University of Alberta

Workload Modeling and Performance Evaluation for Internet Forwarding Systems

by

Weiguang Shi ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**

Department of Computing Science

Edmonton, Alberta
Fall 2003

Your file Votre référence
ISBN: 0-612-88047-8
Our file Notre référence
ISBN: 0-612-88047-8

University of Alberta

Library Release Form

Name of Author: Weiguang Shi

Title of Thesis: Workload Modeling and Performance Evaluation for Internet Forwarding Systems

Degree: Doctor of Philosophy

Year this Degree Granted: 2003

Date: Aug. 29. 2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Workload Modeling and Performance Evaluation for Internet Forwarding Systems** submitted by Weiguang Shi in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dr. Mike H. MacGregor
Co-Supervisor

Dr. Pawel Gburzynski
Co-Supervisor

Dr. Bruce Cockburn

Dr. Joseph Culberson

Dr. Janelle Harms

Dr. Carey Williamson
External Examiner

Date: Aug. 29, 2003

To my parents

Abstract

The exponential growth of the Internet has presented great challenges for its infrastructure. Moreover, the rapid pace of the innovations in Internet applications mandates that the routers that forward packets between networks offer not only high throughput but also huge computation power and maximum flexibility.

In this thesis, we are concerned with the design and performance evaluation of Internet forwarding systems. We attack system performance problems through the study of its workload characteristics, which leads to sound system designs.

First, we characterize temporal locality in router workloads. Temporal locality is critical to cache performance and thus important to overall forwarding system throughput. We derive a mixed distribution which combines the Weibull and Pareto distributions and accurately captures the locality in destination IP address sequences of Internet traffic. Our model is generative; synthetic traffic can be produced with projected temporal locality.

Second, we show that flow popularity characteristics have a significant impact on load balancing in a parallel forwarding system where a hash-based scheduler dispatches incoming packets to individual forwarding engines. We model the flow popularity distribution using Zipf-like distributions and develop a scheme to incorporate large flows into the generative model that captures temporal locality.

Third, we further explore performance implications of flow-level Internet traffic characteristics and develop a highly efficient and effective packet scheduling scheme for parallel forwarding system load balancing. We find that under certain Zipf-like distributions, hash-based scheduling scheme alone can not achieve load balance for a parallel forwarding system. The presence of a few dominating flows in Internet traffic has motivated us to develop a novel load balancer that capitalizes on this phenomenon by scheduling these high-rate flows to balance workload among the forwarding engines in a parallel forwarding system. The effectiveness of our scheme is demonstrated via simulation.

Acknowledgements

This thesis could not have been finished without the help of many people. The most important is my co-supervisor, Mike MacGregor. Throughout my PhD study, he has always been encouraging, understanding, and supportive. Among many other things, Mike motivated and guided my research and taught me how to write. His belief in research and his professionalism have been most inspiring to me. I remember quite well when I just started searching for a topic, I was at a loss and seriously doubted my ability to do useful work. After giving some encouragement, Mike told me: "Have some faith." I don't know about others but I hadn't started to relate "faith" to "research" until that moment when he said it. One never knows, but it just clicked and I went on.

I wish to thank my co-supervisor, Pawel Gburzynski, for his support, encouragement, and insightful comments on my work. I have also benefited from discussions with Janelle Harms, Ioanis Nikolaidis from the communication networks group, Joe Culberson on my thesis committee, and Xiaobo Li. I have had the pleasure of working as a teaching assistant under Paul Lu, Hong Zhang, and Mike MacGregor; their well recognized teaching methods have been educational to me.

Some important inspirations in my work came from the industry. It was at Sprint Advanced Tech. Lab where I conducted performance tests for Internet backbone routers and later realized the discrepancy between the RouterTester-offered workload and the real world traffic. This was one of the motivations for the workload modeling in this thesis. I also wish to thank Andreas Herkersdorf at IBM Zurich Lab for answering my questions on the IBM load balancer. Their paper inspired the load balancing design in this thesis.

I feel fortunate to have conducted my graduate research at the Computing Science Department, University of Alberta, which has provided a friendly, constructive, and efficient academic environment. I wish to thank the supportive staff members who have helped me, especially Edith Drummond for her help with all the paper work and Steve Sutphen for answering my many system questions.

During my study, I have frequently turned to Internet newsgroups for help on various technical questions. To the folks who answered my posts go my sincere thanks.

Finally, I would like to thank my family and friends. Four years as an international student have also been a journey of adjustment to a completely different culture. My beloved wife, Amy, has shared with me my dreams and cheered me through. I am so fortunate to be blessed with some loving and faithful friends whose support I can always rely upon.

Contents

List of Figures

List of Tables

List of Acronyms

AD	Adaptation Disruption
BGP	Border Gateway Protocol
CARP	Cache Array Routing Protocol
CCDF	Complementary Cumulative Distribution Function
CIDR	Classless Inter Domain Routing
CRC	Cyclic Redundancy Code
CV	Coefficient of Variation
DA	Destination (IP) Address
DARPA	Defense Advanced Research Projects Agency
DNS	Domain Name System
DP	Destination (transport layer) Port number
DWDM	Dense Wavelength Division Multiplexing
FE	Forwarding Engine
HRW	Highest Random Weight
HTTP	HyperText Transfer Protocol
ICP	Internet Caching Protocol
IP	Internet Protocol
IRM	Independent Reference Model
ISO/OSI	International Standardization Organization / Open System Interconnection
LAN	Local Area Network
LRUSM	Least Recently Used Stack Model
MAD	Minimum Adaptation Disruption
NP	Network Processor
OSPF	Open Shortest Path First
RFC	Request For Comments
RIP	Routing Information Protocol

SA	Source (IP) Address
SP	Source (transport layer) Port number
SONET	Synchronous Optical NETwork
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WS	Working Set
WWW	World Wide Web

Chapter 1

Introduction

Over the past three decades, the Internet has revolutionized the way that people communicate [1]. What used to be a DARPA (Defense Advanced Research Projects Agency) project to connect four major research centers in the US has evolved into a global medium that carries digitized data in the forms of files, emails, newsgroup messages, etc., between people all over the world. Its growth was comparatively slow until the advent of the World Wide Web (WWW) when computer and network technologies matured to such a degree that people could search the huge amount of information available from the Internet using a Web browser. The tempo suddenly increased. The ease of use of the Internet ignited the imagination of the world as individuals, businesses, and institutions began to put information onto the Web. Recently, important applications e.g., voice over IP (Internet Protocol) and real-time video, have been developed to take advantage of this medium.

The exponential growth of the Internet presents scalability problems for its infrastructure. In a sense, the Internet has become a victim of its own popularity. In particular, the forwarding devices, known as routers or gateways, that connect multiple networks, have to move huge amounts of data from network to network in time to prevent the Internet from degrading in service or even collapsing.

Recent advances in optical transmission technology, such as DWDM (Dense Wavelength Division Multiplexing), have unleashed the potential of seemingly unlimited bandwidth. This has rendered the performance issue of inter-connection devices even more prominent as they have become the major bottleneck of the information delivery system.

In this chapter, we first present the necessary background where the problems addressed

[1] A history of the Internet can be found at the Internet Society's Web site *http://www.isoc.org/internet/history/*. A detailed account of the evolution of Internet backbone can be found in [1].

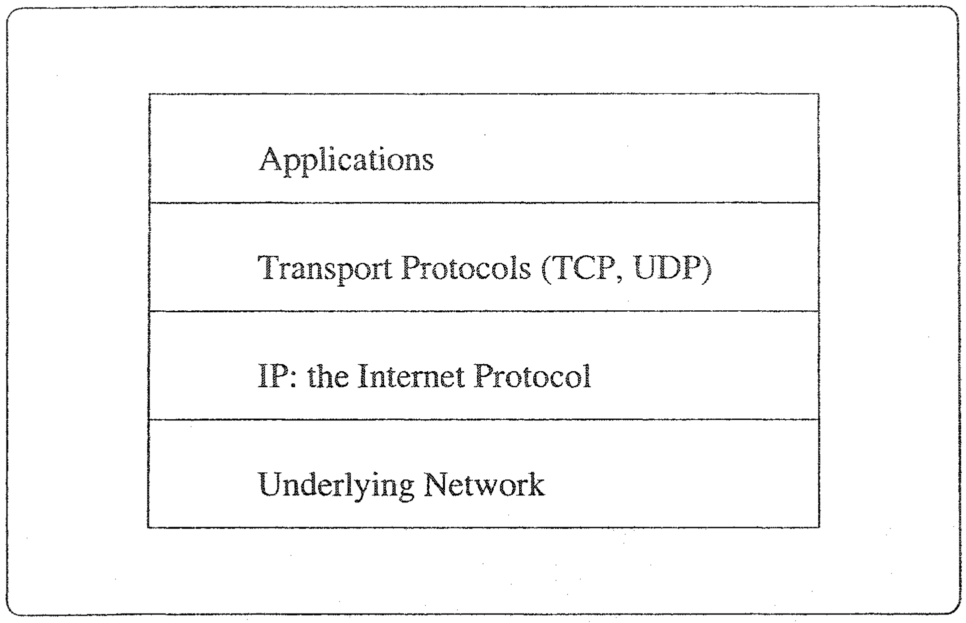


Figure 1.1: TCP/IP Model

in this thesis originate. This includes a brief description of the Internet model. A discussion of a general Internet forwarding system architecture is followed by a more detailed description of one of the major performance bottlenecks, i.e., routing table lookup. Modeling locality in forwarding system workload is the main theme of our work; the concepts of locality and the significance of forwarding system workload characterizations are discussed in Section 1.4. The chapter ends with a brief list of contributions and an outline of this thesis.

1.1 The Internet Protocol Suite: TCP/IP

Fig. 1.1 shows the four-layer TCP/IP networking model. Each protocol layer is a user of the services provided by the layer immediately below and at the same time, provides services to the layer immediately above. Interfaces exist only between adjacent layers.

The Internet connects heterogeneous networks rather than only those of the same architecture. Each particular physical network has its own set of rules with which its hosts comply in order to communicate among themselves. The glue that ties various networks together so that hosts on different networks can communicate is the Internet Protocol, or IP. Immediately above IP are transport layer protocols that provide end-to-end services to applications.

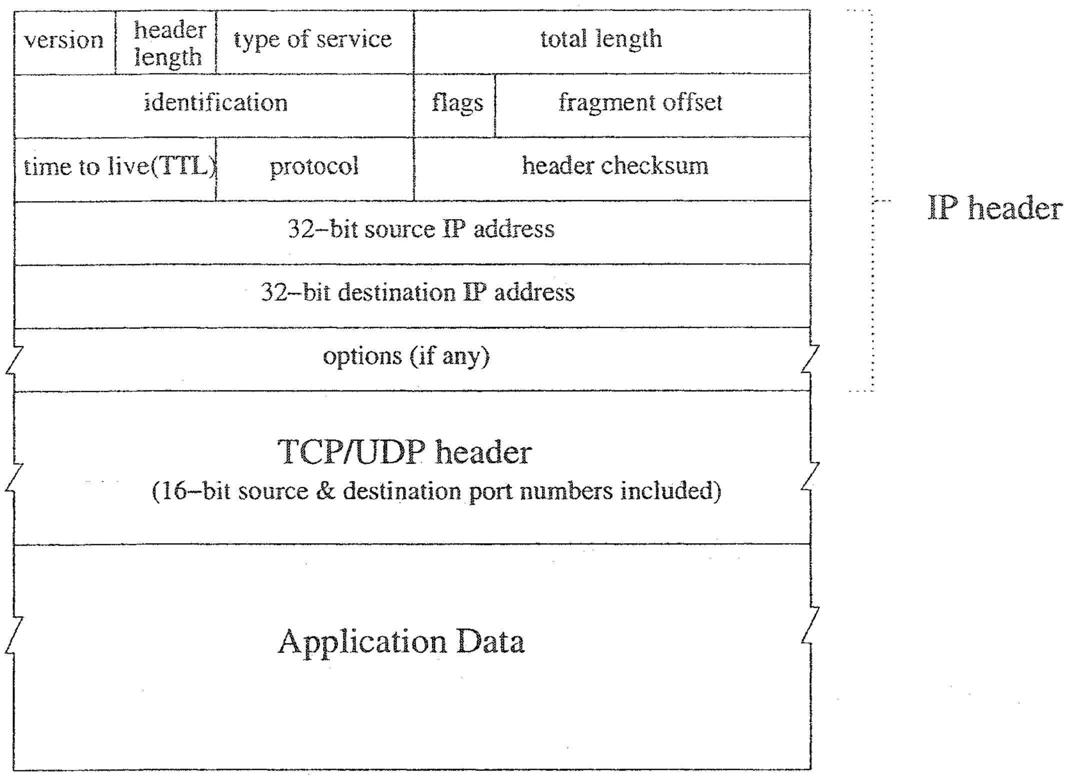


Figure 1.2: The Layout of an IP Packet

The two main transport layer protocols are TCP and UDP. TCP, the Transmission Control Protocol, provides reliable transmission between communicating applications. UDP, on the other hand, simply provides access to the connectionless IP network.

Besides these components, the TCP/IP suite includes protocols for routing, maintenance, diagnostic, and management purposes. Internet protocols are standardized in RFC (Request For Comments) documents. For example, the standard specification for IP is in RFC791 [2] and that for TCP is in RFC793 [3]. A good reference for TCP/IP can be found in [4].

To send data to a host on a different network, the sender chops the data, if necessary, into smaller chunks. A transport layer (TCP/UDP) header is prepended on each chunk to form a *segment*, which includes, besides other information, the 16-bit source and destination ports that are used to locate the specific applications. Each segment, in turn, has an IP header prepended to it to form an IP *packet*. The layout of the IP packet is shown in Fig. 1.2. Among the fields of an IP header, the *protocol* field identifies the transport layer protocol. The 32-bit source and destination addresses uniquely identify the communicating hosts. Finally, each packet is sent onto the physical network after being encapsulated in the native network *frame* (e.g., Ethernet).

As an IP packet carried in one particular network frame is received at an Internet router, it is extracted from its frame. After the next-hop router is decided according to the destination address and other information carried in the headers, the packet is encapsulated using the frame format of the next-hop network and forwarded to the next-hop router. The procedure is repeated until the packet arrives at its destination host. Packets from the same application are forwarded independently through the Internet, possibly over different paths, toward their target hosts. IP does not guarantee successful or in-order delivery of the packets. This is called best-effort forwarding.

IP, as a network layer protocol, delivers an Internet packet to its destination host, according to its destination IP address. In most cases, the communicating entities are two application processes running on two hosts. Transport layer protocols provide information to identify the particular receiver process for a packet. The port numbers in TCP and UDP headers are used to specify the receiver. Thus, the three-tuple: destination port number, transport layer protocol, and IP destination address uniquely identify the intended receiver application for a packet. The five-tuple: source and destination addresses, source and destination port numbers, and the transport layer protocol uniquely identifies a connection between two applications.

1.2 General Internet Forwarding System Architecture and Operation

The Internet is organized in tiers. At the basic level, there are home and small business networks tied to local Internet Service Providers (ISP's). In the middle, small ISP networks, campus networks, and enterprise networks are connected to major ISP networks. The top level consists of backbone networks that connect major ISP's. Routers are found at all levels, connecting networks within the same tier or between adjacent tiers. A router has two main functions:

Routing and Control The router exchanges network topology and routing policy information with peer routers via routing protocols. This information is used to maintain the routing table, including adding, removing, and updating table entries. The router also has to enable system management and control functions. Operations required to perform routing and control functions are relatively infrequent and thus consume only

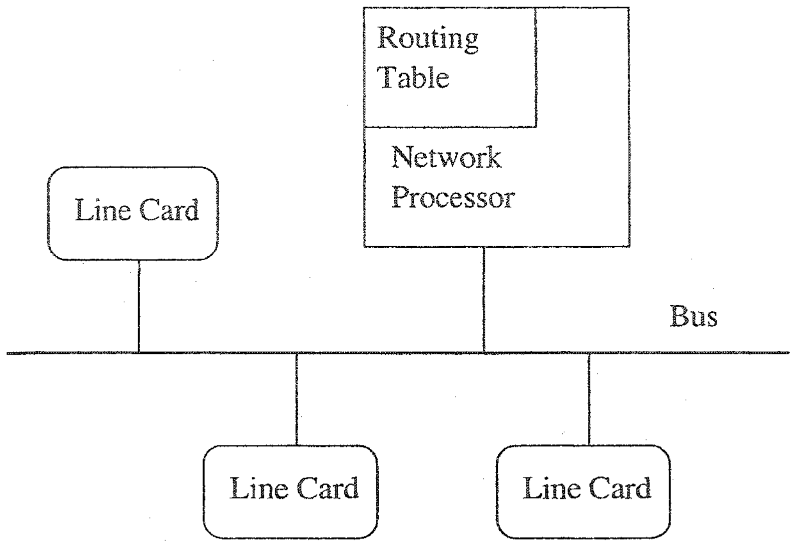


Figure 1.3: A General-purpose Computer as a Router

a small fraction of system resources.

Forwarding The router receives packets from its input ports, looks up the destination IP addresses in the routing table, and forwards the packets to appropriate output ports. Forwarding decisions have to be made for every packet that the router receives. The resources required here are proportional to the traffic volume.

We are interested in the evaluation of the forwarding performance of a router. Throughout this work, we presume that forwarding decisions are based solely on the IP destination address field of a packet.

An Internet router can be implemented as a general-purpose computer system with multiple line cards. Indeed, some operating systems can easily be configured to forward packets. As shown in Fig. 1.3, each card connects the router to a different network. This is a centralized architecture where one processor is responsible for both forwarding and routing. The bus bandwidth is shared among all the ports and each packet traverses the bus at least twice to be forwarded, i.e., from the input port to memory and, once the forwarding decision is made, from memory to the output port. This solution is adequate and cost-efficient for most small networks.

Routers at backbone networks have a much heavier workload. As demands keep growing and advances in transmission technology make gigabit- or even terabit-per-second networks

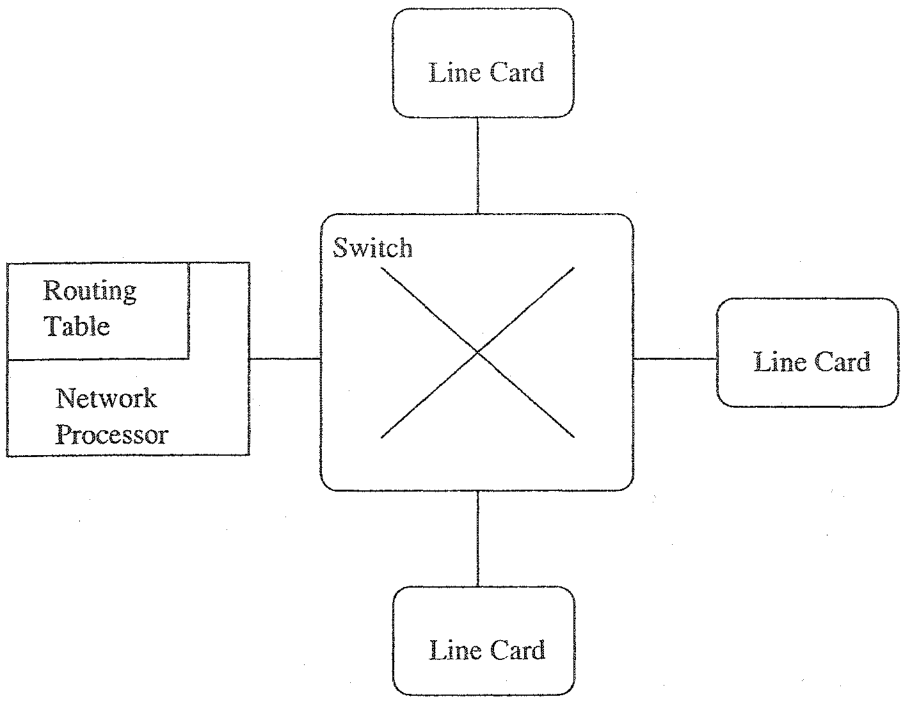


Figure 1.4: A High-end Router

possible, centralized routers such as that in Fig. 1.3 are inappropriate, if at all feasible. To forward over one gigabit per second, a router needs to be able to process at least one million packets per second. At such speeds, the cost of the router can be reduced by two to three orders of magnitude if a distributed architecture, instead of a centralized one, is used. Analytical models have been developed to evaluate the effect of design parameters on the cost of such systems [5].

The high-end router shown in Fig. 1.4 separates routing and forwarding functions. The network processor's job is to run system management software and to maintain a master routing table. Each line card has its own copy of the routing table, which is now called a forwarding table. Line cards are full-fledged forwarding engines with resources to process both input and output traffic. Packets are transferred between line cards through a switch fabric. A switch, unlike a shared bus, allows multiple packets to be transferred simultaneously and thus can provide much higher bandwidth. Moreover, to forward at higher speeds and to cut cost, one current trend in router design is to put multiple network processing units (NPU's) on the same line card to forward incoming packets in parallel.

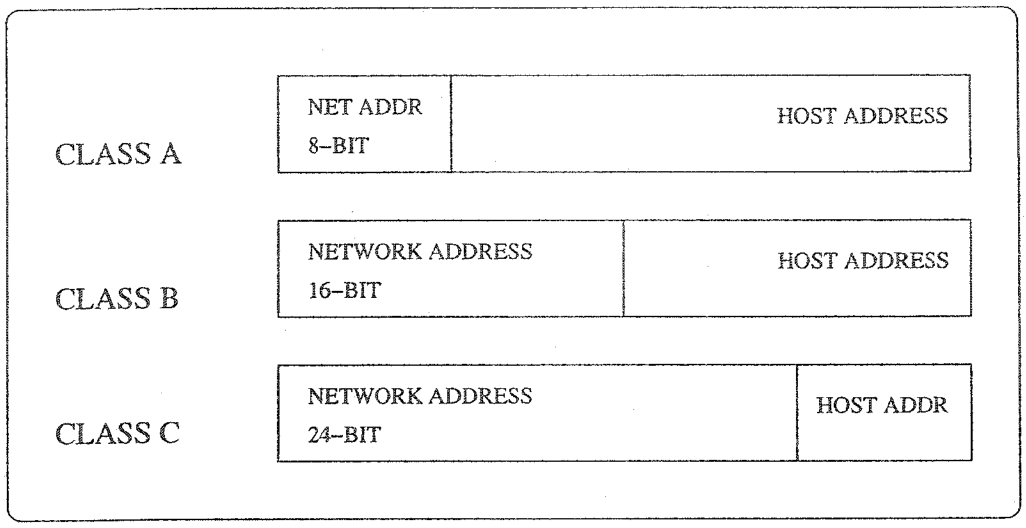


Figure 1.5: IP Address Classes

1.3 The Bottleneck: Routing Table Lookup

One of the most time-consuming operations in packet forwarding is routing table lookup. On one hand, this is because of increasing routing table size as the Internet expands. According to a BGP (Border Gateway Protocol) table size report [6], backbone routing table sizes have increased from fewer than 20,000 entries in 1994 to nearly 140,000 today. Moreover, the complexity of *Longest Prefix Matching*, required by Classless Inter Domain Routing (CIDR) [7, 8] aggravates the problem. The specifics of longest prefix matching will be discussed in Section 1.3.1.

1.3.1 The IP Addressing Scheme

Originally, IP used a class-based addressing scheme where the 32-bit address space was divided into classes A, B, C, D, and E. Classes D and E were reserved for multicast and experimental purposes. IP address blocks were allocated to the unicast classes A, B, and C where a 32-bit address was divided into two parts: the network specifier and the host. Fig. 1.5 shows the ranges of the three address classes. The first bytes of network addresses from class A, B, and C range from 0 to 127, 128 to 191, and 192 to 223, respectively; therefore, by the value of an address's first byte, one can identify to which class it belongs. Class A networks were for large enterprises; they allowed a network to contain as many as 16 million hosts. Class B networks were for middle-sized organizations. Class C networks were for small institutions, and allowed no more than 253 hosts. This address scheme worked fine in the early years of TCP/IP.

As the Internet grew and more middle-sized organizations joined in, the community was faced with the depletion of class B network addresses. Moreover, the growth of routing tables became a serious concern when it was anticipated that the tables would become too large to manage.

CIDR tries to meet both needs by proposing a more flexible addressing scheme. First, it allows the length of network and host addresses to be variable to eliminate the inefficiency of allocating address blocks to organizations that are too large for class C yet too small for class B networks. Second, CIDR allows *address aggregation* to help manage large routing tables. In CIDR, an address is presented as a *prefix/mask* pair where the most significant bits of the mask are set to 1 to identify the portion of the prefix representing the network address. These 1's in the mask are contiguous and start from the most significant bit and the rest of the prefix that corresponds to the 0's of the mask is the host address. In the same spirit, an Internet route can be represented as a pair of *prefix/prefix length*, where *prefix length* is the number of 1's in the mask.

This hierarchical scheme lends itself to route aggregation. For example, if a routing table has two entries "129.128.25.0/24, port A" and "129.128.0.0/16, port A", they can be collapsed into one route, "129.128.0.0/16, port A". That is, if the address range covered by a route enclose the ranges by one or more other routes, these routes can be represented by the first route, which covers the largest range, as long as they point to the same output port.

1.3.2 Routing Table Lookup

Before CIDR, deciding the output port of an IP packet required two steps:

- extract the network address from the destination address.
- use the network address as the index to a routing table of (*network address, output port*) pairs to retrieve the output port.

The deployment of CIDR presented challenges to routing table lookup. For one thing, the length of the network address could theoretically be any value from 0 to 31. Moreover, there was no longer the "class" delineation to help find the network address in an IP address. The route aggregation of CIDR implies that multiple entries in the routing table may match the same address. For example, suppose that a routing table contains the two entries "129.128.25.0/24, port A" and "129.128.0.0/16, port B" and a packet with destination address "129.128.25.8" arrives. Either route matches this address in that

$$129.128.25.8 \ \& \ 255.255.255.0 = 129.128.25.0$$

for the first route and

$$129.128.25.8 \ \& \ 255.255.0.0 = 129.128.0.0$$

for the second, where "&" represents the bitwise logical AND operation. The two routes cannot be aggregated because they have different port numbers, although the range covered by the second route covers that by the first. In this situation, the route lookup process must select the route that has the longer prefix.

The routing table in Berkeley Unix after the 4.3BSD Reno release manages CIDR lookups. It is organized in a radix tree structure [9] where routing entries located at leaf nodes. The bits in the destination address (not necessarily all of them) are compared with the internal nodes. Based on the result, the algorithm branches left or right until a leaf node is reached. This may not necessarily mean a successful longest-prefix match. After comparison of the destination address and the key value in the leaf, a decision will be made if back-tracking is to occur. This algorithm results in a worst case complexity of $O(W)$ where W is the length of the address in bits; the algorithm requires as many as 32 memory accesses per address for IPv4 [10].

Much research has been done to speed up the routing table lookup process. Degermark et al. [11] solve the problem by using a compact complete tree data structure so that the whole routing table fits in the second-level cache of a general-purpose processor. Waldvogel et al. [10] use a combined hashing and binary search method to reduce the number of memory accesses to $log_2(W)$. This means a worst case of five memory accesses for IPv4. Chiueh and Pradhan [12] propose a novel caching scheme which uses a portion of the 32-bit IP address as part of the 32-bit virtual memory address and the rest as "tags" to be compared with those of a destination address. Nilsson and Karlsson [13] use a trie structure that is compressed both in path and level and achieves $O(loglog(n))$ search depth where n is the number of entries in the table. Gupta et al. [14] implement the routing table in hardware. Combined with pipelining, this approach achieves one lookup per memory access time. Shi and MacGregor [15] evaluate the cache performance of three software lookup algorithms in

[9, 11, 13] and find that the algorithms described in [11] and [13], owing to their compact data structures, achieve much higher cache hit ratios than the radix tree approach [9].

1.4 Locality: The Concepts

The performance of routing table lookup became a concern well before the introduction of CIDR. Feldmeier [16] investigated using a route cache to improve gateway performance. A route cache stores the most recently used routes in fast memory so that lookups of future addresses that match cached routes (called *cache hits*) can be done quickly, as searching the full routing table and accessing slower memory are avoided. Caching only works well when sufficient locality exists in system workload. In the case of a router, it is the locality in the sequence of destination IP addresses that makes the cache useful.

The concept of *locality* originates from program memory reference behavior studies [17]. Models built to characterize locality in the workload of computer virtual memory systems are applicable to many other systems. For example, they have been used to capture the locality in workloads of file systems, local area networks, and Web servers.

Temporal locality refers to the phenomenon that when an item is accessed, it is highly possible that it will be accessed again in the near future. It describes the recency of repeated references to the same object. Many workloads exhibit temporal locality. In network traffic, temporal locality stems from bursts of packets transmitted for a chunk of application data that is larger than one single packet can carry. This is known as the *packet train* behavior of network traffic, observed in both local area networks (LAN's) [18] and the Internet [19].

The term *persistence* is used to describe the tendency for an item, once referenced, to be *consecutively* referenced [20] and it is a special case of temporal locality.

In the context of program memory references, spatial locality refers to the phenomenon that when an address is referenced, it is highly likely that neighboring addresses will be accessed in the near future. This concept, however, is difficult to apply to the network environment because the notion of "neighboring" is not clear.

Another form of locality is called *concentration* which means that a small set of items are referenced in the workload [20]. Concentration is by definition related to the working set model [17] and a quantitative measure has been developed in [21, 22].

1.5 Contributions

Due to its heterogeneity, complexity, scale, and fast-evolving nature, the Internet is best studied by models [23]. Besides being able to give a quantitative description of system features, models can be used to predict behaviors. Models capturing salient features of system workload can lead to better designs.

One goal of this thesis is to develop models of locality in router workloads. We have developed a mixed-distribution model, the combination of a Weibull and a Pareto, that accurately describes temporal locality in destination address sequences collected at networks ranging from campus networks to major backbones in the Internet. For the sake of parsimony, we later show that the five-parameter model can be substituted with a four-parameter model by replacing the Weibull by an exponential distribution. Traffic generation is one of the key challenges in modeling and simulating the Internet [23]. Given a set of parameters, our model produces synthetic address traces according to the specified temporal locality and can be used to test cache design alternatives for forwarding systems.

Another important aspect of Internet workloads is their flow-level characteristics. Based on measurements of network traces, we propose a Zipf-like [24] function to describe the non-uniform distribution of flow popularity. We incorporate popularity distributions into the framework of the least-recently-used stack model (LRUSM) in order to generate synthetic traffic that resembles real-world traffic in both temporal locality and skewed flow popularity distribution. This model is useful in evaluating the performance of critical algorithms, e.g., load balancing design, in parallel forwarding systems.

We investigate the effects of two traffic splitting schemes in parallel forwarding systems: round-robin and hash-based. Our results show that hash-based methods improve temporal locality, and that caching can help balance system load. This work also leads to insights into the design of load balancing schemes for parallel forwarding systems.

We propose a novel load balancing design for parallel forwarding systems. Our observation that flow popularity distributions in Internet traffic are Zipf-like leads to the conclusion that hash-only traffic splitting schemes are unable to guarantee load balancing. Given the importance of cache performance, we introduce a new performance metric for load balancing designs, i.e., *adaptation disruption* caused by flow shifting. The key idea in our load balancing scheme is to shift only the high-rate flows in Internet traffic when load adaptation is needed due to load imbalance. High-rate flow scheduling is both effective and efficient and, compared with state-of-the-art load balancing schemes, it minimizes adaptation disruption.

1.6 Thesis Outline

In this thesis, we discuss the characteristics of workloads for Internet forwarding systems. Recognizing Internet routing table lookup as one of the bottlenecks and load balancing as critical to parallel forwarding systems, we model the characteristics of IP traffic that are most relevant to forwarding performance in router workload. This work is organized as follows:

- Chapter 2 discusses related work in modeling computer program memory reference behavior, temporal locality modeling for computer network traffic, flow-level traffic characteristics, and load balancing in parallel forwarding systems. As locality is one of the most exploited concepts in computing system design, previous work on studying its effects is abundant and appears in different areas. These references provide valuable background information for this thesis. Some of the studies have recognized the importance of modeling temporal locality in IP address destination addresses and interesting results have been shown on this particular subject. We will briefly introduce these studies in Chapter 2 and will compare them to our work in more depth in later chapters.

- Chapter 3 presents a model that can describe temporal locality in Internet traffic. We begin by explaining the motivations behind our work and introducing our methodology. We adopt the LRU stack model and develop a flexible mixed distribution function that can accurately describe the temporal locality in IP destination address traces.

- Chapter 4 extends the aggregate traffic model to incorporate flow popularity information. Each of the two workload models, the LRU stack model and the independent reference model, has its strengths and weaknesses. The former captures temporal correlation and the latter characterizes the popularity of distinct addresses. To generate realistic Internet traffic for parallel forwarding system performance evaluation, we propose an algorithm that is based on the LRU stack model but also considers the flow popularity distribution.

- Chapter 5 shows the effects of scheduling schemes on temporal locality in a parallel forwarding system. We study two scheduling methods, round-robin and hashing, which have drastically different effects on caching and load balancing.

- Chapter 6, based on work in Internet forwarding system workload characterization, shows that hash-only load-splitting schemes cannot guarantee load balance in a parallel system. We propose an efficient adaptive load balancing scheme that, when activated, adjusts only the mappings of high-rate flows. In addition, the load adaptation mechanism achieves the desirable goal of low adaptation disruption which is critical to forwarding system performance. We show trace-driven simulation results for different adaptation-triggering policies.
- Chapter 7 summarizes this work and discusses directions for future research.

Chapter 2

Related Work

In this chapter, we review some program memory reference models and later show how they are used in modeling temporal locality in Web document access sequences. We discuss work in measuring locality and using cache to improve system performance in network environments. Flow-level measurement and modeling are important because they lead to better understanding of the burstiness of Internet traffic, a feature that is critical to forwarding system performance. Last we discuss traffic-splitting and load balancing schemes in parallel forwarding systems.

2.1 Program Memory Reference Behavior Models

Much research has been done to model the program page reference behavior of virtual memory (VM) [25]. A VM system provides a much larger address space than physical memory, or main memory. *Paging* is often used in VM systems, where application programs are divided into fixed-size pages to fit into page *frames* of the same size in the main memory. There is no need to load whole programs at once into physical memory; the requirement is that the page(s) that contains the necessary code and data has to be in memory before a program can proceed. Loading pages from external storage into main memory is a costly operation and thus its frequency is critical to system performance. Therefore, page reference sequences of computer programs are important to model.

Since there are usually fewer page frames than required by application programs, most of the time when a page is to be loaded from the disk, a page in memory has to be swapped out to make room. A *replacement policy* decides which page is to be replaced. One well-known policy is *least recently used* (LRU) which states that the page that is least recently used should be swapped out. LRU takes advantage of locality in page reference sequences and assumes that recently-referenced pages are going to be accessed in the near future and thus should be kept in the main memory.

2.1.1 The Working Set Model

The working set (WS) model [26, 27, 17] was developed to model page reference behavior. A program's working set *W(t,T)* at time t is the set of distinct pages referenced in the time interval $[t\text{-}T\text{+}1,\ t]$. The parameter T is called the working set *window size*. The working-set principle of memory management states that a program may use a processor only if its WS is in main memory, and that no working-set page of an active program may be considered for removal from main memory. Because of the locality in program memory reference sequences, the WS model can predict future memory demands based on the past reference pattern.

Given the set of possible pages that a program can access, $N = \{1, 2, ..., n\}$, the subject of the WS model is a page *reference string*, a sequence of the pages accessed by the program during its execution, $\rho = r_1, r_2, ..., r_t, ...$, where $r_t \in N$ is the page referenced at the discrete time t. Denning and Schwartz [17] derived several relations between the average-working-set-size function, the missing-page-rate function, and the interreference-interval density function. The *working-set size w(t, T)* is the number of pages in $W(t,T)$, i.e.,

$$w(t,T) \quad = \quad |W(t,T)|. \tag{2.1}$$

Let

$$s_k(T) = \frac{1}{k}\sum_{t=1}^{k} w(t,T)$$

denote the working-set size averaged over the first k references; the *average working-set size* is defined as

$$s(T) = \lim_{k\to\infty} s_k(T).$$

The *missing-page-rate m(T)* is defined as the number of pages per unit time returning to the working set. $m(T)$ reflects the probability that a new page is referenced given that the current window size is T. The *overall interreference density function*, $f(x)$, is defined as

$$f(x) = \sum_{i=1}^{n} \lambda_i f_i(x)$$

where $f_i(x)$ is the interreference density function of page i and λ_i is the frequency of references to page i. Analysis shows that

$$m(T) = s(T+1) - s(T)$$

and

$$f(T) = m(T-1) - m(T).$$

That is, $f(T)$ is the negative slope of $m(T)$, which is the slope of $s(T)$.

