Load Balancing for Parallel Forwarding

Weiguang Shi, M. H. MacGregor, *Senior Member, IEEE*, and Pawel Gburzynski

***Abstract*—Workload distribution is critical to the performance of network processor based parallel forwarding systems. Scheduling schemes that operate at the packet level, e.g., round-robin, cannot preserve packet-ordering within individual TCP connections. Moreover, these schemes create duplicate information in processor caches and therefore are inefficient in resource utilization. Hashing operates at the flow level and is naturally able to maintain per-connection packet ordering; besides, it does not pollute caches. A pure hash-based system, however, cannot balance processor load in the face of highly skewed flow-size distributions in the Internet; usually, adaptive methods are needed.**

In this paper, based on measurements of Internet traffic, we examine the sources of load imbalance in hash-based scheduling schemes. We prove that under certain Zipf-like flow-size distributions, hashing alone is not able to balance workload. We introduce a new metric to quantify the effects of adaptive load balancing on overall forwarding performance. To achieve both load balancing and efficient system resource utilization, we propose a scheduling scheme that classifies Internet flows into two categories: the aggressive and the normal, and applies different scheduling policies to the two classes of flows. Compared with most state-of-the-art parallel forwarding schemes, our work exploits flow-level Internet traffic characteristics.

***Index Terms*—Load balancing, parallel IP forwarding, Zipf-like distribution.**

I. Introduction

TOGETHER, the continuing Internet bandwidth explosion and the advent of new applications have created great challenges for network forwarding devices, e.g., Internet routers. They have to offer high throughput, computation power, as well as flexibility. One answer to these challenges is network processors (NP) which provide the right balance between performance and flexibility. (In this paper, we use the two terms, *forwarding engine* (FE) and NP, interchangeably.) To achieve high throughput, NPs are optimized for key packet forwarding algorithms and high-speed I/O. More importantly, multiple network processors are employed to forward packets in parallel to achieve scalability.

Although designs from vendors vary, Fig. 1 shows a generalized conceptual model where the forwarding engines are the basic packet processing units. Essential to such a multi-FE system is the *scheduler* that dispatches incoming packets to the FEs. It is necessary for the scheduler to distribute workload in a balanced manner so that the system can achieve its full forwarding potential. In this paper, we divide a scheduler into two functional 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 load distribution to achieve load balance.

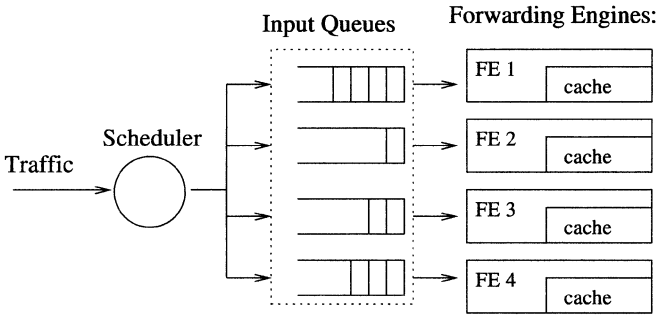


Fig. 1. Multi-processor forwarding system.

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 may be delivered to the FE that is least-loaded. Such schemes schedule workload at the *packet level* and complicate IP forwarding for two reasons. First, reordering of packets within individual TCP connections occurs very frequently in these schemes. Packet reordering within a TCP connection gives TCP false congestion signals and therefore is detrimental to end-to-end system performance [1], [2]. Second, these schemes are not efficient in FE cache utilization [3]: by dispatching packets from the same flow to different FEs, these schemes leave copies of identical data in the caches of the individual FEs.

Hashing is a popular means to distribute load [4]–[9] 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 the packet order of individual TCP connections. Hashing operates at *flow level*. The scheduler typically bases its decision on 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 used 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 FE selected is always 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 being scattered over several FEs, a hashing scheme is efficient in cache utilization [3].

Hashing alone, however, is not able to balance workload under highly variable Internet traffic. Adaptive schemes are needed to accommodate the burstiness and the presence of

Manuscript received June 3, 2003; revised February 20, 2004, May 4, 2004, and August 20, 2004; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Ross.

The authors are with the Department of Computing Science, University of Alberta, Edmonton, AB, T6G 2E8, Canada (e-mail: wgshi@cs.ualberta.ca; macg@cs.ualberta.ca; pawel@cs.ualberta.ca).

Digital Object Identifier 10.1109/TNET.2005.852881

extremely large flows [7], [8]. 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*.

This paper makes the following three contributions.

- First, we prove that due to highly skewed Internet flow size distributions, hashing alone cannot achieve load balance. By characterizing a wide range of IP traces, we localize the sources of load imbalance in a hash-based scheduler.
- Second, we introduce a new metric for *adaptation disruption* to quantify the efficiency of adaptive load balancing schemes. For a system to achieve high forwarding rates, the disruption to FE caches caused by load adaptation should be as small as possible.
- Third, 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 aggressive flows to balance workload among FEs. This design is inspired by IP traffic characterization and the goal to achieve minimum adaptation disruption.

The rest of the paper is organized as follows. In Section II we review studies in flow-level Internet traffic characterization, load-splitting schemes based on highly variable workload distributions in network systems, and hash-based load-splitting schemes in proxy Web cache systems and parallel forwarding systems. In Section III, we present the system model that this study targets and introduce notations used in the paper. Section IV discusses three sources of load imbalance in a hash-based distribution scheme. We prove that generally, hashing alone cannot balance workload given Zipf-like flow-size distributions. We discuss the concept of adaptation disruption and describe the load balancer design in Section V. A critical step in our load balancer is the detection of aggressive Internet flows, which is discussed in Section VI. Section VII presents simulation results for three adaptation policies under varying design parameters. Section VIII concludes this work.

