

Sequence-Preserving Adaptive Load Balancers

Weiguang Shi
Kiyon Inc.
San Diego, CA, USA
weiguang@kiyon.com

Lukas Kencl
Intel Research
Cambridge, UK
lukas.kencl@ieee.org

ABSTRACT

Load balancing in packet-switched networks is a task of ever-growing importance. Network traffic properties, such as the Zipf-like flow length distribution and bursty transmission patterns, and requirements on packet ordering or stable flow mapping, make it a particularly difficult and complex task, needing adaptive heuristic solutions. In this paper, we present two main contributions:

Firstly, we evaluate and compare two recently proposed algorithmic heuristics that attempt to adaptively balance load among the destination units. The evaluation on real life traces confirms the previously conjectured impact of the Zipf-like flow length distribution and traffic burstiness. Furthermore, we identify the distinction between the goals of preserving either the sequence order of packets, or the flow-to-destination mapping, showing different strengths of each algorithm. Secondly, we demonstrate a novel hybrid scheme that combines best of the flow-based and burst-based load balancing techniques and excels in both of the key metrics of flow remapping and packet reordering.

Categories and Subject Descriptors

C.2.1 [COMPUTER-COMMUNICATION NETWORKS]: Network Architecture and Design

General Terms

Algorithms, Performance, Design.

Keywords

Load Balancing, Adaptive Methods, Multiprocessor Systems.

1. INTRODUCTION

For a decade, the Internet has experienced unprecedented growth. It has become an increasingly challenging task to design and implement a correspondingly scalable network infrastructure. In particular, demands for high bandwidth,

computation power, and flexibility have put the key Internet systems, e.g. routers, server farms, web caches, firewalls or load-balancers, under great strain.

An Internet system is a device capable of processing incoming and outgoing IP packets, for example in order to decide onto which output link to forward them. To provide further services, e.g. packet stream reassembly or encryption/decryption, and to guarantee quality of service, extra computation may be performed. Increasingly, solutions employing various types of *multiprocessor architectures* are being adopted to cope with the strain of the massive workloads generated by networking traffic, both within the networks as well as in the end hosts. Routers equipped with multiple forwarding engines or multiple parallel outgoing links, web server farms or distributed web caches are prime examples of such architectures.

As the domain of general-purpose processors has begun the gradual shift towards multi-core architectures, the same issues of packet workload distribution are soon going to be relevant in that domain too. Another example of multiprocessor technology are Network processors (NP) [5], specialized processors customized for networking applications. Their multiprocessor architecture allows for high throughput due to parallelization of packet processing.

Most packet processing tasks involve only one packet and are done on a per-packet basis. This is ideal for parallel processing. However, designing parallel packet processing schemes for IP traffic contains some challenges too:

Firstly, it is critical to spread traffic evenly to ensure high utilization of individual processing units under heavy load. Only when no units are idling can the system operate at its full potential.

Secondly, the packet processing scheme should preserve packet ordering within packet *flows*, which are identified by the common fields of IP headers, e.g. typically the five-tuple of the source and destination IP addresses, source and destination port numbers and the protocol number. Many network protocols are designed based on the assumption of in-order delivery service of the network. For example, TCP on an end host can misinterpret packet reordering as an indication of loss and send duplicate acknowledgements of previous data. After receiving a certain number of these acknowledgements, its peer will re-send packets and halve its transmitting window. Realtime applications, e.g., VoIP, may need to implement buffering to accommodate out-of-order packets.

An alternative formulation of the second goal is to require all packets of a single flow to be processed by the same pro-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ANCS'06, December 3–5, 2006, San Jose, California, USA.
Copyright 2006 ACM 1-59593-580-0/06/0012 ...\$5.00.

cessing unit. This may be important for stateful applications such as flow policing, TCP offload or intrusion detection. As we demonstrate in the paper, these alternatives are not identical and a different solution may cater better to either.

Thirdly, as the gap between microprocessor clock speed and memory latency increases dramatically, it is highly desirable that the performance-critical tasks on the individual processing units see good temporal locality in their workload. For example, the costly IP route lookup operation can benefit from caching the recent results to save full-scale lookups for subsequent packets to the same destinations.

In addition, system scalability, fault tolerance, and resource consumption are important metrics for load balancing designs, however these are not addressed in this paper.

To gain a deeper understanding of the problems and solutions, in this paper, we evaluate two recently proposed load-sharing schemes for parallelizing packet processing in the above aspects. Our main contribution is twofold - we compare the schemes' performance on recent real-world network traces, and draw important conclusions about the factors influencing each method's performance - e.g. burstiness and Zipf-like popularity distribution [23] - with respect to various performance metrics. Furthermore, we introduce a novel algorithm, *Hashing Adapted by Burst Shifting (HABS)*, which combines the strengths of the previously proposed methods and excels in all the metrics considered.

The rest of the paper is organized as follows. After a brief review of related studies in Section 2, we state the load balancing design problem in Section 3. Three schemes are described in Section 4. We present the traces used for this study and the simulation results in Section 5. In Section 6 we further discuss and conclude this work.

2. RELATED WORK

Load balancing, or load sharing, is a well studied problem of computer science [4, 7]. As in recent years much effort has been directed towards optimizing the performance of networking systems, specific sub-formulations of the load balancing problem have attracted great attention too, in step with research focussing on optimizing router architectures and key algorithms, e.g., routing table lookup and packet classification. With the advent of multiprocessor networking systems, e.g. NPs, studying different approaches of parallelizing packet processing has become even more important.

Among others, load balancing is vital to the performance of Web sites and Web caches that use multiple servers to process user requests. For Web cache systems, a hash-based scheme called *highest random weight* (HRW) has been proposed in [22] to achieve high Web cache hit rate, load balancing, and *minimum disruption* in the face of server failure or reconfiguration. To request a Web object, HRW uses the object's name and the identifiers of cache servers, e.g., IP addresses, as the keys of a hash function to produce a list of *weights*. The server whose identifier produces the largest weight is selected to serve the request. If a server fails, only the requests that were mapped to that server are migrated to other servers, while the other requests are not affected, i.e. minimum mapping disruption is achieved. HRW has been extended to accommodate heterogeneous server systems [14] by assigning *weight multipliers* to cache servers to scale the return values of HRW. A recursive algorithm calculates the multipliers such that the object requests are divided among the servers according to a pre-defined distribution.