One important assumption in WS is that the stochastic mechanism that generates the page reference string is stationary, i.e., independent of the absolute time origin. This is largely true as long as the program does not transit to new sets of pages (called *localities*). Unfortunately, computer programs usually exhibit *phase-transition* behavior, jumping from one locality to another during execution. This assumption restricts the results of the WS analysis, which is the motivation for characterizing reference strings using a macro inter-phase model and a micro intra-phase model [28]. Nevertheless, the limitation of WS analysis does not prevent it from being a useful starting point in modeling program memory reference behavior. Moreover, in the contexts of computer network traffic and Web document reference sequences, the stationarity assumption seems to hold, making the WS analysis applicable.

2.1.2 The Independent Reference Model

In the Independent Reference Model (IRM) [29], a page reference string, which is simply the page numbers visited by a program, $R_1, R_2, \ldots, R_t$, is treated as a sequence of independent random variables with a common stationary reference distribution:

$$Pr[R_t = A_i] = p_i, \ 1 \leq i \leq N, \ t > 0$$

where A_i is the ith page out of N unique pages.

The IRM captures the non-uniform page reference behavior of the programs. However, it does not describe the temporal correlation between successive references to the same page. Thus the IRM is inadequate to characterize temporal locality.

Zipf's law [24] is the observation that the frequency of occurrence of some event (P) as a function of rank (R) often obeys the power-law function

$$P(R) \sim 1/R^a \tag{2.2}$$

with the exponent a close to 1.

A similar phenomenon has been observed in the studies of both program memory reference sequences [30, 20] and Web server workload [31, 32]. Zipf-like popularity distributions have been combined with the IRM to produce synthetic workloads [33].

2.1.3 The Least Recently Used Stack Model (LRUSM)

The LRUSM [34, 28, 35] is another program behavior model. Consider a stack as a one-dimensional array containing all possible addresses, each of them a single array element. When an address is referenced, the array (stack) index of the address is output as the stack distance. Note that we use the equivalent term *reuse distance* in the rest of this thesis. All the addresses above this value are moved down by 1 position and the address just referenced is put at position 0 of the array, that is, at the top of the stack. This is equivalent to using the LRU model to update the stack, hence the name "LRU Stack." In the LRUSM, each entry in the address trace produces a stack distance. Corresponding to an address trace, then, there is a sequence of stack distances, which is called the *distance string* of the trace. As an example, a distance string can look like "$38, 1, 0, 143, 1, 162, 1, 0, 40, 97, 1, 150, 63, 311, 80, 312, 1, 3, 0, 313, 127$". The "0"s in the string indicate that the address at the top of the stack, which was just referenced, is referenced again; "1"s mean that the address referenced just prior to the last reference (now at position 1 of the stack) is referenced again, and so on.

The LRUSM captures temporal locality in that the probability of a stack distance of n represents how likely an address just referenced is to be accessed again n *distinct* addresses away in the future. This is not exactly the definition of temporal locality because the number of distinct addresses is not equivalent to the notion of time. As a result, according to the LRUSM, reference strings can have better temporal locality simply because there are fewer distinct addresses. Using the number of distinct addresses to represent time has utilitarian advantage in that LRUSM adapts itself conveniently to cache performance evaluation; recency in real time does not necessarily lead to good cache performance, but a small number of distinct addresses does. The LRUSM treats all addresses the same and thus is unable to characterize non-uniform accesses to the individual addresses.

2.1.4 Synthetic Trace Model for Cache Simulations

Thiebaut et al. [36] developed a synthetic memory reference model for cache simulations. Different from earlier models, this one targets memory addresses instead of page reference sequences. A cache line is assumed to contain only one word. It is found that after an initial linear segment, the number of unique addresses observed at the nth address reference, $u(n)$, can be expressed by the following power law function:

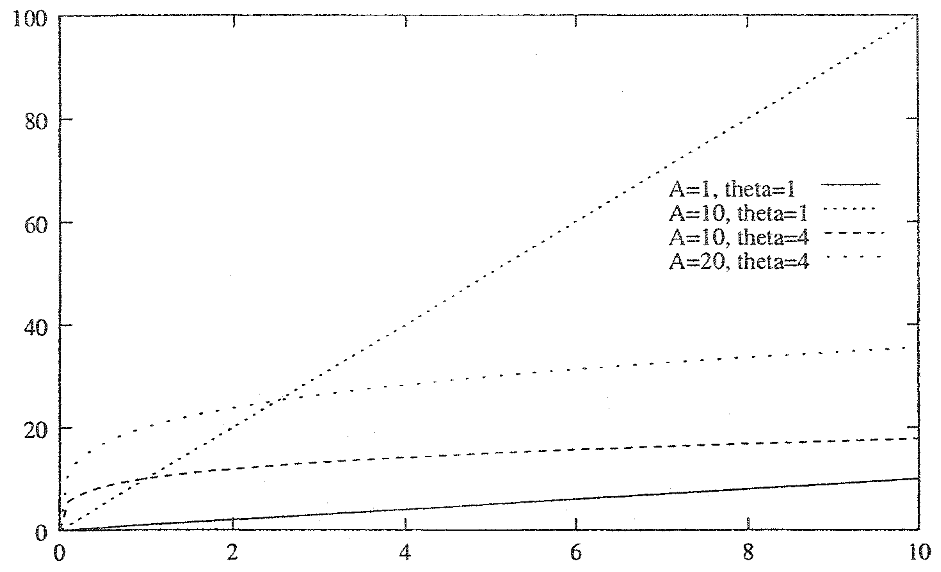


Figure 2.1: Some Hyperbolic Curves

$$u(n) = An^{1/\theta},\ A > 0, \theta > 1 \tag{2.3}$$

where A is a constant and θ is interpreted as a measurement of *spatial locality*. The larger the value of θ, the fewer unique addresses are visited during a certain period of time. The curve of $u(n)$ is called the *footprint* curve of the program. Fig. 2.1 shows some examples of hyperbolic curves. The probability of introducing a new address into the trace at reference n, or the cache miss rate when the cache size is x ($x = u(n)$), is

$$Pr[x] = \begin{cases} (A^\theta/\theta)x^{(1-\theta)} & , \quad x \geq C_c \\ 1 - \frac{x}{C_c\theta} & , \quad x < C_c \end{cases} \tag{2.4}$$

where $C_c = A^{\theta/(\theta-1)}$ is called the CriticalCache, the inflection point of the curve. The initial phase of an empirical footprint curve is linear since the addresses tend to differ from each other and there are no re-appearances. In this *forced mode*, i.e., when $x < C_c$, the number of unique addresses in the trace equals the number of addresses. As more references are made, fewer addresses are new and this forms the hyperbolic part of the curve. Eq. 2.4 differs from the formula in the original article when $x < C_c$. This change ensures that the function is smooth at $x = C_c$ and when $x = 0$, $Pr[x] = 1$.

To generate a synthetic trace, the inverse of the function in Eq. 2.4 is used to generate

LRU stack indexes. This inverse is:

$$x = \begin{cases} (\frac{U}{A^\theta/\theta})^\theta & , \quad U \leq 1/\theta \\ (1-U)\theta C_c & , \quad U > 1/\theta \end{cases} \tag{2.5}$$

where U (representing $Pr[x]$ in Eq. 2.4) is uniformly distributed and is in the range of $(0, 1)$. We will discuss the algorithm in more detail in Section 3.3.

Although satisfactory cache simulation results are shown in [36], this model does not adequately characterize temporal locality in destination IP address traces. An initial examination of this phenomenon and an ad hoc solution can be found in [37]. In the case of IP destination address traces, we show empirically in the next chapter that the footprint curves are the natural outcome of our reuse-distance based trace generation method.

2.2 Locality in Network Environments

Locality has been observed in network environments, from LAN to WAN, and at all protocol layers of the Internet, e.g., the network layer, the transport layer, the application layer, etc. Significant locality makes caching appealing in the design of both host and forwarding systems, especially as the performance gap between microprocessors and main memory continues to widen [38]. In this section, we discuss work in measuring, characterizing, and exploiting network locality.

2.2.1 Locality in Packet Networks

The study in [18] shows that temporal locality exists in LAN traffic. Packets travel in *trains*, where a train is a burst of packets from the same source and to the same destination. The arrival of a packet indicates that there is a high probability that the next packet is to the same destination. At the same time, a sequence of packets heading in one direction is often followed by a second sequence for the reverse direction. This regularity is due to the request-response nature of network protocols. Based on these observations, destination and source address caches can be effective in improving the performance of network devices.

Feldmeier [16] studies a 24-hour trace collected at the router connecting the MIT campus to the Internet. It shows that IP routing table lookup can benefit from even simple caches with a few lines of "destination address, output port" pairs. The destination address reuse distance density curve is analyzed and it is concluded that an LRU cache should be effective for reducing the number of routing table references.

Jain [39] examines destination address traces collected in an inter-connected LAN environment and demonstrates the existence of temporal locality. It is found that two different types of traffic, interactive and non-interactive show different locality behaviors. The locality of the non-interactive traffic is better captured by an LRU cache according to the monotonically decreasing stack distance distribution function. But the stack distance density curve of the interactive traffic has a hump and to capture this characteristic, the requirement for the cache is to have enough entries.

Gulati et al. [21, 22] measure four aspects of locality in LAN traffic: persistence, address reuse, concentration, and reference density. Persistence refers to the tendency of an address, once referenced, to be referenced again and again, consecutively. Address reuse is defined as the tendency for the address used in one network packet to reappear as the destination address of a future packet. Address reuse is a more general measure of temporal locality than persistence. Based on the WS model [17], a measure of concentration C_T is developed.

$$C_T = \frac{T - W_T}{T - 1}, \quad T > 1 \tag{2.6}$$

where T is the WS window size and W_T is the WS size. Reference density is defined as the tendency for a small number of hosts to account for a large proportion of the total network traffic. The measurements show that persistence is low but there is significant address reuse, concentration, and reference density in the traces.

Mogul [40] investigates network locality at a finer grain, the process level. The motivating observation is that the locality visible at the host-address level actually arises because of per-process network locality. Traces collected in 10 Mbps LAN environments are analyzed. It is found that around 75 per cent of the packets arriving at a host have the same destination ports, and thus the same destination processes as their predecessors. Operating systems can cache recent packet header and target process information to accelerate the search for receiver processes for incoming packets. Process level locality is also observed in UDP traffic in [41] and an *improved one-behind cache* is used to improve system performance by recording the last process that received a UDP segment.

One question is what to cache in exploiting temporal locality in packet destination address sequences to speed up routing table lookup. Certainly, caching full addresses can achieve no better a hit ratio than caching the network parts of the addresses or network ID's. The latter, however, requires identifying the network part of an IP packet before the cache lookup. This identification process was non-trivial even in the days when the Internet used class-based addressing [16]. With CIDR, searching for the longest prefix that matches an address is the major part of route lookup, the process we try to avoid with caching. In this thesis, we consider only caching of full IP addresses.

Given the numerous evidences of the presence of locality in networking environments, one should not take it for granted. McKenney [42] analyzes and compares TCP protocol control block (PCB) lookup performance of algorithms proposed in [43, 41], a move-to-front linked list, and a hash-based scheme. The offered workload is that of an on-line banking system [44] and is marked by the lack of locality observed in other work in networking environments. Under this workload and some other assumptions, the work in [42] predicts that both the algorithm in [41] and the move-to-front linked list achieve significant improvement over the single-line caching scheme in [43]. Moreover, the hashing scheme achieves orders of magnitude better performance.

2.2.2 Locality in Web Server Workloads

In [45], the popularity-based IRM is found to be inadequate to capture the temporal locality in the reference strings obtained from Web server access logs. The LRUSM is used instead and it is found that the marginal distribution of stack distance is best fit by the lognormal distribution. A similar approach is taken in [46] to generate synthetic workload for Web servers.

Arlitt and Williamson [47] emphasize that temporal locality is orthogonal to *concentration* in that concentration refers to the aggregate reference counts for documents regardless of the reference order, while temporal locality refers to the relative order in which documents are referenced, regardless of their reference counts. Here, the LRUSM is used to capture temporal locality. The request arrival processes of the aggregate traffic and some frequently referenced documents are investigated and it is found that the former is definitely not Poisson but the latter is.

It is pointed out in [33] that the main drawback of the LRUSM is its inability to distinguish *hot set* (a set of most popular documents) effects from short-term temporal locality, or short-term document reference correlation. That is, a *hot* document may cause many references near the top of the stack, even if there is no correlation between the probability of referencing a particular document and the time since the document was last referenced. The IRM is used to generate synthetic reference sequences on a day-to-day basis. It is found that short-term temporal locality is important in Web workload characterization and that the simple IRM is incapable of capturing this locality.

In [48], the sources of Web document reference temporal locality are identified as short-term temporal correlation and long-term popularity. The model proposed is a combined popularity distribution described by a power-law, and an inter-request time distribution of equally popular documents is described by another power-law. An empirical relationship is established between the two. Based on this work, [49] develops the GreedyDual* Web cache replacement algorithm which takes into account both long-term popularity and short-term temporal locality characteristics in Web server workloads.

Fonseca et al. [50] give a formal definition of temporal locality which helps to differentiate the two sources of locality observed in Web reference sequences: document popularity and temporal correlation. The authors further introduce the *entrophy* [51] of popularity and the coefficient of variation of inter-arrival time as the measurements of the two aspects. These definitions are used in studying the transforms Web reference streams experience as they pass through the Internet. The major findings are that while popularity imbalance can rise and fall, temporal correlation usually declines as the result of transformations.

Work in Web traffic modeling and performance evaluation has shown that both document popularity and reuse-distance distributions are important characteristics of Web traffic. These can be captured by the IRM and LRUSM, respectively. Performance studies using either model tend to emphasize the importance of one feature of the workload, ignoring the other. It would be worthwhile to develop a unified model that combines the strength of the two models. Such a model would be useful in generating representative synthetic traffic.

2.2.3 Synthetic IP Destination Address Generation

Aida and Abe [52, 53] study LRU stack position reference probability under the stationarity assumption of the *inverse stack growth function (ISGF)*, $f(t)$, which is the number of distinct addresses accessed during a time period. Accordingly, the *stack growth function (SGF)* is expressed as $g = f^{-1}$.

By assuming that the ISGF depends only on the number of accesses or WS window size (the stationary assumption), the authors derive the probability function of the stack distance in the LRUSM. Therefore, the proposed method captures temporal locality.

The probability for the next address to be the same as the kth most recently referenced distinct address, a_k, is derived as:

$$a_k := \{f(g(k-1)+1)-(k-1)\} - \{f(g(k)+1)-k\}. \tag{2.7}$$

The ISGF is in a form similar to the empirical power law footprint function in [36]:

$$f(\tau) \simeq \tau^{\alpha} \quad (\tau \gg 1), \tag{2.8}$$

where α is a constant. Unlike the previous work, this is not directly used to generate synthetic traces. With Eq. 2.8, a_k can be expressed as

$$a_k = \{((k-1)^{1/\alpha}+1)^{\alpha}-(k-1)\} - \{(k^{1/\alpha}+1)^{\alpha}-k\}.$$

The LRUSM is then used in the synthetic IP address generation algorithm.

There is an inherent relationship between the LRUSM and the WS when we assume that the WS size w is dependent only on the window size T (see Eq. 2.1). The stack size equals the former and the number of accessed addresses is the latter. As a result, the synthetic address sequences generated using LRUSM resemble real world address sequences not only in temporal locality, but also in working set behavior.

It is observed that Internet addresses are not distributed uniformly [54]; the frequencies of addresses are highly skewed and follow Zipf's law. The proposed address generation scheme in [52, 53], although it captures the WS behavior of Internet address sequences, cannot differentiate address distribution. This is because in the LRUSM, addresses in the stack are all treated the same way.

Aida and Abe [54] extend the proposed algorithm in [52, 53] to accommodate the skewed address distribution. A formula similar to Eq. 2.7 is used to generate stack distances. In the medium/long time scale, i.e., when a generated stack distance is large, the LRUSM is used to produce the output address. When the generated number is less than a threshold value, however, another random number, n, is produced. The meaning of n is that n addresses ago the to-be-generated address appeared in the address sequence. The probability, $\widetilde{a}_n$, is expressed as

$$\widetilde{a}_n = \{f(n)-f(n-1))\} - \{f(n+1)-f(n)\}. \tag{2.9}$$

The synthetic address traces generated using the extended algorithm show WS behavior similar to real world traces. At the same time, they exhibit address distributions that follow Zipf's law.

2.3 Flow-level Traffic Characteristics

The notion of a network *flow* is valuable in describing groups of packets with some common properties traveling in the same direction. For example, a packet train [18] can be seen as a network flow when defined as a burst of packets from the same source to the same destination. In the Internet, some packet header fields can be selected to identify a flow, e.g., the five-tuple { source and destination addresses, source and destination ports, protocol} identifies a connection between two communicating processes. Claffy et al. [55] discuss a parameterizable methodology for flow profiling. The parameters include flow direction, single/double endpoint(s), endpoint granularity and functional layer.

It is worth mentioning that many flow definitions are timeout-based [18, 55, 56, 57]: a flow is considered *active* when the inter-arrival time between two successive packets does not exceed a pre-defined timeout value; and is inactive otherwise. In this work, however, we identify flows simply by the common destination address of the packets and the inter-arrival time is not considered.

Numerous studies have focused on characterizing *aggregate* network traffic, where all the simultaneously active packet transmissions between network hosts are lumped together as a single flow. Recent studies have found that aggregate network traffic exhibits *fractal* or *self-similar* scaling behavior, i.e., the arrival count process of the traffic looks statistically similar on all time scales [58]. Self-similar traffic are long range dependent (LRD), meaning that the auto-correlation function has a long tail which decays slower than exponentially [59]. Erramilli et al. [60] did queueing experiments and found that LRD in packet traffic has significant performance impact.

Sarvotham et al. [61] examine Internet traffic at the connection level. It is found that the burstiness of Internet traffic is not due to a large number of flows transmitting at the same time, as assumed in some aggregate Internet traffic models [62, 63], but rather is caused by a few large files transmitted over high-bandwidth links. These connections contribute to *alpha* traffic and the rest create *beta* traffic. Methods to separate alpha and beta traffic are discussed in [64].

Brownlee and Claffy [65] study the flow patterns of measured Internet traffic, and points out that network streams can be classified by both size (*elephants* and *mice*) and lifetime (*tortoises* and *dragonflies*). Tortoises are flows lasting more than 15 minutes, which contribute to a small portion of the number of flows (one or two percent), but carry fifty to sixty percent of the total traffic.

2.4 Load Balancing for Web Servers

The rapid growth of the Internet manifests itself in, among others, the exponential growth in the request rate to some popular Web sites. To cope with the problem of Web server overload, a collection of Web servers are usually used to service Web requests. Like traditional parallel and distributed computing systems, potentially, a cluster of Web servers provides scalability, but as a system, they can only deliver high performance when the workloads are distributed in a balanced manner. In this section, we will discuss load balancing designs in some multi-server Web systems.

2.4.1 DNS-based Scheduling

One way to provide high performance Web service is via *replication*, i.e., to use multiple mirrored servers with duplicate information to serve the requests. These servers coordinate with each other to create a *distributed Web-server system* [66]. The servers share the same domain name but each has its unique IP address. A local domain name system (DNS) server responds to DNS requests from Web clients with the IP addresses of the individual Web servers. Before sending a hypertext transfer protocol (HTTP) request to a Web server, a client queries the DNS for the server's IP address. By replying with the different IP addresses of the servers, the DNS distributes the workload among multiple Web servers.

The ability of the DNS server to distribute workload in a balanced manner plays a critical role in the performance, scalability, and fault-tolerance of the distributed system. Noticing that the basic round-robin scheme is ineffective for load balancing under skewed WWW request distribution in such a system, Colajanni et al. [67] proposed enhanced round robin schemes to improve distributed Web-server system performance, taking advantage of request statistics at the client side and load information at the server side. One of the algorithms is called two-tier round robin (RR2) which divides domains into two classes: the *normal* class and the *hot* class. Based on the domains where the requests come from, RR2 assigns WWW servers in round robin fashion within each class, independently. This algorithm, combined with single-threshold server workload indicators (called *alarms*), is simple to implement and shown to be able to achieve much better load balancing result than simple RR.

2.4.2 Hash-based Routing

Proxy Web servers, also called shared Web caches (these are different from conventional CPU caches), are an effective way to reduce Web request latency in an organization with internal high-speed network connections, but only low-bandwidth or congested links to the Internet. Usually, an organization employs a collection of shared Web caches instead of just one. The main advantages of using multiple caches are elimination of a potential single point of failure, scalability, and information sharing.

Two popular schemes are used to distribute WWW documents among the caches: the Internet caching protocol (ICP) [68, 69] and the cache array routing protocol (CARP) [70, 71, 72]. When a request to an ICP cache fails, the cache queries other caches for the object and copies it if found. Otherwise, the object has to be retrieved from its original server. Different servers can cache a same object. CARP is based on an extension [71] to the robust hash routing in [70], which allows caches with different processing power and storage capacity.

Noting that it is inefficient to have multiple servers process identical requests [70] proposes the ***highest random weight*** (HRW) mapping. The name of the requested object is combined with the IP addresses of the servers to produce random values, called weights, one for each server. The request is forwarded to the server with the highest weight. This approach improves object "hit ratio" at the servers, and thus reduces response time.

HRW is basically a hash-based scheduler which, in contrast to round robin, naturally partitions objects among servers and thus improves object hit ratio at servers and reduces response time. The novelty of this scheme, however, is to incorporate server addresses in the hash keys and to select the server with the highest weight, which allows the algorithm to achieve *minimum disruption*. Whenever a server comes up or goes down, the number of objects that are remapped to another server is kept as small as possible. This is important to provide fault tolerance in case of server failures, and to ease upgrades and reconfiguration. Compared with simple hashing, the extra cost of HRW, due to minimum disruption, includes a weight calculation and comparison for every request.

Another goal of HRW is to achieve load balancing. The problem, present in all hash-based load distribution schemes, is caused by non-uniform distribution of object popularity, causing uneven workloads for the servers. Suppose that there are m servers and K distinct objects. The popularity of object i is p_i. The sum of the popularities of the objects mapped to server i $(1 \leq i \leq m)$ is q_i, which is defined as the server's popularity. It is proved in [70] that

$$CV[q]^2 = (\frac{m-1}{K-1})CV[p]^2 \tag{2.10}$$

where $CV[q]$ is the coefficient of variation of server popularity and $CV[p]$ is the coefficient of variation of object popularity. When the variance of object popularity is finite,

$$\lim_{K \to \infty} CV[q] = 0. \tag{2.11}$$

Assume that, regardless of the object and server, each request represents the same amount of work. Thus the workload for each server is proportional to its popularity. It is further proved that, as the number of objects increases, the workloads on the servers become more balanced.

In the original HRW, the number of requests are divided evenly among the servers. Ross [71] recognizes that Web caches are more likely to be different in storage and processing capabilities and proposes an extension to HRW. The idea is to assign multipliers to cache servers to scale the return values of HRW. The scaled values are used as weights to select the destination cache servers. A recursive algorithm is provided to calculate the multipliers such that the object requests are divided among the servers according to a pre-defined set of proportions.

2.5 Packet Scheduling and Load Balancing in Parallel Forwarding Systems

As mentioned in Section 1.2, one trend in router design to cope with the growth of the bandwidth demand is to have a number of *network processors* (NP's) working in parallel to increase the system throughput. A major aspect of parallel processing of network processors is their ability to scale to higher data rates of the future.

In this thesis, because our main concern is the forwarding performance of routers, we describe parallel forwarding systems composed of multiple forwarding engines (FE's). The FE's are processors engaged in per-packet processing, e.g., routing table lookup, header checksum calculation, etc. Fig. 2.2 shows a system with four FE's.

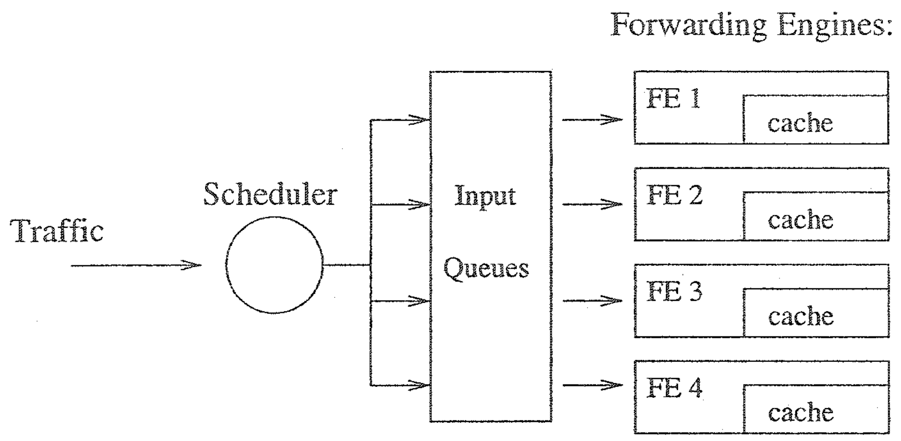


Figure 2.2: A Multi-processor Forwarding System

2.5.1 Basic Scheduling Schemes

The scheduler is responsible for dispatching the next packet in the input queue to an FE where it is processed, i.e., its output port is determined. The various schemes for dispatching packets to FE's have differing performance. There are two popular ways that packets are distributed to FE's:

Round-Robin(RR) Packets are distributed to FE's in a round-robin fashion.

Hash-based A portion of the incoming packet is used as a hash key and an index is produced based on this information. The index is used to determine the FE to which this packet should be sent.

RR is simple and efficient. Moreover, it distributes packets evenly among the FE's; load balancing is achieved naturally. This is important because load balancing mechanisms are usually needed in parallel systems to achieve maximum throughput. There are disadvantages, though. First, RR does not necessarily preserve packet ordering in a connection. Packet reordering can be detrimental to end-to-end protocol performance [73, 74]. Second, RR reduces locality in the workload of the FE compared to that in the aggregate traffic [75].

The hash-based scheme does not have the shortcomings of RR. Usually, packet header fields identifying a connection are used as hash keys by the scheduler, which ensures that packets from the same connection are delivered to the same FE in order. Hash-scheduling is known to improve temporal locality in the scheduled traffic [70, 75]. On the other hand, depending on the traffic characteristics, hashing could lead to serious load imbalance in a parallel system.

2.5.2 Internet Link Load Balancing Schemes

Cao et al. [76] classify hash-based traffic-splitting schemes into two categories: direct and table-based hashing. Direct hashing schemes use fields in a packet's header as a hash key and feed them to a hash function. The result is the index of the outgoing link for the packet. Direct hashing is simple, but has some serious limitations. Among them is the difficulty of load distribution tuning. A table-based hashing scheme adds a mapping stage after splitting the incoming traffic stream into a number of bins. A bin number is used as the index to a table to produce the index of the outgoing link. Table entries can be changed to reflect bin-to-link mapping and thus achieve load tuning.

The ability of a direct hashing scheme to balance workload depends on its abilities to produce uniformly distributed random numbers. Cao et al. [76] evaluate five schemes: *Hashing of Destination Address*, *Hashing Using XOR Folding of Destination Address*, *Hashing Using XOR Folding of Source and Destination Addresses*, *Internet Checksum*, and *CRC16*. The coefficient of variation (CV) [77] of the number of packets forwarded onto each output link is used as the measure of load balance. The results show that CRC16, the 16-bit CRC (Cyclic Redundancy Code) checksum over the 5-tuple (source and destination addresses, source and destination transport layer ports, and transport layer protocol number), produces more uniformly distributed values and achieves better load balance than the other schemes.

Simple hashing alone, however, is not enough to achieve load balancing. The randomness of packet header fields can be exploited by hashing schemes to distribute the number of flows evenly across the outgoing links, but a small number of large flows can easily cause load imbalance at the packet or byte level. Simulations in [76] show that table-based hashing that dynamically distributes traffic by monitoring queue length can achieve better load balancing than direct hashing or simple table-based traffic-splitting methods.

Jo et al. [78] propose dynamic hashing with flow volume (DHFV) which enhances a table-based hashing algorithm such as [76]. The idea is to distribute very large flows/bins during the time of load imbalance from more loaded links to less loaded ones. This approach relies on the temporal locality in Internet traffic to identify flows of large volumes. Simulation results show that this approach is very effective.

On the other hand, no extensive workload characterization has been done in [78] to justify the scheme. The question of why hashing cannot balance workload, even in the long run, is not answered. The effect of shifting different flows remains to be studied.

2.5.3 Load Balancing for Parallel Forwarding Systems

Dittman and Herkersdorf [79] describe a load balancer for parallel forwarding systems. A two-step table-based hashing scheme is used to split traffic. Packet header fields that uniquely identify a flow are used as a hash key and fed to a hash function. In the case of TCP or UDP packets, these fields include source and destination addresses, port numbers, and the protocol type. The hash function return value is used as an index to a look-up memory to derive the processor to which the packet should be forwarded. Flows that yield the same hash value are called a *flow bundle* and are associated with one processor.

In [79] three techniques are used to achieve load balancing. First, a time stamp is kept and updated at every packet arrival for each flow bundle. Before the update, however, this time stamp is compared with the current system time. If the difference is larger than a pre-configured value, the flow bundle is assigned to the processor that is currently least-loaded. Second, flow reassignment monitors the states of the input queues of the processors. Flow bundles are redirected from their current over-loaded processor to the processor with the fewest packets in its queue. Third, excessive flow bundles are detected and repeatedly assigned to the least-loaded processors. This is called flow spraying.

Kencl and Boudec [80] and Kencl [81] propose a scheduling algorithm for parallel IP packet forwarding. Their scheme is based on the HRW developed in [70] and extended in [71]. It is noticed that although HRW provides load balancing over the request object space, load imbalance still occurs due to uneven popularities of the individual objects.

An adaptive scheme is introduced to adjust loads among the FE's to prevent over-utilization of a single processor when the system is under-utilized or under-utilization of a single processor when the system is over-utilized. A second goal is to minimize the amount of packet-to-FE remappings in realizing the adaptive scheduling scheme.

The algorithm includes two parts: the triggering policy and the adaptation. Periodically, the utilization of the system is calculated and compared to a pair of thresholds to determine if the system is under or over-utilized. In either condition, the adaptation is invoked which adjusts the *weights* (called *multipliers* in [71]) for the FE's to affect load distribution. In other words, the algorithm treats over or under-load conditions as changes in the processing power of the FE's. It is shown that the adaptation algorithm can keep the minimal disruption property of HRW.

Although Kencl and Boudec [80] recognize the inability of HRW to provide load balancing, their reasoning is not complete. It is true that during a short term period, the different frequencies of individual objects can affect load balancing. But in the long term, the effect may be averaged out; HRW cannot provide load balancing not because the uneven popularities of the objects, but because the popularity distribution, e.g., the Zipf-like distribution, may not have a finite variance.

The adaptation algorithm treats different IP traffic flows in the same way, although it is realized that their popularities are not the same. It is true that HRW provides minimum disruption in cases when servers go up and down. Adaptation by adjusting weights, however, does not take flow popularity into consideration and thus may not be able to achieve this goal.

2.6 Summary

Program memory reference behavior models are relevant to our work. Memory address sequences and destination IP address sequences are similar in many aspects. The addresses can be of the same length, i.e., both 32 bits. Both sequences exhibit temporal locality. In local network environments, many studies have shown that locality exists at different protocol levels and have proposed caching schemes in network devices, e.g., routers, and network host systems, to exploit locality in their workloads to improve performance.

The IRM and LRUSM have been widely used in modeling Web traffic where the IRM captures the popularity of documents and the LRUSM characterizes temporal locality. Different caching strategies have been proposed to take advantage of both in Web server systems. We will show in later chapters that both features exist in IP destination address sequences.

We have also discussed work on flow and connection level modeling of Internet traffic. Understanding the burstiness of the Internet traffic at the connection level is important. As one application, it explains load-imbalance in parallel forwarding systems. Finally, we compared hash-based traffic-splitting schemes used in these systems.

Chapter 3

Locality Models for Aggregate Internet Traffic

In this chapter, we first discuss the motivation behind modeling locality in IP destination address sequences. On the one hand, many studies have shown that locality has a significant impact on the performance of key forwarding algorithms [13, 12, 82, 83, 84]. On the other hand, locality in IP traffic is largely ignored in network system testing and performance evaluation practices. We believe that this discrepancy stems from the lack of accurate models for IP traffic locality and effective synthetic trace generation methods.

Next, we present our methodology. Our studies are based on the measurement of real-world data. We discuss the IP packet header traces collected from a wide range of networks that are used in our experiments. These networks differ from each other in traffic volume and location in the global Internet hierarchy; yet the temporal locality in their traffic can be captured by a common simple model. Trace-driven simulations for one of the key forwarding algorithms, the IP routing table lookup process, are used to validate the models developed.

We then describe models used to capture locality properties of aggregate Internet traffic, which include the footprint model and the reuse distance model. We show that the two models measure different aspects of locality and differ in capability. The footprint model captures concentration, and was initially developed for cache miss ratio measurement; the LRU cache miss ratio is derived from the average working set size function. The reuse distance model, on the other hand, captures both temporal locality and concentration. We develop a mixed function that accurately describes reuse distance distributions in a wide range of network environments.

Our synthetic trace generation method is directly derived from the reuse distance model and has proved, via routing table lookup simulations, to be able to generate representative router workloads.

3.1 Motivation

One goal of this thesis is to quantify locality in IP destination address sequences. Given sample traces, our model can measure the quantitative difference in locality between them. Our second goal is to generate representative workloads for simulation and forwarding system benchmarking and testing. Synthetic workload models are highly desirable for simulation studies because their flexibility allows not only representative but also projected workload to be generated. Compared with real-world workloads, a synthetic workload is not necessarily less typical or accurate; moreover, for systems in the design phase, real-world workloads may not even exist. The two goals are achieved by a single generative model.

Temporal locality in Internet traffic can have a significant impact on forwarding performance. For example, the experiments on LC-trie routing table lookup algorithm [13] report that lookups with the actual traffic and routing table from the Finnish University and Research Network (FUNET) are almost twice as fast as those with random permutation of routing table entries, although the FUNET routing table is the largest among the routing tables used in the experiments. The authors explain this as due to the locality in the real world traffic.

Recently, network processors have emerged as a flexible technology to accommodate the exponential growth of the Internet [85] and have gained widespread application. Network processors, different from general-purpose processors, specialize in packet processing tasks and I/O. As many vendors now supply network processor products, e.g., the PowerNP^{TM} from IBM and the IXP series network processors from Intel, it has become important to measure and compare the performance of these systems. Locality in IP destination address sequences, although critical to forwarding performance, is not adequately considered or often ignored in network equipment testing systems and network processor benchmarks.

Some router testing systems available today simply generate packets in a round-robin fashion [86, 87] with destination addresses based on the routes in the device under test. In each round, one packet is generated with the destination address for a route in the pool. The next packet's destination address corresponds to the next route. Locality characteristics in real world traffic, which will be the real workload for the router, are completely ignored. In this generated traffic, locality depends on the ordering of the entries in the routing table!

Although the effect of locality has long been recognized in computer benchmarking [38] and incorporated into performance models of general micro-processors [88], there seems to be little consideration of locality in the evaluation of network devices where synthetic instead of real-world workload is popular. For example, in [89], synthetic traces in the benchmark consist of a fixed number of distinct destination IP addresses. On the other hand, benchmarks that do consider temporal locality, e.g., [90], have to turn to real-world traces and, as the destination addresses in these traces are often anonymized, special treatment has to be used to preserve temporal locality.

