Table 4.5 Highest Volume AS Exits

Dest AS	Dest AS Name	Country	Flows%	Octets%	Packets%	Duration%
237	NSFNETTEST14-AS	US	4.083	3.928	3.911	3.652
81	CONCERT	US	3.316	3.457	3.832	3.386
786	JANET	UK	2.021	2.414	1.959	2.154
680	DFN-WIN-AS	DE	1.968	2.157	1.933	2.269
17	PURDUE	US	1.826	2.147	2.443	2.957
137	ITALY-AS	IT	0.994	1.792	1.244	1.407
32	STANFORD	US	1.937	1.613	1.738	1.885
3999	PENN-STATE	US	1.646	1.612	1.704	1.849
2150	CSUNET-SW	US	1.405	1.468	1.251	1.323
87	INDIANA-AS	US	1.553	1.419	2.095	2.606
55	UPENN-CIS	US	1.695	1.411	1.519	1.384
2637	GEORGIA-TECH	US	1.188	1.384	1.372	1.184
27	UMDNET	US	1.879	1.356	1.781	1.953
3582	UONET	US	0.499	1.266	1.061	0.832
111	BOSTONU-AS	US	1.629	1.260	1.787	2.091
3	MIT-GATEWAYS	US	0.710	1.189	1.157	0.803
3794	TAMU	US	0.888	1.178	0.940	1.005
7377	UCSD	US	1.299	1.168	1.298	1.740
2572	MORENET	US	0.866	1.045	0.855	0.800
1224	NCSA-AS	US	0.061	1.024	0.528	0.037

four ACKs were considered. Because of the sampling factor (1:100), we can assume that those flows were long-lived (at least 400 data packets and at least 200 ACK packets). Flow matching was only done within a 5-minute flow file.

Flows with 4:5 data:ACK ratios were considered to be bandwidth limited either before or after Abilene. Nearly equal data:ACK ratios indicate that the data packets are arriving more seldom than the delayed ACK timeout. This implies that the flows are limited by their congestion windows due to consistent pacing losses. This would be typical if the connections were dial-up modems (56 kbps) or that they share a very tight link (typically T1 speed). We draw no conclusions about RTT for these flows.

Flows with data:ACK ratios above 4:5 but below 11:5 are flows that may be memory limited at the sender or receiver and using delayed ACKs. Or they could be limited by a congestion window that is large enough to permit a high percentage of delayed ACKs. We draw no conclusions about RTT from these flows.

Long-lived flows with data:ACK ratios above 11:5 are likely to be memory limited, rather than congestion limited. The loss recovery mechanisms break up stretch ACKs and it takes many rounds for the ACKs to stretch again. Assuming that several connections to the same source (typically a web server, P2P server, an FTP server, or a similar constantly-willing source of large packets) have the same send-window memory limit, the difference between their speeds will be strictly due to differences in RTT. In particular, if two connections have stretch ACKs and one has one-third as much throughput, it has triple the RTT.

As of 2003, memory windows above 64K Bytes or below 16K Bytes are very rare. Since this range is small, we argue it is justified to assume that memory-limited connections hold a fixed number of bytes in flight at all times. We estimated this number to be 32K bytes per window. This allows them to reach a window size of 21 packets, putting them well into the area where TCP's loss recovery mechanisms are effective and efficient. A single lost packet would, at worst, cause the window to drop to 10 packets, allowing the connection to grow back to a window of 21 packets in 11 RTT rounds.

Table 4.6 Achievable Bandwidth At 32 KByte Memory Limit, 1500 Byte Packets

RTT	Windows Per Second	Bits Per Second
0.001	1,000	262,144,000
0.002	500	131,072,000
0.010	100	26,214,400
0.020	50	13,107,200
0.100	10	2,621,440
0.200	5	1,310,720

So the RTT estimation is made based on comparisons of the throughput of long-lived flows with data:ACK ratios above 11:5. Each flow has a duration and a number of bytes seen by the sampler. Flows with 9K Bytes or more sampled are assumed to have at lasted for at least 500K Bytes. Table 4.6 shows how throughput relates to RTT. Flows with RTT less than 2 ms are unlikely to be memory-limited. Flows with RTT 200 ms will still be able to achieve a window of 21 packets giving a throughput of 1.3 Mbps.

An AS with a higher incidence of stretch ACKs is assumed to be closer to its Abilene attachment point. This is because the lowest RTT at which stretch ACKs occur is lower for this AS than for others. We further assume that any packet that crosses the Abilene backbone will travel, at minimum, double the distance of the shortest Abilene link. For example, Chicago to Indianapolis is 210 miles. Light through fiber travels at approximately 66% of the speed of light. Even if there were no time spent getting to the Chicago Abilene site or going from the Indianapolis site, the minimum RTT for a connection would be over 2 milliseconds.

Categories of Exterior Delay

It would be inappropriate to assume high precision in the RTT estimation since the data used to make the determination is a small fraction of the total traffic. We chose to use only two broad categories of Exterior Delay with the intention that one category would represent AS's in or near the same city as an Abilene router, another category would represent AS's that anywhere from regional to trans-oceanic.