Bennett, et al. [2] explain the surprisingly hard problem of preventing packet reordering in a parallel forwarding environment and its negative effects on TCP. The authors outline possible solutions from different angles and point out that at IP layer, hashing as a load-distributing method can preserve packet order within individual flows in ASIC-based parallel forwarding systems; however, under-utilization of individual Forwarding Engines (FEs) can occur with simple, static hashing.

Laor and Gendel [11] revisited the packet reordering problem in a lab environment. The authors predict the usage of more parallel processing in IP forwarding. Besides advocating the use of transport layer mechanisms, e.g., TCP SACK and D-SACK, that can deal with packet reordering to some extent, the study points out that load balancing in a router should be done per source-destination-pair (and not per packet) to preserve order.

Dittmann and Herkersdorf [6] describe a load balancer for parallel forwarding systems. A hashing scheme is used to split the traffic, with packet header fields as a key to the hash function. The return value is used as an index to a look-up memory to retrieve the target FE. Flows that yield the same index value are called a flow bundle. 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, 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 re-assignment 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 minimum number of packets in its queue. Third, excessive flow bundles are detected and repeatedly assigned to the least-loaded processors. This is called flow spraying.

Adishesu et al. [1] propose a load sharing protocol for network traffic across multiple links. At the sender, it performs the *fair load sharing* distribution strategy derived from *fair queueing*. Note that the protocol is general in that it does not distinguish among packets and treats them simply as atomic exchange units. At the receiver side, the protocol proposes a re-sequencing algorithm that provides *quasi-FIFO* delivery where, except for occasional periods of loss of synchronization, packets depart from the receiver in the FIFO order. The work presented in this paper recognizes the *partial* ordering requirement, i.e., that only packets from the same flow need to be delivered FIFO, and takes advantage of Internet packet arrival pattern. Our scheme employs efficient hashing and strives to maintain partial ordering without an explicit synchronization algorithm.

A load-balancer is developed in [18] that actively identifies and spreads dominant flows over multiple FEs. Ref. [9] proposes a similar design. These works demonstrate that achieving load balancing without splitting individual flows over multiple FEs is not always possible. Consequently, preventing packet reordering is incompatible with maximizing the performance of a parallel system. These schemes, and some others in the field of traffic engineering, e.g., [16], are motivated by the observation that a few potent flows in the traffic mix usually cause spikes in traffic volume and can be used to actively balance the system. This observation is also made in [15] where flows are classified into dominant *alpha* and small-volume *beta* flows.

Generally, Internet traffic is considered to be bursty. At the flow level, due to its window-based congestion control mechanism, TCP sends packets in *bursts*. Since TCP packets make the bulk of current Internet traffic, this behavior lends itself to designing scheduling schemes to prevent packet reordering within flows. Sinha, et al. [20] and Shi, et al. [19] propose to take advantage of this phenomenon with simple load balancing algorithms.

3. THE LOAD-SHARING PROBLEM

The problem of sequence-preserving load-sharing (or load-balancing) in packet-processing systems can be formalized as follows: given a system of N processing units, each equipped with processing power μ_j , and a stream P of packets P_i , where each packet belongs to a flow F_i , how do we design a mapping function $f() : P \rightarrow Z_N$, such that:

Load Balance: A metric $\phi(f, P)$, measuring how well the packets are spread among the processing units, is minimized, i.e. $\min_{f()} \phi(|\{f(P_i) = 1\}|, |\{f(P_i) = 2\}|, \dots, |\{f(P_i) = N\}|)$. We can consider ϕ to be various mathematical functions evaluating the performance of the mapping function $f()$, or, in other words, the parameters of the mapped load vector, e.g. standard deviation, coefficient of variation or count of all packets exceeding the capacity of an individual processing unit, i.e. *packet drops*.

Preserving Sequence: A metric $\psi(f, P)$, measuring the amount of *packet sequence reordering* in the packet stream leaving the system in comparison to the packet stream entering the system, is minimized, i.e. if $s(P_i)$ is the sequence number of packet P_i entering the system and $s'(P_i)$ the sequence number of the same packet when leaving the system, then $\psi(f, P) = |\{(i, j) | (s(P_i) < s(P_j)) \wedge (s'(P_i) > s'(P_j))\}|$. The goal of packet sequence preservation can be relaxed to apply to *packets belonging to an individual flow only*, formalized as: $\psi'(f, P) = |\{(i, j) | (P_i, P_j \in F_i) \wedge (s(P_i) < s(P_j)) \wedge (s'(P_i) > s'(P_j))\}|$. Another measure of preserving the sequence of packets is how consistently are *packets from a particular flow mapped to a particular processor* - i.e. we aim to avoid subsequent packets from the same flow being mapped to different processors. This may be important for some application operating at flow level, e.g. flow rate policing or TCP reassembly. This can be formalized as minimizing $\psi''(f, P) = |\{(i, j) | (P_i, P_j \in F_i) \wedge (i < j) \wedge (f(P_i) \neq f(P_j))\}|$. It is vital to realize that although related, the objectives are not identical, as formulation ψ' allows for subsequent packets being re-mapped, as long as they are not reordered in the outgoing sequence.

Kencl and Le Boudec [10] have demonstrated that finding the optimal solution to the problem of balancing the load with minimal packet re-mapping (i.e. using the ψ'' objective function) is NP-hard even if we knew the exact packet sequence in advance, which of course is not true for a realistic load balancer. This leads to a conclusion that a practical heuristic is needed to address the problem.

Shi et al. [18] have shown that the Zipf-like [23] probability distribution $P(R) \sim 1/R^a$, which states that the frequency of some event P is proportional to the function of its rank R , with the exponent a close to 1, can result in vast inefficiency in static load balancing schemes (evaluated using coefficient of variation as ϕ), if traffic properties confirm to it. Yet that is known to be the case in the real world. For distributions with a larger than 1 (very likely in real-world scenarios), the

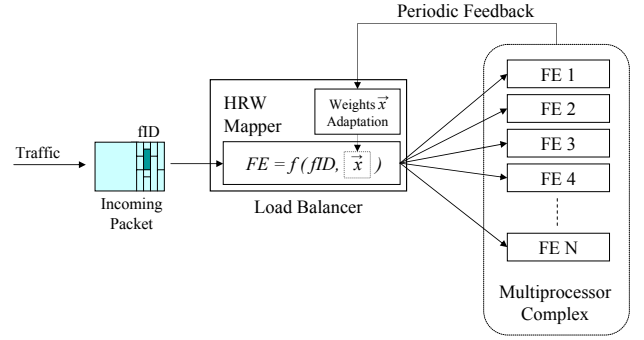


Figure 1: Adaptive HRW Hashing (AHH). Periodically, the Load Balancer gathers information about the workload intensity of the processors. If adaptation is triggered, the balancer adjusts the hash multipliers \vec{x} .

authors prove that static hashing on flow identifiers cannot balance the workload.