In brief, our work is driven by the increasing need to measure the forwarding performance of network devices and the lack of use of locality in current benchmarking and simulation practices.

3.2 Methodology

To validate our locality models, we compare the locality properties of synthetic traces and real-world traces. Given a real-world trace, we extract the model parameters from the data and feed them to our synthetic trace generation program. This produces a synthetic trace that resembles the real world trace in the characteristics captured by the model. Then we compare the plots of the two traces visually. In addition, we use trace-driven application-level simulation to validate models. The radix-tree routing table lookup code extracted from the FreeBSD [91] kernel is run on the SimpleScalar [92] platform. The general cache miss ratio results of the program with synthetic and the real-world traces are compared with each other. This can only be done when we have the routing table for the router where the trace was gathered.

Table 3.1: Traces Used in Experiments

Trace	Length (entries)	Description
UofA	1,000,000	A packet header trace recorded at the gateway connecting the University of Alberta campus network to the Internet backbone. For this trace, the routing table at the gateway is also available.
FUNET	100,000	A destination address trace which is used in evaluating the LC-trie routing table lookup algorithm in [13] from FUNET.
LDestIP	31,518,464	A destination address trace from the PMA (Passive Measurement and Analysis) research project at NLANR [93].

During model-building, we have experimented with the Internet header traces listed in

Table 3.1. To verify the models, we used multiple traces from the following two sets available from NLANR:

Abilene-I Two-hour contiguous bidirectional packet header traces collected from two OC48c Packet-over-SONET links at the Indianapolis router node (IPLS). We used all 48 traces from this set. The Abilene-I set contains traces measured at two router ports. There are 4 groups: CLEV-0, CLEV-1, KSCY-0, KSCY-1, each containing 12 traces.

Auckland-IV A 45-day continuous trace collected at the University of Auckland Internet access link by the Waikato Applied Network Dynamics (WAND) research group between February and April 2001. We retrieved and fit the first 37 out of 94 traces from this set. The Auckland-IV set contains traces from two directions at one port; thus there are two groups: AuckIV-0 (19 traces) and AuckIV-1 (18 traces).

To accurately capture temporal locality in IP address sequences, we use trace-driven simulation. This is in contrast with the approach taken in similar work [52, 53, 54] where the assumption of the average working set size function leads to over-simplified results which prevent them from being practically applied.

3.3 The Footprint Model

Thiebaut et al. [36] use the footprint model to generate synthetic memory reference sequences for computer programs (see Section 2.1.4). The footprint model for a sequence of addresses relates the working set size (see Section 2.1.1), $|W(t,T)|$, to the window size, T. If we assume that the address generation process is stationary, as in [17], the WS size depends only on the window size and can be expressed as $f(T)$. The definition of concentration, one of the locality characteristics (see Section 2.2.1 or [21, 22]), is based on the WS model. The concentration, C_T, in Eq. 2.6, can be determined once the footprint function $f(T)$ is known. It is found in [94] that address-level program memory reference footprint curves converge to the following hyperbola:

$$u(T) = AT^{1/\theta},\ A > 0, \theta > 1$$

Shi and MacGregor [37] capture the footprint behavior of IP destination address traces and apply the synthetic trace generation algorithm described in [36] to produce synthetic IP destination addresses. Two real world traces, UofA and FUNET, are used in the experiments. For each trace, two synthetic traces are generated: one is based on the footprint model, called *synthetic*, and the other is a random trace, called *random*, which retains the same number of unique addresses of the original trace, but is produced by selecting addresses according to the uniform distribution.

Fig. 3.1 shows the footprints of the traces where the values of θ calculated from the two real traces are 2.239 (UofA) and 2.631 (FUNET); the values of A are 11.95 and 23.25 respectively. The synthetic traces based on the footprint model approximate the real traces much more closely than the random traces. The number of unique addresses in the random trace reaches the maximum far more quickly than for either the synthetic or the real trace.

To verify the footprint model, we use both the synthetic and real traces to drive the radix tree routing table lookup algorithm and measure the general cache miss ratios. We used the SimpleScalar tool set, where the cache can be easily configured, for the simulations.

Figure 3.2 shows the results for a 16-KB direct-mapped cache with cache line sizes of 4, 8, 16 and 32 bytes. The miss ratios of the real trace are consistently lower than those of both the random trace and the footprint-based synthetic trace. The miss ratios of the synthetic trace are between those of the real trace and those of the random trace. The synthetic trace curve is closer to that for the random trace than it is to the curve for the real trace.

Locality is the concept behind caching; if a synthetic trace is generated based on a model that accurately captures locality in a real trace, we would expect the cache miss results to be close for both traces. In the case of Fig. 3.2, the synthetic model apparently underestimates the locality in the real trace. From Fig. 3.1, we can conclude that the hyperbolic footprint curve captures concentration well.

The footprint model, however, does not take temporal locality explicitly into consideration. Empirically, we calculated a crude measure of the temporal locality of each trace, namely, the number of times when a destination address was the same as the previous address. This situation occurs 147272 times in the UofA trace, 155 times in the random trace and 6232 times in the synthetic trace. The results for the FUNET traces were very similar.

For the random trace, since the addresses are drawn using a uniformly distributed random number, the expected number of repeats would be

$$\frac{TraceLength}{No.ofUniqueAddresses} = \frac{1000000}{5861} = 171$$

For the synthetic trace generated from the footprint model, we need to take a closer look at the algorithm (see Fig. 3.3 and Eq. 2.5). Repeats only occur when the generated index is 0. When $U < 1/\theta$, the index is no less than $A^{\frac{\theta}{\theta-1}}$, i.e., C_c. Given the A and θ values for the UofA trace, we have an index larger than 88. Obviously, when $U > 1/\theta$, the chance that

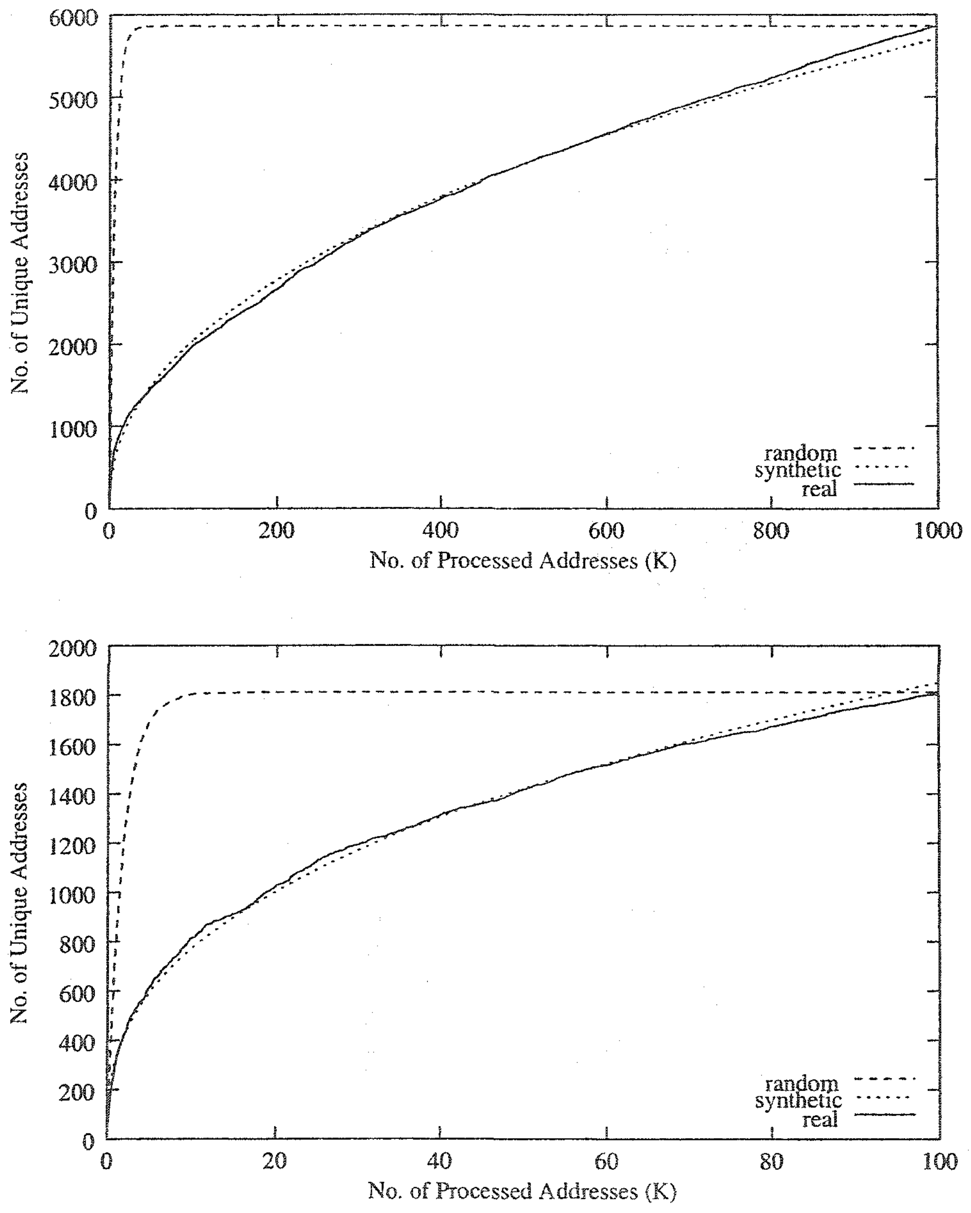


Figure 3.1: a. Footprints of the UofA (above) Traces ($A = 11.95$, $\theta = 2.239$); b. Footprints of the FUNET Traces ($A = 23.25$, $\theta = 2.631$)

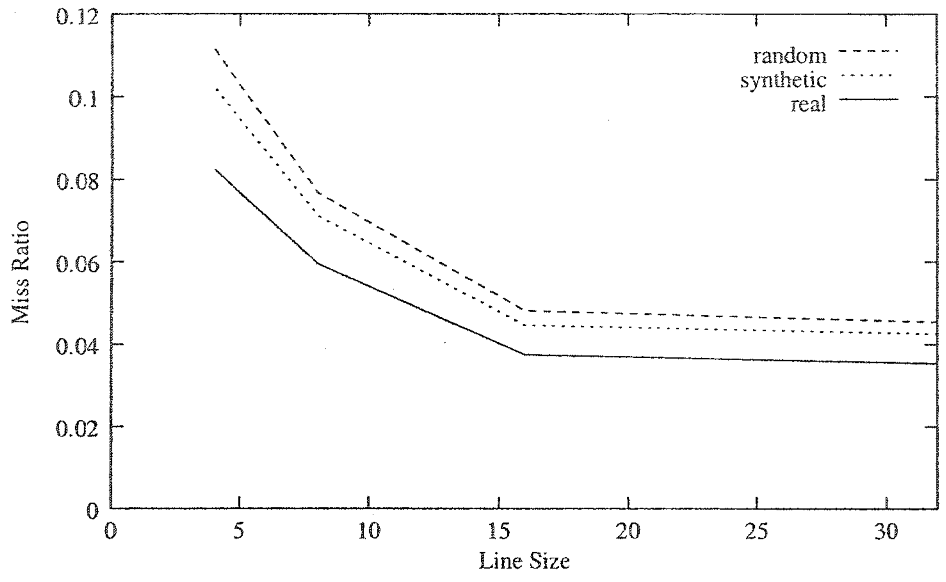


Figure 3.2: Simulation of a 16-KB Direct-Mapped Data Cache

the index is 0 is $1/C_c$. So overall, the probability of repeats in the generated trace should be

$$\frac{1 - 1/\theta}{C_c} = 0.00625,$$

far from that for the real trace.

To remedy this problem, we modified the trace generator by introducing one more parameter, the so-called ***self-repeat ratio*** (In hindsight, the self-repeat ratio is equivalent to the concept of persistence). If the hit index is below the LRU_POINTER, i.e., the address to be generated has been seen already, that address is made identical to the previous one with the probability equal to the self-repeat ratio. Otherwise, with the remaining probability, the address at the hit index is issued as before. The modified algorithm is listed in Fig. 3.3. The four lines starting at "R = Random2" are our modification to the original generator.

The self-repeat ratio observed in the UofA trace is 0.147272. With this modification, the generator will tend to issue not only repeating pairs of addresses, but also runs of three, four, and so on. With this self-repeat ratio, using the algorithm described above, we generated another trace, *syn2*. Measurement and cache simulation results are shown in Fig. 3.4. Fig. 3.4.a compares the measured footprint curves of the synthetic and syn2 traces. The footprint of syn2 matches that of the synthetic trace because we only consider generating repeating addresses when the hit index is below LRU_POINTER. Thus, in terms of the total number

```
{--- fill an LRU stack with unique items ---}
InitializeStack;

{--- generate "SyntheticTraceLength" synthetic addresses ---}

for count:= 1 to SyntheticTraceLength do
    begin
    {--- generate an index from uniform random real in [0, 1] ---}
    U:=Random1;
    if (U<1/Theta) then
        index := round((U/((A^Theta)/Theta)))^(1/(1-Theta)));
    else
        index := round(random * Cc);

    {--- determine the address to output ---}
    R = Random2
    if index<LRU_POINTER and R<SelfRepeatRate then
        address = PreviousAddress;
    else {--- move the item at "index" to stack top ---}
            UpdateLRUStack(index, address, LRU);

    {--- process the new synthetic address ---}
    UserProcess (address);
    end; {for}
```

Figure 3.3: The Synthetic Trace Generation Algorithm

of different addresses that have been issued up to a given point, the two methods produce exactly the same outcome.

Figure 3.4.b shows that the miss ratios of the trace (syn2) from the modified algorithm are closer to those of the real trace (real) than that (synthetic) from the original algorithm. This is due to the improvement in temporal locality in the synthetic trace.

3.4 The Reuse Distance Model

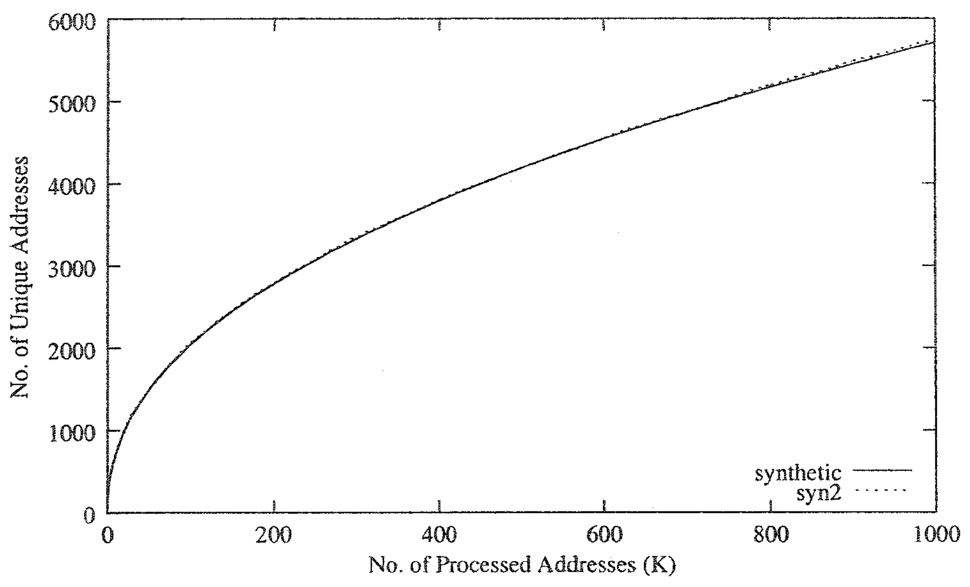
An important concept in this work is *reuse distance*, which is similar to the concept of stack distance in the LRUSM. Let "$r_1, r_2, ..., r_n, ...$" be the distance string of an address sequence "$a_1, a_2, ..., a_n, ...$." Let A_i be the set that contains the unique addresses in the address sequence $a_1, a_2, ..., a_i$. Then $|A_i|$ is the largest reuse distance seen in the trace up to and including a_i. Moreover, we have

$$r_{i+1} = \begin{cases} idx(a_{i+1}) & \text{if} \quad a_{i+1} \in A_i \\ |A_i| + 1 & \text{if} \quad a_{i+1} \notin A_i. \end{cases} \tag{3.1}$$

where $idx(a_i)$ yields the index of a_i in the LRU stack. The first appearance of an address generates the largest reuse distance so far. This differs from [35], where the reuse distance for the first appearance of an address is infinite. This treatment of the first appearances leads to the explanation of the footprint curve in the context of reuse distance.

Aida and Abe [52, 53] use the footprint model combined with the reuse distance model to generate pseudo address sequences. Their approach is to first derive the distribution of reuse distances from the footprint model and then this distribution is used with the LRUSM to generate synthetic traces. Aida and Abe [53] show that the probability distribution of LRU stack distance can be derived from the footprint curve. There it is called the Inverse Stack Growth Function or ISGF.

However, the footprint model is inherently flawed in that it does not approach an upper limit (Eq. 2.3). In reality, however, the address spaces for IP or program virtual memory are both finite. For example, for IP version 4 [2], the size of the address space is 2^{32}. As time goes by, the unique number of addresses should approach and finally reach some limit. This limit, seen at a router, could be far less than the maximum of 2^{32} addresses. This is mainly because the router only forwards packets to a subset of all the networks of the Internet.



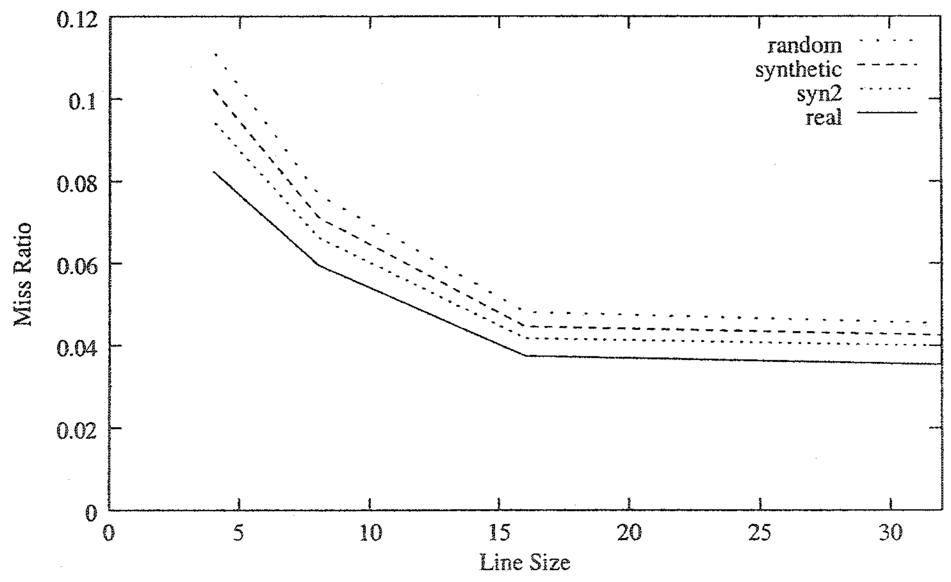
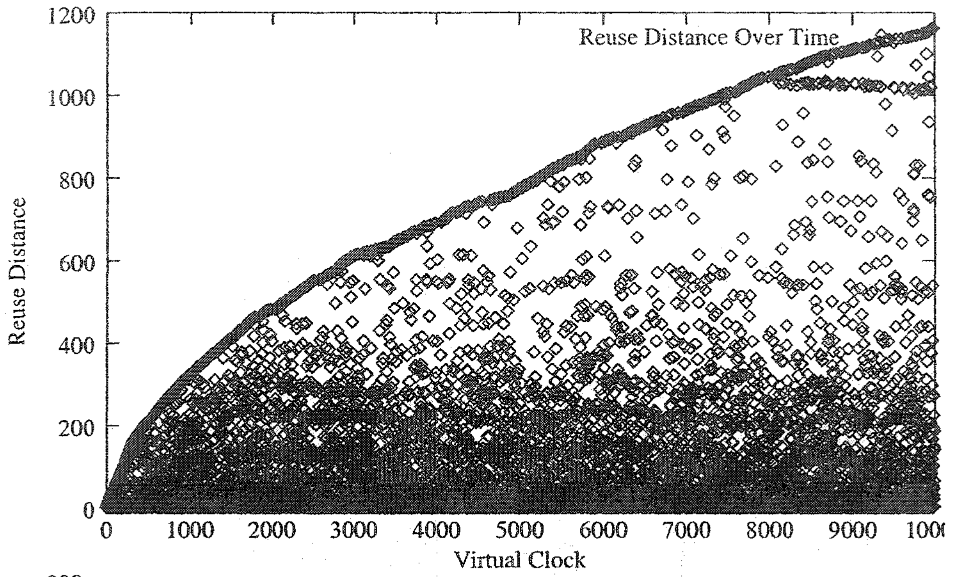


Figure 3.4: a. Footprints of the Two Synthetic Traces (above); b. Comparison of Cache Miss Ratios



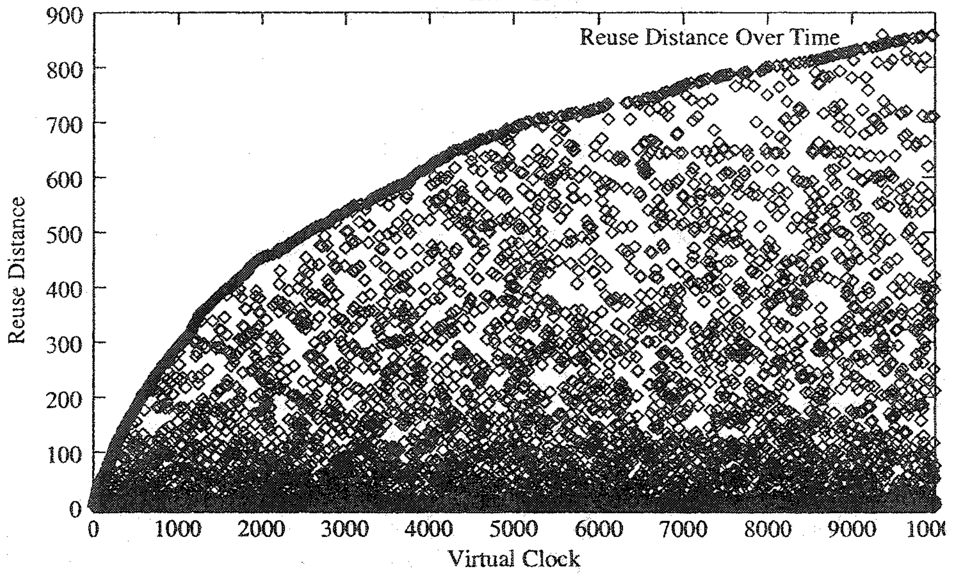


Figure 3.5: Reuse Distance Over Time: LDestIP (above) and UofA Traces

Our approach is to directly characterize the empirical distribution of reuse distances, which automatically captures the footprint behavior of the address sequences at the same time. Let $F(x)$ be the distribution function of the reuse distance and U the random variable that represents the number of distinct reuse distances. Then $E(U)$ depends on the distribution of the random vector $X_1, X_2, \dots X_M$, where $X_j, (j = 1, \dots, M)$ are independently and identically distributed. Thus, the distribution of U depends solely on $F(x)$ and M. $E(U)$ is equivalent to the average WS size, so once $F(x)$ is decided the footprint curve, representing the average WS size, is known.

Fig. 3.5 shows the reuse distance patterns for the first 10,000 entries of the LDestIP and

UofA traces. Two common features stand out:

- The reuse distances of the first appearances of the addresses form an upper border line.
- Under the border line, smaller reuse distances have higher density than larger ones.

The upper border lines in the two figures are the footprint curves of the two traces. According to its definition, a footprint curve fits the points of (*number of addresses referenced, number of unique addresses so far*) where the "number of unique addresses so far" is the same as the "largest reuse distance so far" in Eq. (3.1).

The second feature observed in the figures is the result of temporal locality. Small reuse distances indicate repeated references to the same addresses in short intervals. With the reuse distance probability, we are able to say how likely it is that an address just referenced will reappear in a given number of steps, counted by the number of distinct addresses, in the near future.

3.5 The Reuse Distance Distribution Model

3.5.1 Fitting the CCDF with a Mixed Function

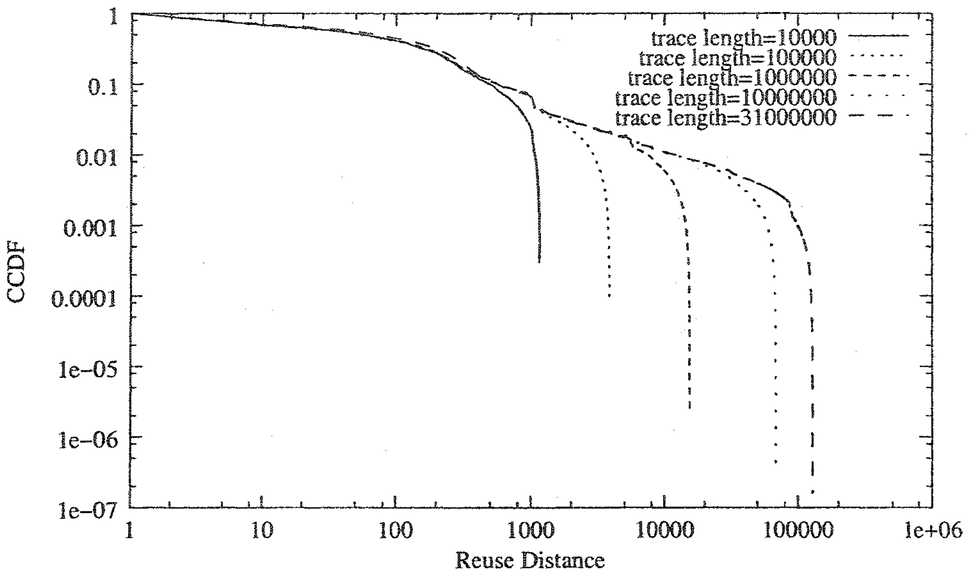
We choose the CCDF (Complementary Cumulative Distribution Function),

$$S(x) = Pr[X > x] = 1 - F(x) \tag{3.2}$$

of reuse distances to characterize temporal locality.

Fitting the CCDF instead of the CDF allows the probability of the occurrence of larger reuse distances to be more accurately captured [35]. This is based on the assumption that the larger the reuse distance of an address, the more likely it will cause a cache miss, and thus the more important it will be in cache performance evaluation. Furthermore, fitting the CCDF makes performance measurement straightforward. That is, given the size of a fully associative cache, the miss ratio of a trace can be derived from the CCDF curve.

Fig. 3.6 shows the reuse distance CCDF's for the LDestIP and UofA traces. Each curve represents a CCDF calculated from a sub-trace of a certain length. All sub-traces are longer than 10,000 entries. The CCDF's follow a pattern. Initially, the curves are all very close. This consistency across different lengths of the traces implies stationarity. Over 60 percent of reuse distances fall in this range. Afterwards, the curves for different lengths diverge. We call this part of the curve the tail. The tails start with segments of roughly straight



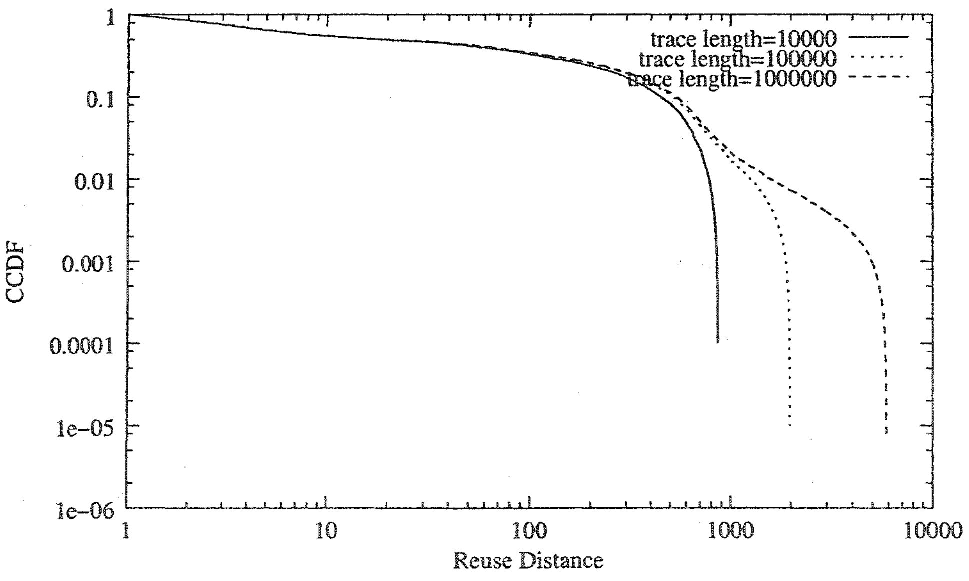


Figure 3.6: Reuse Distance CCDF's of the LDestIP (above) and the UofA Traces

lines, though there are bumps in the middle of these segments. Then they drop off nearly vertically. The diverging tails give the look of branches from a common trunk.

Measurements show that the final segments, i.e., the almost vertical parts of the tails of the CCDF's in Fig. 3.6 are linear. This linearity is due to addresses that are not *reused*, but appear only once in the traces. Using a longer trace to calculate the CCDF eliminates the linear tail of the CCDF for a shorter trace, although this introduces another longer linear tail. The reason is that the addresses that appear only once in a short trace can appear more than once in a long trace. The phenomenon that linear branches stem from a common base in all the CCDF figures reflects instability due to the limited length of the traces. In other words, if we had infinitely-long traces, there would be no linear tails in the CCDF's and they would converge to one common shape.

Given these observations, our approach to fitting the CCDF of a trace is to use the longest trace available, and remove the linear tail first. It was shown in [45] that the distribution of reuse distances for URL's in Web access traces can be fit by the lognormal distribution. However, we have not been able to fit the CCDF's shown in Fig. 3.6 using a single distribution. Instead, we use the sum of two distributions, i.e., the two-parameter Weibull and Pareto distributions [95]:

$$C(x) = pW(x) + (1-p)P(x), \quad 0 < p < 1, \tag{3.3}$$

where $W(x)$ is the CCDF of the Weibull distribution:

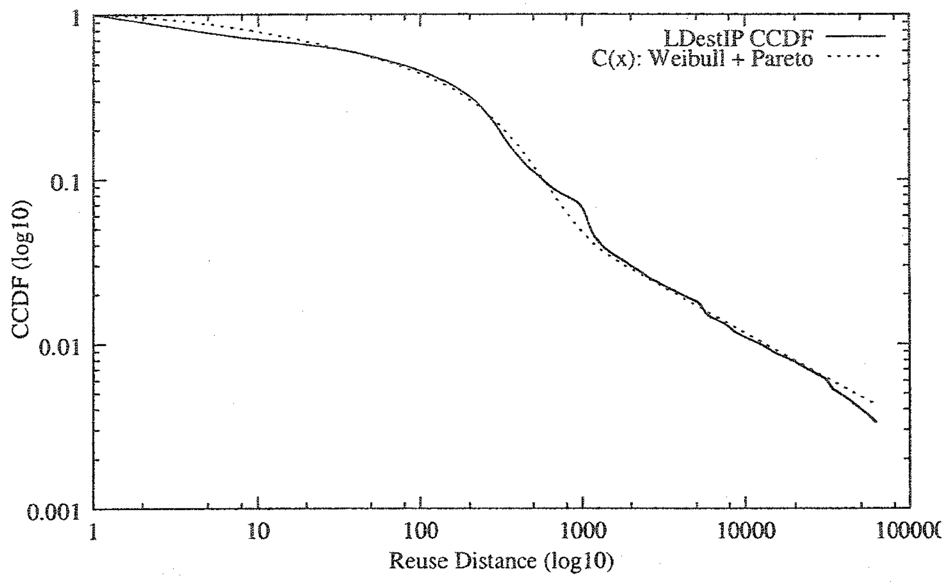
$$W(x) = e^{-(x/d)^c}, \quad c, d > 0, \tag{3.4}$$

and $P(x)$ is the CCDF of the Pareto distribution

$$P(x) = (1 + bx)^{-a}, \quad a, b > 0. \tag{3.5}$$

The c and a are called the *shape* parameters for the Weibull and Pareto distributions, respectively, and the d and b are the *scale* parameters.
Fig. 3.7 shows the fitting of the W+P (shorthand for "Weibull + Pareto") function to the empirical data, where a heuristic method is used to cut off the linear segments in the first place.

For the existing data, our experiments show that the W+P fitting predicts the tails well in that the W+P tail from fitting a shorter sub-trace visually matches the tail of a longer sub-trace. Identifying the decay trend of the tail is important because it helps to generate



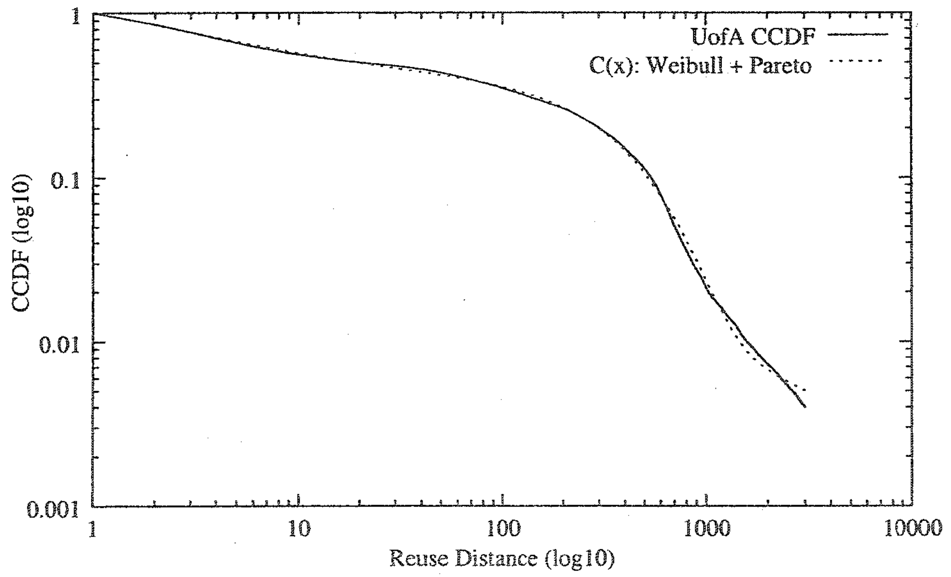


Figure 3.7: Fitting the LDestIP (left) and UofA CCDF's with Weibull + Pareto

longer synthetic traces without losing accuracy. This is the reason that the combination of Weibull and Pareto is chosen. We have tried to fit the CCDF data with other combinations, for example, two Weibull distributions (changing the $P(x)$ in (3.3) to a Weibull CCDF). The result actually shows better visual fit than the W+P scheme. However, further experiments show that for longer traces, CCDF's for synthetic traces based on the two-Weibull fitting have fewer unique addresses and the tails of the CCDF's for the real traces decay more slowly than as predicted by the two-Weibull function. We believe that the faster decay trends in the CCDF tails of the real traces in Fig. 3.7 are caused by the limited trace length.

Recently, the distributions of many important parameters in workload models for communication networks have been found to be *heavy-tailed* or *long-tailed*, meaning that the tails of the CCDF's of the distributions decay slower than exponentially. Empirical data are frequently better fit by such distributions like the Pareto, Weibull, or lognormal. According to the definition given by [96], the Pareto distribution and the Weibull distribution with c of less than 1 are all long-tailed. On the other hand, [97] showed that the lognormal distribution is not long-tailed.

After eliminating the linear segments, we have been able to fit the empirical CCDF's with mixtures of Weibull and Pareto CCDF's. The shape parameters for the Weibull are 1.14 for the LDestIP trace and 1.22 for the UofA trace. Neither satisfies the requirement defined in [96] for being long-tailed. The Pareto segments, however, are long-tailed.

It is generally not easy to tell whether a parameter is long-tail distributed or not by merely obtaining a visually good fit to some specific samples. For example, [98] shows that fitting different samples of certain measures yields different best-fit tail distributions. In our case, the available traces are not long enough to observe a stable shape of the CCDF, which makes it harder to predict the exact tail behavior.