Table 4.7 shows a few examples from the list of equivalence sets. Although MIT might disagree, we considered Harvard and MIT to be equivalent in the sense that their attachment to Abilene was uniquely $nycm$ and they both had similar experience with respect to delayed and stretched ACKs. All packets destined for the Russian Federal Universities Network (AS 3267) also exit Abilene at $nycm$, but their traffic shows a much higher proportion of data packets per ACK packet. The Network Information Service Center (AS 22) has blocks of IP addresses that exit Abilene at different points. As a result, it is not simple to look at the destination AS for a flow and determine it's exit point. We list AS 22 as $ambig$.

Table 4.7 Sample Assignment of AS Numbers to Equivalents

ASNum	equiv	Name	Country
3	nycmN	MIT-GATEWAYS	US
8	hstnN	RICE-AS	US
9	washN	CMU-ROUTER	US
11	nycmN	HARVARD	US
16	snvaN	LBL	US
17	iplsN	PURDUE	US
18	hstnN	UTEXAS	US
22	ambig	NOSC	US
25	snvaN	UCB	US
27	washN	UMDNET	US
29	nycmN	YALE-AS	US
32	snvaN	STANFORD	US
34	washN	UDELNET	US
3267	nycmF	RUNNET	RU
4671	sttlF	GCC-KR	KR
6262	sttlF	CSIRO	AU

Ceiling Estimation

We estimate the ceiling of a flock by adding up the throughput of the connections in that flock. The same filter is applied as in Section 4.4. This ensures that only long-lived flows (> 6 sampled data packets) are considered. Each of those flow records has a λ_r throughput rate in bytes per second. Each record, r, is assigned to a flock based on the equivalence classes of its source and destination. The set of all flow records in flock f is $FlowRec_f$. To correctly sum the throughput rates, λ_r, we have to adjust each one by the ratio of their duration, $Duration_r$, to the duration of the measurement period, $M = 300$ seconds. The total ceiling of all flow records for a given flock,

$$BpsCeiling_f = \sum_{r \in FlowRec_f} \frac{(Duration_r \lambda_r)}{M}$$

This ceiling estimate has inherent inaccuracies. It is derived from sampled data, does not include non-TCP flows, does not include short flows, and does not include traffic from ambiguous sources or destinations. Moreover, the sampling understates the duration of a flow.

The elements of $Ceiling_f$ are then computed from $BpsCeiling_f$ so that they represent packets per tick rather than bytes per second. The selection of a scale factor is sensitive. We scaled the $Ceiling$ vector so that the mean on the busiest link in our Abilene model matched the link utilization at the same time of day in the actual Abilene network. As shown in Figure 4.2, the link from Chicago (CHIN) to Indianapolis (IPLS) had 1.1 Gbps of traffic on test day. The total of all flow records in that direction on that link was 217 Mbps. The ratio of bits per second seen in the flow records to bitsPerSecond from the Abilene weather map is $samplingScale = 5.069$. Each tick is 10 ms, so $ticksPerSecond = 100$. The number of bytes per modeled data packet is $bytesPerPacket = 1500$. So, ceiling values are converted to packets per tick by the formula:

$$Ceiling_f = \frac{samplingScale * BpsCeiling_f}{ticksPerSecond * bytesPerPacket}$$

Thus, an example flow record at 100,000 bytes per second would contribute 3.37 packets per tick using the formula:

$$Ceiling_f = \frac{5.069 BpsCeiling_f}{150000}$$

Simplified Traffic Matrix

Table 4.8 Excerpt from Model Traffic Matrix Estimate

	dest ambig	chinF	chinN	iplsF	iplsN	losaF	losaN	snvaF	snvaN	Other	Total
ambig											
chinF						8				14	22
chinN					11	10		14	23	51	109
iplsF			7							7	14
iplsN		9					19		16	50	94
losaF		19	16	7						42	84
losaN			23							26	49
snvaF					12					19	31
snvaN					12					20	32
Other		6	34	3	20	9	16	4	6	467	565
Total		34	80	10	55	27	35	18	45	696	1000

Table 4.8 shows the final simplification of the traffic matrix based on AS equivalents. All traffic to or from ambiguous AS's is removed, values are normalized so that total (unambiguous) volume is 1000, all values are rounded to the nearest integer, and values smaller than an arbitrary minimum, $\delta = 3$, are merged with a larger flow.

Again, the row and column marked "Other" are purely an artifact of showing the table succinctly in this thesis. Non-zero values in the matrix represent the volume of traffic that must be emulated to present a load to the Abilene emulation that approximates the round-trip times and volumes in the flow data. Blanks are present where the volume is smaller than the minimum δ and values have been aggregated into other flows. The table does not show the ceilings of the flows in the model set.

4.5 Traffic Matrix Summary

We have demonstrated that a succinct traffic matrix can be constructed that greatly simplifies representation of the flows that pass through Abilene for each 5 minute period of a day in the life of Internet2.

Two crucial parameters for reproducing the behavior of large flows were difficult to obtain from the vendor statistics gathered from Abilene equipment. Those were the RTT of the flows and the ceilings (often mis-named external bottleneck bandwidth) of those flows. We showed that both could be inferred from flow data captured in Abilene by noticing delayed ACK counts and stretch ACK counts.

The traffic matrix includes parameters that will allow it to be used in explorations of traffic increases, link additions and link outages. As the demand on Abilene begins to use connections with higher memory limits or with more multiplexing, these compositional changes in traffic characteristics can be easily accommodated to create a new traffic matrix to run against the model in Chapter 3.

Additional nodes can be added to the model, but any traffic migration from old nodes to new nodes and any additional traffic starting or ending at the new nodes would have to be added.