II. Related Work

A system design is hardly sound without taking the characteristics of its workload into consideration. For a parallel forwarding system, it is well known that its workload, the Internet traffic, consists of *elephants* and *mice* [10], [11]: elephants represent the small number of high-volume transmissions that constitute the majority of the traffic mix; mice, on the contrary, are flows that are large in number but consume much less bandwidth. At the connection level, recent measurement studies [12] have found that the burstiness of Internet traffic is solely due to a few aggressive flows dominating the others. These large flows, called *alpha* flows, are the result of large files transmitted over high-bandwidth connections. As a coarser flow aggregation, IP destination address reference patterns have been found [13] to follow Zipf-like [14] distributions:

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

which says that the frequency of some event (P) as a function of its rank (R) often obeys the power-law function with the exponent a close to 1. It is also shown in [13] that the largest flows have a significant impact on load balancing in hash-based scheduling schemes.

The ubiquitous phenomenon of highly biased workload in many Internet systems has motivated a class of schemes which, to achieve performance goals, divide workload into two groups and process them differently. To efficiently transfer diverse traffic, packet switches take advantage of hardware advances and create *short-cuts* for long-lived Internet flows [17]–[19] which represent a large portion of system workload. To improve routing stability and to balance Internet traffic on different links, [20] proposes routing long-lived flows dynamically while forwarding short-lived flows on static, pre-provisioned paths. The idea is to limit load-sensitive routing only to long-lived flows to reduce the frequency of link-state update messages. To balance workload for Web server cluster systems, [21] divides the domains of Web requests into two classes: *hot* and *normal*, and schedules requests from them independently, using two round-robin schemes.

Ref. [5] outlines four design goals of load mapping algorithms in the context of multi-server Web proxy cache systems: low overhead, load balancing, high cache hit rate, and minimum disruption. The authors propose the hash mapping scheme, *highest random weight* (HRW). To map a Web request, HRW takes the combinations of the object name and the identifiers of the proxy servers, e.g., their IP addresses, as keys and returns a list of weight values, one for each server. The server with the largest weight is chosen to serve the request. Since the mapping is hash-based, requests for the same object are usually forwarded to the same server, and therefore cache hit rate is much higher than in replication-based schemes where an object can have multiple copies on the servers (see also [3]). The main strength that distinguishes HRW from other hashing schemes, however, is its ability to achieve fault tolerance with minimum disruption, meaning that only a minimum number of object requests are migrated among the servers during server failures.

HRW is extended to heterogeneous server systems in [22], which leads to the popular cache array routing protocol (CARP). The idea is to assign cache servers with *multipliers* to scale the return values (i.e., the weights) in HRW. A recursive algorithm is developed to calculate the multipliers such that the object requests are divided among the servers according to a pre-defined list of fractions.

Ref. [7] describes a load balancer for parallel forwarding systems. A two-step table-based hashing scheme [6] is used to split traffic. Packet header fields are used as a hash key to a 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 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 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*. (See Section V-A for more discussion.)

Refs. [8] and [9] propose a scheduling algorithm for parallel IP packet forwarding based on HRW [5] and the robust hashing [22]. 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 important goal of the adaptive scheme is to minimize the amount of packet-to-FE re-mappings when balancing workload. 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 [22]) for the FEs to affect load distribution. In other words, the algorithm treats over or under-load conditions as changes of processing power of the FEs. It is proved that the adaptation algorithm can keep the minimal disruption property of HRW.

III. System Model

We consider a parallel forwarding system where m FEs $(\mathrm{FE}_1, \ldots, \mathrm{FE}_m)$ 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 FEs. Logically, the packet is also appended to the input queue of FE_i, Q_i. Since the buffer size is fixed, the length of an input queue is between zero and the buffer size. At any time the limit of a queue's length depends on the number of packets in other queues.

The hash-based load splitter maps the incoming flows onto the individual FEs. 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

$$H(\cdot) : S \rightarrow \{1, 2, \ldots, m\}.$$

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 paper. 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 size distribution is equivalent to that of address popularity distribution. Hereafter, we sometimes use destination addresses to refer to flows and this usage should be clear from the context.

We also measured the flow size distribution (the most important Internet traffic characteristic considered in this paper) where a flow is identified by the five-tuple. The results are similar when the flow identifier is the destination IP address. We therefore believe that the results in this paper apply for other flow definitions.

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 $\lambda (\lambda = \sum_{i=1}^{m} \lambda_i)$ and the mapping scheme H. In this paper, we consider only $\mu_i = \frac{\mu}{m}$, for $1 \leq i \leq m$.

IV. Sources of Load Imbalance

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

A. Hash Function

The mapping scheme F has to be able to generate uniformly distributed random FE identifiers for the source set S. This ensures that, on average, $\frac{|S|}{(m)}$ flows are mapped to each FE. Although for a nonrandom 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 [23]. Refs. [4]–[6] have found that the Internet checksum algorithm and the CRC over the five-tuple {DA, SA, DP, SP, PN} give good random outputs.

B. Burstiness of Internet Traffic

Packet network flows are known to be *bursty*, i.e., packets of a flow travel in groups [24]. 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 FEs may be idling. The bursty nature of Internet traffic can lead to temporary load imbalance and cause packet loss. Aside from adjusting flow mappings adaptively, buffering and provisioning are the common practices to accommodate bursty packet arrivals.