Besides these two traces, we have also examined the reuse distance distribution pattern of traces from two other sets at NLANR, i.e., the Auckland-IV and the Abilene-I set. We experimented with all 48 traces of the Abilene-I set and randomly selected 37 out of 94 traces from the Auckland-IV set. In all these cases, with appropriate parameters, the combination of Weibull and Pareto produced excellent fits. From these experiments, we have found that the values for the parameter p in Eq. 3.3 are generally larger for backbone traffic ($p \in [0.14, 0.19]$) than for campus-level network traffic ($p \in [0.37, 0.90]$). In addition, the scale parameters, d for the Weibull and b for the Pareto CCDF, differ significantly for the two types of network traffic, but the shape parameters, c and a, are relatively constant for different sets of traces. These results are shown in Appendix A.

3.5.2 Discussion

Aida and Abe [53] derive the reuse distance probability distribution by first assuming the stationarity of the underlying address generation process (see also Section 2.2.3). The inverse stack growth function (ISFG), $f(t, \tau)$, which is equivalent to the working set size function in [17], relates the size of the stack (the number of distinct addresses) to discrete time t and the working set window size τ. In addition, the stack growth function (SGF), g, is defined as f^{-1}. The stationarity assumption leads to

$$f(t, \tau) = f(s, \tau) \tag{3.6}$$

where t and s are any points in time. Thus, the stack size is dependent only on τ. $f(\tau)$ is found to follow the power law in Eq. 2.8. The probability of the reuse distance k, a_k, is shown to be that in Eq. 2.7.

A problem is that Eq. 2.8 describes an asymptotic behavior that may not apply to small τ; this could lead to inaccurate probability prediction for smaller reuse distances that appear frequently as the result of temporal correlation in IP address sequences. For this reason, Eq. 2.7 is not acceptable for cache performance evaluations. We will show that the reuse distance distribution predicted by Eq. 2.7 differs significantly from empirical measurement results.

We derive the complementary cumulative distribution function (CCDF) for reuse distances from Eq. 2.7:

$$CCDF(k) = 1 - \sum_{i=1}^{k} a_i = (k^{1/\alpha} + 1)^{\alpha} - k \tag{3.7}$$

The CCDF appears as a straight line in a log-log scale plot (Fig. 3.8). For $\alpha = 2/3$ [53], the CCDF curve of a Pareto distribution (Eq. 3.5) also appears as a straight line in a log-log plot, and with properly tuned parameters ($a = 2.03395, b = 0.507146$) it closely matches the curve of Eq. 3.7. From the empirical result in Fig. 3.8, one may conjecture that any reuse distance CCDF in the form of Eq. 3.7 can be approximated quite well by a Pareto CCDF.

For large k and x, by binomial expansion for real exponents, Eq. 3.7 and Eq. 3.5 lead to

$$CCDF(k) = \alpha x^{1-1/\alpha} + \alpha(\alpha - 1)/2x^{1-2/\alpha} + O(x^{1-3/\alpha}) \tag{3.8}$$

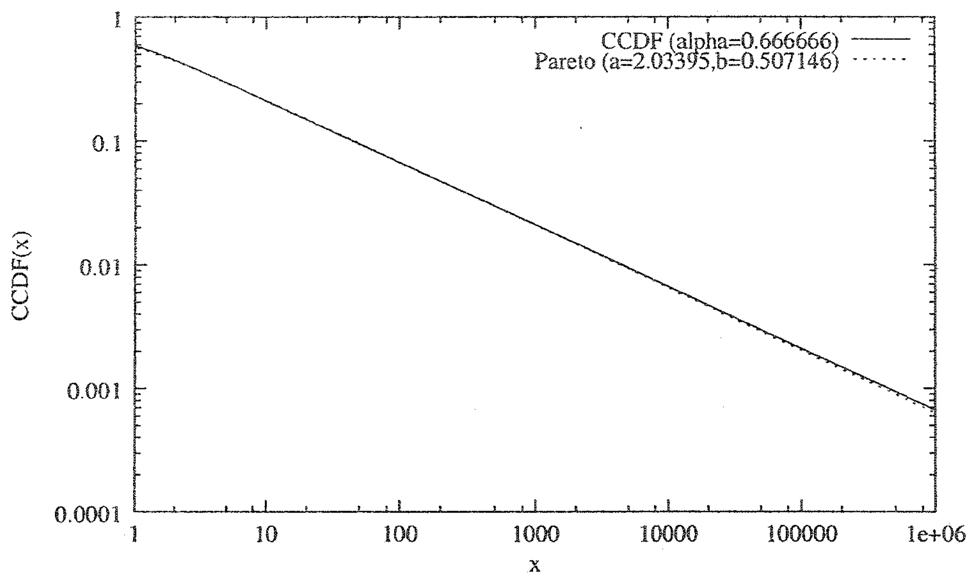


Figure 3.8: The CCDF in Eq. 3.7 matches the Pareto in Eq. 3.5.

and

$$P(x) \quad = \quad a^{-b} - b(ax)^{-b-1} + O(x^{-b-2}), \tag{3.9}$$

respectively. If $b = 1/\alpha - 1$ and $a = \alpha^{-1/b} = \alpha^{\alpha/(\alpha-1)}$, then $P(x)$ is a good approximation to $CCDF(x)$ when x is large.

In the CCDF patterns shown in Fig. 3.6, it is evident that each curve is composed of several distinct segments and cannot be fit well by a straight line in a log-log plot. In conclusion, we believe that the derivation using Eq. 2.7 is not adequate to accurately capture reuse distance distributions and thus temporal locality in IP traffic.

3.6 Synthetic Trace Generation and Simulation Results

3.6.1 Synthetic Trace Generation

With a parameterized reuse distance CCDF, synthetic trace generation is straightforward. We populate a stack with IP addresses and use a random number generator to produce reuse distances according to the CCDF, each of which is used to index the stack to output an address. After an address is produced, the stack is updated accordingly using the LRU criterion. (See Fig. 3.9.)

"InitializeStack()" populates the LRU stack with unique addresses. Though the address space of IPv4 is $4x10^9$, the actual number of unique IP destination addresses in a trace is

```
InitializeStack();
InitializePArray();
while(1) {
    R = random();
    Index = GetStackIndex(R)
    UpdateStack(Index);
}
```

Figure 3.9: Synthetic Trace Generation

limited. For example, for the 31-million LDestIP trace, the number of unique addresses is about 130,000.

The "InitializePArray()" reads an external file containing the discretized CCDF function into an internal "probability array" (*PArray[ ]*) for generating random stack indices. After the initialization, *PArray[i]* is the probability that a reuse distance is larger than *i*. This approach is more flexible than comparing random numbers to the CCDF of a certain formula with a set of parameters to decide what reuse distance to generate.

"GetStackIndex()" takes a random number and compares it sequentially to the probability array elements until one is found that is larger than the random number. Alternatively, since the array is sorted, a binary search can be used to find the smallest array element that is larger than the random number. The index of the array element is output as the reuse distance.

"UpdateStack()" keeps an "LRU_pointer" which equals the number of unique addresses accessed so far. If the "Index" passed in is larger than LRU_pointer, the address at LRU_pointer is moved to the top of the stack, and the addresses previously above it are moved one step down the stack. LRU_pointer is incremented by one. If the Index is less than the LRU_pointer, the same sequence of operations are taken except that the LRU_pointer is not updated. In both cases, the address at the Index position of the stack is output as the synthetic IP destination address.

Fig. 3.10 shows the footprint curve calculated from the synthetic UofA trace along with that from the real trace. Visually, they seem to be close enough to be regarded as generated from the same source. The result for the LDestIP trace is similar. These results show that the reuse distance model alone can capture both features of the reuse distance patterns in

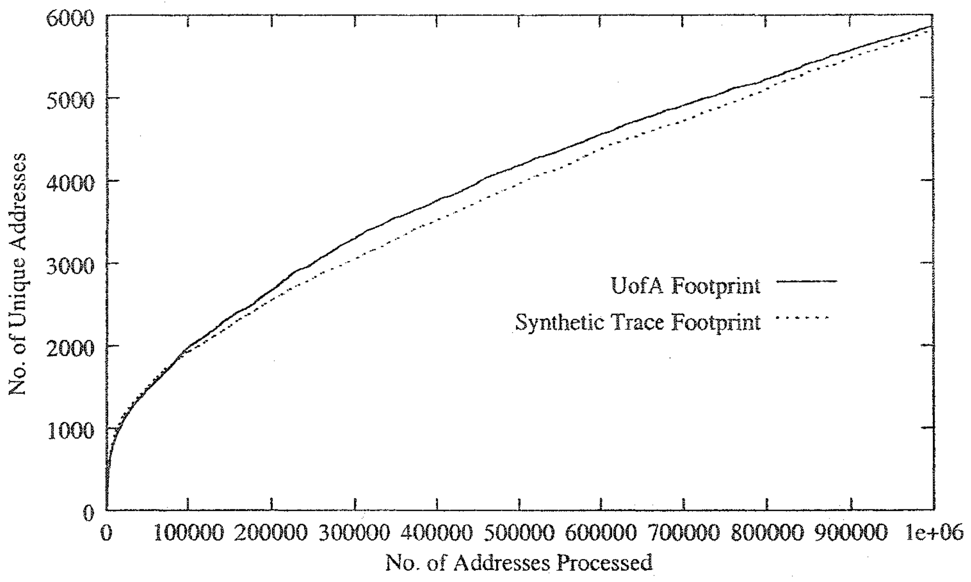


Figure 3.10: Footprints (UofA)

Fig. 3.5.

3.6.2 On the Efficiency of LRU Stack Processing

The LRU stack processing algorithm used in our synthetic trace generation is the same as the original proposed by Mattson et al. [34]: a referenced address in the stack is moved to the top of the stack and the addresses previously before it are pushed down by one. This algorithm is naturally derived from the "least recently used" concept; it keeps the data structure simple and implementation straightforward.

An early work [99] addresses the inefficiency of using the LRU stack to evaluate page reference sequences where a linear search is used to find a page in the stack. But linear search is not necessary in our trace generation algorithm because the location of an address in the stack, the reuse distance, is randomly *generated* according to Eq. 3.3. On the other hand, to improve the efficiency of updating the stack by shifting down addresses with smaller indices than that of the one currently accessed is a hard problem.

The LRU replacement policy is equivalent in concept to the *move-to-front* linear-list update rule. Sleator and Tarjan [100] studies the *amortized* efficiency of the move-to-front update and proves that it is within a constant factor of optimum, assuming that the access cost of the ith element from the front of the list is $\Theta(i)$. The update cost in our stack algorithm can be seen as access cost; thus an access to the ith address from the stack top involves i read and store operations. Thus, the naive form of stack processing is also the most efficient.

3.6.3 General Cache Simulation Results

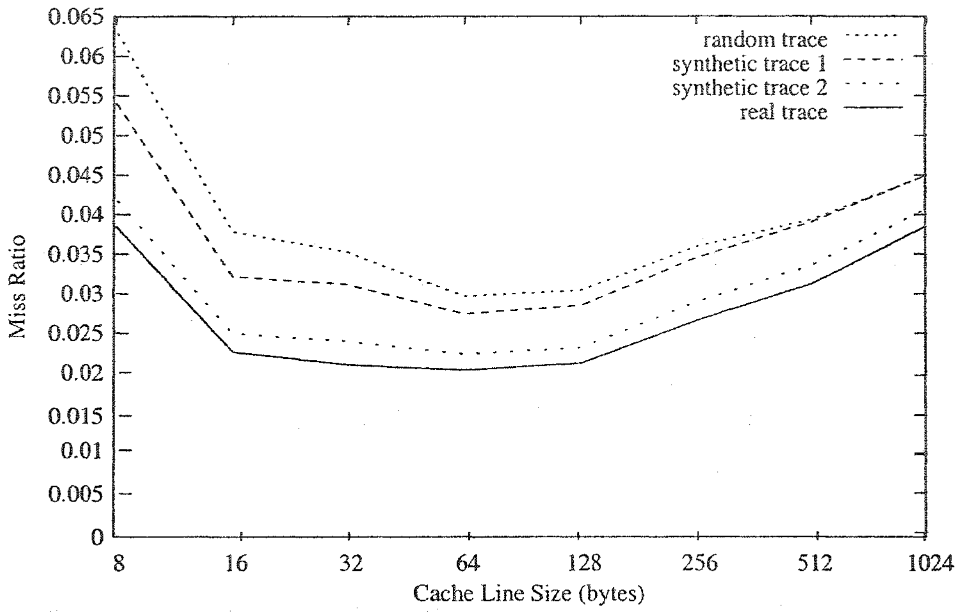
We compare cache miss ratios to verify that our algorithm produces representative synthetic traces. A synthetic trace is generated using the parameters measured from the real UofA trace. Both the synthetic and real trace are used to drive a routing table lookup algorithm under different cache configurations and the miss ratios are recorded. The UofA trace is used because the routing table of the router where the trace was recorded is available. The radix tree algorithm [9] of the FreeBSD [91] kernel is used. SimpleScalar [92] is used as the simulation platform for its cache configuration flexibility.

Fig. 3.11a shows the simulation results with a 64-KB direct-mapped LRU cache. The "random trace" is generated by randomly selecting an address out of the total unique addresses in the real trace; the "synthetic trace 1" is the trace based on the footprint model [36]; "synthetic trace 2" is the trace produced using the method described in Section 3.6.1. The miss ratios at different line sizes for synthetic trace 2 are very close to those of the real trace. The miss ratio curve of synthetic trace 1 is closer to that of the real trace than the curve of the random trace, but not as close as that of synthetic trace 2. This is because synthetic trace 1, compared with the random trace, captures one more characteristic of the real trace, i.e., the footprint curve. Synthetic trace 2 is even closer because it also captures the reuse distance distribution of the real trace. The reuse distance distribution, as discussed earlier, essentially answers the question "How likely is it that a just-accessed address is going to be referenced in the near future?". The differences between the curves for the real trace and synthetic trace 2 are mainly due to the inaccuracy of fitting the CCDF. Fig. 3.11b shows similar results for the four traces with varying cache sizes. In brief, the temporal locality of synthetic trace 2 approximates that of the real trace very well.

3.6.4 Route Cache Simulation Results

A "route cache" simulates a simple caching mechanism in routers [16]. Each cache entry contains an IP address and the corresponding output port. When a packet arrives, the router uses its destination address as an index to look up the output port where the packet should be forwarded. Again, we are interested in cache miss ratios.

There are several reasons that we use simple route caches where each entry contains



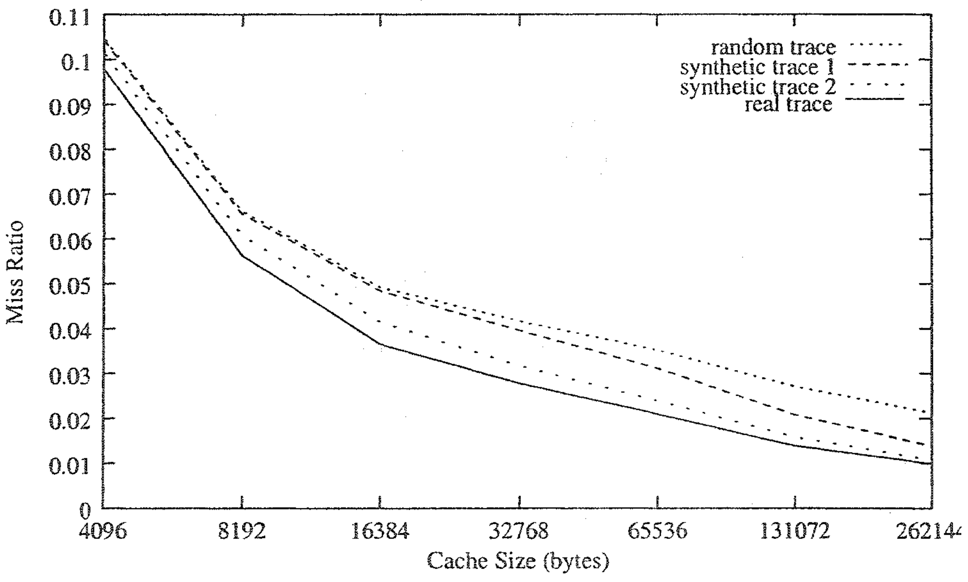


Figure 3.11: General Cache Simulation Results: a. 64-KB Direct-Mapped Data Cache (above) b. Different Cache Sizes with 32-byte Cache Lines

a whole 32-bit destination address instead of a variable-length *prefix* in the routing table. First, due to the longest-prefix-matching requirement for routing table lookup, caching prefixes is an interesting problem itself. Matching prefixes involves rather tricky operations that are likely to increase the cost of the cache. Caching full IP addresses is considerably simpler. Second, for most traces that we work on, corresponding routing tables are not available. Third, the destination addresses for traces publicly available are usually "sanitized" for security reasons, making it difficult to run lookup simulations even if the routing tables are available.

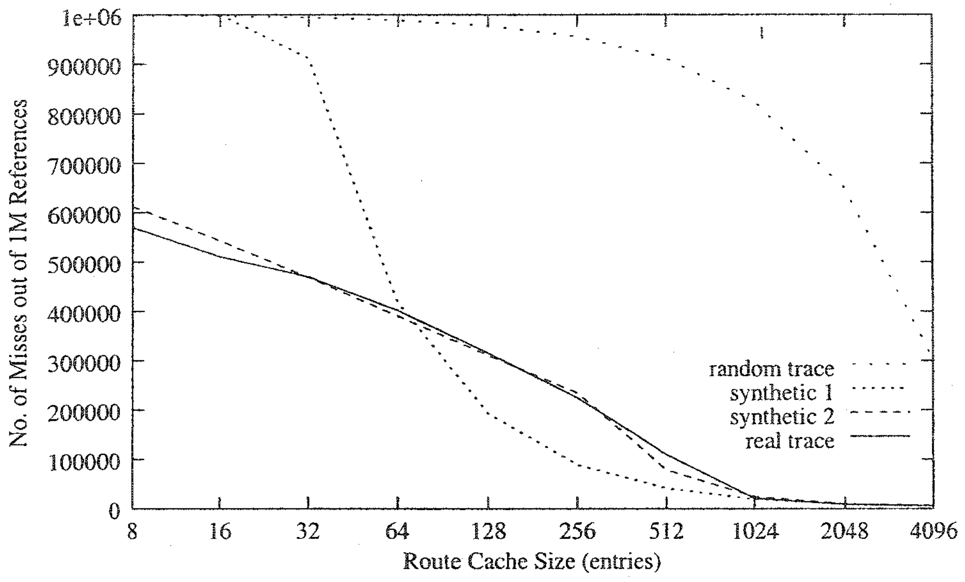
Fig. 3.12 shows the number of route cache misses for traces that contain 1 million destination IP addresses, where the real traces are the UofA and LDestIP traces (first 1 million entries), respectively, and the synthetic traces are generated using the algorithm in Fig. 3.9. The replacement algorithm is LRU. The route cache simulation results support the idea that the synthetic trace produced using the method described in Section 3.6.1 has locality similar to that of the real trace.

3.7 Summary

We discuss the motivation and methodology for modeling locality in the Internet traffic at the beginning of this chapter. The lack of consideration in network system evaluation and the obvious impact of locality on forwarding system performance is the incentive for our work. Our study of network traffic locality is based on experiments with real world traces. Locality, however, is only one of the important characteristics of forwarding system workload. As we proceed to study other aspects of Internet traffic, we follow the same methodology.

Models previously used to characterize program memory reference behavior are adapted in this chapter to the analysis of IP destination address traces. We reject the footprint model but accept the LRUSM as an effective model to capture temporal locality. Our work to fit the IP address reuse distance distribution produces a flexible model that can accurately describe the temporal locality in the various traces we have obtained. Compared with similar work on this subject, our model is more accurate in that it captures both the head and the tail of the distribution. Using this model, we have been able to generate representative IP destination address traces.

The distribution of address frequency is a traffic characteristic that cannot be conve-



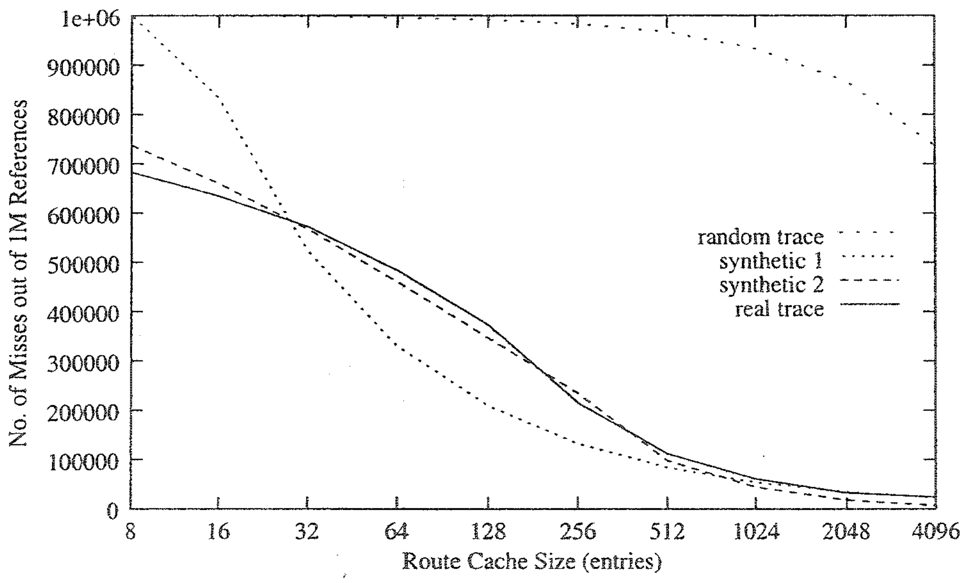


Figure 3.12: Route Cache Misses: UofA (above) and LDestIP Traces

niently generated by the simple LRUSM which treats individual addresses uniformly. In trace generation, incorporating address frequency distribution into the LRUSM is challenging. We discuss in Chapter 4 how this feature is integrated with our synthetic trace generator.

Chapter 4

An Integrated Workload Model

The LRUSM provides a means to capture temporal locality. By applying the LRU stack operations, a sequence of references is transformed into a series of reuse distances. Locality can then be quantified by characterizing the reuse distance distribution. The transformation, however, anonymizes the trace; information, such as the frequencies of distinct references, is lost. As an example, in Web workload characterization, it is noted that the LRUSM is not able to tell the difference between short-term temporal correlation and long-term popularity [48, 33, 101]. As a result, although the LRUSM can be used to characterize locality, additional models have to be built to describe other aspects of the reference strings, e.g., the distribution of reference popularity.

In the particular case of modeling IP address sequences, using the LRUSM ignores the address popularity distribution. The algorithm in Fig. 3.9 generates IP address traces with reuse distance distributions and an LRU stack populated with IP addresses. This is the reverse transformation to that in the modeling phase: from a series of randomly generated reuse distances to a sequence of IP addresses. The assumption is that the reuse distances are a realization of a stochastic process where all random variables are identically, and independently distributed. In the synthetic trace, the flows tend to be similar in rates and arrival patterns because the addresses in the stack have the same opportunity to be referenced, even though the reuse distance distribution may be skewed.

This contradicts the traffic patterns observed in [61, 65]: destination addresses are *not* evenly distributed and popular addresses are *very* popular. It has been observed that the accesses to forwarding data structures, such as the routing table, are biased toward the extremely popular IP addresses. Several data structures [82, 102, 84] are proposed to take advantage of temporal locality and the biased reference pattern in Internet traffic. The depth-constrained alphabetic tree in [82] and the biased skip list in [102] put the most recently or most frequently accessed routes near the top of the trees or at higher level link nodes to achieve high lookup performance.

The skewed popularity distribution of IP addresses has particular significance in parallel forwarding system load balancing design, especially for a hash-based scheduling system because a hash scheduler assigns packets of the same flow to the same FE. It does not help to balance the load if the hash function produces perfectly random forwarding engine identifiers, because the flows are different in rates.

In this chapter, we show that the address popularity distribution is generally Zipf-like, i.e., the pattern of "rank, popularity" pairs of the addresses can be modeled by a straight line in a log-log plot. Moreover, we show that the skewed popularity distribution and in particular, the presence of several high-rate flows, are the major source of load imbalance. This observation motivates us to incorporate flow-level information in our IP address generation model to produce realistic synthetic traffic. In addition, the observation also indicates that efficient load balancing schemes can be built by exploring flow-level Internet traffic characteristics.

4.1 Flow-level Internet Traffic Characteristics

Targeting routing table lookup in routers, we define a flow as a sequence of packets with the same destination address. This definition is sufficient for our discussion of alpha flows and performance evaluation of IP forwarding systems. In trace analysis, we do not consider the start and end of flows. We identify an alpha flow based on the popularity of its destination address.

4.1.1 Zipf's Law Applies to Internet Traffic

Zipf's law [24] is the observation that frequency of occurrence of some event (P) as a function of rank (R) is a power-law function

$$P(R) \sim 1/R^{\alpha}, \quad 1 < R \leq N \tag{4.1}$$

with the exponent α close to 1. N is the number of ranks or the number of unique events. α is the shape parameter and N is the scale parameter of the distribution.

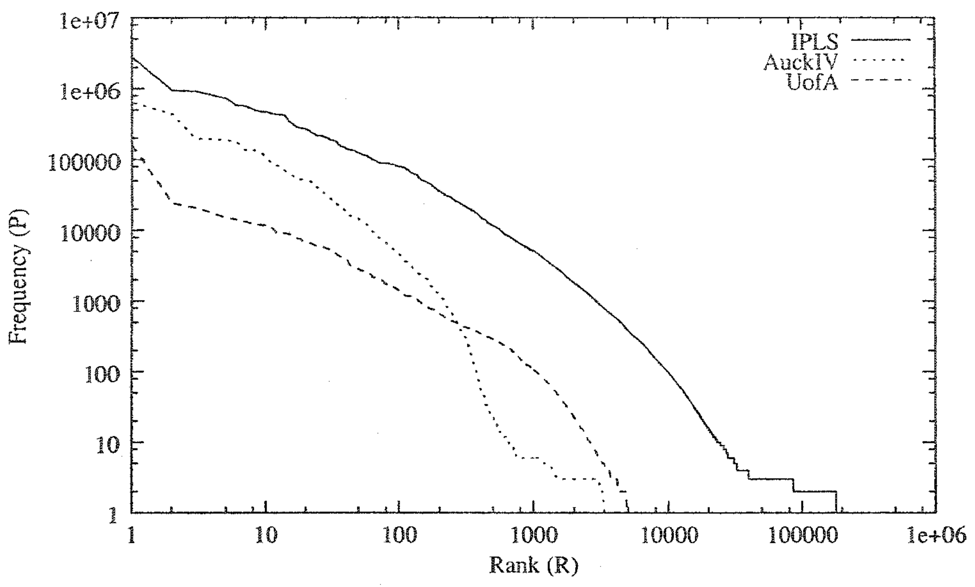


Figure 4.1: Zipf's Law for Internet Flows

The most famous examples of Zipf's law include the frequency distribution of English words, the population of communities, and the revenue of a company as a function of its rank. The existence of a small number of very popular objects seems to be a general phenomenon and has been noticed in the workloads of many computer systems. Similar observations were made in reference popularity characterization studies for Web documents [103, 47], files [104], video objects [105], Internet flows [61], etc. See the Web site *http://linkage.rockefeller.edu/wli/zipf* for a bibliography of works related to the Zipf's law. The Zipf distribution has been used in Web workload generators such as in [46] and in the streaming media generator, GISMO [106].

We experimented with three traces: the 1-million-entry UofA trace, the 4.5-million-entry AuckIV trace randomly selected from the Auckland-IV set, and the 44.8-million-entry IPLS trace randomly selected from the Abilene-I set. (See Section 3.2 for the descriptions of these sets.) Fig. 4.1 shows that Zipf's law applies to the popularity distribution of IP addresses in all three traces. Although the reasons for this distribution are unclear, we believe that the flows with the highest ranks are caused by the transmission of large files. It is evident from the figure that the values of α vary for the three traces and that the curves may not be fit with straight lines of slope (-1). Generally, however, we believe that Zipf's law is adequate for modeling the popularity distribution.

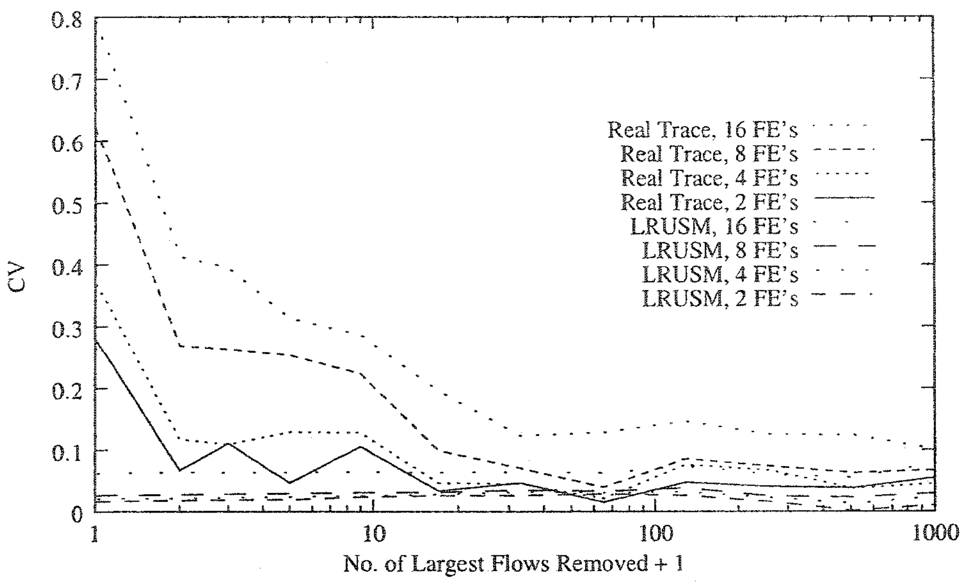


Figure 4.2: Effect of Largest Flows on Load Balancing (UofA Trace)

4.1.2 Impact on Load Balancing

We consider a parallel forwarding system similar to that in Fig. 2.2, where a simple hash-based scheduler is used. The destination address of an IP packet is fed to the hash function which returns the ID of the FE to which the packet should be forwarded.

We use B_i to represent the total busy time of the ith FE. The coefficient of variation (CV) of busy time is a measure of the degree of system load imbalance:

$$CV = \frac{\sigma_B}{\mu_B} \tag{4.2}$$

where σ_B is the standard deviation and μ_B is the mean of B_i. We assume that B_i is proportional to the number of packets forwarded by the ith FE; thus in the following simulations, we only need to record the number of packets to represent the busy time of the ith FE, which is independent of its actual forwarding rate.

CV is chosen for its independence from the units of data. It is a measure of the combined effects of the traffic being skewed toward a few flows and the system's ability to balance the load. With the simulation input fixed, CV measures the latter. In the ideal case where each FE has exactly the same load, σ_B and CV would both become zero.

To appreciate the importance of flow-level specifics in Internet forwarding systems, we show the load balance difference after removing a number of the largest flows from the

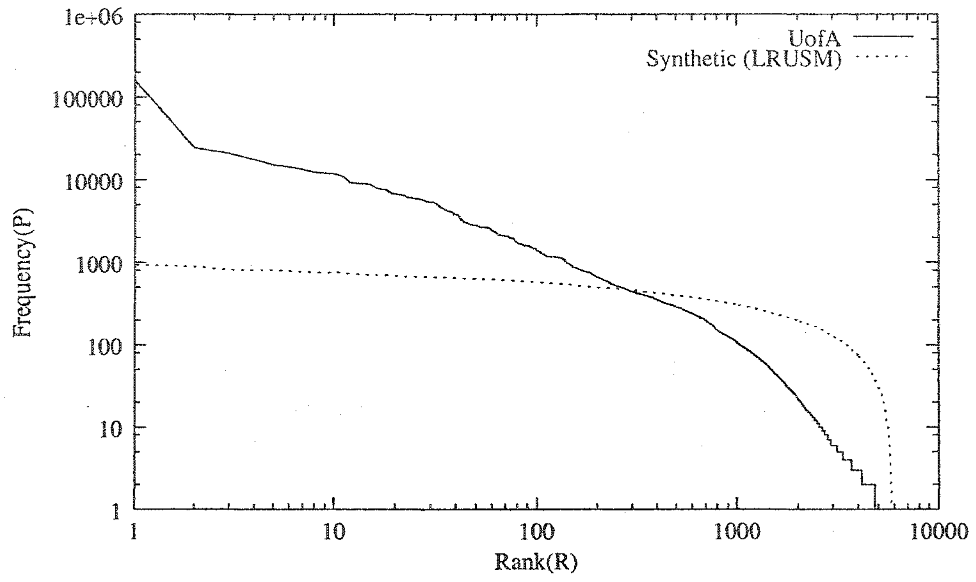


Figure 4.3: Address Popularity Patterns: Real and LRUSM-based Synthetic Traces

UofA trace in several configurations (2, 4, 8, 16 FE's) in Fig. 4.2. In each configuration, for the real trace, removing one or two of the largest flows drastically reduces CV_B values; for the synthetic trace produced using LRUSM (See Fig. 3.9), however, there are no obvious changes when the largest flows are removed. This is because the LRUSM does not have a mechanism to produce traffic with the similar address popularity distribution to that of the real traffic. For both traces, however, further removing flows has much less effect on load imbalance than removing the first several largest. From the figure, we can conclude that high-rate flows are the major source of load imbalance in a parallel forwarding system. The results indicate that balancing the alpha flows over the forwarding engines will be far more effective than scheduling flows randomly.

Furthermore, from the figure, it is obvious that the more parallelism the more serious the load imbalance for the real trace. This indicates that as more forwarding processors are used to cope with bandwidth growth, the need for efficient schemes to balance workload becomes more urgent.

4.2 Synthetic Trace Generation

Having developed an accurate reuse distance model, we try to extend the synthetic trace generation algorithm in Fig. 3.9 to take into account address popularity distribution. The resulting algorithm should be able to generate traces with the specified temporal locality as well as address popularity distribution.

The challenge, however, is that reuse distances measure the *recency* by which addresses are reused but do not describe the popularity of specific addresses. This is the reason that the synthetic traces produced by the method in Fig. 3.9 do not have the Zipf-like popularity pattern. As an example, Fig. 4.3 shows the address popularity patterns in the real and synthetic UofA traces. The difference is obvious. The numbers of addresses are much more homogeneous in the synthetic trace. There are no addresses that are popular enough to indicate dominating alpha flows.

Aida and Abe [54] try to combine the Zipf distribution of the destination address frequency with the LRU recency model. The proposed scheme divides the address generation into medium and long term behaviors, which obey the LRU model, and short time scale operations, which follow an inter-reference frequency model.

The inverse stack growth function ISGF, $f(x)$, is equivalent to the average working set size function, $s(T)$, in [17] or the footprint curve in [36]. On the other hand, the inter-reference frequency density,

$$\tilde{a}_n = \{f(n) - f(n-1)\} - \{f(n+1) - f(n)\}$$

is the negative of the numerical solution for $s''(T)$, the second derivative of $s(T)$, when $T = n$ [107], and thus is the discrete form of the *overall* inter-reference interval density function in [17].

The problem with this approach is that the $\tilde{a}_n$ is an average effect that is derived based on the stationarity assumption of the address generation process. In this sense, different flows have the same opportunity to dominate the traffic, which is not true in the real world. Although an accurate match between the Zipf curves of the synthetic and real traces is shown in [54], it is at least questionable to what extent this match can be maintained. In particular, both the traces have the length of 350,000 entries, which might be too small to observe any discrepancy.

This approach, although employing two different models, is based solely on the stationarity of the average working set size function, which is not adequate to capture the Zipf distribution of the address frequency because it does not include any flow-specific information: all the addresses have the same opportunity to be referenced. We have implemented the algorithm described in [54] and have found that the generated traces do not exhibit a Zipf-like popularity distribution.