Traffic Matrix Future Work

The traffic matrix forms the basis for delivering traffic to a laboratory-based Abilene emulation. To apply the traffic to actual routers, PCs will have to accurately emulate the quantity and composition of the traffic from each source equivalence class to each destination equivalence class. Delays will be needed before entering the Abilene cloud, inside the cloud, and after exiting the cloud. Monitoring and measurement will be needed to see if the loss rates and queue delays accurately reflect those given by the actual Abilene network. This effort will be difficult partly because the actual Abilene network is very fast, it is difficult to separate out Abilene queuing delay from other delays, and many Abilene links are nearly lossless.

A major goal of the traffic matrix estimation project was to study the effect of window synchronization to validate that a flock-based model has sufficient texture to predict the likelihood that congestion events will be successful. Traffic engineering that can avoid chronic congestion is a worthy goal. If the traffic matrix causes the model to predict congestion events of comparable duration and intensity to the actual Abilene, it will be powerful traffic engineering tool. To do this, we will need to find ways to isolate and measure bursts of losses in both the actual Abilene and the emulated Abilene.

Much work is needed to validate that the traffic matrix is itself accurate enough for congestion study. Round trip times are easily measured and the total volume of traffic is straight-forward. But, the addition of flow record rates to create a $Ceiling_f$ for each flock is problematic. If future work could test the reaction of flocks to congestion events, we could watch the rate at which the traffic grows back after the event. This improved understanding of the traffic's elasticity and ability to accelerate back to its ceiling would help us validate or improve our computation for $Ceiling_f$.

The current traffic matrix does not include the $Floor_f$ used to indicate the unwillingness of the flock to go below a lowest traffic rate. A significant fraction of Abilene traffic is *open loop* traffic that is either non-responsive to congestion signaling or so short-lived that the response is insignificant. This includes constant bit rate traffic, ICMP and most UDP traffic, and short connections. Discovering a mechanism to measure $Floor_f$ would improve the accuracy of the model in Chapter 3. It would be particularly useful to measure long-lived unresponsive open loop traffic. The proposals for active queue management that disproportionately drop packets from non-responsive flows could be validated in an emulation setting if we knew how much volume was non-responsive in Abilene.

Chapter 5

Related Work

In this chapter we discuss the studies that are basis for our investigation of global Internet topology and traffic. Much of the pioneering work has been done by simulating busy links at the packet-level with repeatable sources of data. These provided substantial insight into the dynamics of TCP connections or the statistics of packet-level and connection-level behavior.

Our work is particularly informed by the early topology studies using BGP tables to try to draw useful graphs of the global Internet. Researchers wanted to visualize the Internet and wanted to model the Internet using simple rules about out-degrees.

We are also indebted to the researchers who created the tools that we used to simulate the Internet, to trace routes through the Internet, and to measure flows through the Internet. No listing of related work would be complete without giving credit to the writers of the flowtools and to the many operators who allow their servers to be used as traceroute servers.

5.1 Topology Related Work

Both router level and inter-domain topology have been studied over the past five years [37, 67, 86, 36, 24]. Our clustering algorithm uses BGP data thus inter-domain topology is most relevant to this work. In [36], Govindan and Reddy characterize inter-domain topology and route stability using BGP routing table information collected over a one year period. In that work the authors describe inter-domain topology in terms of diameter, degree distribution and connectivity characteristics. Inter-domain routing information can be collected from a number of public sites including NLANR [32], Merit [38] and Route Views [90] (our source of routing information). These sites provide BGP tables from looking glass routers located in various places in the Internet and peered with a large number in ISP's.

Routing characteristics have also been widely studied in the context of topology. Examples include [3, 36, 70]. These studies inform our work with respect to the structural characteristics of end-to-end Internet paths.

Clustering, Caching and Content Delivery

Our clustering algorithm is analogous to generating a spanning tree for the AS graph. Prim's algorithm [75] is a standard method for constructing a minimum spanning tree if the root of the tree is known in advance. Starting at the root, use a breadth-first search to find all nodes. Each edge that lies on a shortest path from the root to any other node is a member of the minimum spanning tree. We cannot use Prim's algorithm since our graph does not have a pre-defined root. Kruskal's algorithm [48] does not require a starting point. It constructs a spanning forest that initially contains a tiny tree for each vertex. Trees are then combined by coalescing them at the shortest edges first. Any edge that does not cross between trees is redundant and any edge left over after all of the vertices have been visited is similarly not needed. Although Kruskal's algorithm serves as the inspiration for our algorithm, we still had to address the stopping criteria, since declaring a single root for the entire Internet would have artificially added several hops in the core of the Internet, where a dozen of the biggest transit providers are almost completely interconnected. Kruskal's algorithm finds a minimal spanning tree in the sense that the total of the edge lengths is minimized, even if it makes the tree deep. Our goal was subtly different, since we want a tree that has maximum fidelity to the traffic flow in the Internet. In particular, we want a shallow tree so that node representations are not mistakenly far from the backbone. Even outside the core, Kruskal's algorithm produces trees that are inappropriately deep when presented with neighborhoods of completely interconnected vertices.