4. LOAD-SHARING SCHEMES

In this section we describe three adaptive load-sharing schemes, two of them proposed previously and a third, novel scheme, improving on the known solutions.

4.1 Adaptive HRW Hashing (AHH)

The scheme proposed in [10] balances the load among the processing units by mapping packet flows to processors using a weighted hash function (see Fig. 1). The mapping is computed using the Highest Random Weight (HRW) [22, 14] function over a flow identifier (fid) carried in the packet (e.g. a protocol 5-tuple):

HRW Packet-to-Processor Mapping f : Let a packet arrive carrying a flow identifier vector fid . The mapping $f(fid, \vec{x})$ is then computed as follows:

$$f(fid, \vec{x}) = j \quad (1)$$

$$\Leftrightarrow$$

$$x_j g(fid, j) = \max_{k \in \{1, \dots, N\}} x_k g(fid, k), \quad (2)$$

where $x_j \in \mathbb{R}^+$ is a weight multiplier assigned to each processing unit FE j , $j \in \{1, \dots, N\}$, and $g(fid, j)$ a pseudo-random function $g : fid \times \{1, 2, \dots, N\} \rightarrow (0, 1)$, implemented as a hash function in practice. The weights $\vec{x} = (x_1, \dots, x_N)$ determine the fraction of the identifier vector space assigned to each processing unit [14].

Periodically, the balance of load is evaluated and the mapping *weight multipliers* are adjusted. The adaptation algorithm consists of two parts: a triggering policy and an adaptation policy. Using a pair of dynamic thresholds to determine whether some unit is under- or over-utilized, an adaptation is triggered that adjusts the weights. In other words, the algorithm treats over- or under-load conditions as changes of processing power of the processors. The adaptation algorithm preserves the minimal disruption property of HRW, i.e. that only the minimal necessary amount of flows is re-mapped [10].

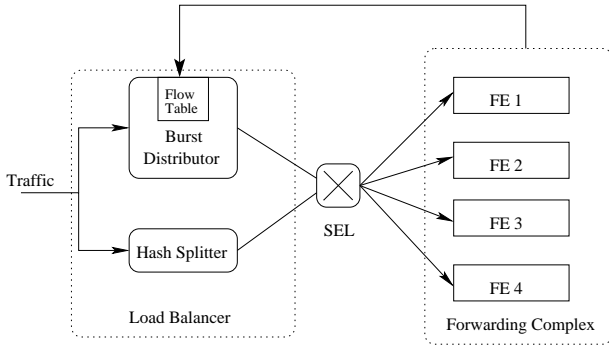


Figure 2: Adaptive Burst Shifting (ABS)

To evaluate the status of individual processors, periodic processor workload intensity $\rho_j(t)$ is measured and a smoothed, low-pass filtered processor workload intensity $\bar{\rho}_j(t) = \frac{1}{r} \rho_j(t) + \frac{r-1}{r} \bar{\rho}_j(t - \Delta t)$ is computed (r is an integer constant). A similar measure is used for total system workload intensity: $\bar{\rho}(t) = \frac{1}{r} \rho(t) + \frac{r-1}{r} \bar{\rho}(t - \Delta t)$. The filtering reduces the influence of the short-term load fluctuations.

Triggering Policy specifies conditions for action, by using a dynamic workload intensity threshold $\epsilon'_\rho(t) = \frac{1}{2} (1 + \bar{\rho}(t))$, and a fixed hysteresis bound $\epsilon_h > 0$, preventing adaptation within the interval $((1 - \epsilon_h) \bar{\rho}(t), (1 + \epsilon_h) \bar{\rho}(t))$. The ϵ_h is typically set to a value close to 0, for example 0.01, thus preventing adaptation when the load stays within 1 percent of the total system workload intensity. The result of the comparison of the filtered workload intensity to the threshold then acts as a trigger for the adaptation to start.

Weights-Vector \vec{x} Adaptation Policy specifies how to act when triggered. If the system is underutilized, the adaptation lowers the weights for the exceedingly overutilized processors, or, if the system in total is overutilized, the adaptation raises the weights for the exceedingly underutilized processors. The lowering or raising of weights is carried out using a multiplicative factor proportional either to the minimal workload intensity $\bar{\rho}_j(t)$ exceeding the threshold $\epsilon_\rho(t)$, or to the maximal workload intensity $\bar{\rho}_j(t)$ below $\epsilon_\rho(t)$.

Unlike other schemes, AHH does not maintain per-flow information, which gives it the advantage of being less resource-demanding. For a detailed explanation of the AHH method and its mathematical background, the reader is best referred to [10].

4.2 Adaptive Burst Shifting (ABS)

The ABS scheme is described in [19]. This approach takes advantage of the ubiquitous burst transmission pattern of TCP traffic, as the result of TCP's window-based congestion control algorithm. TCP allows one window's worth of data to be transmitted during one round-trip time (RTT) interval between the sender and receiver.

The insight is that gaps between adjacent bursts are much larger than those between consecutive packets within a burst and therefore can allow bursts of one flow to be scheduled onto different target processors without causing packet re-ordering. In addition, since traffic is scheduled on a much smaller scale, i.e. bursts instead of flows, load balancing can be easily achieved.

Fig. 2 shows an example design of a four-FE system. The

- STEP 1. Search the table using the flow ID of the packet.
- STEP 2. IF a valid entry is found
- STEP 3. THEN return the FE field of the entry
- STEP 4. IF the table is not full
- STEP 5. THEN insert the entry and return the minimum-loaded FE
- STEP 6. Return an invalid FE index

Figure 3: The ABS Algorithm

design parameter in this scheme is simply the number of entries in the flow table. The larger this number, the more effective the scheme in balancing load. The load adjustment algorithm is shown in Fig. 3.

The *burst distributor* is responsible for looking up the present flows, inserting the newly arrived flows in the *flow table* and assigning them to the least-loaded processor. Each flow table entry contains a flow ID and number of its packets currently in the system. The system continuously monitors and updates the flow table. When the packet number field of an entry becomes zero, the entry is removed from the table to make room for new flows. Working in parallel, the *hash splitter* implements hashing over flow identifiers and outputs the index of the processor where the packet is to be forwarded to. For each packet, the *selector* receives forwarding decisions from both the distributor and the hash splitter and always honors that from the distributor, when active.