Another Web traffic evaluation environment, WebTraff [108], provides a visually interactive front end to a set of Web traffic generation and analysis and Web proxy cache simulation tools, where the Web traffic generation tool, ProWGen, is described in [109, 110]. ProWGen characterizes four aspects of Web traffic: a Zipf-like document popularity distribution, a high degree of one-time referencing, heavy-tailed file size distributions, and temporal locality. The proposed *dynamic* LRU stack generator re-calculates cumulative probability distribution every time a document that is not on the top of the stack is referenced.

4.2.1 Incorporating High-rate Flows

Based on the discussion in Section 4.1.2, we take a different approach to synthetic trace generation. Instead of taking all the flows into account, we consider the largest ones, which represent the major portion of the overall traffic and have significant impact on performance. Our model obeys Zipf's law for these high-rate flows. Fig. 4.2 indicates that this simplification is justified for load balancing performance evaluation.

Given the number of addresses to generate, K, we can derive the number of unique addresses N, the scale parameter for a Zipf-like function (Eq. 4.1), using the footprint model (Eq. 2.3):

$$N = AK^{1/\theta}.$$

The discrete form of a Zipf-like popularity distribution is defined as

$$f(x) = \frac{1}{\Omega x^{\alpha}}, \quad x = 1, 2, ..., N, \tag{4.3}$$

where $\Omega = \sum_{i=1}^{N} i^{-\alpha}$.

4.2.2 Trace Generation Algorithm

In the following discussions, we consider the *number of inter-arrival packets* (NIP), the number of packets between two successive packets of the flow in question, as the measure of inter-arrival time between packets in the same flow. This concept enables us to integrate flows into the LRUSM.

```
0. Initialization
       Generate a reuse distance sequence in array RD[ ]
       Populate the array Stack[ ] with unique IP addresses
       Identify the flow's destination address A
       p <--- 0
1. Generate next arrival time t
2. While (p < t) Do
3.     Find i, so that i>=p, RD[i] != StackIndex(A)
4.     Swap RD[i] and RD[p]
5.     UpdateStack(RD[p])
6.     p <--- p + 1
7. Find i, so that i>=p, RD[i] == StackIndex(A)
8. Swap RD[p] and RD[i]
9. UpdateStack(RD[p])
10. p <--- p + 1
11. Go to step 1
```

Figure 4.4: Synthetic Trace Generation: LRUSM + 1 Alpha Flow

The goal of our algorithm is to implement alpha flows in the framework of the LRUSM which ensures that the temporal locality of generated traffic approximate that in real world traces. The basic idea is to reorder LRUSM-generated reuse distances to meet the NIP's generated from some packet arrival model for the alpha flows.

A stack is first initialized to contain all the distinct IP addresses. Let $r_1, r_2, ...,$ be a sequence of reuse distances generated by the LRUSM with desired temporal locality. Suppose that N alpha flows are to be integrated. At time T, the arrival model of the N flows is used to produce the $\{$*next address,* $NIP\}$ pair, say $\{A,\ NIP_A\}$. Thus, the address A is scheduled at future time t, $t = T + NIP_A$, in the generated traffic.

While $0 \leq i < NIP_A$, generating the $T+i$th address is done by producing the address at the stack position r_{T+i} with stack updating. To generate the tth address, i.e., A, we search the stack for address A (alternatively, the stack position of the alpha flows can be tracked during the generation process). Assuming that A is found at index r_A, we search forward in the reuse distance sequence $r_t, r_{t+1}, ...,$ for r_m equal to r_A. r_m is then removed from the reuse distance sequence, the address A is output, and the stack is updated. (Alternatively, A can be simply output and as later when the generating process proceeds to the point where r_A appears in the reuse distance sequence, it is removed.) We generate another pair of $\{A,\ NIP_A\}$ and the above process repeats itself, and so on.

An important question is when searching for the desired reuse distance, r_A, how far into the future (say K_A reuse distances) the algorithm is expected to look. The answer depends on the rates of the alpha flows. Suppose the arrival rate of flow A is λ_A, then $E[NIP_A] = 1/\lambda_A$, where $E[NIP_A]$ is the expected value of NIP_A. The definition of reuse distance implies that $E[r_A] \leq E[NIP_A]$, thus the upper bound of $E[K_A]$ can be expressed as

$$UpperBound(E[K_A]) = \frac{1}{P(E[NIP_A])}, \tag{4.4}$$

where $P(x)$ is the reuse distance probability density function which can be derived from Eq. 3.3.

In practice, we would like K_A to be as small as possible to avoid lengthy searching. Fortunately, since alpha flows are high-rate, i.e., any two successive packets from the alpha flow K_A are not far-apart, and therefore the reuse distances for alpha flow addresses should be relatively small. According to Eq. 4.4, this, in turn, indicates that K_A should be small. Even if the inter-arrival time (NIP) distribution is skewed, larger NIP's are still rare for high-rate alpha flows. For these reasons, the trace generation procedure usually succeeds without having to scan long sequences of reuse distances.

The algorithm that combines the LRUSM and one alpha flow is shown in Fig. 4.4. The function $StackIndex(A)$ returns the index of address A in the stack. $UpdateStack(n)$ outputs the address at stack index n and updates the stack using LRU as described in Section 3.6.1. The next arrival time is generated by the flow model (step 1). It is trivial to incorporate more alpha flows because the address A is controlled by the packet arrival model. We only need to mix all the alpha flows and generate NIP's within packets of different flows. Since this process is independent from the LRUSM, the reuse distance search and stack update procedures need not be changed.

To summarize, our algorithm needs two groups of parameters to generate synthetic traces:

Locality Parameters: the five parameters, a, b, c, d, p, of the reuse distance distribution (Eqs. 3.3, 3.4, and 3.5).

Popularity Parameters: α, the shape parameter, and N, the scale of the Zipf-like function (Eq. 4.1), and the number of alpha flows to incorporate.

4.3 Simulation

We use two real world traces in the experiments, the UofA trace and the first 1 million entries of the IPLS trace. Two synthetic traces are generated for each, one based on the LRUSM model and Eq. 3.3, the other based on the algorithm described in Section 4.2.2. 20 large flows are generated in the second synthetic trace. To better show the impact that large flows have on load balance, in the simulations we use the measured rates of the large flows from the real traces. The destination addresses of these flows are also used as those of the corresponding synthetic flows; this is necessary for simulating the load of the FE's under hash-scheduling.

We thus have two sets of three traces, UofA and IPLS. For each set, we compare the reuse distance CCDF's, the flow popularity distributions, and the load balance curves as large flows are removed from the trace. Comparing the popularity and reuse distance distributions shows how well the synthetic traces approximate the real traces in temporal locality and skewness of references. Comparing load balance validates the integrated trace generation algorithm.

4.3.1 Packet Arrival Model

Although other models, e.g., the two-state Markov chain used in [83], or the two-level arrival process described in [111], may capture the arrival pattern of packets within a flow, for simplicity, we use the Poisson model for the alpha flows. This is adequate for our discussion of load balancing.

4.3.2 Results

Fig. 4.5 shows that no significant difference exists between the temporal locality of the real trace, the LRUSM-based synthetic trace, and the integrated-model-based trace, as the CCDF's for these traces in each set are almost identical. This is easily understandable as reordering reuse distances does not disturb the distribution. This result also shows how well Eq. 3.3 captures the temporal locality of aggregate traffic.

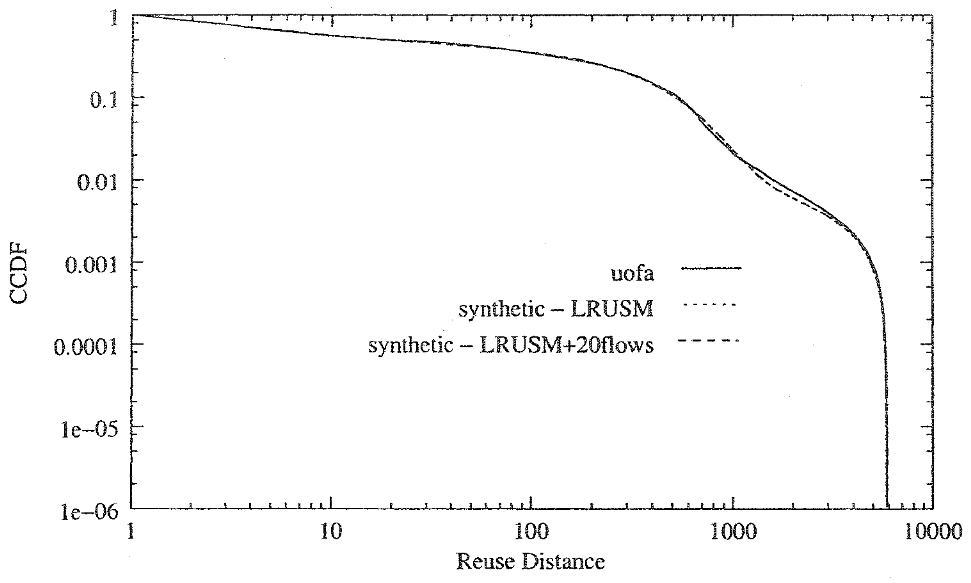
However, the flow popularity distributions of the three traces differ from each other. In the results shown in Fig. 4.6, the curve for the LRUSM-based synthetic trace is far from the Zipf-like curve of the real trace. For the integrated-model-based synthetic trace, the popularities of its largest 20 flows match those of the real trace. The popularities of the remaining flows generated by the integrated model return to a similar pattern as those of the LRUSM-based synthetic trace. The "search forward and reorder" scheme in the integrated algorithm works in practice. For trace-driven performance evaluation purposes, we do not need to simulate all the flows. As indicated by Fig. 4.2, a small number of popular flows contribute the most to the load imbalance situation in the parallel forwarding system.

This is confirmed by the load balance simulation results shown in Fig. 4.7. The *CV* value from the real trace and that from the integrated model are very close, especially when fewer than 20 large flows are removed.

4.4 Summary

In this chapter, we first note that the LRUSM does not capture the biased address reference pattern in IP traffic. We then study the popularity distribution of the addresses; we show that this distribution can be modeled by Zipf's law.

We observe that the skewed address popularity distribution and, in particular, the presence of alpha flows have significant implications for the load balancing design in parallel forwarding systems. To evaluate the performance of such systems, we developed a synthetic



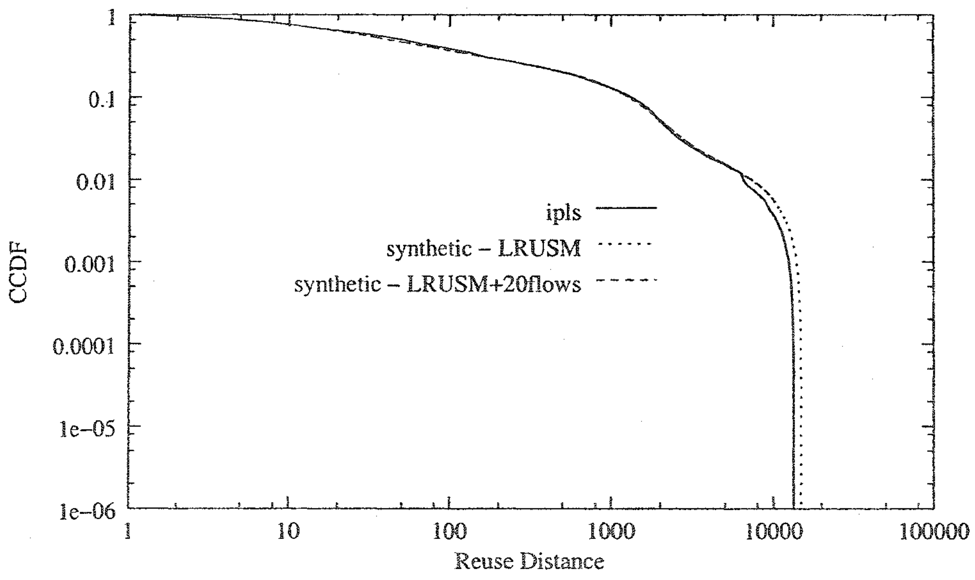
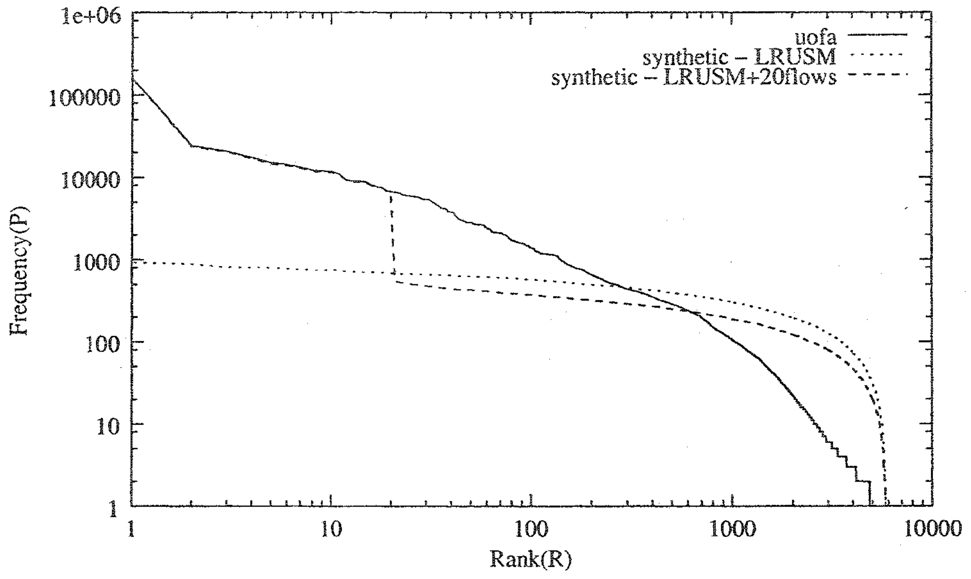


Figure 4.5: Reuse Distance CCDF's: UofA (above) and IPLS



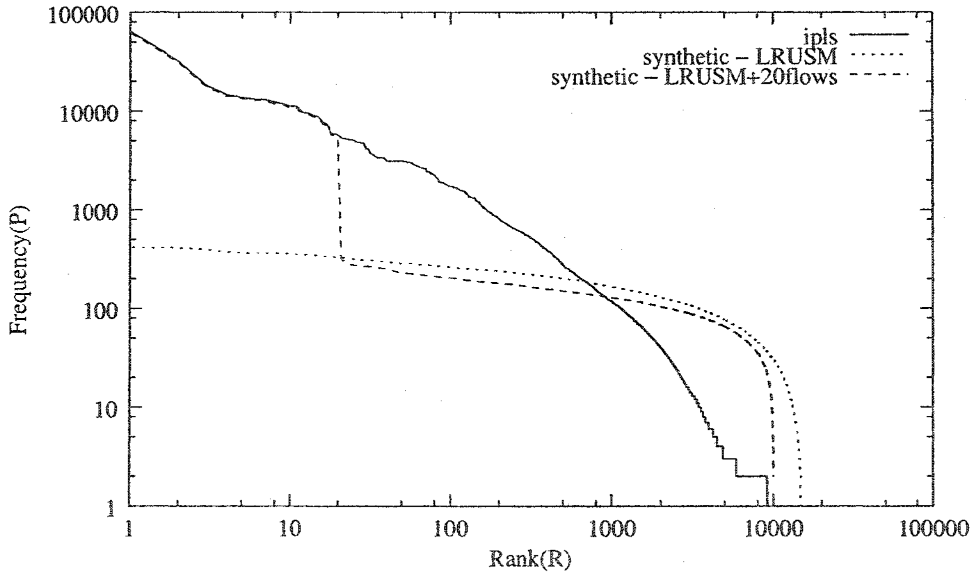
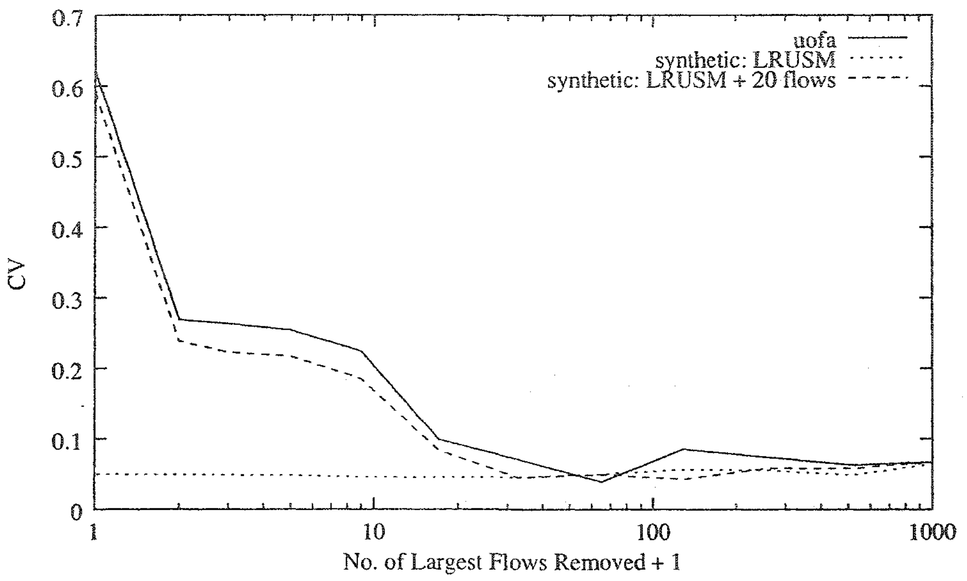


Figure 4.6: Flow Popularity Distributions: UofA (above) and IPLS



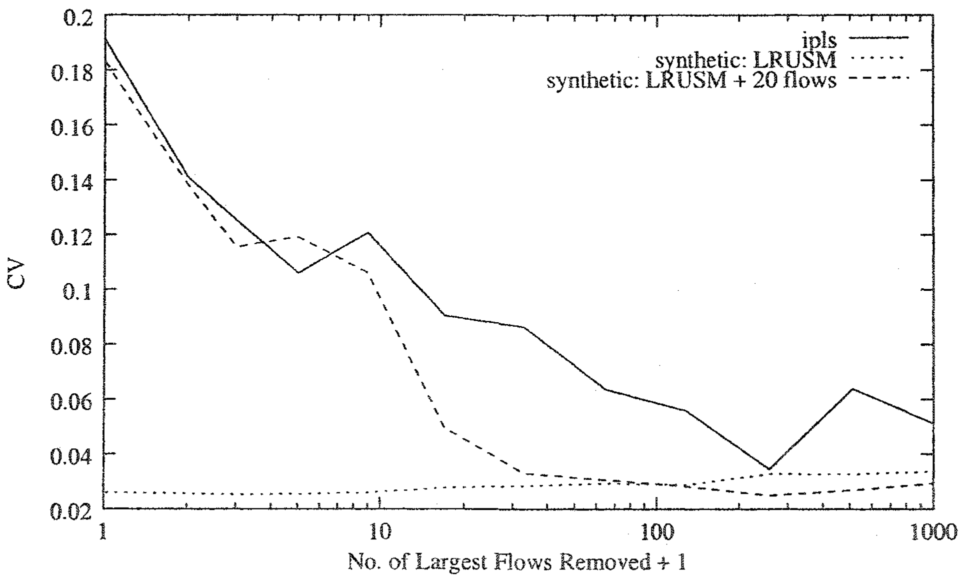


Figure 4.7: Load Imbalance for an 8-FE System: UofA (above) and IPLS

trace generation algorithm that incorporates the alpha flows into the LRUSM. The reuse distance sequence produced by the LRUSM is reordered when necessary to generate specific addresses dictated by the packet arrival model for the high-rate flows. The synthetic traffic preserves two salient features of Internet traffic: temporal locality and the presence of high-rate flows.

Simulation results validate our observation on the impact of alpha flows on load balancing. This discovery leads to the design of efficient packet distribution schemes which effectively balance workloads among multiple forwarding engines.

Chapter 5

Traffic Locality in Parallel Forwarding Systems

We consider a parallel forwarding system similar to Fig. 2.2. We call these processors forwarding engines (FE's) because one of their major tasks is to do the IP address lookup in the routing table to find the output port for the incoming packet. The caches in the figure are simple route caches that contain entries of (destination IP address, output port) pairs.

Packet dispatching schemes are critical to parallel forwarding system performance. For an incoming IP packet, the scheduler's job is to find the appropriate forwarding engine and deliver the packet to it. Usually, to accommodate variances in the packet arrival process and in the packet processing at FE's, input buffers, or queues, are put in front of the processors.

In this chapter we first discuss the impact on temporal locality of different scheduling schemes. Second, we study the effects on cache performance of the presence of high-rate flows in IP traffic.

5.1 General Impact of Scheduling Schemes on Locality

In this section, we investigate FE cache performance under two scheduling schemes: round robin and hashing:

Round Robin (RR) Given the last packet was delivered to FE i, the scheduler forward the current packet to FE $(i+1)\%N$ where % is the modulo operator and N is the number of FE's. This can happen when the per-packet processing time is fixed and the same for all the FE's. The scheduler simply delivers the next packet to the next FE. We consider strict round robin mainly for the ease of simulation. The results, however, should be extensible to similar scheduling schemes, e.g., random selection in available FE's.

Hash-based The destination address is used as the key to a hash function and the return value is the index of the target FE. For the particular hash function, we examined the simple Fletcher checksum [112] bits used in the Internet protocols and the 16-bit CRC (Cyclic Redundancy Check), as both have been shown to be able to produce uniformly distributed random numbers.

Using the model described in Section 3.5, we compare the temporal locality in two traces using their reuse distance CCDF's. Trace i is said to have better temporal locality than trace j if $CCDF_i(x) \leq CCDF_j(x)$ for all $x > 0$.

Fig. 5.1 shows the CCDF's for a 4-FE system driven by the UofA and LDestIP traces. Fig. 5.2 shows the results for the LDestIP trace in a 16-FE system. The notations used in these figures are:

$CCDF_{aggr}(x)$ is the CCDF of the aggregate traffic.

$CCDF_{rr1}(x)$ is the CCDF of the traffic processed at the first FE, under Round-Robin.

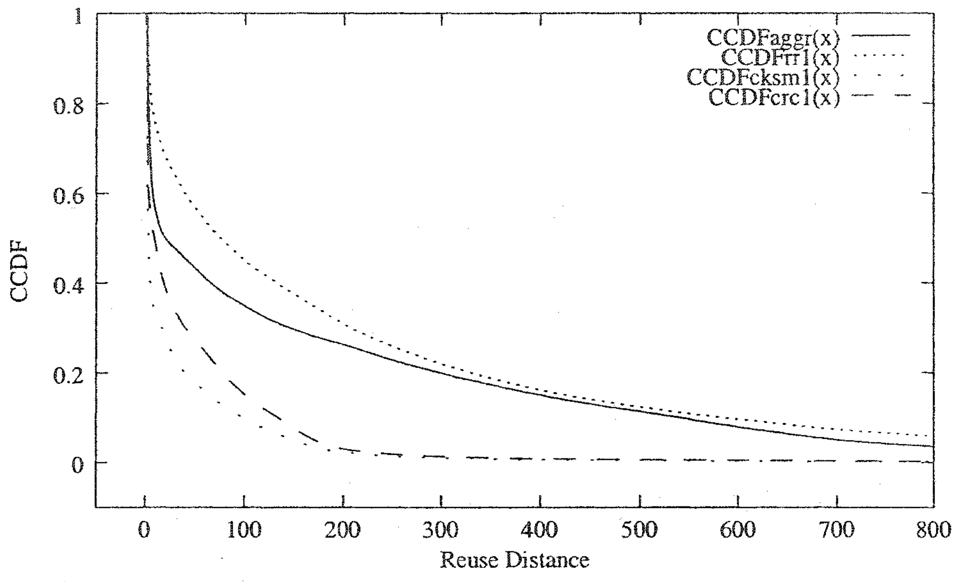
$CCDF_{cksm1}(x)$ is the CCDF of the traffic processed at the first FE, under checksum hashing.

$CCDF_{crc1}(x)$ is the CCDF of the traffic processed at the first FE, under 16-bit CRC hashing.

The overall patterns of the CCDF's at other FE's under a given scheduling scheme are very similar to that given for the first FE. Since the results are similar for the checksum and CRC hashing functions we will only discuss the results for the checksum function.

Fig. 5.1 shows the impact that the two scheduling schemes have on the locality of scheduled traffic. The CCDF's can also be seen as the curves of miss ratio versus cache size with simple destination route caches [16]. The workload at each FE under hashing has much better temporal locality and thus better cache performance than for both RR and the original aggregate traffic. For example, for the UofA trace, with 50 cache lines, the miss ratio for cksm1 is 0.178755, less than one third of that for rr1, 0.568349. With larger caches or more processors, the difference between the two disciplines is larger.

It is evident that the RR-scheduled traffic has less temporal locality and hash-scheduled traffic has more temporal locality than the original unscheduled traffic. Intuitively, RR disperses the original traffic over FE's but hashing groups packets that belong to the same flow and sends them to a particular FE, thus improving temporal locality.



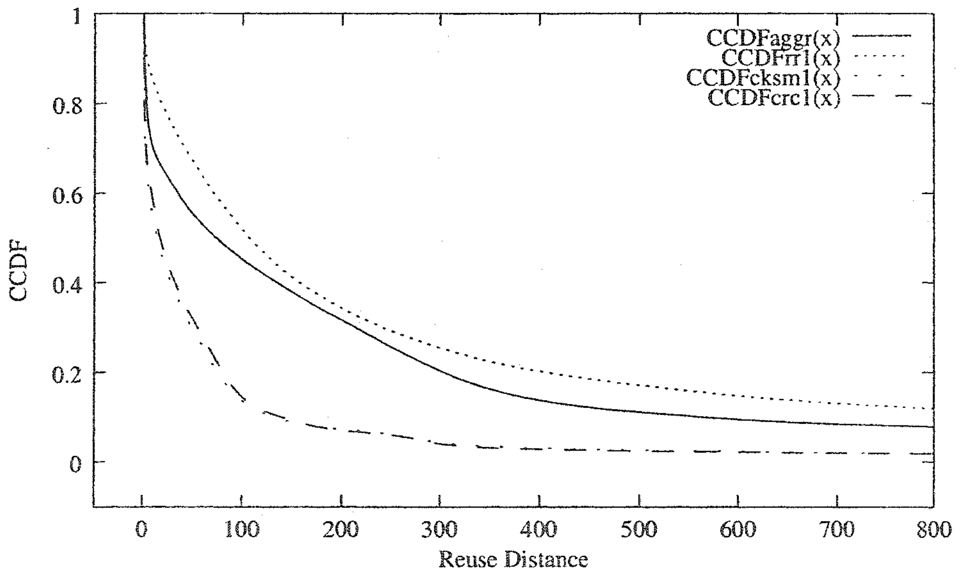


Figure 5.1: CCDF's in a 4-FE System with the UofA(above) and the LDestIP Traces

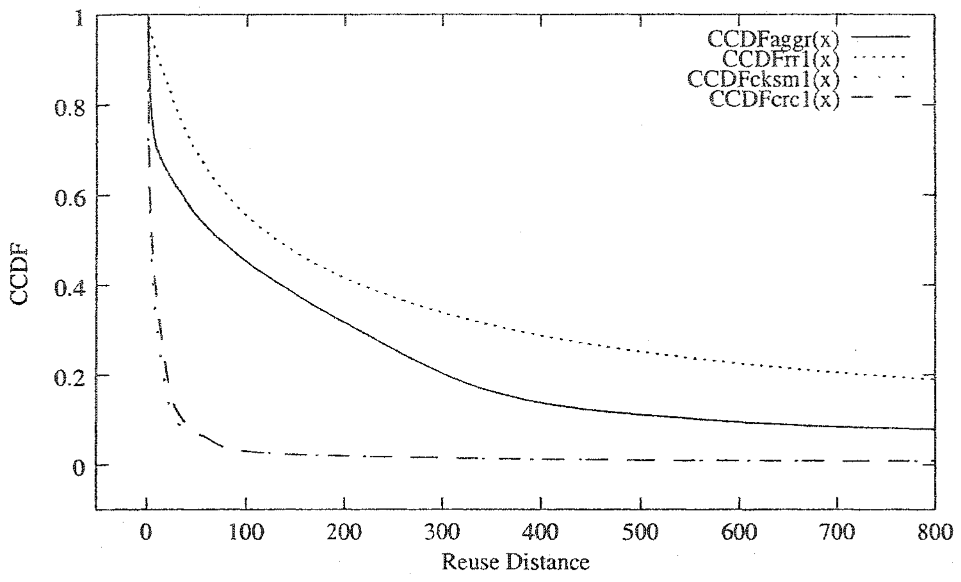


Figure 5.2: CCDF's in a 16-FE System with the LDestIP Trace

The above observations are abstracted and shown in [70] as the "Partitioning Non-Harmful" theorem which says that the expected hit ratio in a partitioned mapping (e.g., hashing) is greater than or equal to that in a non-partitioned mapping (e.g., round-robin).

5.2 Impact of Scheduling Schemes on Per-Processor Locality

Besides the improvement in temporal locality with hash-scheduling, we observed that the hash-dispatched workload in terms of packets is not the same for all processors in a parallel forwarding system. In this section, we explain the problem qualitatively. For hashing schemes, we use CRC as an example.

As shown in Table 5.1, under either CRC-hashing or Round-Robin, each FE sees a similar number of flows, although overall, the number of flows seen by an FE under hashing is significantly smaller than that under RR. The total number of flows in the UofA trace is 5861. Under RR, the number of packets seen at each FE is the same. However, under hashing, the number of packets seen at FE1 is more than twice that seen at FE2. In other words, the load under RR is perfectly balanced but skewed under hashing. The problem is that although hashing divides the number of flows almost evenly among FE's, due to the difference in flow rates, the numbers of packets processed by different FE's can differ from

Table 5.1: No. of Flows vs. No. of Packets Seen at Each FE (UofA Trace, 4 FE's)

Hashing

FE	No. of Flows	No. of Packets
1	1473	385801
2	1436	174544
3	1525	213375
4	1427	226273

Round-Robin

FE	No. of Flows	No. of Packets
1	4282	249998
2	4268	249999
3	4303	249998
4	4258	249998

each other.

Sarvotham et al. [61] class Internet traffic flows into *alpha* and *beta* traffic, where alpha traffic "is caused by large file transmissions over high-bandwidth links and is extremely bursty" and beta traffic is caused by file transmission over low-bandwidth links. When an alpha flow exists in the workload, under hashing, all packets of that flow will be dispatched to one particular FE. There are relatively few alpha flows compared to the number of beta flows, but when one alpha flow is scheduled to an FE, this FE has many more packets to process than another FE processing only short-lived and low-volume beta flows.

In a system where no cache is used, the observed skewness in workload distribution for the processors creates a load imbalance which can significantly reduce the system utilization and overall performance. However, as will be discussed in the rest of this section, per-processor locality measurements show that better temporal locality exists in the workload for the most loaded processor.

Fig. 5.3 shows the CCDF curves for the workload for each processor in a 4-FE system. They are plotted on a log-log scale to emphasize the differences between the curves under either scheduling scheme, CRC or Round-Robin. The Round-Robin curves (RR0-3) do not show noticeable difference from each other. However, the CRC curves differ from each other, with CRC0 seemingly in its own class. The other three curves under CRC, i.e., CRC1-3, are much closer to each other, but still distinguishable, for example, CRC3 is lower than the other two.

Generally speaking, under hash-scheduling, it seems that the workload for heavier loaded FE's has better temporal locality than that for relatively lightly loaded ones. This is con-

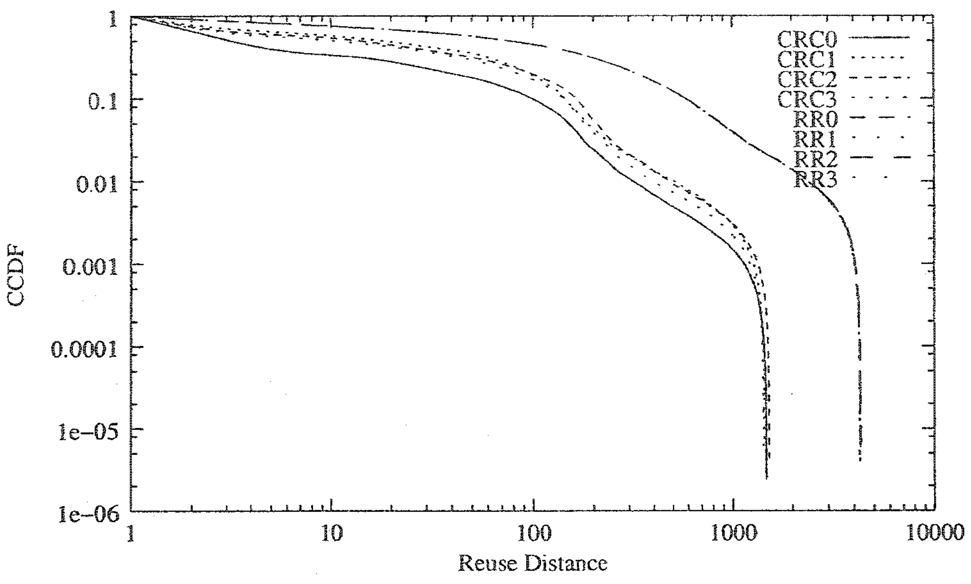


Figure 5.3: Impact of Scheduling on Per Processor Locality (UofA Trace with 4 FE's)

sistent with the results shown in Table 5.1 and Fig. 5.3. The implication is that under hash-scheduling, caching is not only effective in improving overall forwarding performance, but is also helpful in mitigating load-imbalance as a result of hashing. In other words, with a cache taking advantage of locality differences in the workload, the more heavily loaded an FE, the more efficient it becomes.

5.3 Simulations

Based on the discussion in the previous section, we can expect that in a parallel forwarding system with a hash-based scheduler, caching would be an effective way to improve system performance. Moreover, differences in temporal locality in per-processor workloads indicate that caching could also be helpful in mitigating load imbalance. In this section, we describe the simulations we conducted to verify these ideas. Simulations are simplified by the assumption that only routing table lookup operations are performed by the FE's. It is further assumed that the system has an infinite buffer that stores the incoming packets. Finally, the cache replacement algorithm is assumed to be LRU.

5.3.1 Metrics

We first consider system throughput T as the metric which is measured in terms of number of packets forwarded during some unit time period:

$$T = \frac{N}{T_d(p_N) - T_e(p_1)}$$

where

- N is the number of packets.
- $T_d()$ returns the time when a packet is dequeued.
- $T_e()$ returns the time when a packet is scheduled for lookup.
- $p_1, p_2, \ldots, p_N$ are the packets in their arrival order.

The cost of a route lookup for a destination IP address depends on the cache state. When the route is in the cache, it takes T_h to finish the lookup. When the route is not in the cache, the time is T_m which includes T_h plus the cache miss penalty. All the variables measuring time will be expressed in terms of T_m and T_h. T_m/T_h is usually much larger than 1. For example, for the BBN multigigabit router [113], it is at least 5. As the speed gap between off-chip memory and CPU widens, this ratio will become much larger. For example, in [114], it takes the μEngine 30 cycles to transfer a word both to and from memory. Even with hardware assistance, it takes 30 cycles to finish an IP lookup.

We use the coefficient of variation (CV) (Eq. 4.2) described in Section 4.3 to measure load imbalance.

5.3.2 Results

Table 5.2a shows the simulation results for a 4-FE system for the two traces. In both cases, a small amount of cache (4 entries in the UofA trace and 8 in the LDestIP trace) doubles the throughput. The differences between the results for the two traces are due to the peculiarities of each trace, for example, the composition of the trace at the flow level.

Before showing the results for the effect of caching on CV, we should note that with the hashing scheme fixed, the composition of traces in terms of flow rates affects the value of CV. Generally, the more skewed the flow rate distribution, i.e., the more dominant a few flows are in the trace, the larger the value of CV. To give an appreciation of the flow rate composition of the two traces, Table 5.3 lists the largest 10 flows in each.