C. Skewed Flow Size Distribution

1) Flow-Level Internet Traffic Characteristics: 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 results for five traces (see Table I). The address popularity distributions in these traces are shown in Fig. 2. Although their scales differ, each curve can be matched by a straight line, i.e., a Zipf-like function, in the log-log plot. The slopes fitted 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 II shows the the number of packets in the ten most popular flows of each trace. This common phenomenon is the motivation of the load balancing scheme developed in this paper.

2) Implications for Load Balancing: The flow size distribution adds another dimension to the load balancing problem. In [6], 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 [7].

While hashing may manage to balance workload in the average sense when the flow size distribution is homogeneous, i.e., with a finite variance, as proved for HRW in [5], it cannot

TABLE I
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 [15] 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) [16]. 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.

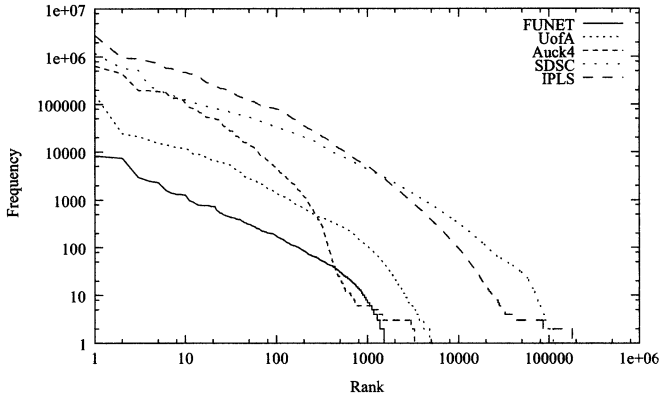


Fig. 2. IP address popularity distribution follows Zipf's law.

TABLE II
NUMBER OF PACKETS OF 10 LARGEST FLOWS IN THE TRACES

R	FUNET	UofA	Auck4	SDSC	IPLS
1	8233	158707	640730	1183834	2788273
2	7424	24245	440149	581495	944253
3	2971	20769	196513	524542	919088
4	2470	17482	194757	235363	808773
5	2298	15146	186095	212150	732339
6	1614	14305	177388	168384	582367
7	1387	13308	135286	160798	570316
8	1317	12348	135033	138657	510043
9	1309	12028	132812	125531	473562
10	1258	11824	104716	125389	470072

when the distribution is so skewed that the coefficient of variation (CV) is infinite.

Let m be the number of identical FEs in a parallel forwarding system and 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 [5] that HRW, or any hash function that generates uniformly distributed random numbers over its hash key space, distributes workload in a balanced way. This occurs when the the load imbalance of the system, expressed as the CV of q_j:

$$\mathrm{CV}[q_j]^2 = \left(\frac{m-1}{K-1}\right)\mathrm{CV}[p_i]^2 \tag{2}$$

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

The discrete-form probability density function (PDF) of a Zipf-like distribution (1) is

$$P(X=i) = \frac{1}{Z} i^{-\alpha}, \quad i = 1, 2, \ldots, K, \quad \alpha > 1 \tag{3}$$

where Z is a normalizing constant:

$$Z = \sum_{i=1}^{K} i^{-\alpha}. \tag{4}$$

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

$$\begin{aligned}\mathrm{CV}[p_i]^2 &= \frac{\mathrm{Var}(p_i)}{E[p_i]^2} \\ &= \frac{E[p_i^2] - E[p_i]^2}{E[p_i]^2} \\ &= \frac{\frac{1}{K}\sum_{i=1}^{K}\frac{1}{Z^2} i^{-2\alpha}}{\frac{1}{K^2}} - 1 \\ &= \frac{K}{Z^2}\sum_{i=1}^{K} i^{-2\alpha} - 1.\end{aligned} \tag{5}$$

Substituting the $\mathrm{CV}[p_i]^2$ in (2), we have

$$\mathrm{CV}[q_j]^2 \sim \frac{K(m-1)}{Z^2(K-1)}\sum_{i=1}^{K} i^{-2\alpha}. \tag{6}$$

As $\alpha > 1$ and $K \to \infty$, items Z and $\sum_{i=1}^{K} i^{-2\alpha}$ converge, and thus $\mathrm{CV}[q_j]^2$ is nonzero.

Zipf-like distributions (1) 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 [5], 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 with $\alpha > 1$.

V. Load Balancer

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

A. Goals

The goals of load-splitting algorithms [5] for Web proxy cache systems apply for the packet schedulers in parallel forwarding systems. First, the scheduler shown in Fig. 1 sits 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 proven in Section IV, hashing alone cannot achieve load balancing; it is therefore necessary for the scheduler to monitor the workload on the FEs and perform adjustment at appropriate times. Third, since each FE usually has its own local fast storage functioning as cache, higher hit rate is desirable. FE cache hit rate is mainly determined by temporal locality in IP traffic. Scheduling schemes have a big impact on temporal locality seen at each FE [3]. Finally, the system has to be fault-tolerant to provide reliability and graceful degradation when one or more FEs 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 [7], [8]. The result is that some flows will be shifted from the most loaded processors to less loaded ones.

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 FEs cache. We call this phenomenon *adaptation disruption*. Obviously, flow migration is harmful to forwarding performance and should be done as infrequently as possible. Thus, in a hash-based parallel forwarding system, another feature is desirable; we call it minimum adaptation disruption (MAD). For N_P packets forwarded, adaptation disruption, denoted by ζ, is quantified as follows:

$$\zeta = \frac{N_S}{N_P} \tag{7}$$

where N_S is the number of flow-shifts. Thus, $0 \leq \zeta \leq 1$.

Note that MAD 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 FEs. Adaptation disruption, on the other hand, is caused by flow migrations among FEs 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. The parameter ζ is introduced to relate cache performance to the frequency of flow-shifts.