Initial work on clustering clients and proxy placement was done by Cuñha in [18]. That work described a process of using *traceroute* to generate a tree graph of client accesses (using IP addresses collected from a Web server's logs). Proxies were then placed in the tree using three different algorithms and the effects on reduction of server load and network traffic were evaluated. Our work differs from this in our use of AS level information from BGP routing tables to create a tree which is simpler and more efficient. Our cache placement algorithms differ in that the coarser aggregation allows us to use a method that guarantees optimal placement. The next significant work on client clustering was done by Krishnamurthy and Wang in [47]. In that work, the authors merge the longest prefix entries (i.e.. those with the most detail) from a set of 14 BGP routing tables. This creates a prefix/netmask table of approximately 390K possible clusters. IP addresses from Web server logs are then clustered by finding the longest prefix match in the prefix/netmask table. While this approach generates client clusters which are topologically close and of minimal size, it does not provide for further levels of aggregation of clusters.

Content distribution companies (e.g.. Akamai) and wide area load balancing product vendors (e.g.. Cisco, Foundry and Nortel) also use the notion of client clustering to redirect client requests to distributed caches. These companies use the Domain Name System (DNS) [61] as a means for both determining client location and redirecting requests. The assumption made in DNS-redirection is that clients whose DNS requests come from the same DNS server are topologically close to each other. Initial work in [46] evaluates the performance of redirection schemes that access documents from multiple proxies versus a single proxy and shows that retrieving embedded objects from a single page from different servers is sub-optimal. Subsequent work in [85] indicates that clients and their nameservers are frequently *neither* topologically close *nor* close from the perspective of packet latency. However, Myers *et al.* show that the ranking of download times of the same three sites from 47 different mirrors was stable [62].

Caching has been widely studied as a means for enhancing performance in the Internet during the 1990's. These studies include cache traffic evaluation [6, 8], replacement algorithm performance [91, 19], cache hierarchy architecture [34, 58] and cache appliance design [11, 15]. A number of recent papers have addressed the issue of proxy placement based on assumptions about the underlying topological structure of the Internet [52, 43, 79]. In [52], Li *et al.* describe an optimal dynamic programming algorithm for placing multiple proxies in a tree-based topology. Their algorithm is comparable to ours although it is less efficient. It places M proxies in a tree with N nodes and operates in $O(N^3M^2)$ time where as our algorithm operates in $O(NM^2logN)$. Jamin *et al.* examine a number of proxy placement algorithms under the assumption that the underlying topological structure is not a tree. Their results show quickly diminishing benefits of placing additional mirrors (defined as proxies which service all client requests directed to them) even using sophisticated and computationally intensive techniques. In [79], Qiu *et al.* also evaluate the effectiveness of a number of graph theoretic proxy placement techniques. They find that proxy placement that considers both distance and request load performs a factor of 2 to 5 better than a random proxy placement. They also find that a greedy algorithm for mirror placement (one which simply iteratively chooses the best node as the site for the next mirror) performs better than a tree based algorithm.

5.2 Backbone Delay and Loss Related Work

Packet delay and loss behavior in the Internet has been widely studied. Examples include [5] which established basic properties of end-to-end packet delay and loss based on analysis of active probe measurements between two Internet hosts. That work is similar to ours in terms of evaluating different aspects of packet delay distributions. Paxson provided one of the most thorough studies of packet dynamics in the wide area in [72]. While that work treats a broad range of end-to-end behaviors, the sections that are most relevant to our work are the statistical characterizations of delays and loss. The important aspects of scaling and correlation structures in local and wide area packet traces are established in [51, 73]. Feldmann *et al.* investigate multifractal behavior of packet traffic in [26]. That simulation-based work identifies important scaling characteristics of packet traffic at both short and long timescales. Yajnik *et al.* evaluated correlation structures in loss events and developed Markov models for temporal dependence structures [92]. Recent work by Zhang *et al.* [80] assesses three different aspects of *constancy* in delay and loss rates.

There are a number of widely deployed measurement infrastructures which actively measure wide area network characteristics [77, 63, 55]. These infrastructures use a variety of active probe tools to measure loss, delay, connectivity and routing from an end-to-end perspective. Recent work by Pasztor and Veitch identifies limitations in active measurements, and proposes an infrastructure using the Global Positioning System (GPS) as a means for improving accuracy of active probes [69]. That infrastructure is quite similar to Surveyor [77] which was used to gather data used in our study.

A variety of methods have been employed to model network packet traffic including queuing and auto-regressive techniques [42]. While these models can be parameterized to recreate observed packet traffic time series, parameters for these models often do not relate to network properties. Models for TCP throughput have also been developed in [54, 65, 16]. These models use RTT and packet loss rates to predict throughput, and are based on characteristics of TCP's different operating regimes. Our work uses simpler parameters that are more directly tuned by traffic engineering.

Fluid-Based Analysis

Determining the capacity of a network with multiple congested links is a complex problem. Misra proposed fluid-based analysis [60] employing stochastic differential equations to model flows almost as though they were water pressure in water pipes. Bu used a fixed-point approach [10] that focuses on predicting router average queue lengths. Both methods are fast enough to use in “what if” scenarios for capacity planning or performance analysis. Both methods take, as input parameters, a set of link capacities, the associated buffer capacities, and a set of sessions where each session takes a path that includes an ordered list of links. Our model uses essentially the same input parameters. We expect that the results of these models would be complementary to our results and suggest that traffic engineers use one fluid-based analysis to compare to our window synchronization model. Our expectation is that fluid-based analyses might overstate capacity when our model would understate.