Table 5.2: Simulation Results ($T_m = 6T_h$)

Cache Size	Performance($Pkts/T_h$)	
(Entries)	UofA	LDestIP
0	0.431999	0.590651
1	0.606034	0.708440
2	0.727566	0.773254
4	0.867115	0.843081
8	0.956509	0.921658
16	1.040634	1.038129
32	1.227534	1.222992
64	1.481491	1.602527
128	1.927787	2.328812
256	2.415757	2.727164
512	2.515509	3.131331
1024	2.540820	3.262716
2048	2.543437	3.359013

a. Forwarding Performance (4FE)

Cache Size	CV	
(Entries)	UofA	LDestIP
0	0.621621	0.146373
1	0.492448	0.124491
2	0.393326	0.113517
4	0.325211	0.105923
8	0.320008	0.101943
16	0.297577	0.106039
32	0.317192	0.131886
64	0.405188	0.117430
128	0.565930	0.127425
256	0.595407	0.129325
512	0.603497	0.132768
1024	0.604258	0.137589
2048	0.604258	0.139844

b. Load Balancing (8FE)

Table 5.3: High Rate Flows in the Traces

Trace	UofA	LDestIP
No. of Pkts	999,993	31,518,464
No. of Flows	5,861	130,163
No. of Pkts in 10 Largest Flows	158,707 (15.9%)	1,183,834(3.7%)
	24,245 (2.4%)	581,495 (1.8%)
	20,769 (2.1%)	524,542 (1.7%)
	17,482 (1.7%)	235,363 (0.7%)
	15,146 (1.5%)	212,150 (0.7%)
	14,305 (1.4%)	168,384 (0.5%)
	13,308 (1.3%)	160,798 (0.5%)
	12,348 (1.2%)	138,657 (0.5%)
	12,028 (1.2%)	125,531 (0.5%)
	11,824 (1.1%)	125,389 (0.5%)

Table 5.2b shows the simulation results for an 8-FE system. Generally, caches of all sizes help to reduce the CV. However, it is apparent that certain cache sizes are optimal. For the UofA trace, the optimal size is 16 entries and for the LDestIP trace, it is 32. As cache size increases, caching tends to be less beneficial in terms of helping balancing the load. In the extreme case that there are only compulsory misses, the CV approaches a fixed value. This is the case with the UofA trace (see also Table 5.1 for the number of flows for each FE.)

5.4 Summary

We have investigated the effects of two packet dispatching schemes, round robin and hashing, on the temporal locality in the scheduled workload. RR dispatches packets belonging to the same flow over multiple FE's, and thus reduces temporal locality in the workload seen by an FE. On the other hand, good hashing algorithms evenly divide the flow identifier space and assign each flow to an FE. As a result, hashing improves temporal locality in the workload of individual FE's simply by reducing the number of different flows an FE has to process. Our findings indicate that, under hashing packet distribution, caching in forwarding engines is effective.

We also study the temporal locality differences in the workloads of different FE's under a hashing scheduling. We have found that although high-rate flows dispatched to an FE tend to overload the processor, they also improve the temporal locality in its workload. This effect can mitigate load imbalance in a parallel forwarding system.

Chapter 6

Parallel Forwarding System Load Balancing

Essential to the performance of a multi-FE system like the one in Fig. 2.2 is the *scheduler* that dispatches incoming packets to the FE's. It is necessary for the scheduler to distribute workloads in a balanced manner so that the system can achieve its full forwarding potential. In this chapter, we divide a scheduler into two function units: the load *splitter* and the *balancer/adapter*. The former implements a general packet distribution policy and the latter is invoked when necessary to adjust the load distribution to improve the load balance.

In some scheduling schemes, the two functions are naturally integrated. For example, workload may be distributed in a round-robin fashion, or an incoming packet can be delivered to the FE that is least-loaded. Such schemes schedule workload at the *packet level* and are not appropriate for IP forwarding for two reasons. First, reordering of packets from individual TCP connections easily occurs in these schemes. Packet reordering within a TCP connection can give TCP a false congestion signal and be detrimental to end-to-end system performance [74, 73]. The second reason is that these schemes are not efficient in FE cache utilization [75]: by dispatching packets from the same flow to different FE's, these schemes leave copies of identical data in the caches of the individual FE's.

Hashing is a popular means to distribute load [115, 70, 76, 79, 80, 81, 78] in network systems. It is used in parallel IP forwarding systems because, in contrast to round-robin or minimum-load mapping, it is able to maintain packet order in individual TCP connections. Hashing operates at the *flow level.* The scheduler typically selects one or more header fields of an incoming IP packet, e.g., the destination address (DA), the source address (SA), the destination port (DP), the source port (SP), and the transport layer protocol number (PN). These fields define a flow and are fed as a key to a hash function; the return value is used to decide the target FE that the packet should be forwarded to. Since the selected fields remain constant for all the packets transmitted over a TCP connection, the target FE selected is the same and therefore packet order within individual TCP connections is maintained. In addition, since packets from one flow are directed sequentially to the same FE instead of scattered over several FE's, a hashing scheme is efficient in cache utilization [75].

Hashing alone, however, is not able to balance workloads under highly variable Internet traffic. Adaptive schemes are needed to accommodate the burstiness and the presence of extremely large flows [79, 80, 78]. According to our terminology, in a load scheduler, the splitter implements the hashing scheme and the balancer/adapter implements load adjustment. We call such a scheduler *hash-based.*

In this chapter,

- first, by characterizing a wide range of IP traces, we trace the sources of load imbalance in a hash-based scheduler. We show that due to highly skewed Internet flow popularity distributions, hashing alone cannot achieve load balance.
- second, we introduce a new metric, *adaptation disruption*, to measure the efficiency of adaptive load balancing schemes. For a system to achieve a high forwarding rate, disruption to FE caches, caused by load adaptation, should be as small as possible.
- last, we develop a highly efficient load balancer which, compared with state-of-the-art scheduling schemes, is unique in capitalizing on flow-level Internet traffic characteristics. The balancer implements an adaptation algorithm that shifts only high-rate flows to balance workload among FE's. This design is inspired by IP traffic characterization and the goal to achieve minimum adaptation disruption.

In Section 6.1, we present the system model that this study targets and introduce notations used throughout the chapter. Section 6.2 discusses three sources of load imbalance in a hash-based load distribution scheme. In this section, we show that generally, hashing alone cannot balance workload given Zipf-like flow popularity distributions. We introduce the concept of adaptation disruption and describe the load balancer design in Section 6.3. A critical step in our load balancer is the detection of high-rate Internet flows, which is discussed in Section 6.4. Section 6.5 presents simulation results for three adaptation policies under varying design parameters. Section 6.6 concludes this chapter and discusses future directions.

6.1 System Model

We consider a parallel forwarding system where M FE's (FE_1, ..., FE_M) process packets dispatched from the scheduler. A packet destined to FE_i is processed at once if FE_i is idle; otherwise, it is stored in a shared buffer of size B (in packets) in front of the FE's. Logically, the packet is also appended to the input queue, Q_i, of FE_i. No limits are imposed on queue lengths; only the buffer size is fixed.

The hash-based load splitter maps the incoming flows onto the individual FE's. The mapping scheme is a function H that establishes relationships between two sets, the set of flow identifiers S and the set of FE indices. That is

$$H(\cdot) : S \to \{1, 2, \ldots, M\}$$

A flow identifier is defined as a vector of one or more fields of a packet header that remain the same for all the packets in the flow. It can be one or a combination of DA, SA, DP, SP, PN. We use the destination IP addresses of incoming packets as flow identifiers in this chapter. This is a coarser level of aggregation than the popular definition of a flow, identified by the five-tuple, {DA, DP, SA, SP, PN}. The justification here is that DA sequences represent workload for major forwarding algorithms, e.g., routing table lookup and filtering. Thus, S contains all the possible destination IP addresses and the notion of flow popularity distribution is equivalent to that of address popularity distribution. Hereafter, we sometimes use destination addresses to refer to flows and it should be clear from context.

The processing power of FE_i is defined as its service rate μ_i. The total processing power is $\mu = \sum_{i=1}^{M} \mu_i$. The packet arrival rate at FE_i is λ_i which is determined by the aggregate arrival rate λ ($\lambda = \sum_{i=1}^{M} \lambda_i$) and the mapping scheme F. In this chapter, we consider only $\mu_i = \frac{\mu}{M}$, for $1 \leq i \leq M$.

6.2 Sources of Load Imbalance

We discuss three sources of load imbalance in a hash-based traffic splitting scheme.

6.2.1 Hash Function

The mapping scheme F has to be able to generate uniformly distributed random FE identifiers for the source set S. The result is that, on average, K/M flows are mapped to each FE. Although for a non-random input, it is theoretically impossible to define a hash function that generates random output, it is not difficult in practice to find a scheme that approximates random data generation [116]. Jain [115] and Cao et al. [76] have found that the Internet checksum algorithm and the CRC over the five-tuple {DA, SA, DP, SP, PN} give good random outputs.

6.2.2 Burstiness of Internet Traffic

Packet network flows are known to be *bursty*, i.e., packets of a flow travel in groups [18]. A large number of packets from one flow arriving at one FE in a short period can swamp the processor. At the same time, other FE's may be idling. The bursty nature of Internet traffic can lead to temporary load imbalance and cause packet dropping. Aside from adjusting flow mappings adaptively, buffering and provisioning are the common practices to accommodate bursty packet arrivals.

6.2.3 Skewed Flow Size Distribution

In this section, we extend the discussion on skewed Internet flow popularity distributions in Chapter 4 and show that load distribution schemes based on hash only cannot balance workloads in the Internet environment.

Flow-level Internet Traffic Characteristics

Table 6.1: Traces Used in Experiments

Trace	Length(entries)	Description
FUNET	100,000	A destination address trace which is used in evaluating the LC-trie routing table lookup algorithm in [13] from Finnish University and Research Network (FUNET).
UofA	1,000,000	A 71-second packet header trace recorded in 2001 at the gateway connecting the University of Alberta campus network to the Internet backbone.
Auck4	4,504,396	A 5-hour packet header trace from National Laboratory of Applied Network Research (NLANR) [93]. This is one from a set of traces (AuckIV) captured at the University of Auckland Internet uplink by the WAND research group between February and April 2000.
SDSC	31,518,464	A 2.7-hour packet header trace from NLANR. Extracted from outgoing traffic at San Diego Supercomputer Center (SDSC) around the year 2000.
IPLS	44,765,243	A 2-hour packet header trace from NLANR. This is from a set of traces (Abilene-I) collected from an OC48c Packet-over-SONET links at the Indianapolis router node.

To study flow-level Internet traffic characteristics, we have experimented with traces collected from networks ranging from campus to major Internet backbones. We show the

Table 6.2: High Rate Flows in the Traces

Trace	FUNET	Auck4	IPLS
No. of	8,233 (8.2%)	640,730 (14.2%)	2,788,273 (6.2%)
Pkts	7,424 (7.4%)	440,149 (9.8%)	944,253 (2.1%)
(Percent)	2,971 (3.0%)	196,513 (4.4%)	919,088 (2.1%)
	2,470 (2.5%)	194,757 (4.3%)	808,773 (1.8%)
	2,298 (2.3%)	186,095 (4.1%)	732,339 (1.6%)
	1,614 (1.6%)	177,388 (3.9%)	582,367 (1.3%)
	1,387 (1.4%)	135,286 (3.0%)	570,316 (1.3%)
	1,317 (1.3%)	135,033 (3.0%)	510,043 (1.1%)
	1,309 (1.3%)	132,812 (2.9%)	473,562 (1.1%)
	1,258 (1.3%)	104,716 (2.3%)	470,072 (1.1%)

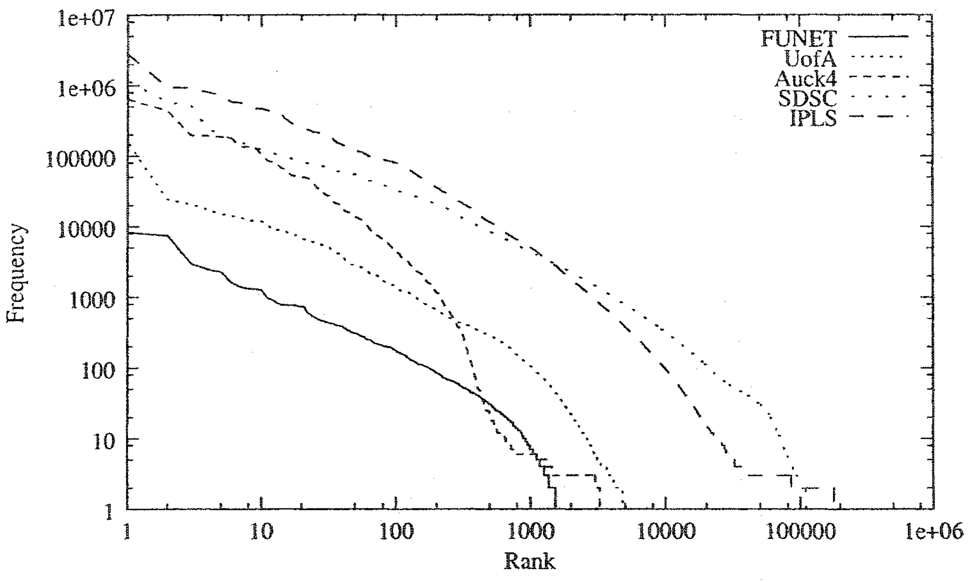


Figure 6.1: IP Address Popularity Distribution Follows Zipf's Law

results for three traces (see Table 6.1). The address popularity distributions in these traces are shown in Fig. 6.1. We match each curve by a straight line, i.e., a Zipf-like function, in the log-log plot. The slopes fit for the five traces, SDSC, FUNET, UofA, IPLS, and Auck4, are -0.905, -0.929, -1.04, -1.21, and -1.66, respectively. Common to all traces is the presence of several popular addresses dominating a large number of less popular addresses. Table 6.2 shows the number of packets in the ten most popular flows of three traces (the statistics for the other two traces can be found in Table 5.3). This common phenomenon is the motivation of the load balancing scheme developed in this chapter.

Implications for Load Balancing

The flow popularity distribution adds another dimension to the load balancing problem. In [76], it is realized that "long packet trains will have negative effects on traffic splitting performance", and "traffic splitting is significantly harder when there is a small number of large flows." Their solution is a table-based hashing scheme where mapping can be tuned by adaptive load monitoring mechanisms, which forms the basis for the load balancing scheme described in [79].

While hashing may manage to balance workloads in the average sense when the flow popularity distribution is homogeneous, i.e., with a finite variance, as proved for HRW in [70], it cannot when the distribution is so skewed that the coefficient of variation (CV) is infinite.

Let K be the number of distinct addresses, i.e., the size of S. Let p_i $(0 < i \leq K)$ be the popularity of address i and let q_j $(0 < j \leq M)$ be the number of distinct addresses distributed to FE j. It is derived in [70] that HRW, or any hash function that generates uniformly distributed random numbers over its hash key space, distributes workloads in a balanced way. This occurs when the load imbalance of the system, expressed as the CV of q_i:

$$CV[q_i]^2 = (\frac{M-1}{K-1})CV[p_i]^2, \tag{6.1}$$

approaches zero as K and the number of packets approach infinity. The condition here is that $CV[p_i]$ should be finite.

The discrete-form probability density function (PDF) of a Zipf-like distribution (Eq. 2.2) is:

$$P(X = i) = \frac{1}{Z} i^{-\alpha}, \quad i = 1, 2, \ldots, K, \ \alpha > 1 \tag{6.2}$$

where Z is a normalizing constant:

$$Z = \sum_{i=1}^{K} i^{-\alpha} \tag{6.3}$$

Given that the average popularity of the K objects, $E[p_i]$, is $\frac{1}{K}$, we have

$$\begin{aligned} CV[p_i]^2 &= \frac{Var(p_i)}{E[p_i]^2} \\ &= \frac{E[p_i^2] - E[p_i]^2}{E[p_i]^2} \end{aligned} \tag{6.4}$$

$$= \frac{\frac{1}{K}\sum_{i=1}^{K}\frac{1}{Z^2}i^{-2\alpha}}{\frac{1}{K^2}} - 1$$
$$= \frac{K}{Z^2}\sum_{i=1}^{K} i^{-2\alpha} - 1$$

Substituting the $CV[p_i]^2$ in Eq. 6.1, we have

$$CV[q_i]^2 \sim \frac{K(M-1)}{Z^2(K-1)}\sum_{i=1}^{K} i^{-2\alpha} \tag{6.5}$$

As $\alpha > 1$ and $K \to \infty$, items Z and $\sum_{i=1}^{K} i^{-2\alpha}$ converge, and thus $CV[q_i]^2$ is non-zero. Zipf-like distributions (Eq. 2.2) are known to have infinite variance when $\alpha \leq 3$ and infinite mean when $\alpha \leq 2$. This is the reason that a hash-based scheme, such as HRW [70], is not able to achieve load balancing when the population distribution of objects in its input space, in our case destination IP addresses, is Zipf-like.

6.3 Load Balancer

In addition to general desirable features for load-splitting schemes, to measure the efficiency of adaptive load balancing schemes, we introduce the concept of *adaptation disruption*. Minimizing this metric is achieved by scheduling only high-rate flows.

6.3.1 Goals

The goals of load-splitting algorithms [70] for Web proxy cache systems apply for the packet schedulers in parallel forwarding systems. First, the scheduler shown in Fig. 2.2 is in the data forwarding path and therefore should be as efficient as possible to reduce delay. Second, load balancing is crucial for the system to achieve its full forwarding potential. As discussed in Section 6.2, hashing alone cannot achieve load balancing; it is therefore necessary for the scheduler to monitor the workloads on the FE's and perform adjustments at appropriate times. Third, since each FE usually has its own local fast storage functioning like cache, higher hit ratio is desirable. FE cache hit ratio is mainly determined by temporal locality in IP traffic. Scheduling schemes have a big impact on temporal locality seen at each FE [75]. Finally, the system has to be fault-tolerant to provide reliability and graceful degradation when one or more FE's fail.

Typically, when a system is unbalanced to some degree, the adaptation mechanism will be triggered to make adjustments to the mapping from the system's input to output [79, 80]. The result is that some flows will be shifted from the most loaded processors to less loaded ones. When remapping happens, it is desirable that the number of flows shifted is small to cause minimum disruption to FE caches.

Most state-of-the-art schedulers migrate flows without considering their rates, but this is ineffective. The probability of shifting low-rate flows is high since there are many of them. Shifting these flows does not help re-balance the system much, and causes unnecessary disruption. The high-rate flows are few so it is unlikely that they would be shifted, but it is usually these flows that cause trouble [117]. While the scheduler is busy shifting low-rate flows, the high-rate ones keep swamping the overloaded processor(s).

Thus in a hash-based parallel forwarding system, another feature is desirable; we call it minimum adaptation disruption (MAD). In adaptation, migration of flows from one FE to another renders some previously cached data in the source FE useless and causes *cold start* in the target FE's cache. We call this phenomenon *adaptation disruption*. Obviously, flow migration is harmful to forwarding performance and should be done as infrequently as possible. At the same time, when migrating, the number of flows to be shifted should also be minimized. For N_F packets forwarded, adaptation disruption, denoted by ζ, is defined as follows:

$$\zeta = \frac{N_S}{N_F}, \qquad (6.6)$$

where N_S is the number of flow-shifts.

Note that minimum adaptation disruption is different from the minimum disruption in HRW which describes the desirable behavior of a distributed system in the face of partial failure. Redirecting only flows for a failed FE causes least disruption to the states of other FE's. Adaptation disruption, on the other hand, is caused by flow migrations among FE's as a result of load balancing efforts. It measures the degree of disturbance to cache during forwarding. As the performance gap between computer processor and memory keeps widening, it is important for an adaptive scheduler to achieve MAD to maintain overall forwarding performance.

In addition, MAD is also desirable for maintaining packet order within TCP connections. When flows are shifted from a heavily loaded FE to a less loaded one as the result of adaptive load balancing, it is hard to maintain the original packet order for these flows. Packets of the shifted flows arriving after the migration are very likely forwarded before some previous packets that still wait in the queue of the previously heavily loaded FE. For this reason,

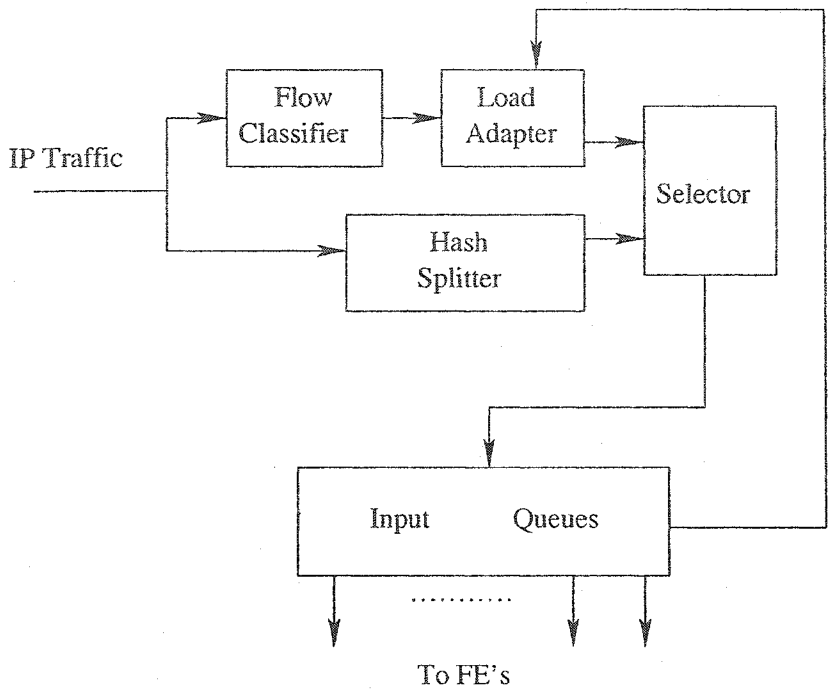


Figure 6.2: Load Balancing Packet Scheduler

minimizing adaptation disruption also minimizes the occurrence of packet reordering, which is important for maintaining end-to-end TCP performance.

6.3.2 Design

The Zipf-like flow popularity distribution and, in particular, the small number of very popular addresses, indicate that scheduling high-rate flows should be effective in balancing workloads among parallel forwarding processors. Since there are few high-rate flows, the adaptation disruption should be small. Our scheduler design takes advantage of this observation and divides Internet flows into two categories: the high-rate and the normal. By applying different forwarding policies to the two classes of flows, the scheduler achieves load balancing effectively and efficiently.

Fig. 6.2 shows the design of our packet scheduler. When the system is in a balanced state, packets flow through the hash splitter to be assigned to an FE. When the system is unbalanced, the load adapter may decide to override the decisions of the hash splitter. When making its decisions, the load adapter refers to a table of high-rate flows developed by the flow classifier.

The hash splitter uses the packet's destination address as input to a hash function. The packet is assigned to the FE whose identifier is returned by the hash function. There are several possible choices for the hash function. For example, the function could use the low order bits of the address and calculate the FE as the modulus of the number of FE's. Alternatively, HRW could be used to minimize disruption in the case of FE failures.

The load adapter becomes active when the system is unbalanced. The load adapter checks each passing packet to see whether it belongs to one of the high-rate flows identified by the classifier. If the packet belongs to one of these flows, the load adapter sets it to be forwarded to the FE that is least loaded at that instant. Any forwarding decisions made by the load adapter override those from the hash splitter; the selector gives priority to the decisions of the load adapter. In this sense, the hash splitter decides the *default* target FE for every flow.

As noted above, the load balancer functions only when the system is unbalanced. Periodically, the system is checked and if it is unbalanced, the load balancer is activated: the least loaded (possibly idle) FE is identified and the high-rate flows are shifted to it from their default FE's decided by the hash splitter. Later if, as a result of the adaptation, the system becomes balanced, the balancer is inactivated and consequently, the high-rate flows are automatically shifted back to their default FE's. After the system becomes balanced, it is not desirable to keep these high-rate flows mapped to the FE decided by the balancer because even though they can be used very effectively to balance workloads, they can quickly swamp the FE and cause load imbalance again.

An important design parameter is F, the size of the balancer's flow table. Generally, shifting more high-rate flows, i.e., having more flows in the table, is more effective as far as load balancing is concerned. Nevertheless, to reduce cost, speedup the lookup operation, and minimize adaptation disruption, the flow table should be as small as possible.

Another component in the system that is critical to the success of the load balancing scheme described above is the *flow classifier* (See Fig. 6.2). The flow classifier monitors the incoming traffic to decide which flows are high-rate flows and should be put in the balancer's flow table. We discuss in detail the high-rate flow identification procedure in Section 6.4.

6.3.3 Triggering Policies

The adapter implements the scheduling scheme that decides *when* to remap flows (the triggering policy), *which* flows to remap, and *where* to direct the packets. To effectively achieve load balancing with minimum adaptation disruption, the adapter only schedules packets in the largest flows. Packets in the smaller flows are mapped to FE's by the hash scheduler.

There are multiple choices for deciding when the adapter should be activated to redirect packets. For example, the adapter can be invoked periodically, i.e., triggered by a clock after every fixed period of time. This scheme is easy to implement, as it does not require any load information from the system. It may not be efficient, however, as far as minimizing adaptation disruption is concerned since it could shift load unnecessarily, i.e., when the system is not unbalanced.

The adapter can also monitor the lengths of the input queues, using them as indicators of the workloads of the FE's. Remapping can be triggered by events indicating that the system is unbalanced to some degree, based on the input buffer occupancy, the largest queue length, or the CV of the queue length growing above some pre-defined threshold. The system load condition could be checked at every packet arrival. This overhead can be reduced by periodic checking. We simulate several triggering policies in Section 6.5.

As another design dimension, the remapping policy decides to which processor(s) the largest flows should be migrated. One solution is to redirect all the largest flows to the shortest queue.

6.4 Detecting High-rate Flows

In this section, we describe the mechanism used in the flow classifier to identify high-rate flows.

6.4.1 Definition of High-rate Flows

We define high-rate as flows that are both large and fast; these are the source of long-term load imbalance and are most effective when shifted to balance load. These flows are similar to the alpha flows in [61]. In addition, taking the bursty nature of Internet traffic into consideration, we also classify flows that are smaller in size but are fast enough to cause short-term load imbalance or buffer-overflow as high-rate flows.

It is pointed out in [65] that flow size and lifetime are independent dimensions. There might be correlation between flow size and rate but generally, the notion of long-lived flows in most previous studies is not accurate for our purposes. As a result, short-cut establishment triggering [118] for long-lived flows cannot be used to detect high-rate flows. Instead, we need a mechanism that takes into account both the number of packets and the length of time during which the packets arrive.

6.4.2 Detecting High-rate Flows

We define *window size*, W, as the number of packets over which flow information is collected. Therefore, the incoming IP traffic is a sequence of windows: $W_1, W_2, \ldots, W_n$, $n \to \infty$, each containing W packets. Suppose we are receiving packets in W_i. We find the set F_i that contains the largest flows in W_i. The number of flows in F_i equals the size of the flow table, F, $|F_i| = F$. $F_0 = \{\}$. At the end of W_i, we replace the flows in the flow table by those in F_i. This mechanism benefits from the phenomenon of ***temporal locality*** in network traffic. Due to the *packet train* [18] behavior of network flows, it is highly possible that flows in F_i are also some of the largest ones over the next W packets. That is $F_i \cap F_{i+1} \neq \{\}$.

Let $\delta_i = |F_{i-1} \cap F_i|$. To measure the effect of W on the continuity of the content of the flow table due to temporal locality, we define

$$\Delta = \frac{\sum_{i=1}^{n} \delta_i / F}{n} \tag{6.7}$$

where

$$n = \frac{N_F}{W}$$

Thus, $0 \leq \Delta \leq 1$. The larger the value of Δ, the better flow information collected in the current window predicts high-rate flows for the next window.

Small W values are preferred when the input buffer size is small and load adjustment must be made to reflect the existence of smaller scale, short-term bursty flows. Larger W values can be used for larger buffers where the system can tolerate the load imbalance caused by bursts of small flows. Fig. 6.3 shows the effects of W on Δ for the first one million entries of the four larger traces in Table 6.1 with $F = 5$. The larger the value of W, the better the current high-rate flows predict the future. This high predictability is critical to the success of the flow classifier. Despite the window size, however, experiments show that, the largest flow of the entire trace is almost always identified as the largest flow of every window (the smallest W experimented with is 100). And we will see that shifting even only the one largest flow is very effective in balancing workloads.

To implement high-rate flow detection, another traffic model, the hyperbolic *footprint* curve [36, 37]:

$$u(W) = AW^{1/\theta}, \; A > 0, \; \theta > 1, \tag{6.8}$$

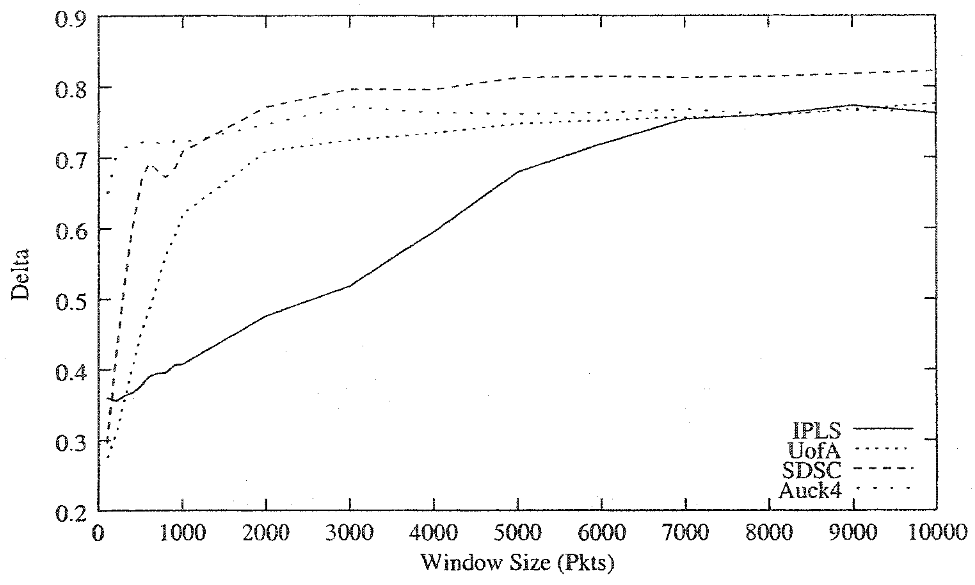


Figure 6.3: Effects of W on Δ ($F = 5$)

Table 6.3: Arrival Rates (No. of Packets/Second) of Four Traces

IPLS	UofA	SDSC	Auck4
74,608.742	14,007.337	3,210.378	251.394

could be used to relate the W to the total number of flows expected for W packets, $u(W)$.

6.5 Simulations

In this section, we conduct trace-driven simulations of an eight-FE system under static hash mapping and adaptive load balancing schemes. In the former, packets are directed to the FE's by the hash splitter only and the results serve as performance bounds for the adaptive load balancing scheme. For the latter, we simulate three adaptation triggering policies for the balancer.

6.5.1 Trace Driven Simulation

The average packet arrival rates (λ) are measured for the four larger traces (Table 6.3 [1]). IP traffic is well known for its large variability; here λ serves only as a gross estimation and is used to derive the service rates for the FE's given some system utilization ρ:

$$\mu_i = \frac{\lambda/M}{\rho}, \quad i = 1, \ldots, M. \tag{6.9}$$

[1]The FUNET trace does not have arrival timestamp information.

Given a trace (so that λ is fixed) and an overall service rate (μ), parameters that have major effects on system performance include: the input buffer size B, the number of FE's (M), the number of high-rate flows in the flow table, F, the adaptation policy, and classifier window size W. We are mainly concerned, however, about the effects of scheduling policies and the input buffer size (B) on two performance metrics: the packet loss rate (η) and the adaptation disruption (ζ). Throughout the simulations, $M = 8$, $\rho = .8$, and $W = 1000$.

6.5.2 Hash Splitter

The hash splitter implements a modulo operation to dispatch a packet, i.e.,

$$H(IPAddress) = (IPAddress)\%M$$

where % is the modulo operation and M is the number of FE's. This is equivalent to retrieving the least significant $log_2(M)$ bits of the IP address, which is deprecated in [116] as easily leading to significant bias, especially when M is a power of the radix of the computer. According to the study of hash function performance in [76], however, the low order bits in source and destination IP addresses tend to be more random than the high order bits.

This is another advantage of scheduling high-rate flows: we do not need to use complex hash functions to generate perfectly uniformly distributed random FE identifiers. The reason is that there are many low-rate flows but their contribution is insignificant compared with that of a few high-rate flows. Uniform distribution of the hash return values is not important as far as load balancing is concerned. As a result, our scheduling scheme is capable of balancing the load with low-complexity and efficient hash splitter implementations.

6.5.3 Triggering Policies

We tested three triggering policies:

- *Periodic Mapping* (PM): The adapter schedules high-rate flows periodically (after each interval of P packets), regardless of system load situation.

- *Buffer Occupancy Threshold* (BOT): The adapter is invoked if the buffer is filled above some percentage. The term *buffer* refers to the physical shared storage for all FE's.

- *Maximum Queue Length Threshold* (MQLT): The adapter is invoked if the length of the largest queue grows above some pre-defined threshold, also expressed as a percentage of the total buffer size. The term *queue* refers to the logical input queue that holds the incoming packets for an individual FE.

For comparison purposes, we also simulated hash-based load splitting without adaptation. For BOT and MQLT, periodic checking of the system workload condition is implied; for comparison purposes, we would assume this period is the same as that in PM. Thus, the results for PM set upper bounds on the frequency by which the high-rate flows are shifted from one FE to another and the amount of adaptation disruption for BOT and MQLT.

6.5.4 Adaptation Disruption

Two sources in our load balancing scheduler contribute most to adaptation disruption (AD).

First is the decision of the adapter to re-map high-rate flows to the least loaded FE. If the flows in the flow table are not currently destined to the target FE, flow-shifts occur. We call this type of flow-shift *explicit disruption* (ED). $ED \simeq N_S * F$. For the PM balancing policy, the number of flow-shifts is the length of the trace divided by the period P. For BOT and MQLT, this number should be smaller since the balancer is not always activated.

Second, after processing a window of packets, the flow classifier replaces the content of the current flow table with the largest flows calculated during the past window. This implicitly moves the flows that were not in the table from their current destination FE, determined by the hash splitter, to the FE decided by the adapter and, at the same time, shifts the replaced flows to the FE's determined by the splitter. Flow-shifting caused by the flow classifier is called *implicit disruption* (ID). When the classifier updates the content of the flow table at the end of window i, the total number of flows to be shifted is $|F_{i-1} \cup F_i| - |F_{i-1} \cap F_i|$. For the PM balancing policy,

$$ID = \sum_{i=1}^{n} |F_{i-1} \cup F_i| - |F_{i-1} \cap F_i|$$

For the other two adaptive policies, the balancer is not always on, and therefore their ID values should be smaller.

In addition, when the system is balanced and the adapter is inactive, the high-rate flows are shifted back from the balancer-decided FE to their splitter-decided default FE's, causing disruptions to the FE's involved.

6.5.5 Packet Reordering

Adaptive load balancing in hash-based distribution schemes comes at the price of packet reordering. Whenever a flow is shifted from a busy FE to a less loaded one, there is the risk of packet reordering within this flow. Therefore, the sources of adaptation disruption

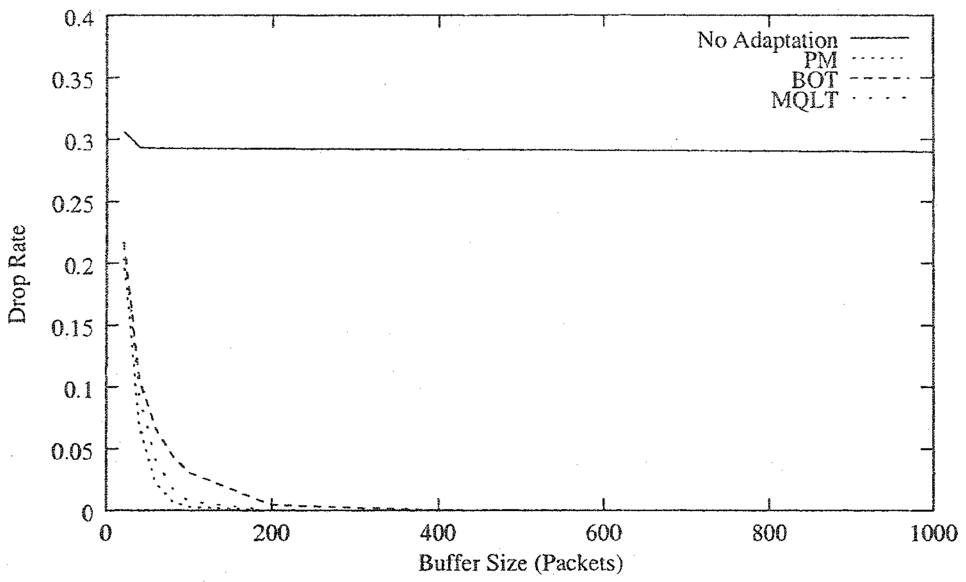


Figure 6.4: Drop Rate vs. Buffer Size (For PM, BOT, and MQLT, the system load condition checking is done every 20 packets. For BOT, the threshold is 80 percent of the buffer size. For the MQLT, the threshold is 30 percent of the buffer size. There are eight FE's and the system utilization $\rho = 0.8$. For this simulation, the number of high-rate flows in the flow table is 1.

are also the sources of potential packet reordering. Shifting a few high-rate flows minimizes adaptation disruption and for the same reason, causes less packet reordering than adaptation schemes that shift flows with no regard to their rates.