4.3 Hashing Adapted by Burst Shifting (HABS)

This novel algorithm builds on previous research, especially AHH and ABS [10, 19]; it introduces innovative ideas to address the inefficiencies of the existing schemes.

4.3.1 The HABS Algorithm

First, as shown in Fig. 4, the patent change made in HABS is to place the hash splitter *sequentially* in front of the burst load adapter. This arrangement makes it possible for the adapter to decide the target FE of a burst when its flow is not found in the flow table. This capability is important since the burst adapter now has more information than merely the binary state of the flow table (full or not-full) to make a decision on flow admission. For example, instead of admitting any new burst whenever there is room in the flow table and (re-)directing the burst to the less-busy FEs, the adapter now has the choice only to re-direct bursts that are targeted to, say, the busiest FE by default, i.e., as decided

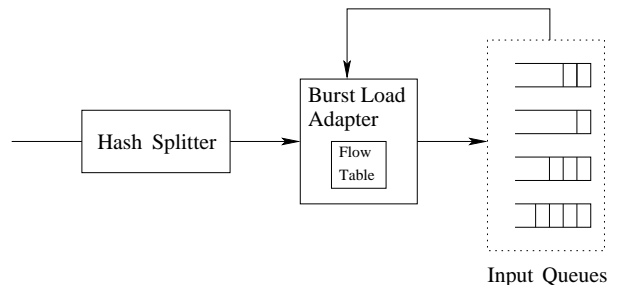


Figure 4: Hashing Adapted by Burst Shifting (HABS)

- STEP 1. The hash splitter assigns an FE, FE-hash, for the incoming packet.
- STEP 2. If an entry is found for the flow in the flow table, return the FE in the entry.
- STEP 3. If the flow table is full, return FE-hash.
- STEP 4. If FE-hash is overloaded, assign the least-loaded FE to the flow and insert the entry.
- STEP 5. Return FE-hash.

Figure 5: The HABS Algorithm

by the hash splitter. This extra information is critical in reducing flow-remapping and thus helping to preserve locality at the FEs.

Second, HABS introduces major innovations in the burst load adapter by adding triggering conditions. The insight behind the burst adapter is the same as in ABS that the dominant transport protocol TCP transmits packets in bursts with relatively large gaps in between, yet HABS redirects only bursts destined to the overloaded FEs. To decide whether an FE is overloaded, we introduce a triggering process constructed similarly as in AHH, using the (non-filtered) immediate status of the input queues and a second hysteresis bound, ϵ'_h , for burst shifting. Mathematically, if N is the number of processors, $bufsz$ the total buffer capacity available to store the input queues and $npkts$ the total number of packets currently in the system, we define the trigger as

$$(1 + \epsilon'_h) \frac{1}{2} \frac{(bufsz + npkts)}{N}. \quad (3)$$

As in AHH, the trigger is thus positioned halfway between the proportionally ideal (at the time) and the (proportionally) maximum-possible queue length (we do not consider overutilization of the entire system possible, $bufsz \geq npkts$ is always true). If a queue's length is larger than the trigger, the burst heading to the FE behind the queue is redirected to the least loaded FE. Note that this condition decides both *when* and *what* to switch.

To eliminate reordering, when a packet arrives, we simply check if a previous packet from the same flow exists in the system. If there is such a packet, it is not safe to treat the incoming packet as the start of a burst and switch it to the least loaded FE.

In summary, the HABS algorithm executes steps shown in Fig. 5. STEP 1 is performed by the hash splitter and the rest are performed by the burst distributor. In STEP 4, the simplest way to balance the system would be to redirect the bursts destined for the *most* loaded FE in the system, however, redirecting bursts destined to any overloaded (triggered) FE is more effective. With a background process that removes the entries when no matching packets exist in the system, this method maintains order during bursts and thus eliminates packet reordering.

4.3.2 Improvements on AHH and ABS

In agreement with the findings in [18], HABS augments the initial static hash with an adaptive routine, burst shifting, to compensate for the skewed flow popularity distribution and to exploit the bursty nature of traffic. The sequential addition of burst shifting to the more traditional flow hashing allows to build an adaptive, yet almost reorder-free parallel system. The use of the initial hash splitter to sug-

gest a destination processing unit, together with strong constraints on triggering a burst shift, likewise leads to dramatic reductions in flow remapping.

The original ABS load balancer [19] does not consider two factors that make burst distribution more effective in balancing load: first, bursts are not created equal and some are more amenable to switching than others. Bursts can be classified in different ways. They can be grouped according to the load of the FEs they are sent onto. Intuitively, those that are destined to the most-loaded FEs should be switched, relieving the most loaded FEs and adding load to the idling ones; those to the lightly loaded FEs should be left alone.

Second, burst switching, or any other load balancing scheme, does not have to be invoked all the time. In a load balancing system, the goals are to improve forwarding rates and to reduce packet reordering and flow remapping. Spreading load evenly, as e.g. indicated by a small coefficient of variation in the number of packets processed by the FEs, is not a *necessary* condition for any of the objectives. The improved triggering policy allows to switch less frequently, while the flow table allows to preserve the shifted mapping.

5. SIMULATIONS

We conducted trace-driven simulations for the three load-balancing schemes described in Section 4.

5.1 Traces Used

To evaluate the load balancing schemes, we used two different sets of traces, one representing a more central position within the network, and one recorded at a position at the network edge. The network core is represented by two traces, described in more detail in Table 1, made available by the national laboratory of applied network research (NLNR) [13] in the USA. To represent the network edge, we used real data collected by the network monitor described in [12]. The site we monitored is host to several biology-related facilities that employ about 1,000 researchers, administrators and technical staff. It is connected to the Internet with a full-duplex Gigabit Ethernet link, where the monitor was placed. Full-duplex traffic was monitored for each traffic-set.

Fig. 6 verifies the idea that the popularity distribution of the flows can be characterized as Zipf-like with a values close to 1. This observation applies to all the traces that we

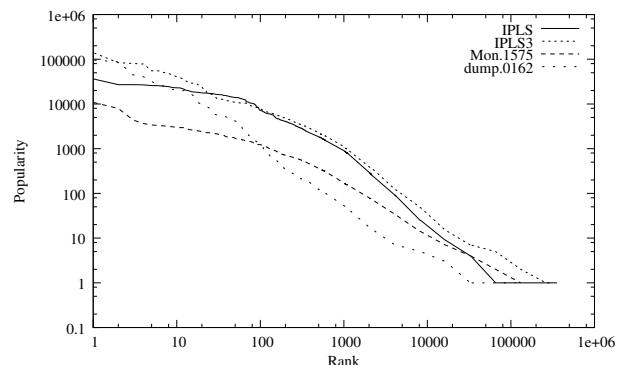


Figure 6: Flow popularities in the traces used.