Other Forms of Global Synchronization

The tendency of traffic to synchronize was first reported by Floyd and Jacobson [29]. Their study found resonance at the packet level when packets arrived at gateways from two nearly equal senders. Deterministic queue management algorithms like drop-tail could systematically discriminate against some connections. This paper formed the earliest arguments in favor of RED. This form of global synchronization is the synchronization of losses when a router drops many consecutive packets in a short period of time. Fast retransmit was added to TCP to mitigate the immediate effects. The next form of global synchronization was synchronization of retransmissions when the TCP senders retransmit dropped packets virtually in unison.

In contrast, window synchronization is the alignment of congestion window saw-tooth behavior. Packet level resonance was never shown to extend to more than a few connections. Qui, Zhang and Keshav [80] found that global synchronization can result when a small number of connections share a bottleneck at a slow link with a large buffer, independent of the mixture of RTTs. Increasing the number of connections prevents the resonance. Window synchronization is the opposite.

Window synchronization scales to large numbers of connections, but a broad mixture of RTTs prevents the resonance.

5.3 Related Work in Traffic Matrix Estimation

Much of the prior work on traffic matrix estimation starts with the assumption that sources and destinations are not known. Our work differs in that the volume of data is directly read from flow data, rather than trying to find a way to infer volume from link utilization and other SNMP statistics. In contrast, this thesis focused on ways to infer exterior delay and exterior ceilings for connections. Very few techniques have been proposed that take into account the way TCP (and TCP friendly) connections react to changes in the interior of the network.

Linear Programming Approach

Goldschmidt [35] suggested an innovative technique for discovering a set of source-to-destination flows that satisfy a list of link utilizations. Goldschmidt saw this as an optimization problem and posed a linear program (LP) to attempt to compute the traffic matrix directly. Since there are an infinite number of feasible solutions that correctly satisfy the link utilizations, Goldschmidt imposes a linear constraints on the solution based on the differences across time. Subsequent researchers [57] found that the technique produced error rates that were "probably too high to be acceptable by ISPs" and that the technique is "highly sensitive to noise" in the raw input data.

Gravity Modeling Approach

Zhang *et al.* [95] developed a very fast technique for estimating the traffic matrix using gravity modeling. If one simply assumes a proportionality relationship between the total traffic entering the network and the total traffic leaving the network at each perimeter point, the points in the interior can be inferred. Starting with the edges, they incorporate both BGP data exchanged with peer networks and routing information about the interior of the ISP. The relative strength of the interaction between any two nodes is modeled as though they had gravity according to Newton's law of gravitation. They call this mixture of gravity techniques and tomography techniques *tomogravity*.

It would be interesting to use Zhang's techniques to model more than simply the traffic volume. Gravity techniques could be very useful in estimating the demands that would move to a new node if a node were to be added to our existing Abilene model.

Expectation Maximization Approach

Cao *et al.* [14] incorporated multiple sets of link measurements and, assuming these were IID variables. There are many situations where a maximum likelihood estimate (MLE) is not straightforward due to the absence of data. So they applied an Expectation Maximization (EM) algorithm that provides an iterative procedure for computing MLEs. That, in turn, led them to the problem of estimating an initial matrix prior to initiating the iterative procedure.

This also could be interesting work if it could be turned to the problem of estimating connection ceiling throughput.

Reaction of TCP to Congestion

Memory limitations were quickly recognized as an impediment to throughput, but the research community lost interest after RFC 1323 [41] established a mechanism for window scaling. This allowed a single TCP connection to have a very large amount of data (potentially 2^{30} bytes) unacknowledged in transit. It is now possible to allocate large amounts of memory for a single TCP connection, but the default settings for popular operating systems are typically much smaller.

Stretch ACKs

Delayed ACKs and Stretch ACKs have been common in TCP since RFC 1122. Although that RFC was written clearly, there was a period of confusion among the vendors when it was not clear if stretch ACKs were considered legal. RFC 2525 [23] discusses specific bugs that cause stretch ACKs and describes the impact of stretch ACKs. RFC 2581 [88] establishes that a TCP implementation *SHOULD* generate an ACK for at least every second full-sized segment. RFC 2581 unambiguously states that an implementation may generate ACKs less frequently "after careful consideration of the implications".

The importance of adding RTT into any study of TCP congestion has been widely reported [2, 30] and cannot be overstated.

Parameters in the Traffic Matrix Estimate

Medina *et al.* [57] use *choice models* in the Sprint Network Analysis Toolkit to generate high quality starting points to improve the behavior of earlier statistical techniques. The choice model acts as though each ingress node chooses an egress node for each packet so as to maximize a utility function. The combination of features of an egress POP (total capacity, number of customers / peers, etc.) make it more or less attractive to a particular ingress node. The results were applied to a tier-1 ISP and found to be appropriately accurate to match actual Internet volume data.

This effectively increased the number of parameters in the model space to include information about each egress. While our thesis has already increased the parameter space by adding RTT and ceiling, we can imagine ways to further improve accuracy by characterizing egress points by the composition of the traffic they attract. For example, an egress point that is popular for streaming video may be statistically very different from an egress point that emphasizes very short HTTP transactions.

LIST OF REFERENCES

[1] Internet2 Abilene Project. http://abilene.internet2.edu, 2003.

[2] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, March 2000.

[3] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proceedings of ACM SIGCOMM '99*, Boston, MA, September 1999.

[4] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way delay metric for ippm. RFC 2679, September 1999.