Let L_i be a flow in a trace, where $0 < i \leq |S|$ and S is the set that contains all the flows in the trace. Let $P_{i,j}$ be a packet in L_i, where $0 < j \leq N_i$ and N_i is the number of packets in L_i. Let $T_{i,j}$ be the time that the packet $P_{i,j}$ is observed. At the input port, $T_{i,j} < T_{i,j+1}$, $0 < j < N_i$. At the output port, however, due to possible packet reordering, $T_{i,j}$ might be larger than $T_{i,j+1}$. If

$$r(i,j) = \begin{cases} 1 & if \;\; T_{i,j} > T_{i,j+1} \\ 0 & otherwise \end{cases}$$

then the packet reordering rate R_r for N_F packets forwarded is

$$R_r = \frac{\sum_{i=1}^{|S|} \sum_{j=1}^{N_i} r(i,j)}{N_F}$$

6.5.6 Simulation Results

Fig. 6.4 shows packet drop rates of different adaptation policies under varying buffer sizes for the UofA trace. The hash-only scheme (no adaptation) has the highest drop rate and,

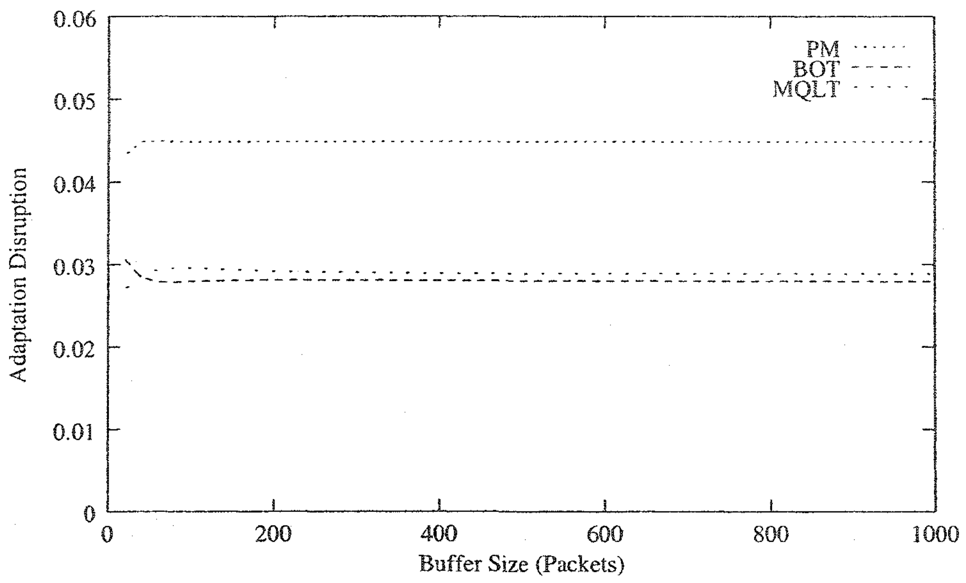


Figure 6.5: Adaptation Disruption vs. Buffer Size (The same setting with Fig. 6.4)

moreover, increasing buffer size does not help. This is because the trace is short and thus variability in arrival rate is small. On the other hand, the three adaptation schemes all respond positively to buffer increases and beyond certain buffer sizes, the drop rates reach zero. PM achieves the best drop rates compared to BOT and MQLT.

Fig. 6.5 shows that changes in buffer size have very slight effects on adaptation disruption for the three adaptation schemes, except when the sizes are small. The hash-only policy does not shift flows from one FE to another and therefore does not incur any adaptation disruption. The PM strategy has the highest adaptation disruption and this explains why it achieves the lowest drop rate: it re-maps the high-rate flow much more frequently than BOT and MQLT. The difference in adaptation disruption between MQLT and BOT is small; it seems that MQLT achieves lower drop rates (Fig. 6.4) than BOT at the cost of a little more adaption disruption.

An important parameter of the adaptation policies is the checking period. It controls the system's responsiveness to load imbalance. The smaller the interval, the more quickly the system responds to load imbalance; this leads to a lower packet drop rate. On the other hand, system load checking is one of the major parts of the adaptation overhead and could cause more adaptation disruption. Frequent load checking also consumes more CPU cycles.

Figs. 6.6 and 6.7 show how the checking interval affects drop rate and adaptation

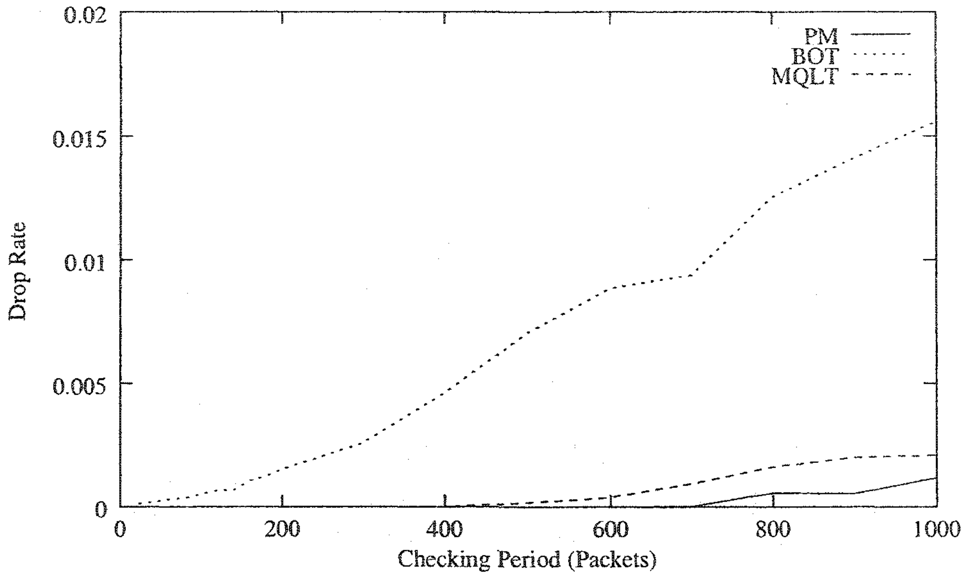


Figure 6.6: Drop Rate vs. Checking Period (The buffer size is 400 packets. The other parameters are the same as those of Fig. 6.4)

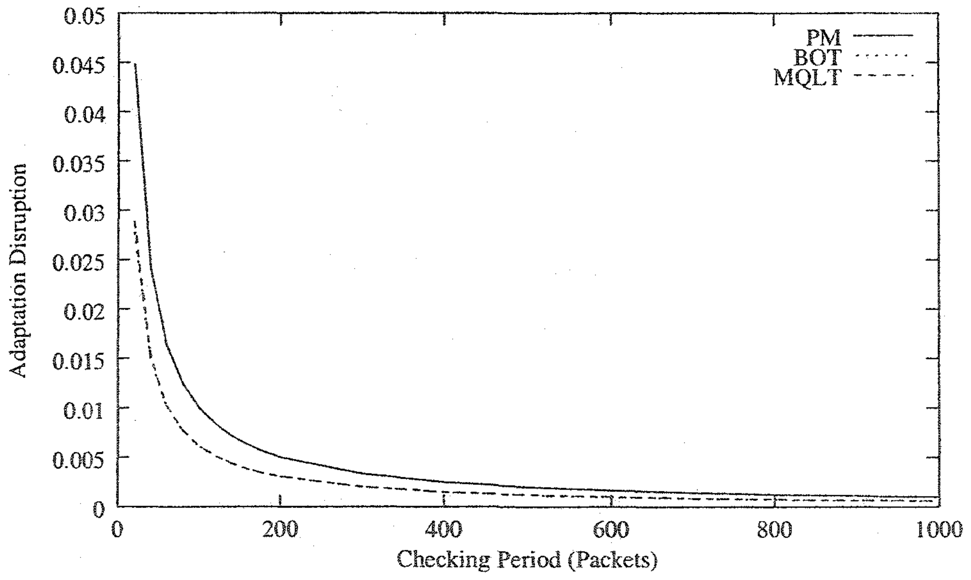


Figure 6.7: Adaptation Disruption vs. Checking Period (The same setting as Fig. 6.6)

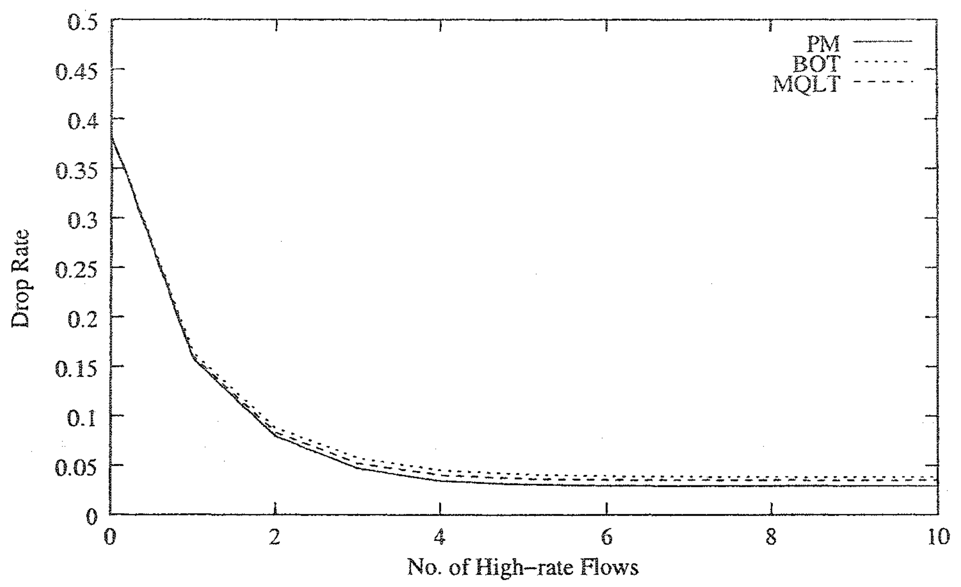


Figure 6.8: The Effectiveness of Scheduling More High-rate Flows (The checking period is 20 and the buffer size is 400. The other parameters are the same with those in Fig. 6.4

disruption. Generally, the decrease in responsiveness to load imbalance leads to packet dropping. Fig. 6.6 shows that compared with PM and MQLT, BOT (with 80 percent occupancy threshold value) is more susceptible to checking period increases. Fig. 6.7 shows that increasing the checking period is effective in reducing adaptation disruption.

Simulations with other traces show similar trends to the above results for the UofA trace. Differences in scale are caused by the peculiarities of the largest flows in the individual traces. For example, as shown in Table 6.2, the largest flow in the Auck4 trace is not significantly larger than the second, which is unlike the UofA trace where a single largest flow dominates. This implies that, for the Auck4 trace, scheduling only the one largest flow might not be able to spread load evenly over multiple processors. This can be solved partly by adding more flows into the flow table at the cost of degradation in adaptation disruption.

In the following simulations, we experiment with the Auck4 trace to study the effect of scheduling more high-rate flows on packet drop rate, adaptation disruption, and packet reordering. The results are shown in Figs. 6.8, 6.9, and 6.10. In each figure, the x axis denotes the number of high-rate flows. That is, $x = 1$ represents the case when only the largest flow in the trace is remapped to balance load; $x = 2$ means the largest two flows are scheduled, and so on.

Fig. 6.8 shows the effectiveness of scheduling more high-rate flows in reducing drop rates

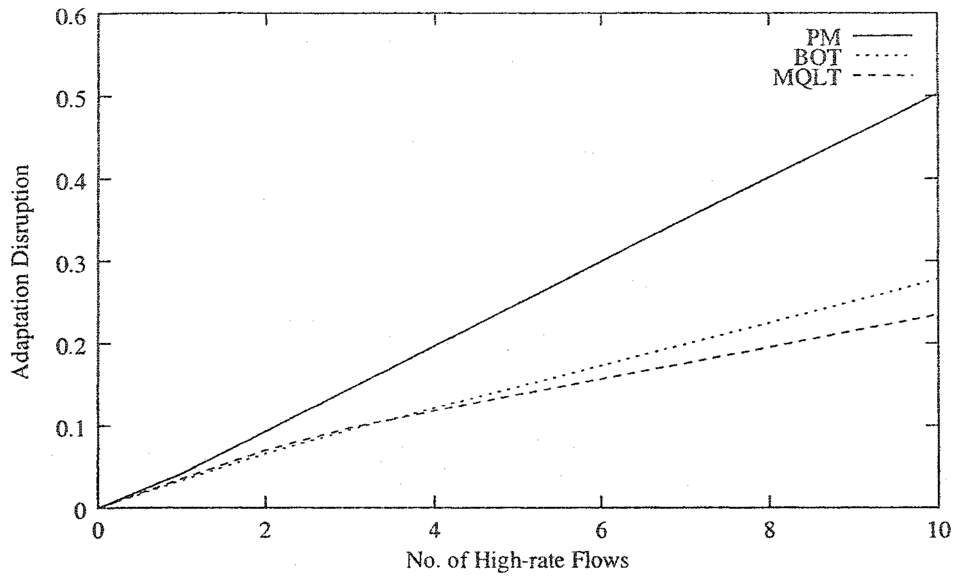


Figure 6.9: The Effects of Scheduling More Flows on Adaptation Disruption (with the same setting as Fig. 6.8)

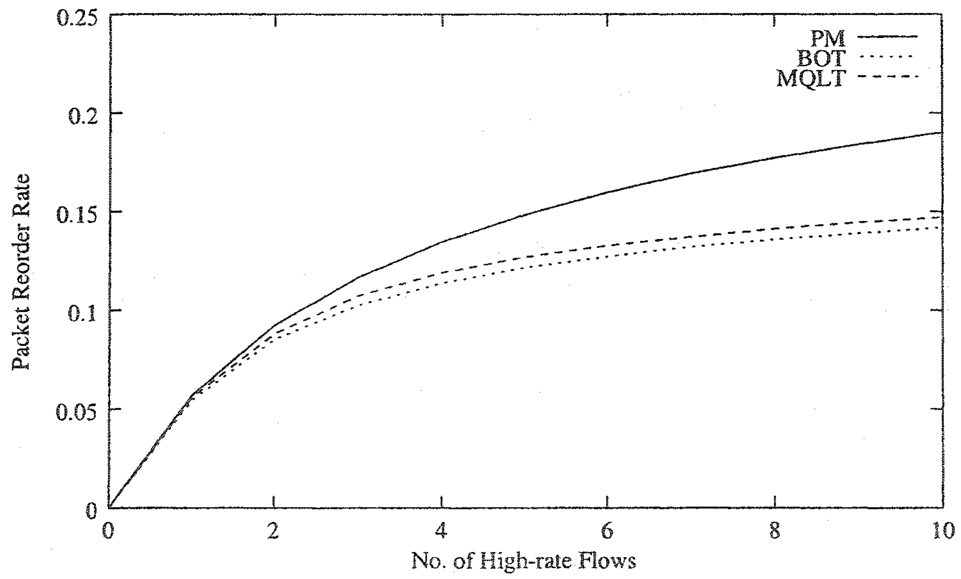


Figure 6.10: The Effects of Scheduling More Flows on Packet Reordering (with the same setting as Fig. 6.8)

Table 6.4: Comparison between Shifting Only the Largest Flow and Shifting Only Smaller Ones

Simulation	Auck4 LF	Auck4 SF	IPLS LF	IPLS SF
No. of Flows	1	37	1	439
$CV[q_i]$	.172	.265	.0782	.137
η	.133	.133	.0103	0.0121
ζ	.0413	1.84	.0357	22.2
Rr	.0612	.146	.00965	0.0626

Simulation	SDSC LF	SDSC SF	UofA LF	UofA SF
No. of Flows	1	2	1	500
$CV[q_i]$	.181	.164	.143	.288
η	.0904	.0749	0	0.103
ζ	.0342	.0714	.0357	25.2
Rr	.00546	.00740	.00965	0.0511

for the Auck4 trace for the three adaptive policies. It seems that for a given configuration, beyond a certain number of high-rate flows, the benefit of scheduling more flows becomes negligible. On the other hand, as shown in Fig. 6.9, adaptation disruption increases linearly with the number of flows scheduled. Therefore, it is both important and desirable to limit the number of flows in the flow table.

Fig. 6.10 shows simulation results of packet reordering rates for the Auck4 trace. Like adaptation disruption, packet reordering is affected mainly by the number of flows shifted. It is apparent from the figure that shifting a larger flow causes more packet reordering than migrating a smaller one. This is different from adaptation disruption where each flow-shift contributes the same to the overall disruption regardless of the nature of the individual flow.

To further illustrate the advantage of shifting the largest flows, we compare the results of two simulations: scheduling only the largest flow (LF) and scheduling only smaller flows (SF) to achieve similar drop rates as with LF. We simulate the PM policy with a 20-packet checking period. Table 6.4 summarizes the results for four traces. With similar packet drop rates (η), scheduling the largest flow always causes less adaptation disruption (ζ) and packet reorders (R_r). For the Auck4, IPLS, UofA traces, scheduling the largest flow also achieves a smaller $CV[q_i]$. Unanimously, more than one smaller flows are needed to achieve similar packet drop rates as scheduling the largest flow. The least number of smaller flows needed is two, as in the SDSC case where scheduling smaller flows achieves a lower miss ratio and $CV[q_i]$. One reason might be that in the SDSC trace, the largest flows identified by the mechanism in Section 6.4 only accounts for a small portion of the total traffic, not significant enough for the scheduling-the-largest-flow strategy to outperform scheduling-the-smaller-flows by a large margin. The other extreme is the UofA trace, where the largest flow by itself represents around 16 per cent of the aggregate traffic; when it is scheduled onto an FE, even if the rest of the traffic are spread evenly among the other seven FE's (each 12 per cent), the system is still not perfectly balanced.

It is important to note that the arrival rate λ for the Auck4 trace (see Table 6.3) used to decide the FE service rates (Eq. 6.9) in the simulations of Figs. 6.8, 6.9, and 6.10 is the *average* rate over five hours. Arrival rates during shorter intervals may be much higher. For example, the arrival rate for the first one million packets in the Auck4 trace is 1.3 times the average rate. The service rate of the system, however, is only 1.25 times the average arrival rate. In such situations, packet losses occur regardless of the scheduling scheme. Therefore, under similar adaptation configurations, differences in arrival rate variability account for different drop rates, adaptation disruption, and packet reordering rates, for different traces.

6.6 Summary

The highly skewed Internet flow popularity distribution has profound implications for Internet forwarding system design. First, we have shown in this chapter that the Zipf-like flow popularity distribution, which has infinite mean and variance, is one of the major sources of load imbalance in a hash-based packet dispatching scheme. Second, to measure the efficiency of adaptive scheduling schemes, we introduce a new metric, the adaptation disruption, which reflects the effect of adaptive algorithms on cache performance and is an important touchstone for evaluating overall parallel forwarding system performance. Third, flow-level Internet traffic characterization inspires the classification of flows into two categories: the high-rate and the normal. By applying different scheduling policies to the two classes, we have been able to build a highly effective and efficient scheduler that can be used in parallel Internet forwarding devices.

Instead of migrating flows, regardless of their nature, from heavily load FE's to less loaded ones, our scheduler shifts only a few high-rate flows when the system is unbalanced. Manipulating these flows is effective because they are the major source of load imbalance. At the same time, since their number is small, migrating these flows causes minimum adaptation disruption to the FE's cache. We expect much higher disruption in adaptive load balancing schemes that do not take flow popularity distribution into account. Experiments show that due to temporal locality in Internet traffic, the high-rate flows can be readily identified, which indicates that the proposed load balancer is highly feasible.

The data in Table 6.4 indicates that, as α increases, a larger number of smaller flows must be scheduled in order to achieve the same packet drop rate as scheduling the largest flow. Intuitively, as α increases, the largest flow contributes more to the aggregate traffic and thus it is more effective in balancing workloads. Analytical work is underway to determine the bounds of the number of the largest flows to schedule under the constraints of α and performance requirements of load balancing, adaptation disruption, and packet reordering.

Chapter 7

Conclusions and Future Work

The phenomenal development of the Internet poses challenges to forwarding systems. To keep up with the growing bandwidth demand, Internet routers have to process packets at line speed; to accommodate new applications and standards, these devices have also to be flexible to allow easy updates; finally, new applications, e.g., secure transactions, require computation power. Over the past years a wide variety of solutions have been proposed. One trend in forwarding system design is to employ network processors and to implement key forwarding algorithms in software.

Due to a huge design space and a diverse market, a large variety of these systems have been developed. Workload characterization is critical to performance evaluation of systems from different vendors, and essential to understanding the trade-offs in system design. This thesis addresses this need by modeling two salient features critical to performance in the workload for Internet forwarding systems: the temporal locality in IP destination address sequences and the skewed address popularity distribution. The results from our work can be applied in network forwarding system testing and benchmarking. Moreover, our work on Internet traffic modeling leads to the design of an efficient and effective load scheduling scheme for parallel forwarding systems.

7.1 Conclusions

First, after examining several existing schemes in similar areas, we use the LRUSM to capture temporal locality in destination IP address sequences. Based on the analysis of real world traces gathered from networks ranging from the campus level to major sections of the Internet backbone, we propose a five-parameter mixed reuse distance distribution function (which can be further reduced to four parameters). This distribution accurately models the temporal locality in Internet traffic. Furthermore, we propose a synthetic trace generation algorithm based on this model. Simulation results of the cache miss ratio of a routing table lookup algorithm validate our model.

Second, we study the flow-level characteristics of Internet traffic streams, which are important in the design of parallel forwarding systems. We observed that a few high-rate flows usually dominate Internet traffic and we found that the popularity distribution of IP addresses generally follows Zipf's law. We demonstrated that with a simple hash-based packet scheduler, the load imbalance in a parallel forwarding system is caused by a few high-rate flows. This conclusion indicates that skewed popularity distributions and, in particular, the existence of the few high-rate flows, are important to model.

The LRUSM, although it captures temporal locality, does not differentiate among flows. In a synthetic trace generated using LRUSM alone, the flows tend to be similar in the rate and arrival patterns. We propose an algorithm based on the LRUSM, to generate traffic with the desired temporal locality, which also accommodates high-rate flows. Synthetic traces generated by this algorithm are shown to induce similar load imbalance behavior of parallel forwarding systems.

Third, we show the impacts of packet dispatching schemes on cache performance, and the effects of caching on load balancing in parallel forwarding systems. We have found significant differences in temporal locality in the traffic scheduled by two schemes: hashing and round-robin.

Based on the work on Internet traffic characterization, we developed an efficient and effective load balancing scheme for parallel forwarding systems. Compared with state-of-the-art designs in this area, ours is unique in taking advantage of flow-level traffic characteristics. We introduced an important metric for load balancing design, i.e., adaptation disruption, which measures the disruption to cache states in the individual FE's caused by load shifting schemes. Our load adaptation scheme is effective at balancing workloads and achieving minimum adaptation disruption.

7.2 Areas of Future Research

The development of this thesis is an example of how a deeper understanding of computing system workload characteristics can lead to sound system designs. Workload characterization, therefore, will be one of our directions for future work. At the same time, the methodology and many results in this thesis are applicable to other networks and more general computing systems. In this section, we discuss several future research directions.

7.2.1 Integration of the IRM and the LRUSM

We described in Chapter 4 an algorithm that generates high-rate Internet flows within the LRUSM framework. The synthetic traffic exhibits the desired temporal locality and can be used to evaluate the performance of hash-based load balancing schemes in parallel forwarding systems. This is because the effectiveness of these schemes is strongly influenced by a few high-rate flows in the traffic.

Incorporating a small number of flows, however, is only a partial solution to a more general problem, i.e., how to integrate the IRM and LRUSM. Given a reference string, in the IRM we calculate the frequency distribution of the objects; in the LRUSM we model the reuse distance distribution, which quantifies temporal locality. It is desirable to develop an algorithm that generates a sequence of objects that exhibits both features according to the model parameters. Such an algorithm, besides capturing temporal locality, would be able to produce a synthetic object reference sequence encompassing the whole spectrum of object frequency distribution. As an example, the algorithm would allow us to generate both "elephants" and "mice" in synthetic IP destination address sequences.

Few previous studies have focused on this topic. In [54] the authors develop a synthetic IP address generation algorithm based on a hybrid model. The model does not explicitly take into account individual address frequencies, yet is able to generate Zipf-like distributions. Our preliminary implementation does not yield the same results. Further verification of this model is part of our future work on incorporating the IRM and LRUSM.

7.2.2 High-performance Designs Based on Traffic Characteristics

Highly skewed popularity distributions exist in the workloads for many network systems. Dividing these workloads into two or more categories and treating each group differently is a general idea that could be effective in improving system performance. For example, WWW server cluster systems could benefit from hash-based load distribution schemes, e.g., HRW, to improve cache hit ratio and to reduce response time. It is pointed out in [70], however, that requests for one hot object alone could present enough load to swamp a server. Such systems could implement object *replication* for the most popular objects so that these objects have copies on more than one server and object space *partition* by hashing for the other not-so-popular objects so that each server only hosts a partition of these objects. A load distribution scheme similar to the one outlined in this chapter could then be used to balance the load. For such systems, a centralized scheduling mechanism is essential.

Zipf-like popularity distributions have been found in workloads of many Internet systems and novel designs have been developed to capitalize on this characteristic. Recently, Chvets and MacGregor [119] and MacGregor [120] proposed a novel IP route caching scheme that divides a cache into zones, where each cache routing table lookup results of certain prefix lengths. According to the frequencies of prefix lengths, the sizes of the zones are assigned. Results show that the best configuration can reduce the miss ratio of an LRU cache that contains "IP Address, Output Port" pairs by half.

We observe that generally, LRU performance is degraded by the presence of the large number of Internet "mice". For example, IP addresses that appear only once in a trace evict addresses that are to be referenced in the future. In a preliminary experiment, we divide an LRU IP address cache into two sections, both using LRU. Only addresses that are referenced more than once in the first section migrates to the second. This simple scheme eliminates the "mice" effect described above and achieves better performance than an LRU cache of the combined size of both sections. It is worth noting that this method is orthogonal to the scheme proposed in [119, 120]. More experiments are underway.

Bibliography

[1] B. Halabi. *Internet Routing Architecture.* Cisco Press, 1997.

[2] J. Postel. RFC 791: Internet protocol, September 1981.

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

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

[5] H. C. B. Chan, H. M. Alnuweiri, V. C. M. Leung. A framework for optimizing the cost and performance of next-generation IP routers. *IEEE Journal on Selected Areas in Communications*, 17(6):1013–1029, 1999.

[6] G. Huston. BGP table data report. Statistics of Internet Routing Tables collected at Telstra (available at http://bgp.potaroo.net/).

[7] Y. Rekhter, T. Li. RFC 1518: An architecture for IP address allocation with CIDR, September 1993.

[8] V. Fuller, T. Li, J. Yu, K. Varadhan. RFC 1519: Classless inter-domain routing (CIDR): an address assignment and aggregation strategy, September 1993.

[9] K. Sklower. A tree-based packet routing table for Berkeley Unix. In *USENIX Winter 1991*, pages 93–104, Dallas, TX, USA, January 1991.

[10] M. Waldvogel, G. Varghese, J. Turner, B. Plattner. Scalable high speed IP routing lookups. In *ACM SIGCOMM 1997*, pages 25–38, Cannes, France, September 1997.

[11] M. Degermark, A. Brodnik, S. Carlsson, S. Pink. Small forwarding tables for fast routing lookups. In *ACM SIGCOMM 1997*, pages 3–14, Cannes, France, September 1997.

[12] T. Chiueh, P. Pradhan. High performance IP routing table lookup using CPU caching. In *IEEE INFOCOM 1999*, pages 1421–1428, New York, NY, USA, March 1999.

[13] S. Nilsson, G. Karlsson. IP-address lookup using LC-tries. *IEEE Journal on Selected Areas in Communications*, 17(6):1083–1092, June 1997.

[14] P. Gupta, S. Lin, N. McKeown. Routing lookups in hardware at memory access speeds. In *IEEE INFOCOM 1998*, pages 1240–1247, San Francisco, CA, USA, March 1998.

[15] W. Shi, M. H. MacGregor. Cache reference behavior of three IP routing table lookup algorithms. In *The 5th Multi-Conference on Systemics, Cybernetics and Informatics (SCI-2001)*, pages 318–320, Orlando, FL, USA, July 2001.

[16] D. Feldmeier. Improving gateway performance with a routing table cache. In *IEEE INFOCOM 1988*, pages 298–307, New Orleans, LA, USA, April 1988.

[17] P. J. Denning, S. Schwartz. Properties of the working set model. *Communications of the ACM*, 15(3):191–198, 1972.

[18] R. Jain, S. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal of Selected Areas in Communications*, SAC-4(6):986–995, September 1986.

[19] K. Claffy. *Internet Workload Characterization.* PhD thesis, University of California, San Diego, June 1994.

[20] R. B. Bunt, J. M. Murphy, S. Majumdar. A measure of program locality and its application. *Performance Evaluation Review*, 12(3):28–40, 1984.

[21] N. Gulati, C. Williamson, R. Bunt. Local area network traffic locality: Characteristics and application. In *The First International Conference on LAN Interconnection*, pages 233–250, Research Triangle Park, NC, USA, October 1993.

[22] N. Gulati. Local area network traffic locality: Characteristics and application. Master's thesis, University of Saskatchewan, July 1992.

[23] S. Floyd, V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, Febrary 2001.

[24] G. K. Zipf. *Human Behavior and the Principle of Least-Effort.* Addison-Wesley, Cambridge, MA, 1949.

[25] R. Glass, editor. *In the Beginning: Recollections of Software Pioneers*, chapter 6. IEEE Press, 1997.

[26] P. J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[27] P. J. Denning. Working sets past and present. *IEEE Transactions on Software Engineering*, SE-6(1):64–84, January 1980.

[28] J. Spirn. Distance string models for program behavior. *IEEE Computer*, 9(11):14–20, November 1976.

[29] A. V. Aho, P. J. Denning, J. D. Ullman. Principles of optimal page replacement. *Journal of the ACM*, 18(1):80–93, 1971.

[30] R. B. Bunt, J. M. Murphy. The measurement of locality and the behavior of programs. *The Computer Journal*, 27(3):238–245, 1984.

[31] S. Glassman. A caching relay for the World Wide Web. *Computer Networks and ISDN Systems*, 27(2):165–173, 1994.

[32] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *IEEE INFOCOM 1999*, pages 126–134, New York, NY, USA, March 1999.

[33] A. Mahanti, D. Eager, C. Williamson. Temporal locality and its impact on Web proxy cache performance. *Performance Evaluation Journal: Special Issue on Internet Performance Modeling*, 42(2-3):187–203, 2000.

[34] R. Mattson, J. Gecsei, D. Slutz, I. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[35] J. Spirn. *Program Behavior: Models and Measurements.* Elsevier-North Holland, N.Y., 1977.

[36] D. Thiebaut, J. L. Wolf, H. S. Stone. Synthetic traces for trace-driven simulation of cache memories. *IEEE Transactions on Computers*, 41(4):388–410, 1992.

[37] W. Shi, M. H. MacGregor, P. Gburzynski. Synthetic trace generation for the Internet. In *The 4th IEEE Workshop on Workload Characterization (WWC-4)*, pages 169–174, Austin, TX, USA, December 2001.

[38] J. L. Hennessy, D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 3rd edition, 2003.

[39] R. Jain. Characteristics of destination address locality in computer networks: a comparison of caching schemes. *Computer Networks and ISDN Systems*, 18(4):243–254, May 1990.

[40] J. Mogul. Network locality at the scale of processes. *ACM Transactions on Computer Systems (TOCS)*, 10(2):81–109, May 1992.

[41] C. Partridge, S. Pink. A faster UDP. *IEEE/ACM Transactions on Networking*, 1(4):429–440, 1993.

[42] P. McKenney, K. Dove. Efficient demultiplexing of incoming TCP packets. In *ACM SIGCOMM 1992*, pages 269–279, Baltimore, MD, USA, August 1992.

[43] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM 1988*, pages 314–329, Stanford, CA, USA, August 1988.

[44] J. Gray. *The Benchmark Handbook for Database and Transaction Processing Systems.* Morgan Kaufmann, 1991.

[45] V. Almeida, A. Bestavros, M. Crovella, A. Oliveira. Characterizing reference locality in the WWW. In *The IEEE Conference on Parallel and Distributed Information Systems (PDIS 1996)*, pages 92–107, Miami Beach, FL, USA, December 1996.

[46] P. Barford, M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *ACM SIGMETRICS 1998*, pages 151–160, Madison, WI, USA, July 1998.

[47] M. F. Arlitt, C. L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, Oct. 1997.

[48] S. Jin, A. Bestavros. Sources and characteristics of Web temporal locality. In *MASCOTS 2000*, pages 28–35, San Fransisco, CA, USA, August 2000.

[49] S. Jin, A. Bestavros. GreedyDual* Web caching algorithm: exploiting the two sources of temporal locality in Web request streams. *Computer Communications*, 24(2):174–183, 2001.

[50] R. Fonseca, V. Almeida, M. Crovella, B. Abrahao. On the intrinsic locality properties of Web reference streams. In *IEEE INFOCOM 2003*, pages 448–458, San Francisco, CA, USA, April 2003.

[51] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[52] M. Aida, T. Abe. Pseudo-address generation algorithm of packet destinations for Internet performance simulation. In *IEEE INFOCOM 2001*, pages 1425–1433, Anchorage, AK, USA, April 2001.

[53] M. Aida, T. Abe. Stochastic model of Internet access patterns. *IEICE Transactions on Communications*, E84-B(8):2142–2150, 2001.

[54] M. Aida, T. Abe. Stochastic model of Internet access patterns: coexistence of stationarity and Zipf-type distributions. *IEICE Transactions on Communications*, E85-B(8):1469–1478, 2002.

[55] K. C. Claffy, H. W. Braun, G. C. Polyzos. A parameterizable methodology for Internet traffic flow profiling. *IEEE Journal of Selected Areas in Communications*, 13(8):1481–1494, 1995.

[56] R. Caceres, P. Danzig, S. Jamin, D. Mitzel. Characteristics of wide-area TCP/IP conversations. In *ACM SIGCOMM 1991*, pages 101–112, Zürich, Switzerland, September 1991.

[57] M. Acharya, B. Bhalla, R. E. Newman-Wolfe, H. Latchman, R. Chow. A flow model for computer network traffic using real-time measurements. In *Second International Conference on Telecommunications Systems, Modeling and Analysis*, pages 141–149, Nashville, TN, USA, March 1994.

[58] W. E. Leland, M. S. Taqq, W. Willinger, D. V. Wilson. On the self-similar nature of Ethernet traffic. In *ACM SIGCOMM 1993*, pages 183–193, San Francisco, CA, USA, September 1993.

[59] J. Beran. *Statistics for Long-Memory Processes.* Chapman & Hall, 1994.

[60] W. Willinger A. Erramilli, O. Narayan. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, 1996.