Table 1: Traces Used in Experiments

Trace	Length (# entries)	5-Tuple Flows	Variability: #pkts/10ms max, mean, min	Description
IPLS-1	5,047,760	355,786	1152, 841, 619	The first minute of a 10-minute and 45 million-entry 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.
IPLS-3	6,656,950	308,197	1586, 1109,785	The first minute of a 9-minute and 57 million-entry trace from a set of traces (Abilene-III) collected from an OC192 Packet-over-SONET links.
Mon.1575	1,186,070	130,722	711, 322, 197	A 40s trace from a Gigabit Ethernet edge Internet link of a medium-sized facility.
dump.0162	1,236,681	113,596	234, 51, 4	A 4 minute trace of the same link under low-load conditions.

examined. In our traces, indeed many TCP flows exhibit bursty behavior, as shown in the example in Fig. 7.

5.2 Metrics

As discussed in Section 1, we measure three aspects of the load-sharing schemes: load-sharing capabilities, packet reordering or re-mapping, and temporal locality in the scheduled traffic. The first is dynamically calculated during simulations and the other two are computed passively.

5.2.1 Load-sharing

When evaluating an adaptive load-sharing scheme, comparing the numbers of packets processed by the different FEs, using e.g. the coefficient of variation, can be misleading. For example, when traffic rates stay low for extended periods; adaptive load-adjusting algorithms may not be activated, since the system is not overloaded in these situations. On the other hand, given fixed processing power and buffer sizes, drop rates indicate the system overall throughput. In our simulations, only load-imbalance can cause packet drops, since locality and packet reordering are evaluated passively using the traces after scheduling. It is therefore reasonable to use drop rates as the metric for the load-sharing capability of the schemes.

5.2.2 Packet Reordering and Flow Remapping

We use a simple method to calculate packet reordering rates: packets are numbered sequentially and incrementally upon arrival. At the output of the forwarding complex, the largest sequence number seen so far, S_i , is recorded for flow i ($0 < i \leq \text{The Total Number of Flows}$). Upon exit, the sequence number of a packet from flow i , s , is compared with S_i . If $s > S_i$, $S_i = s$. Otherwise, the reorder counter is incremented.

We also monitor flow re-mappings, i.e. a count of events when a subsequent packet in a flow is mapped to a different processor than its predecessor. We maintain a table of the current flow-to-processor mappings and increase a counter upon a packet bringing a mapping change.

5.2.3 Locality

Temporal locality in the workload reveals another quality of the traffic scheduled onto an FE, i.e., its *cache-friendliness*. To evaluate this quality of the algorithms, one has to select an application. We choose one of the most time-consuming operations in packet forwarding, i.e., flow classification, for this purpose.

We use the cumulative complementary distribution function (CCDF) of the *reuse distance* (which is equivalent to the *stack distance* in [21]) in a flow sequence to measure its locality. This method has been popular in measuring temporal locality in the workload of different systems.

Basically, the reuse distance of an item is the number of unique items referenced after its previous appearance. Let “ $r_1, r_2, \dots, r_n, \dots$ ” represent the corresponding reuse distance sequence of a flow sequence “ $f_1, f_2, \dots, f_n, \dots$ ”. The reuse distance sequence of an address sequence can be generated using the LRU stack [21]. Imagine a stack that contains all possible flows, one per entry. The index of the stack increases from top (0) to bottom. Each time an address is referenced, the index of the stack entry that contains the flow is output as the reuse distance and the flow is removed and pushed onto the top of the stack.

Let F_i be the set that contains the unique flows in the flow sequence f_1, f_2, \dots, f_i and $|F_i|$ the size of F_i . Then $|F_i|$ is the largest reuse distance seen in the trace up to and including f_i . Therefore, we have

$$r_{i+1} = \begin{cases} idx(f_{i+1}) & \text{if } f_{i+1} \in F_i \\ |F_i| + 1 & \text{if } f_{i+1} \notin F_i. \end{cases} \quad (4)$$

where $idx(f_i)$ yields the index of f_i in the LRU stack.

An advantage of using the CCDF is that the curve actually

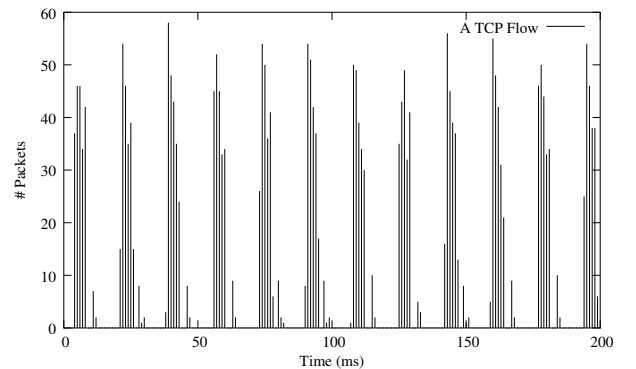


Figure 7: ON-OFF transmission pattern typical of unidirectional TCP flows (2nd largest flow, IPLS-3 trace).

represents miss rates of a fully-associated cache of different sizes (the x axis). Therefore, the CCDF manifests cache performance.

5.3 Experiment Setup

For our simulations, we have used Python scripts to implement the load sharing methods and executed them on Linux desktop machines. Correct timing has been maintained by reading the timestamps from the processed traces. For the basic hash functions in the three algorithms, we have used the standard Python CRC32 library function `zlib.crc32()`. CRC has been shown to have good spreading properties over networking traffic [3]. To compute the $g(fid, j)$ pseudo-random function in AHH, we have used `zlib.crc32(fid + str(j))` per each processor j . The variable parameters of the AHH algorithm are $r = 3$ (the filtering coefficient) and $\epsilon_h = 0.1$ (the hysteresis bound). The ϵ'_h in HABS is set to 0.15 to reflect the fluctuation of the non-filtered utilization.