[5] J. Bolot. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of ACM SIGCOMM '93*, San Francisco, Setpember 1993.

[6] H. Braun and K. Claffy. Web traffic characterization: An assessment of the impact of caching documents from NCSA's Web server. In *Proceedings of the Second International WWW Conference*, Chicago, IL, October 1994.

[7] H. Braun, K. Claffy, and G. Polyzos. A framework for flow-based accounting on the internet. In *Singapore International Conference on Networks, SICON93*, Singapore, 1993.

[8] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[9] A. Broido and kc claffy. Internet topology: connectivity of IP graphs. Technical report, CAIDA, http://www.caida.org/outreach/papers/topologylocal, 2001.

[10] T. Bu and D. Towsley. Fixed point approximations for TCP behavior in an AQM network. In *Proceedings of ACM SIGMETRICS '01*, 2001.

[11] Squid Internet Object Cache. http://www.nlanr.net/squid, 2001.

[12] J. Cao, W. Cleveland, D. Lin, and D. Sun. The effect of statistical multiplexing on the long range dependence of Internet packet traffic. *Bell Labs Tech Report*, 2002.

[13] J. Cao, W. Cleveland, D. Lin, and D. Sun. Internet traffic: Statistical multiplexing gains. *DIMACS Workshop on Internet and WWW Measurement, Mapping and Modeling, 2002*, 2002.

[14] J. Cao, D. Davis, S. Vanderweil, and B.Yu. Time-Varying network tomography. In *Journal of the American Statistical Association*, 2000.

[15] P. Cao, J.Zhang, and K. Beach. Active cache: Caching dynamic contents on the Web. *Distributed Systems Engineering*, 6(1), 1999.

[16] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *Proceedings of IEEE INFOCOM '00*, Tel-Aviv, Israel, March 2000.

[17] H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Towards capturing representative AS-level Internet topologies. In *ACM SIGMETRICS*, 2002.

[18] C. Cuñha. *Trace Analysis and its Applications to Performance Enhancements of Distributed Information Systems*. PhD thesis, Boston University, 1997.

[19] J. Dilly and M. Arlitt. Improving proxy cache performance: Analysis of three replacement policies. *IEEE Internet Computing*, 3(6), November 1999.

[20] D. Clark et. al. Looking over the fence at networks, a neighbors view of networking research. *Sigcomm*, 2001.

[21] H. Balakrishnan et al. TCP performance implications of network path asymmetry. RFC 3449, 2002.

[22] R. Braden et al. Requirements for internet hosts – communications layers. IETF RFC 1122, 1989.

[23] V. Paxson et al. Known TCP implementation problems. RFC 2525, 1999.

[24] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of ACM SIGCOMM '99*, Boston, Massachusetts, September 1999.

[25] A. Feldmann, A. Gilbert, W. Willinger, and T. Kurtz. The changing nature of network traffic: Scaling phenomena. *Computer Communications Review*, 28(2), April 1998.

[26] A. Feldmann, P. Huang, A. Gilbert, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM '99*, Boston, MA, September 1999.

[27] S. Floyd. Connections with multiple congested gateways in packet-switched networks part 1: One-way traffic. *ACM Computer Communications Review*, 21(5):30–47, Oct 1991.

[28] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[29] S. Floyd and V. Jacobson. Traffic phase effects in packet-switched gateways. *Journal of Internetworking:Practice and Experience*, 3(3):115–156, September, 1992.

[30] S. Floyd and E. Kohler. Internet research needs better models. *Hotnets I*, Oct 2002.

[31] S. Floyd and V. Paxson. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

[32] National Laboratory for Applied Network Research. http://www.nlanr.net, 1998.

[33] L. Gao. On inferring autonomous system relationships in the internet. In *IEEE Global Internet Symposium*, November 2000.

[34] S. Glassman. A caching relay for the World Wide Web. *Computer Networks and ISDN Systems*, 27(2), 1994.

[35] O. Goldschmidt. ISP backbone traffic inference methods to support traffic engineering. In *Internet Statistics and Metrics Workshop '00*, San Diego, California, USA, December 2000.

[36] R. Govindan and A. Reddy. An analysis of internet inter-domain topology and route stability. In *Proceedings of IEEE INFOCOM '97*, Kobe, Japan, April 1997.

[37] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *Proceedings of IEEE INFOCOM '00*, April 2000.

[38] Merit Internet Performance Measurement and Analysis Project. http://nic.merit.edu/ipma/, 1998.

[39] Internet Protocol Performance Metrics. http://www.ietf.org/html.charters/ippm-charter.html, 1998.

[40] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, pages 314–332, August 1988.

[41] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. IETF RFC 1323, May 1992.

[42] D. Jagerman, B. Melamed, and W. Willinger. *Stochastic Modeling of Traffic Processes*. Frontiers in Queuing: Models, Methods and Problems, CRC Press, 1996.

[43] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, April 2001.

[44] S. Kalidindi. OWDP implementation, v1.0, http://telesto.advanced.org/ kalidindi, 1998.

[45] S. Kalidindi and M. Zekauskas. Surveyor: An infrastructure for internet performance measurements. In *Proceedings of INET '99*, June 1999.

[46] J. Kangasharju, K. Ross, and J. Roberts. Performance evaluation of redirection schemes in content distribution networks. In *Proceedings of 5th Web Caching and Content Distribution Workshop*, Lisbon, Portugal, June 2000.