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

A goal of the load sharing scheme in [8] is to minimize flow re-mapping and thus to minimize packet reordering. FE states, e.g., cache, are not taken into account. Ref. [8] extends the work on HRW in [5] and [22]. Although nonuniform object popularity is realized as a potential reason for load imbalance, this aspect of the workload is not characterized or explored in [8] but is listed as future work.

Ref. [8] uses the fraction of flow remappings over all flows existing during a time interval as a measure of disruption. This is different from the concept of adaptation disruption introduced by (7).

The denominator in (7), N_P, is the number of *packets* instead of the number of *flows* forwarded. The former corresponds to the router performance metric, i.e., packet-per-second (pps), not flow-per-second. This is essential because flows vary in size and therefore cannot be used to measure throughput.

The numerator, N_S in (7), is defined as "the number of flow-shifts" instead of "the number of flows remapped" as in [8, Table I]. N_S is incremented by one each time a flow is shifted, or remapped, from one FE to another; it does not matter if this flow has been in the set of flows shifted hitherto. As each flow-shift can result in a cold-start of the cache in the target FE, (7) represents the upper-bound of the cache disturbance caused by shifting flows. On the other hand, "the number of flows remapped" in [8, Table I] should be understood as the size of the set of flows remapped and therefore does not reflect cache disturbance.

It is worth noting that the *packet spraying* in [7] is proposed to deal with "rare" "emergency" situations when an excessive flow bundle by itself exceeds the processing power of one FE. The scheme does not *actively* spray to achieve load balance. In addition, both flow reassignment and packet spraying in [7] operate on bundles instead of individual flows. A bundle by definition contains more than one flow. The larger the number of flows shifted, the more disruption is caused to the states of the FEs. It is possible that shifting one bundle causes a large portion of the target FE cache to be flushed. In contrast, our goal is specifically to balance load with minimum disruption. We achieve this goal by identifying and shifting only aggressive flows.

B. Design

Most state-of-the-art schedulers migrate flows without considering their rates, but this is ineffective. As indicated by our measurements in Section IV, the probability of shifting low-rate flows should be high since there are many of them.

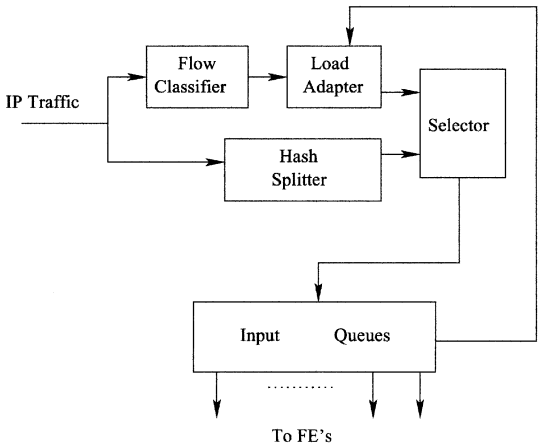


Fig. 3. Load balancing packet scheduler.

Shifting these flows does not help re-balance the system much, but 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 [13]. While the scheduler is busy shifting low-rate flows, the high-rate ones keep swamping the overloaded processor(s).

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

Fig. 3 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 FEs. Alternatively, HRW could be used to minimize disruption in the case of FE failures.

The load adapter becomes active when the system is unbalanced. It 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 it to the FE with the shortest queue. 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, which is decided by the triggering policy (see Section V-C). 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 FEs decided by the hash splitter. Later even if, as a result of the adaptation, the system becomes balanced and the balancer is inactivated, the flows previously identified in the flow table are still directed to the FE assigned by the balancer. This is to prevent unnecessary flow migration.

An important design parameter is F, the size of the balancer's flow table. Generally, shifting more aggressive 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. 3). The flow classifier monitors the incoming traffic to decide which flows are aggressive and should be put in the balancer's flow table. We discuss in detail the aggressive flow identification procedure in Section VI.

C. Triggering Policies

The adapter implements the scheduling scheme that decides *when* to remap flows (the triggering policy), *what* 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 FEs by the hash scheduler. There are multiple choices for deciding when the system is unbalanced and 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 workload of the FEs. 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 VII.

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 FE with the shortest queue.

VI. Detecting Aggressive Flows

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

A. Definition of Aggressive Flows

We define aggressive flows as high-rate flows. Flows that are both large and fast 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 [12]. 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 aggressive flows.

It is pointed out in [11] 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 [19] for long-lived flows cannot be used to detect aggressive 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.

B. Detecting Aggressive 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 to 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* [24] 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

$$\Delta = \frac{\sum_{i=1}^{n} \delta_i / F}{n} \tag{8}$$

where

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

and N_P is the number of packets forwarded during the measurement. Thus, $0 \leq \Delta \leq 1$. The larger the value of Δ, the better flow information collected in the current window predicts aggressive 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. 4 shows the effects of W on Δ for the first one million entries of the four larger traces in Table I with $F = 5$. The larger the value of W, the better the current aggressive 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 an entire trace is almost always identified as the largest flow of every window (the smallest W experimented with is 100). We will see that shifting even only the one largest flow is very effective in balancing workload.

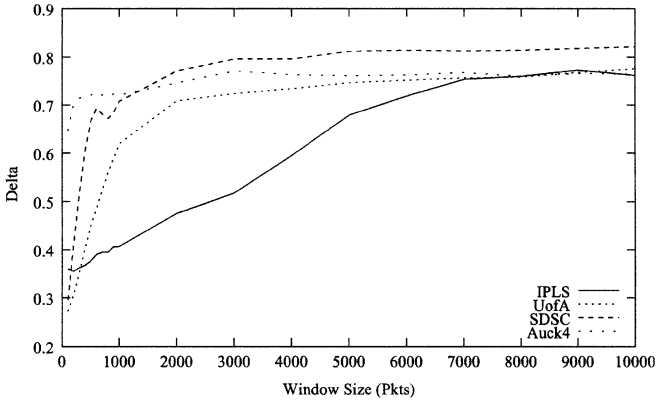


Fig. 4. Effects of W on $\Delta(F = 5)$.