[61] S. Sarvotham, R. Riedi, R. Baraniuk. Connection-level analysis and modeling of network traffic. In *ACM SIGCOMM Internet Measurement Workshop*, pages 99–103, San Francisco, CA, USA, November 2001.

[62] W. Willinger, M. Taqqu, R. Sherman, D. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, Feb 1997.

[63] M. Crovella, A. Bestavros. Self-similarity in World Wide Web traffic. *IEEE/ACM Transactions on Networking*, 5(6):835–846, Dec. 1997.

[64] S. Sarvotham, R. Riedi, R. Baraniuk. Network traffic analysis and modeling at the connection level. Technical report, Electrical and Computer Engineering Department, Rice University, June 2001.

[65] N. Brownlee, K. Claffy. Understanding Internet traffic streams: Dragonflies and tortoises. *IEEE Communications*, 40(10):110–117, Oct. 2002.

[66] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, P. Sturm. Enhancing the Web's infrastructure: From caching to replication. *IEEE Internet Computing*, 1(2):18–27, Mar. 1997.

[67] M. Colajanni, P. S. Yu, D. M. Dias. Analysis of task assignment policies in scalable distributed Web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):585–600, 1998.

[68] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, K. J. Worrell. a hierarchical Internet object cache. In *USENIX Annual Technical Conference*, pages 153–164, San Diego, CA, USA, January 1996.

[69] D. Wessels and K. Claffy. RFC 2186: Internet cache protocol (ICP), version 2, September 1997.

[70] D. G. Thaler, C. V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking*, 6(1):1–14, February 1998.

[71] K. W. Ross. Hash routing for collections of shared Web caches. *IEEE Network*, 11(7):37–44, Nov-Dec 1997.

[72] N. Phadnis, V. Valloppillil, K. W. Ross J. Cohen. Cache array routing protocol v1.1. Internet Draft, http://ds1.internic.net/internet-drafts/draft-vinod-carp-v1-01.txt, September 1997.

[73] E. Blanton, M. Allman. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, 32(1):20–30, Jan. 2002.

[74] J. Bennett, C. Partridge, N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, Dec. 1999.

[75] W. Shi, M. H. MacGregor, P. Gburzynski. Effects of a hash-based scheduler on cache performance in a parallel forwarding system. In *Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2003)*, pages 130–138, Orlando, FL, USA, January 2003.

[76] Z. Cao, Z. Wang, E. Zegura. Performance of hashing-based schemes for Internet load balancing. In *IEEE INFOCOM 2000*, pages 332–341, Tel-Aviv, Israel, March 2000.

[77] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, Inc., 1991.

[78] J. Jo, Y. Kim, H. Chao, F. Merat. Internet traffic load balancing using dynamic hashing with flow volumes. In *Internet Performance and Control of Network Systems III at SPIE ITCOM 2002*, pages 154–165, Boston, MA, USA, July 2002.

[79] G. Dittmann, A. Herkersdorf. Network processor load balancing for high-speed links. In *2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002)*, pages 727–735, San Diego, CA, USA, July 2002.

[80] L. Kencl, J. Le Boudec. Adaptive load sharing for network processors. In *IEEE INFOCOM 2002*, pages 545–554, New York, NY, USA, June 2002.

[81] L. Kencl. *Load Sharing for Multiprocessor Network Nodes.* PhD thesis, Swiss Federal Institute of Technology (EPFL), January 2003.

[82] P. Gupta, B. Prabhakar, S. P. Boyd. Near optimal routing lookups with bounded worst case performance. In *IEEE INFOCOM 2000*, pages 1184–1192, Tel-Aviv, Israel, March 2000.

[83] X. Chen. Effect of caching on routing-table lookup in multimedia environment. In *IEEE INFOCOM 1991*, pages 1228–1236, Bal Harbour, FL, USA, April 1991.

[84] G. Cheung, S. McCanne. Optimal routing table design for IP address lookups under memory constraints. In *IEEE INFOCOM 1999*, pages 1437–1444, New York, NY, USA, March 1999.

[85] N. Shah. Understanding network processors. Master's thesis, U. C. Berkeley, September 2001.

[86] Agilent Technologies. Router tester. http://advanced.comms.agilent.com/RouterTester/.

[87] J. Kuan. Private communication, 2001.

[88] R. H. Saavedra-Barrera. *CPU Performance Evaluation and Execution Time Prediction Using narrow Spectrum Benchmarking.* PhD thesis, University of California, Berkeley, February 1992.

[89] M. Tsai, C. Kulkarni, C. Sauer, N. Shah, K. Keutzer. A benchmarking methodology for network processors. In *Workshop on Network Processors*, pages 75–85, Cambridge MA, USA, February 2002.

[90] M. Franklin, P. Crowley, H. Hadimioglu, P. Onufryk, editor. *Network Processor Design*, chapter 2. Morgan Kaufmann, 2002.

[91] FreeBSD. The FreeBSD project. http://www.freebsd.org.

[92] T. M. Austin, E. Larson, D. Ernst. SimpleScalar: an infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[93] NLANR (National Laboratory for Applied Network Research) Measurement and Operations Analysis Team (MOAT). Packet header traces. http://moat.nlanr.net.

[94] D. Thiebaut. On the fractal dimension of computer programs and its application to the computation of the cache miss-ratio. *IEEE Transactions on Computers*, 38(7):1012–1026, 1989.

[95] N. L. Johnson, S. Kotz. *Continuous Univariate Distributions.* Hougton Mifflin, 1970.

[96] A. Feldmann, W. Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. In *IEEE INFOCOM 1997*, pages 1096–1104, Kobe, Japan, April 1997.

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

[98] A. B. Downey. Evidence for long-tailed distributions in the Internet. In *ACM SIGCOMM Internet Measurement Workshop*, pages 229–241, San Francisco, CA, USA, November 2001.

[99] B. T. Bennett, V. J. Kruskal. LRU stack processing. *IBM Journal of Research and Development*, 19(4):353–357, 1975.

[100] D. D. Sleator, R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[101] A. Mahanti. Web proxy workload characterization and modeling. Master's thesis, University of Saskatchewan, September 1999.

[102] F. Ergun, S. Mittra, S. C. Sahinalp, J. Sharp, R. K. Sinha. A dynamic lookup scheme for bursty access patterns. In *IEEE INFOCOM 2001*, pages 1444–1453, Anchorage, AL, USA, April 2001.

[103] M. Arlitt, C. Williamson. Trace-driven simulation of document caching strategies for Internet Web servers. *Simulation Journal*, 68(1):23–33, 1997.

[104] A. B. Downey. The structural cause of file size distributions. In *MASCOTS 2001*, pages 361–370, Cincinnati, OH, USA, August 2001.

[105] C. C. Aggarwal, J. L. Wolf, P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *International Conference on Multimedia Computing and Systems (ICMCS)*, pages 253–258, Hiroshima, Japan, June 1996.

[106] Shudong Jin and Azer Bestavros. GISMO: A generator of Internet streaming media objects and workloads. *ACM SIGMETRICS Performance Evaluation Review*, 29(3):2–10, November 2001.

[107] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes in C: The Art Of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[108] N. Markatchev, C. Williamson. WebTraff: a GUI for Web proxy cache workload modeling and analysis. In *MASCOTS 2002*, pages 356–363, Fort Worth, TX, USA, October 2002.

[109] M. Busari, C. Williamson. ProWGen: a synthetic workload generation tool for simulation evaluation of Web proxy caches. *Computer Networks*, 38(6):779–794, 2002.

[110] M. Busari. Simulation evaluation of Web caching hierarchies. Master's thesis, University of Saskatchewan, June 2000.

[111] P. P. Ware, T. W. Page Jr., B. L. Nelson. Modeling file-system input traces via a two-level arrival process. In *Winter Simulation Conference*, pages 1230–1237, Coronado, CA, USA, December 1996.

[112] J. G. Fletcher. An arithmetic checksum for serial transmissions. *IEEE Transactions on Communications*, COM-30(1):247–252, January 1982.

[113] C. Partridge, et al. A fifty gigabit per second IP router. *IEEE/ACM Transactions on Networking*, 6(3):237–248, 1998.

[114] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb. Building a robust software-based router using network processors. In *The 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, pages 216–229, Banff, AB, Canada, December 2001.

[115] R. Jain. A comparison of hashing schemes for address lookup in computer networks. *IEEE Transactions on Communications*, 40(3):1570–1573, October 1992.

[116] D. E. Knuth. *The Art Of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, 1st edition, 1969.

[117] W. Shi, M. H. MacGregor, P. Gburzynski. Synthetic trace generation for the Internet: An integrated model, 2003. Submitted.

[118] A. Feldmann, J. Rexford, R. Cáceres. Efficient policies for carrying Web traffic over flow-switched networks. *IEEE/ACM Transactions on Networking*, 6(6):673–685, 1998.

[119] I. Chvets, M. MacGregor. Multi-zone caches for accelerating IP routing table lookups. In *High Performance Switching and Routing (HPSR 2002)*, pages 121–126, Kobe, Japan, May 2002.

[120] M. H. MacGregor. Design algorithm for multi-zone IP address caches. In *High Performance Switching and Routing (HPSR 2003)*, Torino, Italy, June 2003.

[121] G. E. P. Box, G. M. Jenkins. *Time Series Analysis: forecasting and control*. Holden-Day Inc., 1970.

Appendix A

More CCDF Fitting Experiments

In this appendix, we first show the results of fitting the W+P model developed in Section 3.5 to more data sets, to further validate Eq. 3.3. We show further that to achieve parsimonious modeling, the Weibull component in Eq. 3.3 can be replaced by an exponential distribution.

A.1 Fitting More Data Sets with the W+P Model

The traces used in this section are from the Abilene-I and Auckland-IV sets, both from NLANR. They are divided into groups by their directions. The Abilene-I set contains traces measured at two router ports. So there are four groups: CLEV-0, CLEV-1, KSCY-0, KSCY-1, each containing 12 traces. The Auckland-IV set contains 94 traces from two directions at one port. We select a subset of traces for each direction: AuckIV-0 (19 traces) and AuckIV-1 (18 traces). The reuse distance CCDF's of these traces are shown in Figs. A.1 and A.2. The CCDF's for the KSCY traces are similar to those for the CLEV traces and are not shown.

During our experiments, we have found that temporal locality characteristics of traces of packets traveling in the same direction, i.e., arriving at or departing from a measurement port, are very similar. For example, using the fitting results of one trace as initial values, we have been able to automate the fitting procedure to fit all the other CCDF's of traces in the same group. This is especially true for the traces in the Abilene-I set where one set of parameters was used as initial values to obtain the fitting of all the CCDF's, even those of traces from different groups. This is also true for the AuckIV-1 group and most of the traces (17 out of 19) in the Auck-0 group. The similarity of the CCDF curves in the figures indicates that the backbone traces are relatively consistent in terms of temporal locality whereas the traces from lower-bandwidth links have more variation.

As shown in Figs. A.3 and A.4, our model is successful in describing the temporal locality characteristics of a wide range of traces gathered at different levels in the Internet, from campus networks to Internet backbones.

We have also found that the parameters in the model are similar within groups of traces. We are especially interested in the parameter "p" in Eq. 3.3, which represents the percentage that the Weibull contributes to the mixed-CCDF model. The values of p are in the ranges of $[0.62, 0.90]$ and $[0.37, 0.65]$ for the Auck-0 and Auck-1 traces, respectively. They are in the range of $[0.14, 0.16]$ and $[0.16, 0.19]$ for the CLEV-0 and CLEV-1 traces. It seems that p tends to be larger for a campus level network but smaller for backbone networks. The p values for the UofA and LDestIP traces collected at campus-level networks and those for the KSCY-0 and KSCY-1 traces gathered at backbone networks support this hypothesis. Figs. A.5 and A.6 show the effects of p by decomposing the fit CCDF's for some traces into the two components of the model, $pW(x)$ and $(1-p)P(x)$.

The other parameters that differ significantly across trace sets are the scale parameters, i.e., d for the Weibull and b for the Pareto CCDF. b for the AuckIV set is in the range of $[1.21, 253]$ and for the IPLS set is within $[0.059, 0.154]$. d is within $[6.23, 47.0]$ for the AuckIV set but $[1067, 1325]$ for the IPLS set. Both scale parameters differ by more than one order of magnitude, respectively. On the other hand, the shape parameters, i.e., c for the Weibull and a for the Pareto, are relatively constant for different trace sets.

Thus the set of values of p, b, d, or even only one of them, can be used to indicate a trace's origin. Moreover, we can generate synthetic traces using different parameter values for the evaluation of network devices at different levels.

A.2 On Parsimonious Modeling

Parsimony is desirable in modeling [121]; the goal is to use as few parameters as possible in a model without losing accuracy. In our reuse distance distribution model (Eq. 3.3), there are a total of five parameters, i.e., the parameter p, and the scale and shape parameters of the Weibull and Pareto functions.

A.2.1 Replacing the Weibull Distribution

Our intuition about reducing the number of parameters comes from the observation that Eq. 3.3 consists of a long-tailed component, the Pareto, and a short-tailed component, the Weibull. The tails of the CCDF's are largely contributed by the Pareto distribution; any short-tailed distribution might be able to replace the Weibull without fundamentally changing the shape of the tails.

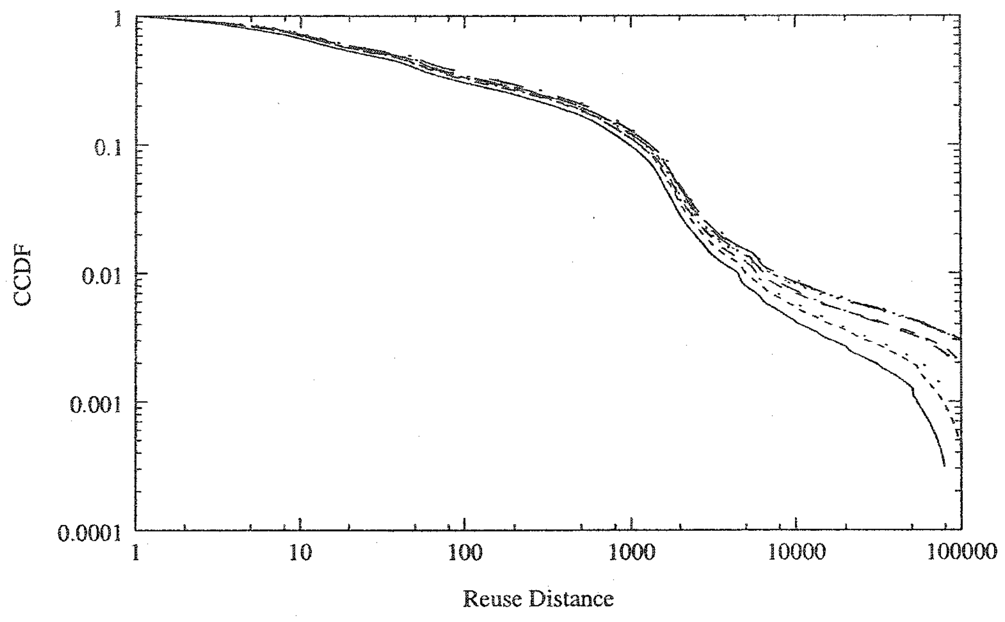
Feldmann and Whitt [96] show that long-tailed distributions can be approximated by hyper-exponentials. In light of this work, both distributions in Eq. 3.3 can be expanded as a mixture of exponentials. This leads us to experiment with replacing the Weibull with the CCDF of the exponential distribution

$$E(x) = e^{-\lambda x} \tag{A.1}$$

which leads to the reuse distance CCDF

$$C'(x) = pE(x) + (1-p)P(x), \quad 0 < p < 1. \tag{A.2}$$

The results show that fitting with Eq. A.2 achieves comparable accuracy with fitting with Eq. 3.3.



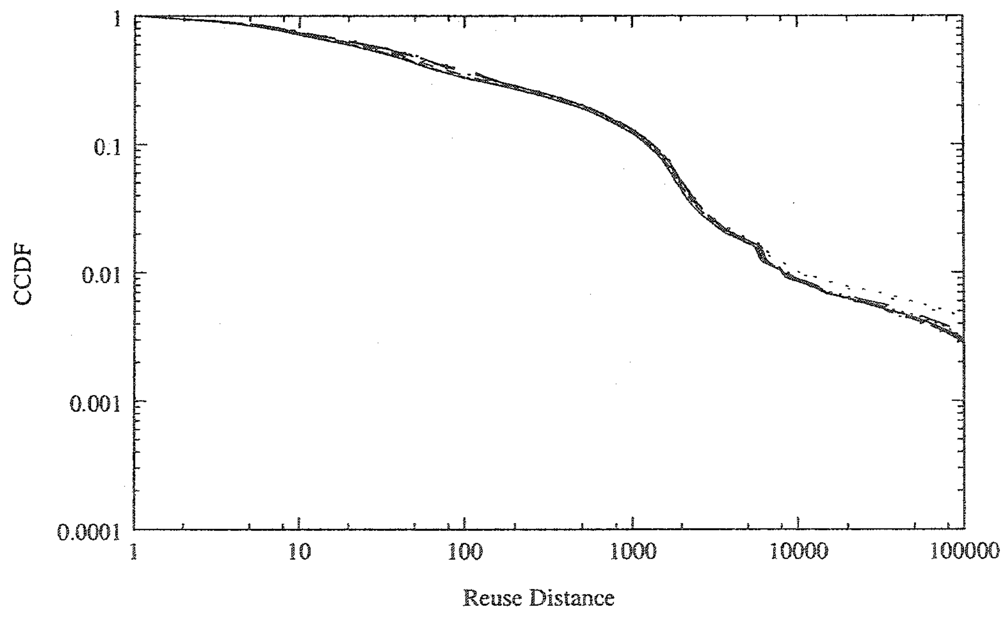
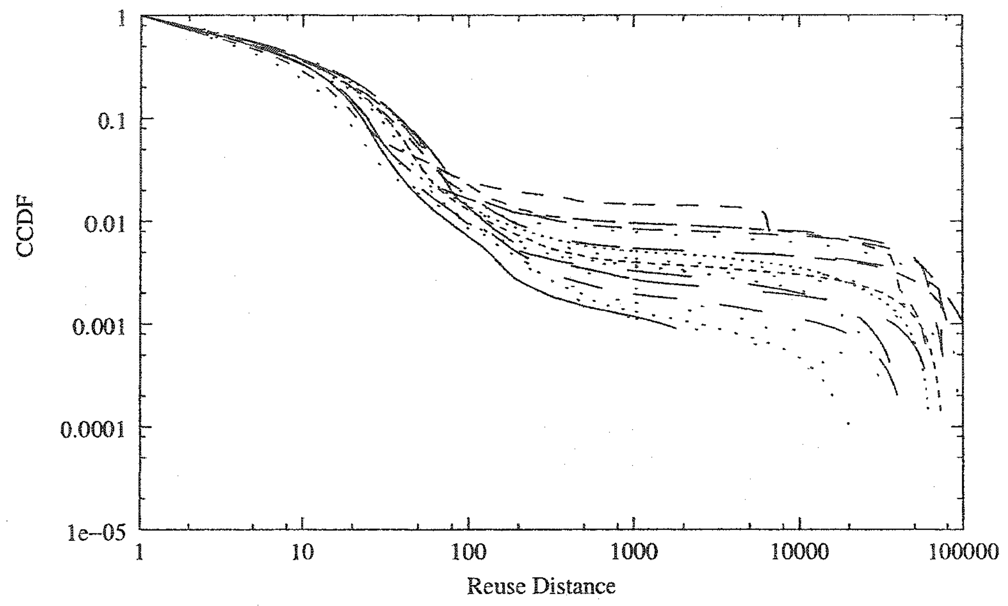


Figure A.1: CCDF's for the traces in the CLEV-0(left) and CLEV-1 groups



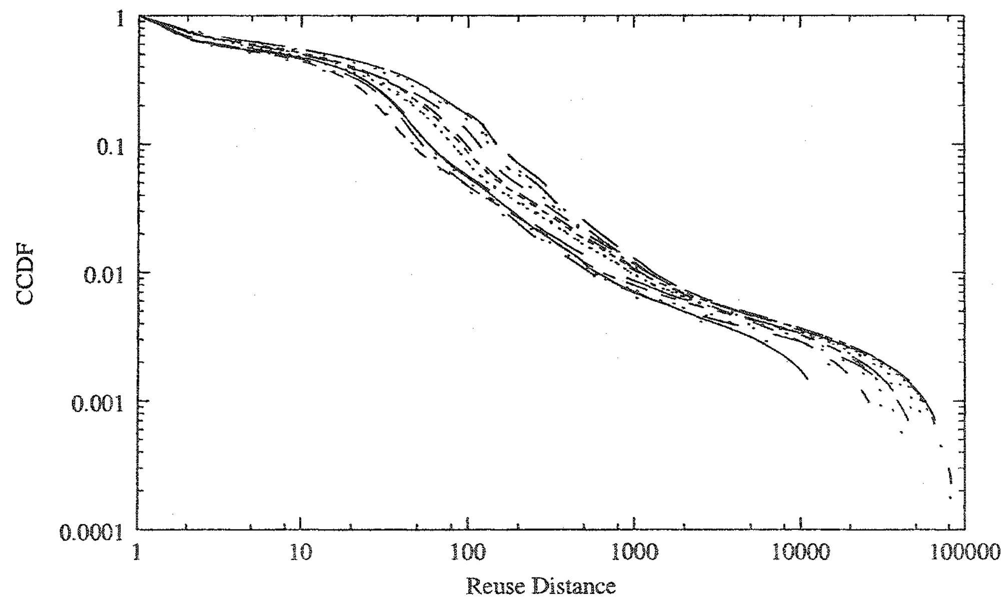
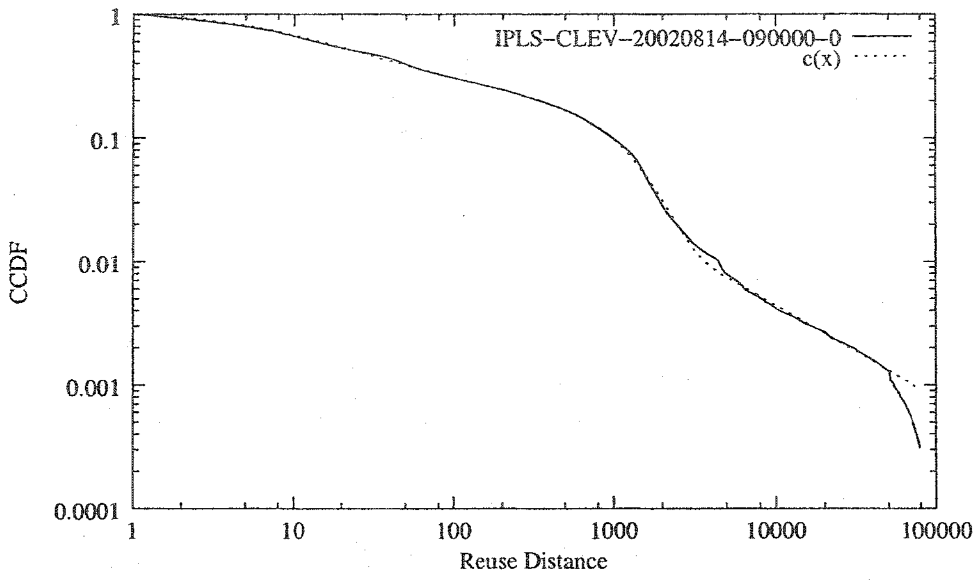


Figure A.2: CCDF's for the traces in the AuckIV-0(left) and AuckIV-1 groups



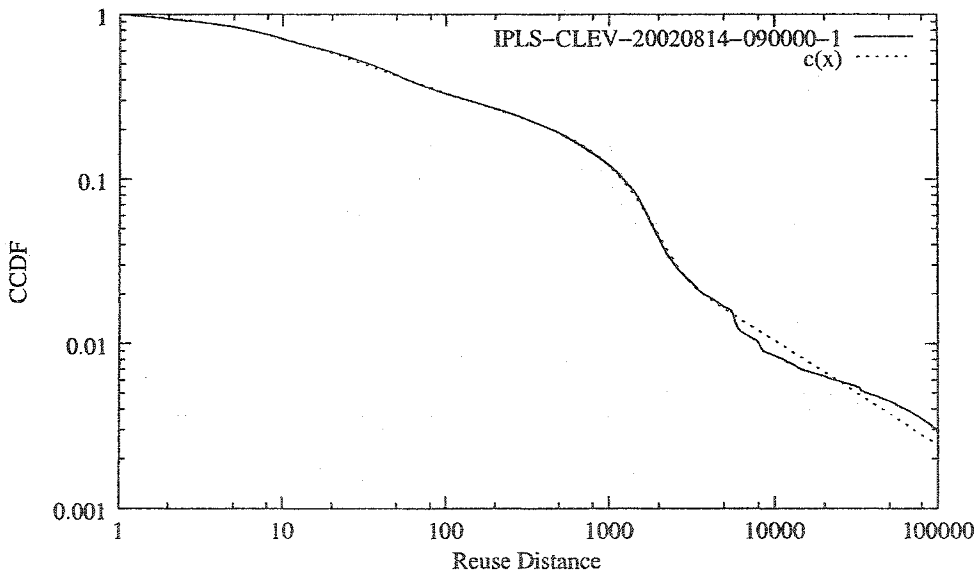
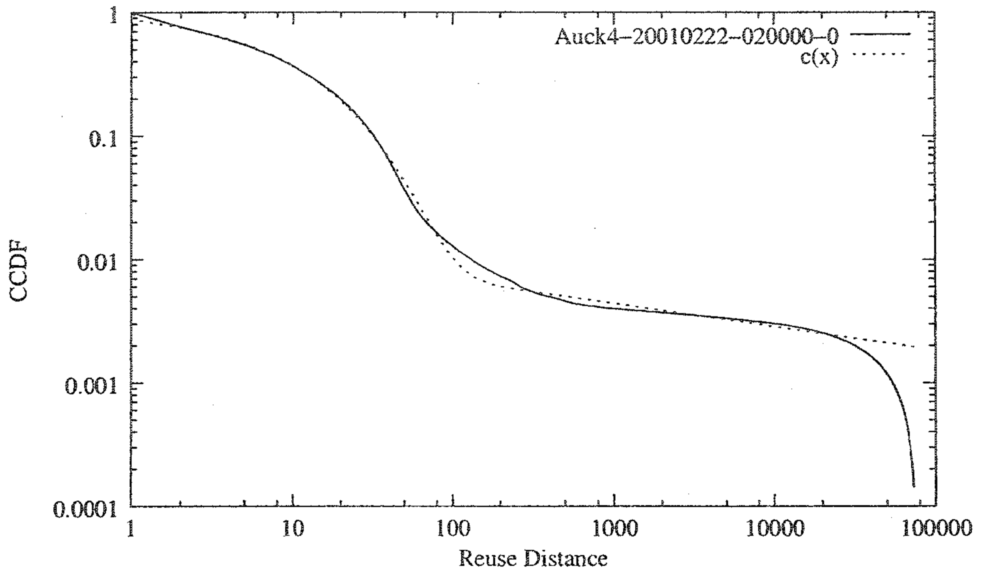


Figure A.3: Some CCDF Fittings(CLEV)



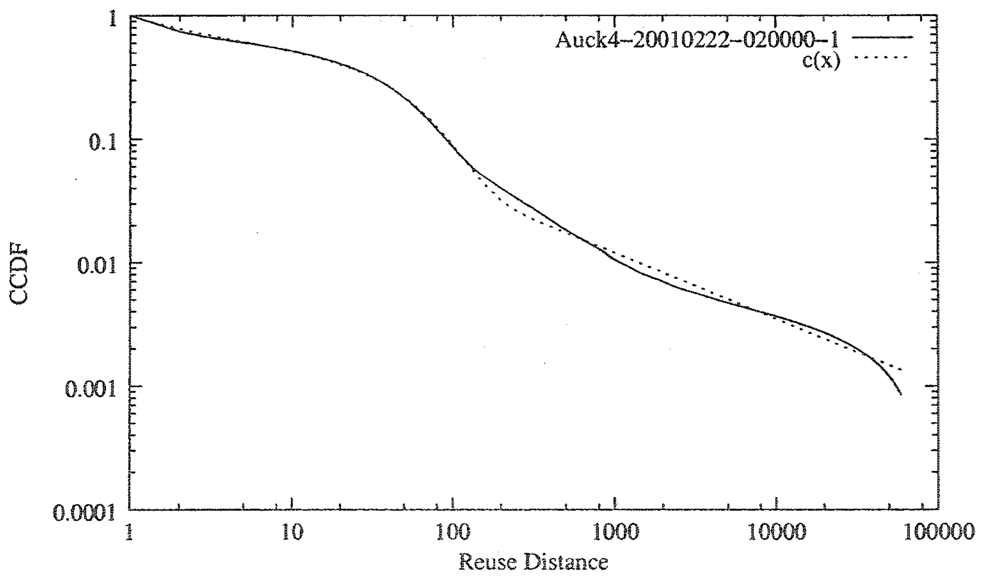
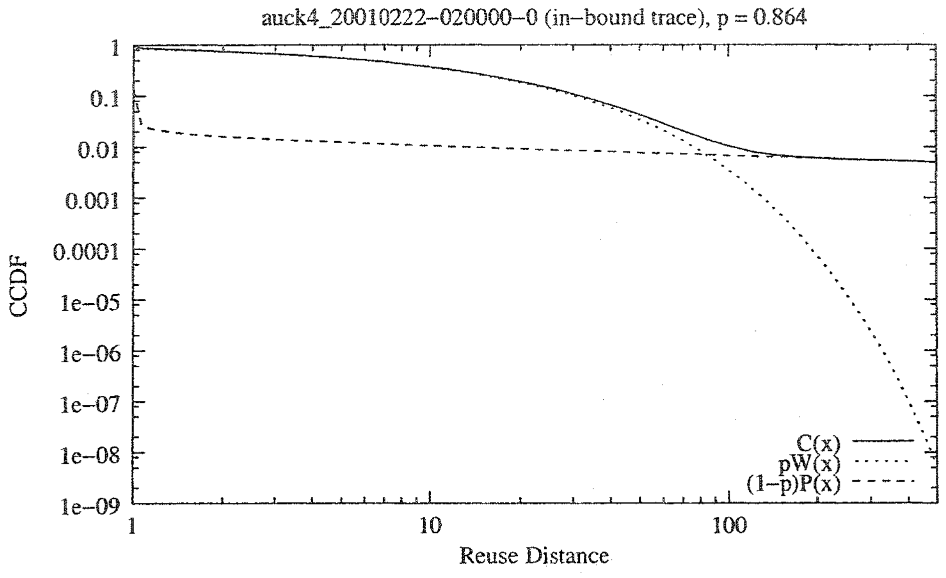


Figure A.4: Some CCDF Fittings(AuckIV)



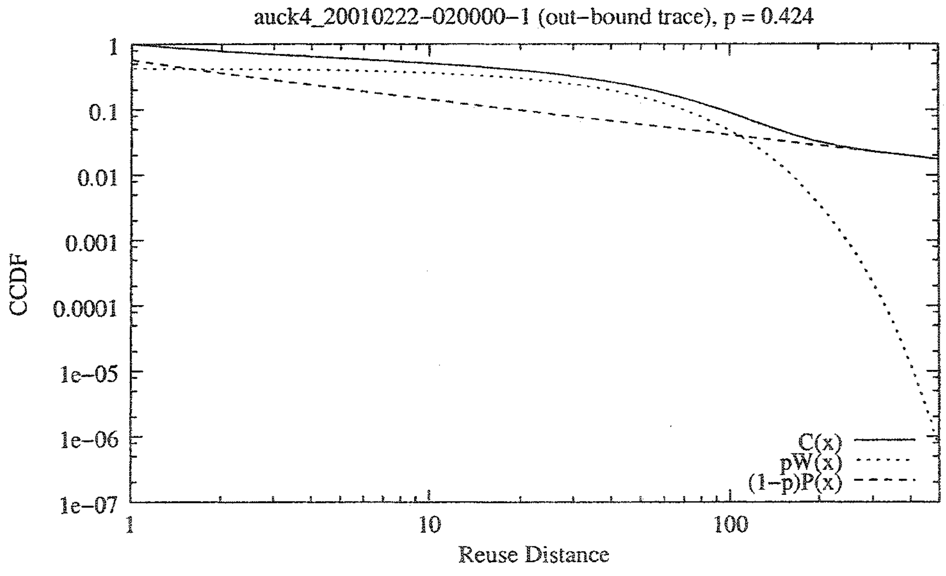
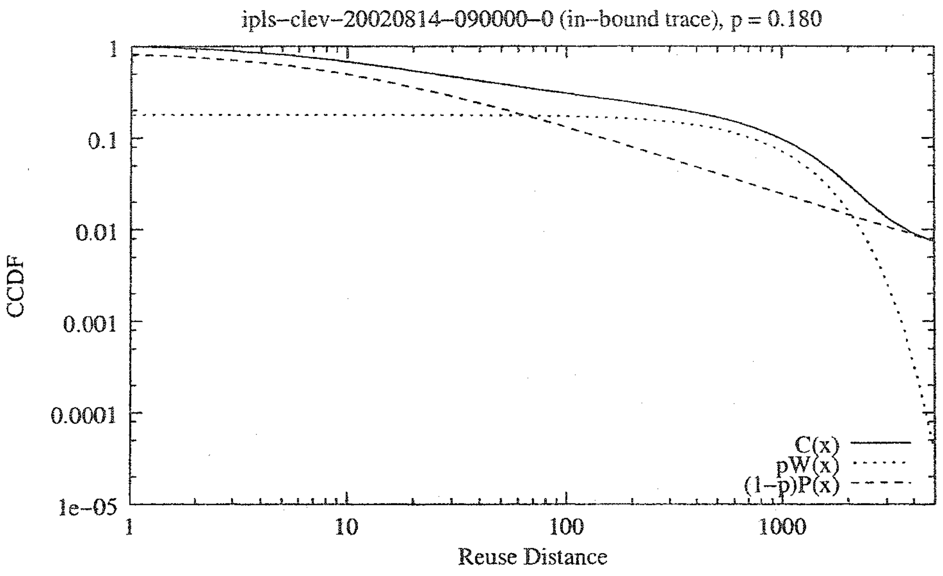


Figure A.5: Effects of p: Fitted CCDF's for Two AuckIV Traces and Their Components



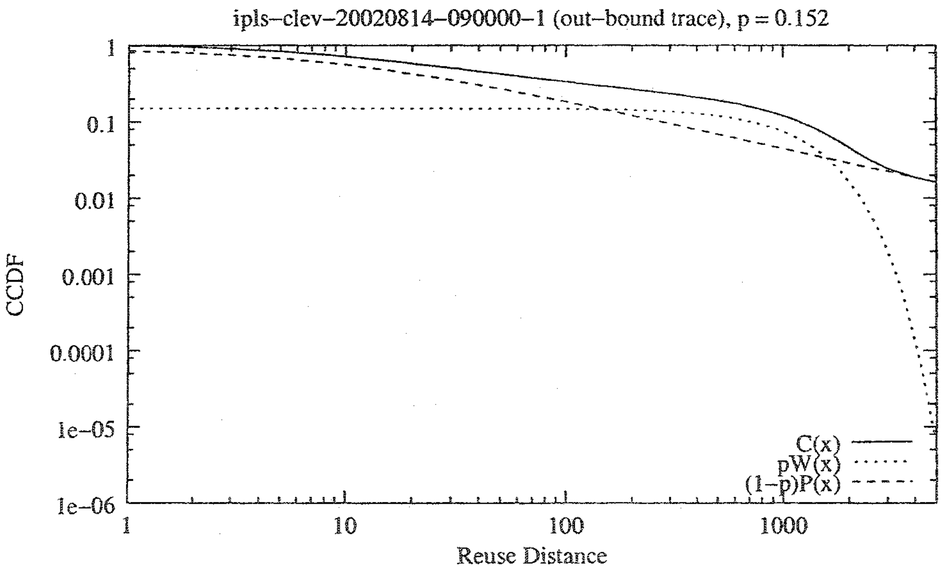


Figure A.6: Effects of p: Fitted CCDF's for Two IPLS-CLEV Traces and Their Components