[47] B. Krishnamurthy and J. Wang. On network aware clustering of Web clients. In *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweden, September 2000.

[48] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, 1956.

[49] Wisconsin Advanced Internet Lab. http://wail.cs.wisc.edu, 2002.

[50] K. Lahey. TCP problems with path MTU discovery. RFC 2923, 2000.

[51] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, pages 2:1–15, 1994.

[52] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby. On the optimal placement of Web proxies in the Internet. In *Proceedings of IEEE INFOCOM '99*, New York, New York, March 1999.

[53] M. Mathis and J. Mahdavi. Enabling high performance data transfers; http://www.psc.edu/networking/perf_tune.html, 2003.

[54] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

[55] W. Matthews and L. Cottrell. The PINGer Project: Active Internet Performance Monitoring for the HENP Community. *IEEE Communications Magazine*, May 2000.

[56] M. May, T. Bonald, and J-C. Bolot. Analytic evaluation of RED performance. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Isreal, March 2000.

[57] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. In *SIGCOMM 2002*, August 2002.

[58] S. Michel, K. Nguyen, A. Rosenstein, S. Floyd, and V.Jacobson. Adaptive Web caching: Towards a new global caching architecture. In *Proceedings of the 3rd Web Caching Workshop*, Manchester, England, June 1998.

[59] J. S. Mill. A system of logic, ratiocinative and inductive: Being a connected view of the principles of evidence, and methods of scientific investigation. J.W. Parker, London, 1843.

[60] V. Misra, W. Gong, and D. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *SIGCOMM*, pages 151–160, 2000.

[61] P. Mockapetris. Deomain names - concepts and facilities. IETF RFC 1034, November 1987.

[62] A. Myers, P. Dinda, and H. Zhang. Performance characteristics of mirror servers on the Internet. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[63] NLANR Acitve Measurement Program - AMP. http://moat.nlanr.net/AMP.

[64] Abilene NOC. http://loadrunner.uits.iu.edu/weathermaps/abilene, 2003.

[65] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, Setpember 1998.

[66] V. Padmanabhan, L. Qiu, and H. Wang. Server-based inference of internet link lossiness. In *Infocomm '03*, 2003.

[67] J.-J. Pansiot and D. Grad. On Routes and Multicast Trees in the Internet. *Computer Communications Review*, 28(1), January 1998.

[68] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot. Analysis of measured single-hop delay from an operational backbone network. In *Proceedings of IEEE INFOCOM '02*, March 2002.

[69] A. Pasztor and D. Veitch. A precision infrastructure for active probing. In *PAM2001, Workshop on Passive and Active Networking*, Amsterdam, Holland, April 2001.

[70] V. Paxson. End-to-end routing behavior in the Internet. In *Proceedings of ACM SIGCOMM '96*, Palo Alto, CA, August 1996.

[71] V. Paxson. End-to-end Internet packet dynamics. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[72] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California Berkeley, 1997.

[73] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[74] D. Plonka. FlowScan: A network traffic flow reporting and visualization tool. In *LISA 2000*, December 2000.

[75] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[76] The Netcity Project. http://www.cs.wisc.edu/netcity, 2001.

[77] The Surveyor Project. http://www.advanced.org/surveyor, 1998.

[78] The Web100 Project. $http://www.web100.org$, 2002.

[79] L. Qiu, V. Padmanabhan, and G. Voelker. On the placement of Web server replicas. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, April 2001.

[80] L. Qiu, Y. Zhang, and S. Keshav. Understanding the performance of many TCP flows. *Computer Networks (Amsterdam, Netherlands: 1999)*, 37(3–4):277–306, 2001.

[81] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification ECN to IP. IETF RFC 2481, January 1999.

[82] Y. Rekhter and P. Gross. Application of the border gateway protocol in the Internet. IETF RFC 1772, 1995.

[83] Y. Rekhter and T. Li. A border gateway protocol 4. IETF RFC 1771, 1995.

[84] R.Hamming. Error detecting and error correcting codes. Technical Report 29-147, Bell System Technical Journal, 1950.

[85] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, April 2001.

[86] R. Siamwalla, R. Sharma, and S. Keshav. Discovering internet topology. Technical report, Cornell University Computer Science Department, July 1998. http://www.cs.cornell.edu/skeshav/papers/discovery.pdf.

[87] W. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[88] W. Stevens, M. Allman, and V. Paxson. TCP congestion control. RFC 2581, April 1999.

[89] UCB/LBNL/VINT Network Simulator - ns (version 2). $http://www.isi.edu/nsnam/ns/$, 2000.

[90] Route Views. University of oregon. http://www.antc.uoregon.edu/routeviews.

[91] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Sixth First International World Wide Web Conference*, Santa Clara, California, 1997.

[92] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modeling of temporal dependence in packet loss. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[93] L. Zhang, S. Shenkar, and D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *Proceedings of ACM SIGCOMM*, 1991.

[94] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the constancy of Internet path properties. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop '01*, San Francisco, November 2001.

[95] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale IP traffic matrices from link loads. In *Proceedings of ACM SIGMETRICS*, 2003.

Vita

James Alan Gast was born in Milwaukee, Wisconsin U.S.A. on September 14, 1950 to Patricia Aronson Gast and Irving Bernard Gast. His third grade teacher was his mother and there were two other boys in the class named “James”. One became Jim, one became Jimmy, and James Gast (to this very day) signs his name the way he learned in third grade: James A. Gast. Friends call him “Jim” so that he won’t descend into classroom courtesy.