TABLE III
ARRIVAL RATES (NUMBER OF PACKETS/SECOND) OF FOUR TRACES

IPLS	UofA	SDSC	Auck4
74608.742	14007.337	3210.378	251.394

VII. 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 FEs 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.

A. Trace Driven Simulation

The average packet arrival rates (λ) are measured for the four larger traces (Table III[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 FEs given some system utilization ρ

$$\mu_i = \frac{\lambda/m}{\rho}, \quad i = 1, \ldots, m. \tag{9}$$

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 FEs (m), the number of aggressive 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 $W = 1000$.

B. Hash Splitter

The hash splitter implements the following operation:

$$\mathrm{CRC16}(\mathrm{DestIPAddress})\%m$$

where CRC16 is the 16-bit cyclic redundancy check, % is the modulo operator, and m is the number of FEs. According to previous studies of hash function performance [4], [6], the CRC is a very good hash function.

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

C. Triggering Policies

We tested three triggering policies:

- *Periodic Mapping* (PM): The adapter schedules aggressive 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.
- *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.

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 aggressive flows are shifted from one FE to another and the amount of adaptation disruption for BOT and MQLT.

D. 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 aggressive 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 $\simeq N_S * F$.

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 FEs 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

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

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

E. Packet Reordering and Loss

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 are also the sources of potential packet reordering. Shifting a few aggressive 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 and $T'_{i,j}$ the time that it is observed at the output port. 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

$$r(i,j) = \begin{cases} 1, & \text{if } T'_{i,j} > T'_{i,j+1} \\ 0, & \text{otherwise} \end{cases}$$

then the packet reordering rate R_r for N_P packets forwarded is

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

In our simulations, there are two reasons for packet loss. First, the load may not be properly balanced among the FEs. The service capability of the whole system is adequate, but while some FEs are busy forwarding, other FEs can be idling. Therefore, the system is not utilized at its full potential. Over time, the number of packets in the busy FE's queues increases to the limit of the buffer size and newly arriving packets are dropped. The second reason for packet loss has little to do with scheduling schemes: the service rate of each FE is calculated based on the *average* arrival rate of packets; during traffic bursts, packet arrival rates can be much more than the system can handle.

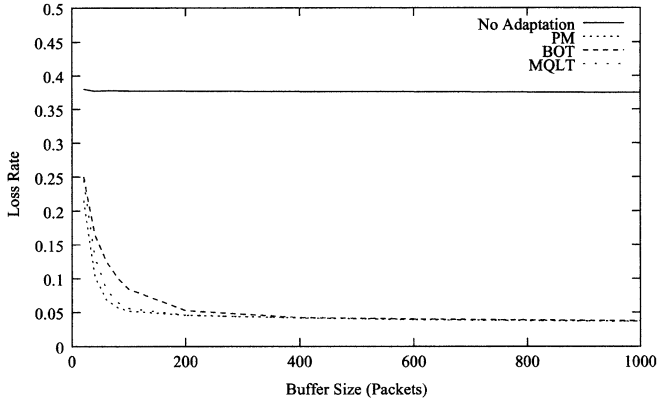


Fig. 5. Loss Rate vs Buffer Size (For PM, BOT, and MQLT, the system load condition checking is done every 20 packets. For BOT, the threshold is 80% of the buffer size. For the MQLT, the threshold is 30% of the buffer size. There are eight FEs and the system utilization $\rho = 0.8$. For this simulation, the number of aggressive flows in the flow table is 1.

F. Simulation Results

Figs. 5 and 6 show packet loss rates of different adaptation policies under varying buffer sizes for the UofA and the IPLS traces. For both traces, the hash-only scheme (no adaptation) has the highest loss rate and, moreover, increasing buffer size does not help. On the other hand, the three adaptation schemes all respond positively to buffer increases. PM achieves the best loss rates compared to BOT and MQLT.

Fig. 7 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 the

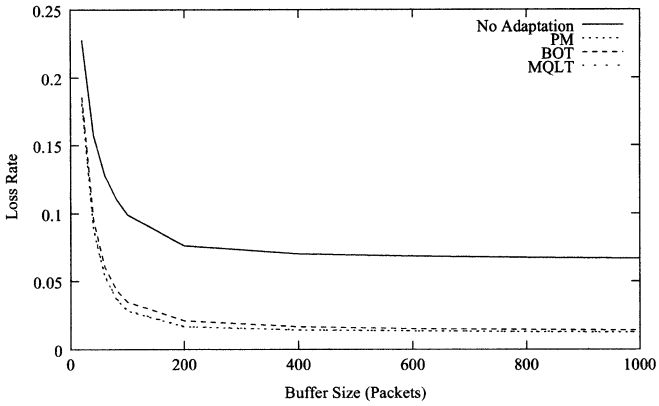


Fig. 6. Loss rate versus buffer size (with the IPLS trace).