We simulated an eight-unit forwarding system, with homogeneous processing power. In the simulations, we vary the processing power according to the chosen system utilization (we alternate 0.8 and 0.9). In addition, we assume that the service time is constant and is derived from the average arrival rate and the system utilization. For periodic adaptive algorithms such as AHH, we use fixed intervals between adjacent adaptations. The challenge for this approach and the schemes in general is that the optimal adaptation interval may not be easy to come by. In this paper, as our emphasis is to compare the algorithms instead of searching for sound adaptation intervals, we experimented with different values, and eventually settled for a 100 ms adaptation interval. The size of the system buffer, shared by all the FEs, is 200 packets.

5.4 Results

In this section we first present the simulation results for load-sharing, packet reordering, and flow remapping with one trace, IPLS, and for the two utilization ratios 0.8 and 0.9. We then calculate temporal locality for the overall system and for one FE under the three algorithms. Last, we discuss the results for the other traces.

5.4.1 Load-Sharing, Packet Reordering, and Flow Remapping

Fig. 8 summarizes the results for the three key metrics of load distribution. Under high utilizations, ABS achieves the lowest drop rates. This should be ascribed to two factors: first the general burst transmission pattern of TCP flows and, second, ABS's aggressive scheduling policy, i.e., to switch bursts whenever possible. AHH cannot cope with short-term bursts due to its periodic flow-based scheduling scheme. Of course, we can shorten the period for AHH, but this simple tuning results in much reordering and flow remapping. The HABS algorithm achieves comparable drop rates with ABS. The slightly higher drop rates for the HABS is because only bursts to the highly loaded FEs have to be switched. This constraint, however, brings the major benefits of much lower flow remapping and better temporal locality as we show in the rest of this Section.

As far as packet reordering is concerned, AHH, with its periodic and flow-based scheduling, achieves low rates. On the other hand, with small table sizes ABS and HABS can perform poorly. Between the two, the HABS does better

most of the time due to the extra constraint, i.e., only bursts to heavily loaded FE's are eligible for switching, it imposes. On the other hand, larger flow table sizes help in the case of the burst-based schemes: both ABS and HABS are able to achieve zero reordering rates.

For flow remapping, AHH, again with its least-disruption property [22] and relatively infrequent rescheduling, performs well, and ABS's aggressive burst shifting policy results in much higher remapping rates. HABS fares much better compared with ABS. The subtle burst-shifting policy (3) is the main contributor to the improvement over ABS.

The results for the two utilization ratios 0.8 and 0.9 are juxtaposed in Fig. 8. It is obvious that the results for each metric differ only in magnitudes.

5.4.2 Temporal Locality

Fig. 9 shows the averaged reuse distance CCDFs under different schemes. Each CCDF is obtained by averaging the flow reuse distance CCDFs observed in the traffic seen by every FE. For comparison, also shown in the figure is the reuse distance CCDF of the flows in the original trace on which the measurement was conducted, i.e., the IPLS trace.

Pure hashing usually improves locality seen on each FE over that of the original trace [17]. This is because the diversity of flows is reduced; each FE is allocated a certain portion of the incoming flows, decided by the hash algorithm and the number of FEs in the system.

Adaptation in general disturbs the cache of each FE by feeding flows otherwise unseen to the FE from time to time. Fig. 9 shows that while most of the algorithms produce averaged reused distance CCDFs below that of the original, IPLS, the scheme ABS can result in very poor locality with large flow tables. This is because the algorithm does not discriminate flows and focuses all resources to balance load. It lacks the more restraint triggering policies as in AHH and HABS. This leads to excessive flow re-mapping. AHH, on the other hand, only reschedules every 100 ms, resulting in much less frequent remapping and thus its results largely overlap those from hashing.

The right part of figure 9 further compares effects of the three schemes on temporal locality. It was produced using the IPLS-1 trace, $\rho = 0.8$, a buffer size of 200 packets, and 200 table entries in each flow table. The traffic over which the results are gathered is collected on the first FE, i.e., FE-0 in the figure, under each load distribution scheme. With the large flow tables, neither HABS nor ABS reorders packets. We have seen that remapping was significantly improved in HABS (Fig. 8) over ABS, which directly translates to better temporal locality. As shown in the figure, the reuse distance CCDF of the HABS is very close to that of the pure hashing where no remapping or reordering happens and to that of the AHH where remapping happens much less frequently.

5.4.3 Results for More Traces

Fig. 10 compares AHH, ABS and HABS in the contexts of the four traces. In general, the results confirm what we have seen in the previous sections, i.e., HABS achieves comparable drop rates and reorder rates to ABS, while improving by an order of magnitude in remapping (almost comparable to AHH on the dump trace). For readability, we have not included AHH in the detailed results in Fig. 10, but general trends are clearly similar for all traces.