The Gast family moved to Park Forest, IL in 1955, were Jim met his bride-to-be in kindergarten at Dogwood School. His primary and secondary education were spent in the south suburbs of Chicago. During Jim’s Sophomore year in High School, the Gast family hosted a foreign exchange student from Mexico. Although Jim had 2 years of Latin and only brief training in Hebrew, Spanish came easily and he enrolled in Spanish III, skipping Spanish I and II. In the summer of 1967, before his Senior year in High School, Jim studied in Durango, Mexico. Jim’s two years of High School Spanish are Spanish III and Spanish V.

Jim took his Bachelor’s Degree at the University of Illinois in Urbana. In 1970, he married Anne Stafford and started accumulating dogs, cats, and, eventually, sons. He changed majors from Electrical Engineering (with Computer Science) to Math (with Computer Science) to Philosophy (Logic) before the University finally approved a Computer Science major. There was a problem, however, because the College of Engineering required Physics 107 (Electricity) and Physics 108 (Magnetism). The Dean accepted 2 semesters of Spanish Literature as replacement credit, and Jim graduated with a Bachelor of Science in Computer Science in 1973. To this day both his Spanish and his Physics are rusty.

While in school, Jim worked at the Computer-Based Education Research Lab on the PLATO project in the team that wrote TUTOR, a courseware language that was still actively being used 20 years later. After graduating, Jim took a position on academic staff at the Center for Advanced Computation, writing many early applications to enable the use of ILLIAC IV over the ARPANET, including a remote job entry system. Jim was active in the early standardization of Initial Connection Protocols that became parts of TCP/IP.

In 1976, Jim co-founded Champaign Computer Company, making 8-bit computers for hobbyists and local businesses. The only persistent storage was floppies (80 KBytes per side), and a computer with 48 KBytes of RAM was considered huge.

In 1980, Jim signed on with Systems and Programming Resources to be a consultant to Bell Labs in Naperville, IL. During the next 3 years, Jim worked as a senior developer for Bell Labs Network, a 7-layer ISO-modeled network connecting Western Electric and Bell Labs mainframes with UNIX computers all over the United States. At that time, files on mainframe disks had no notion of ownership or permissions, since disks are just temporary storage. Permanent files were on tapes.

Jim was Product Development Manager at Tellabs in Lisle, IL where he designed the data switching products. When X.25 was standardized, Jim's team created the multiplexers and packet switches that were sold by A T & T. Jim was active in the standardization of X.25 and X.75.

In 1987, Jim co-founded Palindrome Corporation with his 2 best friends. All 3 were from Tellabs and, before that, Bell Labs. The software development lifecycle was formalized before the very first product was written. All changes went through change management and bugs and suggestions were tracked through the entire process until product end-of-life. Jim was the architect and designer of the entire line of network backup, archiving, file migration and business continuity planning products. Jim was also active in the Optical Storage Technology Association and was the founding secretary of the System Independent Data Format Association. In 1994, after Palindrome had grown to 150 employees, it was acquired by Seagate.

During this time, Jim served at the local level as an officer in the Local Area Network Dealers Association and the Novell Users Group. Jim wrote articles in Computer Technology Review and was quoted several times in Byte Magazine (Jerry Pournelle called Jim "an information preservation fanatic") and LAN Magazine. At the International Level, he was Chairman of the Professionalism and Ethics Committee of the Network Professionals Association.

In 1995, Jim joined Novell and worked his way up to Corporate Software Architect. During his tenure, the 64-bit journaled file system was developed and unveiled and replication services were completed.

During this time Jim hosted the formation meeting of the Storage Networking Industry Association. He was also Novell's representative to the The Open Group (the merger of X-Open and the Open Systems Foundation). Jim sat on the Architecture Board and the Technical Managers Forum when The Open Group standardized UNIX95. Jim was the Chairman of the Professionalism and Ethics Committee of the Network Professionals Association while it grew to 10,000 members. He was also a founder of the System Independent Data Format Association (SIDF) and served as the secretary during the entire process of making SIDF into the ECMA-208 and ISO-14863 tape and optical disk file formats. He served on the Optical Storage Technology Association from the design of the Universal Data Format (UDF) until it was adopted for DVDs.

One day in 1996, while Jim was flying to yet another meeting, a fellow traveler started talking about retirement. Each had a lifelong love of teaching and heartfelt respect for teachers at all levels. The two complete strangers decided that they would teach undergrads when they retired. But, to do that Jim needed a Ph.D.

So, in 1998, Jim stopped being an empty suit and started attending graduate school at the University of Wisconsin - Madison. He was fortunate to be at Madison during the creation of the Wisconsin Advanced Internet Lab, and spent many pleasant hours there learning alongside the smartest (and most genuine) people in the world.

Jim has an older brother Michael (who beat him to a Ph.D. by 26 years) and sons, Peter (who got his BS/Computer Science from MIT in 1993), Brian (Iowa State University, Ames), Jeremy (University of Illinois, Urbana), and Daniel (Daniel Webster College, Nashua, NH).

Jim will become a member of the Computer Science and Software Engineering faculty at the University of Wisconsin at Platteville in August, 2003.

`jgast@cs.wisc.edu`
August 4, 2003
Madison, Wisconsin