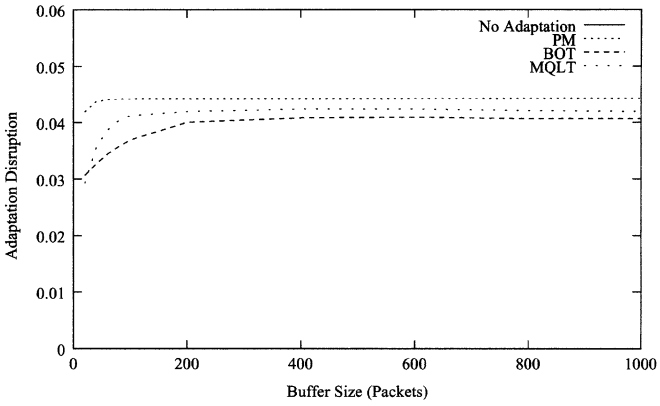


Fig. 7. Adaptation disruption versus buffer size (the same setting as Fig. 5).

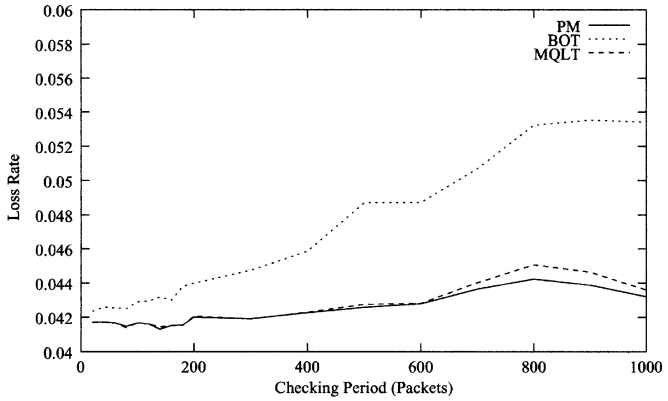


Fig. 8. Loss rate versus checking period. (The buffer size is 400 packets. The other parameters are the same as those of Fig. 5.)

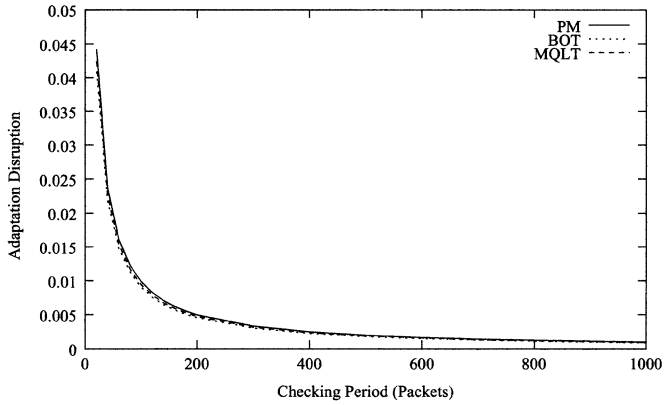


Fig. 9. Adaptation disruption versus checking period (the same setting as Fig. 8).

best loss rate: it re-maps the aggressive flow much more frequently than BOT and MQLT. The difference in adaptation disruption between MQLT and BOT is small; it seems that MQLT achieves better loss rates (Fig. 5) 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 lower packet loss 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. 8 and 9 show how the checking interval affects loss rate and adaptation disruption. Generally, the decrease in responsiveness to load imbalance leads to more packet loss. Fig. 8 shows that compared with PM and MQLT, BOT (with 80% occupancy threshold value) is more susceptible to checking period increases. Fig. 9 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 and IPLS traces. Differences in scale are caused by the peculiarities of the largest flows in the individual traces. For example, as shown in Table II, 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 more adaptation disruption.

In the following simulations, we experiment with the Auck4 trace to study the effect of scheduling a larger number of aggressive flows on packet loss rate, adaptation disruption, and packet reordering. The results are shown in Figs. 10–12. In each figure, the x axis denotes the number of most aggressive flows. That is, $x = 1$ represents the case when only the most aggressive flow in the trace is used in load balancing; $x = 2$ means the largest two flows are scheduled, and so on.

Fig. 10 shows the effectiveness of scheduling more aggressive flows in reducing loss rates for the Auck4 trace for the three adaptive policies. It seems that for a given configuration, beyond a certain number of aggressive flows, the benefit of scheduling more flows becomes negligible. On the other hand, as shown in Fig. 11, 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. 12 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. Among the three triggering policies, BOT performs best.

To further illustrate the advantages of shifting the most aggressive flows, we compare the results of two simulations:

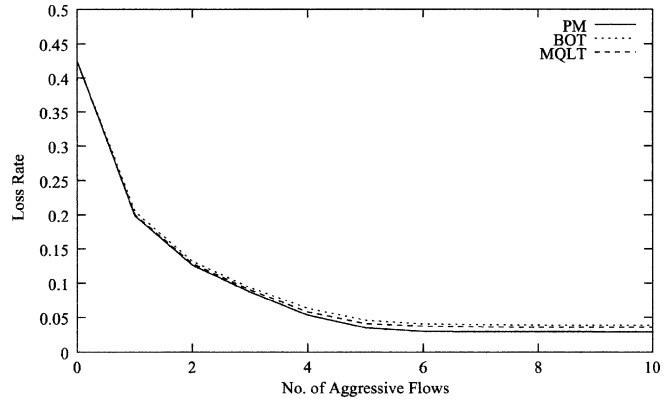


Fig. 10. The effectiveness of scheduling more aggressive flows. (The checking period is 20 and the buffer size is 400. The other parameters are the same as those in Fig. 5.)