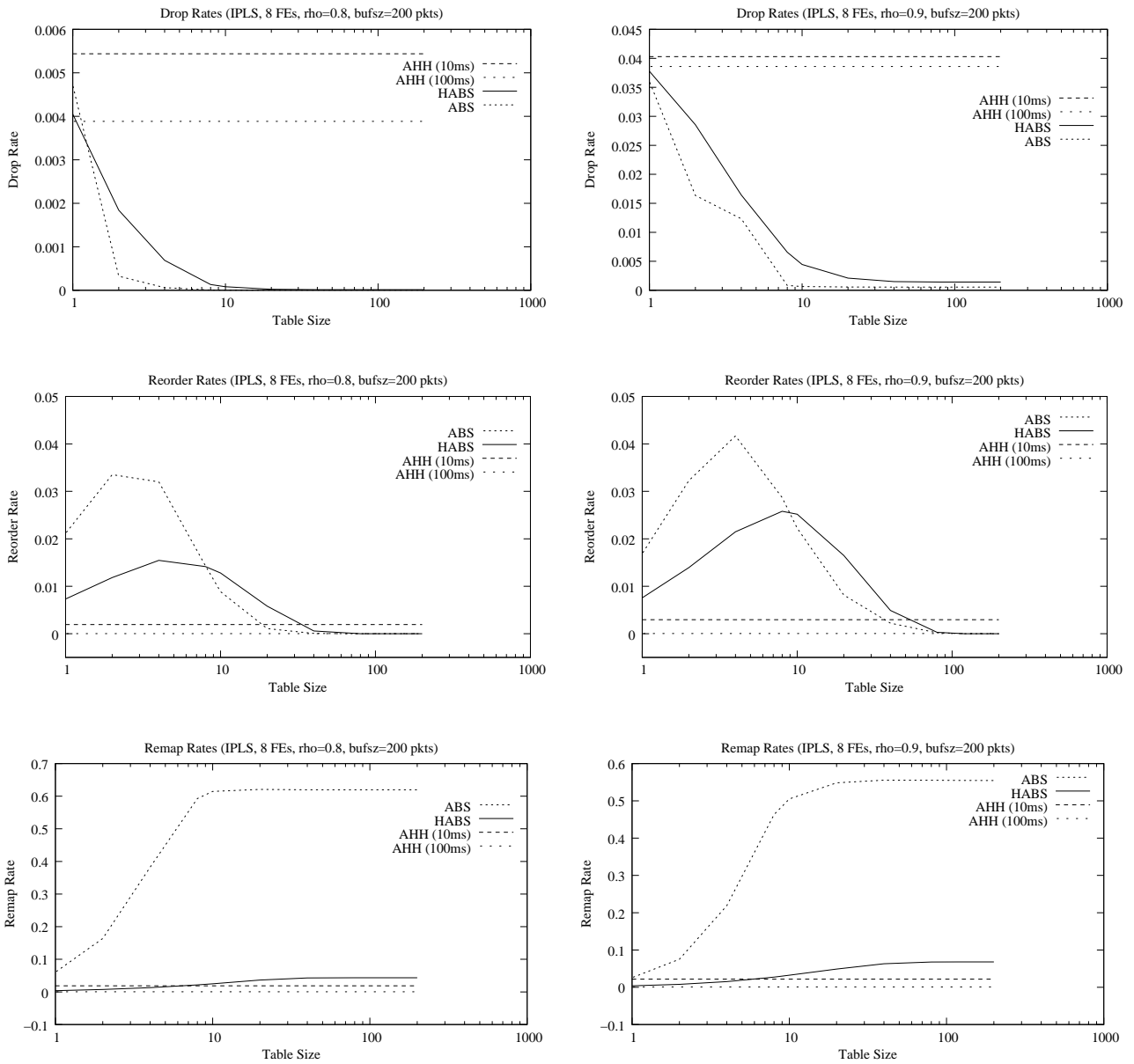


Figure 8: Drop Rates (Top), Packet Reordering Rates (Middle), and Flow Remapping Rates (Bottom)

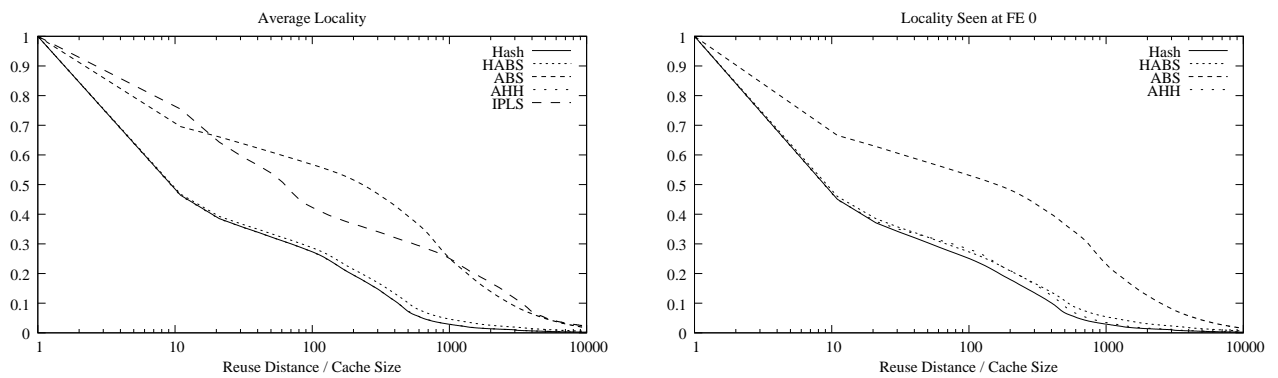


Figure 9: Average Locality (Left) and Locality at FE-0 (Right)