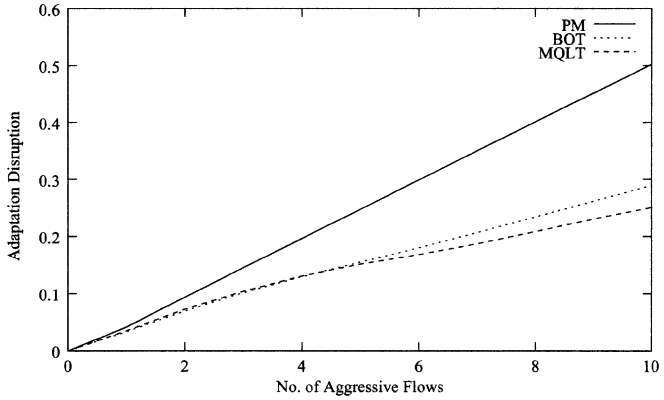


Fig. 11. The effects of scheduling more flows on adaptation disruption (with the same setting as Fig. 10).

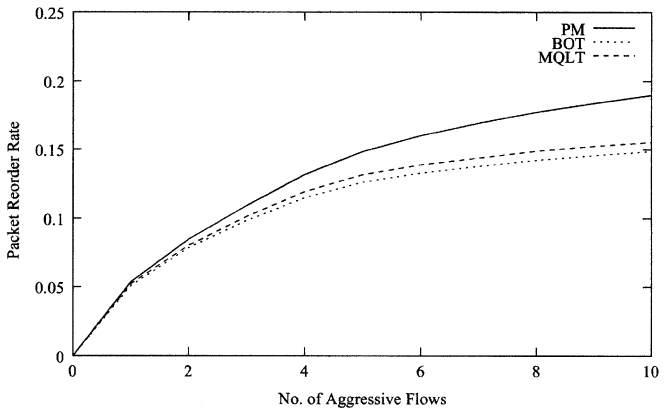


Fig. 12. The effects of scheduling more flows on packet reordering (with the same setting as Fig. 10).

scheduling only the most aggressive flow (MAF) and scheduling only a number of less aggressive flows (LAF) to achieve similar loss rates as with shifting MAF. In the simulations, the MAF is the largest flow identified in the flow table by the aggressive flow detection mechanism described in Section VI. The LAFs are the second largest, the third largest, etc., in the same flow table. We simulate the PM policy with a 20-packet checking period.

TABLE IV
COMPARISON BETWEEN SHIFTING ONLY THE MOST AGGRESSIVE FLOW AND SHIFTING ONLY LESS AGGRESSIVE ONES

Simulation	Auck4 MAF	Auck4 LAF	IPLS MAF	IPLS LAF
No. of Flows	1	14	1	3
η	.176	.177	.0260	.0218
ζ	.0417	.688	.0364	.110
Rr	.0581	.111	.00958	.0888
$CV[q_j]$	.213	.250	.114	.127

Simulation	SDSC MAF	SDSC LAF	UofA MAF	UofA LAF
No. of Flows	1	2	1	500
η	.0277	.0239	0.0367	.111
ζ	.0345	.0716	.0443	25.2
Rr	.00615	.00764	.0336	.0503
$CV[q_j]$	.0907	.0776	.167	.285

Note that the simulations for each trace are designed to achieve similar loss rates. If system throughput can be expressed as forwarding rate, the throughput achieved by the two scheduling strategies is similar, too. What we want to show are the differences in the CV, the adaptation disruption, and the reordering rate under the two schemes for each trace.

Table IV summarizes the results for four traces. With similar packet loss rates (η), MAF scheduling always causes less adaptation disruption (ζ) and packet reorders (R_r). For the Auck4, IPLS, UofA traces, MAF scheduling also balances load better, as shown by the smaller CV. More than one LAF is always needed to achieve similar packet loss rates as MAF scheduling. The least number of LAFs needed is two, as in the SDSC case where scheduling LAFs achieves a lower miss rate and $\mathrm{CV}[q_i]$. One reason might be that in the SDSC trace, the MAF identified by the mechanism in Section VI only accounts for a small portion of the total traffic, not significant enough for the MAF scheduling strategy to outperform LAF scheduling by a large margin. The other extreme is the UofA trace, where the MAF by itself represents around 16% of the aggregate traffic; when it is scheduled onto an FE, even if the rest of the traffic is spread evenly among the other seven FEs (each 12%), the system is still not perfectly balanced.

It is important to note that the arrival rate λ for the Auck4 trace (see Table III) used to decide the FE service rates (9) in the simulations of Figs. 10–12 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 loss rates, adaptation disruption, and packet reordering rates, for different traces.

VIII. CONCLUSION

The highly skewed Internet flow size distribution has profound implications for Internet forwarding system design. First,

we have proved in this paper that the Zipf-like flow popularity distribution, which has infinite mean and variance, is a major source 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 quantifies 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 aggressive 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 FEs to less loaded ones, our scheduler shifts only a few aggressive 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 only these flows has the potential to cause little adaptation disruption to the FE's cache. We expect much higher disruption in adaptive load balancing schemes that do not take flow size distribution into account. Experiments show that due to temporal locality in Internet traffic, the aggressive flows can be readily identified, which indicates that the proposed load balancer is highly feasible.

Highly skewed popularity distributions exist in 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 rate and to reduce response time. It is pointed out in [5], however, that requests for a 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 servers 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 paper could then be used to balance the load. For such systems, a centralized scheduling mechanism is essential.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments and constructive criticism.

REFERENCES

[1] J. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, pp. 789–798, Dec. 1999.

[2] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM Comput. Commun. Rev.*, vol. 32, no. 1, pp. 20–30, Jan. 2002.

[3] W. Shi, M. H. MacGregor, and P. Gburzynski, "Effects of a hash-based scheduler on cache performance in a parallel forwarding system," in *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conf. (CNDS'2003)*, Orlando, FL, Jan. 2003, pp. 130–138.

[4] R. Jain, "A comparison of hashing schemes for address lookup in computer networks," *IEEE Trans. Commun.*, vol. 40, no. 3, pp. 1570–1573, Oct. 1992.

[5] D. G. Thaler and C. V. Ravishankar, "Using name-based mappings to increase hit rates," *IEEE/ACM Trans. Netw.*, vol. 6, no. 1, pp. 1–14, Feb. 1998.

[6] Z. Cao, Z. Wang, and E. Zegura, "Performance of hashing-based schemes for Internet load balancing," in *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000, pp. 332–341.

[7] G. Dittmann and A. Herkersdorf, "Network processor load balancing for high-speed links," in *Proc. Int. Symp. Performance Evaluation of Computer and Telecommunication Systems (SPECTS'2002)*, San Diego, CA, Jul. 2002, pp. 727–735.

[8] L. Kencl and J. L. Boudec, "Adaptive load sharing for network processors," in *Proc. IEEE INFOCOM*, New York, Jun. 2002, pp. 545–554.

[9] L. Kencl, "Load Sharing for Multiprocessor Network Nodes," Ph.D. dissertation, Swiss Federal Institute of Technology (EPFL), Jan. 2003.

[10] L. Guo and I. Matta, "The war between mice and elephants," in *Proc. 9th IEEE Int. Conf. Network Protocols (ICNP)*, Riverside, CA, 2001, pp. 180–188.

[11] N. Brownlee and K. Claffy, "Understanding Internet traffic streams: Dragonflies and tortoises," *IEEE Commun.*, vol. 40, no. 10, pp. 110–117, Oct. 2002.

[12] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," in *Proc. ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, Nov. 2001, pp. 99–103.

[13] W. Shi, M. H. MacGregor, and P. Gburzynski, "Synthetic trace generation for the Internet: An integrated model," in *Proc. SPECTS'04*, San Jose, CA, Jul. 2004, pp. 471–477.

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

[15] S. Nilsson and G. Karlsson, "IP-address lookup using LC-tries," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 6, pp. 1083–1092, Jun. 1997.

[16] Packet Header Traces. NLANR (National Laboratory for Applied Network Research) Measurement and Operations Analysis Team (MOAT). [Online]. Available: http://moat.nlanr.net

[17] P. Newman, G. Minshall, and L. Huston, "IP switching and gigabit routers," *IEEE Commun. Mag.*, vol. 35, no. 1, pp. 64–68, Jan. 1997.

[18] Y. Rekhter, B. Davie, E. Rosen, G. Swallow, D. Farinacci, and D. Datz, "Tag switching architecture overview," *Proc. IEEE*, vol. 85, no. 12, pp. 1973–1983, Dec. 1997.

[19] A. Feldmann, J. Rexford, and R. Cáceres, "Efficient policies for carrying web traffic over flow-switched networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, pp. 673–685, Dec. 1998.

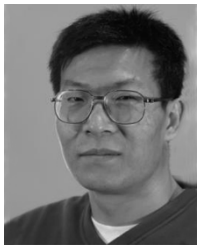
[20] A. Shaikh, J. Rexford, and K. G. Shin, "Load-sensitive routing of long-lived IP flows," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, Oct. 1999, pp. 215–226.

[21] M. Colajanni, P. S. Yu, and D. M. Dias, "Analysis of task assignment policies in scalable distributed Web-server systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 6, pp. 585–600, Jun. 1998.

[22] K. W. Ross, "Hash routing for collections of shared web caches," *IEEE Network*, vol. 11, no. 7, pp. 37–44, Nov.–Dec. 1997.

[23] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, 1st ed. Reading, MA: Addison-Wesley, 1969, vol. 3.

[24] R. Jain and S. Routhier, "Packet trains: Measurements and a new model for computer network traffic," *IEEE J. Sel. Areas Commun.*, vol. SAC-4, no. 6, pp. 986–995, Sep. 1986.



Weiguang Shi received the B.Eng. degree in system engineering in 1995 and the M.Sc. degree in computer science in 1998, both from Beijing University of Aeronautics and Astronautics, Beijing, China. He received the Ph.D. degree in computing science from the University of Alberta, Alberta, Canada, in 2003.

He worked at Sprint Advanced Technology Lab in 2000 as an intern. He is now a Research Assistant in the Electrical and Computer Engineering Department, University of Alberta. His research interests include workload modeling and performance evaluation of Internet forwarding systems, network QoS, and network performance issues in host systems.

M. H. MacGregor (SM'05) received the M.Sc. degree in process control from the University of Alberta, Alberta, Canada, in 1984 for his work in minimum variance modeling and control, and the Ph.D. degree in computing science from the University of Alberta in 1991 for his dissertation on self-traffic-engineering networks, the research for which was conducted at TRLabs.

He is an Associate Professor in the Department of Computing Science at the University of Alberta. His current research interests are in the areas of router and switch architectures, and the design and optimization of optical networks. He has over ten years of industrial experience in the areas of real-time process control, broadband Internet service architectures, and optical switching

Dr. MacGregor has been a Registered Professional Engineer in the Province of Alberta since 1980. He is a member of APEGGA, ACM, and IEICE.

Pawel Gburzynski received the M.Sc. and Ph.D. degrees in computer science from the University of Warsaw, Warsaw, Poland, in 1976 and 1982, respectively.

Before coming to Canada in 1984, he had been a research associate, systems programmer, and consultant in the Department of Mathematics, Informatics, and Mechanics at the University of Warsaw. Since 1985, he has been with the Department of Computing Science, University of Alberta, Alberta, Canada, where he is a Professor. His research interests are in communication networks, operating systems, simulation, and performance evaluation.