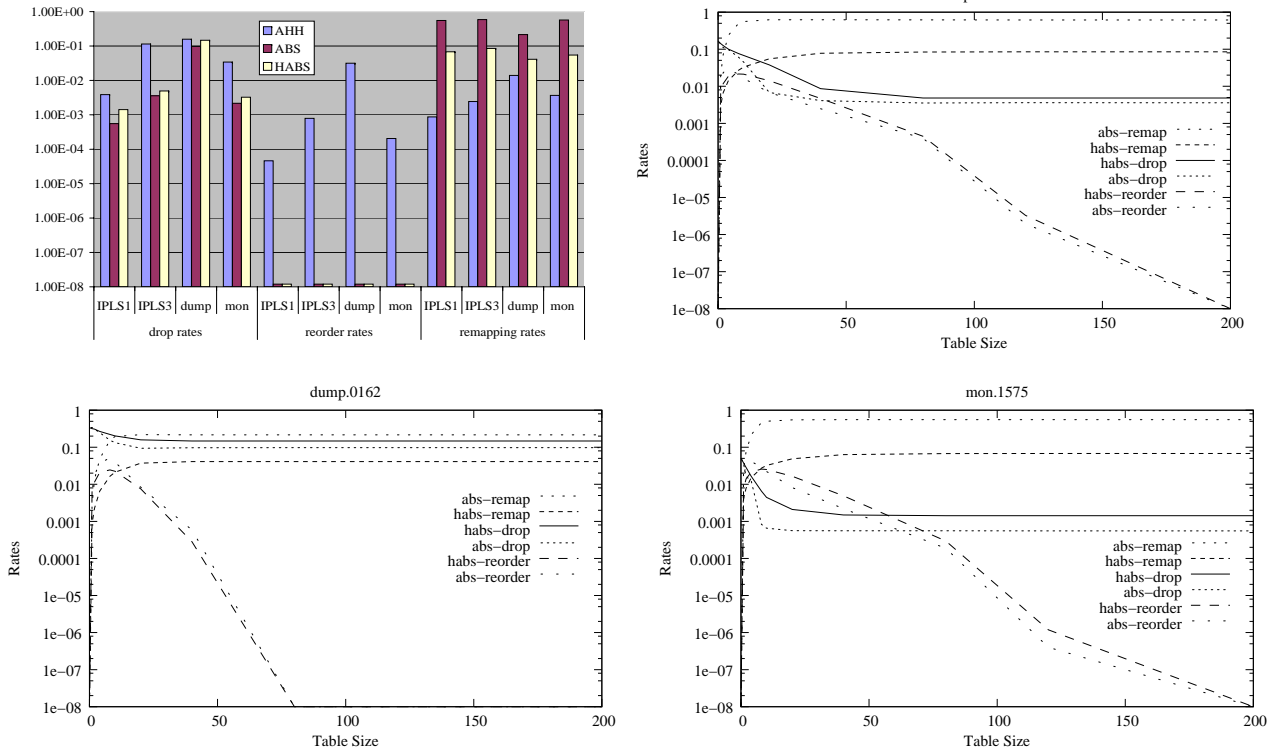


Figure 10: Results with Four Traces

In most of our simulations with the backbone traces, ABS achieves excellent drop rates. On the other hand, its performance, especially the drop rate, degrades with the edge trace dump.0162. This trace has far fewer flows than the backbone traces and a large variance in volume, as shown in Table 1, where we binned the traces into 10ms-slots. The ABS scheduler performs worse with larger flow tables, because when a flow is inserted in the flow table, it will not be removed until there are no packets from that flow in the system; the flow is *locked*. When the table grows large, the majority of the traffic is handled by the burst distributor instead of the hash splitter. When a large portion of the traffic is locked onto one processor, the load is not well balanced anymore. In this case, a timeout to break the locks would be helpful. Similarly, the large variance causes the high reorder rate of AHH on the dump trace, as groups of packets are moved among processors.

In the simulation, the service rate was calculated based on the mean packet arrival rate of the trace and the chosen utilization, 0.8 or 0.9. For utilization 0.9, the overall service rate will be $51/0.9=56$ packets/10ms and for one FE in an eight FE system, each FE can process 7 packets/10ms. A burst larger than 40 packets over a 10ms period will often swamp an FE that has a buffer that can hold only $200/8=25$ packets. In summary, the poor results from the edge trace dump.0162 are due to the bursty nature of the traffic and the way system utilization is simulated.

6. CONCLUSIONS

This work demonstrates that a detailed study of Internet traffic characteristics can be beneficial in designing load-balancing algorithms for multiprocessor systems.

The simulation results show that burst scheduling [19] is a promising direction for future research. As gaps between bursts are long enough, flows can be dispersed among the processing units without causing any packet reordering. The ability to react at very short time scale allows burst shifting techniques to be a powerful tool for eliminating processor overload and packet drops.

For some applications, it might be entirely unacceptable for packets from the same flow to be processed by different processors. For that purpose, burst scheduling alone is not an answer, as it incites vast number of re-mappings. Methods based on adaptive hashing [10, 8] that seem to be a good fit for such a problem statement, however, can be rather wasteful in comparison to burst scheduling in buffer size or capacity over-provisioning. Packet drops increase, as the timescales of the adaptive control loop cannot address the short-term packet bursts. However, adaptive hashing addresses well the imbalance caused by the longer-term Zipf-like distribution of packet flow popularity and by uneven packet spreading due to imperfect hash functions.

The major contribution of this work rests in demonstrating that a design combining the two approaches is not only possible, but can achieve the holy grail of exhibiting the good properties of each individual algorithm with respect to the particular performance metric. In combining the methods, the designer must carefully consider the behavior of the triggering and adaptation mechanisms, and the order of the techniques of hashing and burst switching. While the design presented in this work scores well in all the vital performance metrics of packet drop, reordering and remapping, there certainly are more intricate possibilities of combining multiple adaptive load balancing techniques together. Investigation of such mechanisms is in scope of our future research.

7. REFERENCES

- [1] H. Adishesu, G. Parulkar, and G. Varghese. A reliable and scalable striping protocol. In *Proceedings of ACM SIGCOMM*, Stanford University, CA, USA, September 1996.
- [2] J. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, Dec. 1999.
- [3] Z. Cao, Z. Wang, and E. Zegura. Performance of hashing-based schemes for Internet load balancing. In *IEEE INFOCOM 2000*, pages 332–341, Tel-Aviv, Israel, March 2000.
- [4] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988.
- [5] D. E. Comer. *Network Systems Design using Network Processors*. Prentice Hall, 2004.
- [6] G. Dittmann and 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.
- [7] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogenous distributed systems. *IEEE Transactions on Software Engineering*, 12(5):662–675, May 1986.
- [8] R. Haas, P. Droz, L. Kencl, A. Kind, B. Metzler, C. Jeffries, R. Pletka, and M. Waldvogel. Creating advanced functions on network processors: Experience and perspectives. *IEEE Network*, 2003.
- [9] J. Jo, Y. Kim, H. Chao, and 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.
- [10] L. Kencl and J.-Y. L. Boudec. Adaptive load sharing for network processors. In *Proceedings of IEEE Infocom*, New York, 2002.
- [11] M. Laor and L. Gendel. The effect of packet reordering in a backbone link on application throughput. *IEEE Network*, 16(5):28–36, 2002.
- [12] A. Moore, J. Hall, E. Harris, C. Kreibich, and I. Pratt. Architecture of a network monitor. In *Proceedings of the Fourth Passive and Active Measurement (PAM) Workshop*, April 2003.
- [13] NLANR (National Laboratory for Applied Network Research) Measurement and Operations Analysis Team (MOAT). Packet header traces. <http://moat.nlanr.net>.
- [14] K. W. Ross. Hash routing for collections of shared Web caches. *IEEE Network*, 11(7):37–44, Nov-Dec 1997.
- [15] S. Sarvotham, R. Riedi, and 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.
- [16] A. Shaikh, J. Rexford, and K. Shin. Load-sensitive routing of long-lived IP flows. In *Proceedings of ACM SIGCOMM*, pages 215–226, 1999.
- [17] W. Shi, M. H. MacGregor, and P. Gburzynski. Traffic locality characteristics in a parallel forwarding system. *International Journal of Communication Systems*, 16(9):823–839, November 2003.
- [18] W. Shi, M. H. MacGregor, and P. Gburzynski. Load balancing for parallel forwarding. *IEEE/ACM Transactions on Networking*, 2004. to appear.
- [19] W. Shi, M. H. MacGregor, and P. Gburzynski. A scalable load balancer for forwarding Internet traffic: Exploiting flow-level burstiness. In *Symposium on Architectures for Networking and Communications Systems (ANCS)*, Princeton, NJ, USA, October 2005.
- [20] S. Sinha, S. Kandula, and D. Katabi. Harnessing TCP’s burstiness with flowlet switching. In *HotNets 2004*, San Diego, CA, USA, November 2004.
- [21] J. Spirn. *Program Behavior: Models and Measurements*. Elsevier-North Holland, N.Y., 1977.
- [22] D. G. Thaler and C. V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking*, 6(1):1–14, February 1998.
- [23] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.