An Adaptive Load Balancer for Multiprocessor Routers

W. Shi, M. H. MacGregor and P. Gburzynski
Department of Computing Science,
University of Alberta,
Edmonton, AB, T6G 2E8, Canada
{wgshi, macg, pawel}@cs.ualberta.ca
Phone: (780)492-1930, fax: (780)492-1071

Keywords: Workload Modeling, Parallel Forwarding, Hashing, Load Balancing

Abstract
We develop a novel load-balancing packet scheduler for parallel forwarding systems. By investigating flow level characteristics of Internet traffic, we are able to trace the root of load imbalance in hash-based load-splitting schemes. Our scheduler capitalizes on the non-uniform flow reference pattern and especially the presence of a few high-rate flows in typical Internet traffic mix. We show that detecting and scheduling these flows can be very effective in balancing workloads among network processors. Moreover, since the number of these flows is small, reassigning them to different processors makes the least disruption to the states of individual processors. Our ideas are validated by simulation results.

INTRODUCTION

Together, the continuing Internet bandwidth explosion and the advent of new applications have created challenges for Internet routers. They have to offer high throughput, computation power, and flexibility. One answer to these challenges is network processors (NP) [1] which provide a balance between performance and flexibility.

To achieve high throughput, NP's are optimized for key packet forwarding algorithms and high-speed I/O. More importantly, multiple network processors can be 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 (FE's) are the basic packet processing units.

Essential to multi-NP system performance is the scheduler that dispatches packets to the FE's. It is necessary for the scheduler to distribute workloads in a load-balanced manner such that the system can achieve its full forwarding potential.

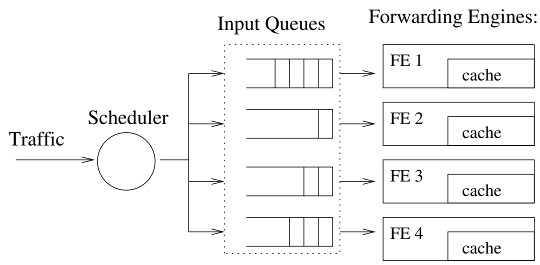

Figure 1: A Multi-processor Forwarding System

Hashing is a popular packet dispatching scheme [2, 3] where for each packet, one or more header fields are selected as input to a hash function. The return value is used to select the FE to deliver the packet to. Hashing is efficient, improves locality, and can preserve packet order within TCP connections [4]. The major disadvantage of simple hashing is that it does not balance workloads. This is caused by the skewed distribution of the sizes of flows, and especially the presence of a few high-rate flows.

In the context of IP forwarding, we define a *flow* as a sequence of packets with the same destination IP address. This represents a coarser aggregation of network traffic than used by other popular flow definitions such as the five-tuple of source and destination IP addresses, source and destination transport layer port numbers, and transport layer protocol identifier. Our definition is targeted at the packet forwarding process where IP destination address lookup [5, 7–9] is the bottleneck. Our approach is extensible to other flow definitions and to more general load balancing problems in other contexts,

Table 1: Traces Used in Experiments

Trace	Length(entries)	Description
FUNET	100,000	A destination address trace which is used in evaluating the LC-trie routing table lookup algorithm in [5] 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) [6]. 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.

e.g., Web server systems.

The rest of the paper is organized as follows. We first discuss relevant studies on Internet workload characterization and parallel forwarding system load balancing. Next, we show the flow level characteristics of Internet traffic and explain their implications for load balancing. We then describe a load balancing scheduler design and verify the scheme by simulation. Finally, we summarize the work.

RELATED WORK

Load Balancing for WWW Caches

Load balancing is important to the performance of Web sites and Web caches that use multiple servers to serve user requests. For Web cache systems, [10] proposes *highest random weight* (HRW) to achieve high Web cache hit rate, load balancing, and *minimum disruption* in the face of server failure or reconfiguration. HRW is a hash-based scheme. To request a Web object, the object's name and the identifiers of cache servers, e.g., IP addresses, are used as keys in 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 object requests that were mapped to this server are migrated to other servers; the other requests are not affected. This achieves minimum disruption.

HRW is extended to accommodate heterogeneous server systems in [11], which leads to the popular cache array routing protocol (CARP). The idea is to assign *multipliers* to cache servers to scale the return values of HRW. A recursive algorithm is provided to calculate the multipliers such that the object requests are divided among the servers according to a pre-defined fraction list.

Internet Workload Characterization

Ref. [12] examines Internet traffic at the connection level. It is found that the burstiness of Internet traffic is *not* due to a large number of active flows being active at the same time, as assumed in some Internet traffic models [13], but rather is caused by a few large files transmitted over high-bandwidth links. These connections contribute to *alpha* traffic and the rest create *beta* traffic.

Ref. [14] studies the flow patterns of measured Internet traffic, and points out that network streams can be classified by both size (*elephants* and *mice*) and lifetime (*tortoises* and *dragonflies*). Tortoises are flows lasting more than 15 minutes, which contribute to a small portion of the number of flows (one or two percent), but carry fifty to sixty percent of the total traffic.

Load Balancing for Parallel Forwarding Systems

Ref. [2] describes a load balancer for parallel forwarding systems. A two-step table-based hashing scheme is used to split traffic. Packet header fields that uniquely identify a flow are used as a hash key and fed to a hash function. The return value is used as an index to a look-up memory to derive the processor to which the packet should be forwarded. Flows that yield the same value are called a *flow bundle* and are associated with one processor.

Three techniques are proposed to achieve load balancing. First, a *time stamp* is kept and updated at every packet arrival for each flow bundle. Before the update, the 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 pro-

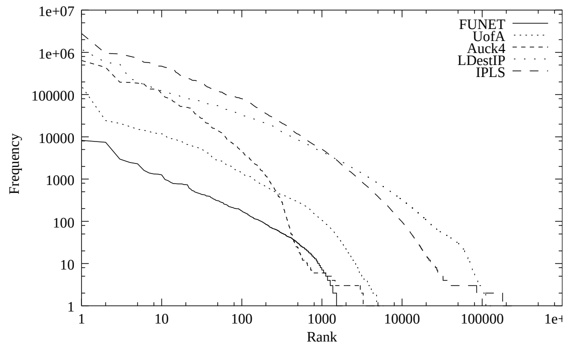

Figure 2: IP Address Popularity Distribution Follows Zipf's Law

cessor 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 *packet spraying*.

Refs. [3] proposes a scheduling algorithm for parallel IP packet forwarding. Their scheme is based on HRW [10] [11]. Although HRW provides load balancing over the request object space, load imbalances still occur due to uneven popularities of the individual objects.

An adaptive scheduling scheme is introduced to cope with this problem. It 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 [11]) for the FE's to affect load distribution. In other words, the algorithm treats over or under-load conditions as changes of processing power of the FE's. It is proved that the adaptation algorithm keeps the minimal disruption property of HRW.

FLOW LEVEL INTERNET TRAFFIC CHARACTERISTICS

Zipf-like Distribution of IP Address Popularity

To study flow level Internet traffic characteristics, we have experimented with traces collected from networks ranging from campus to major Internet backbones (see Table 1). The address popularity distributions in these traces are shown in Fig. 2. Although their scales differ, each curve can be approximated by a straight line on a log-log plot. This is a Zipf-like function [15],

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

where $a \approx 1$. To get a proper fit we bin the data into exponentially wider bins [16] so that they appear evenly spaced on a log scale as shown in Fig. 3. 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 2 shows the number of packets in the ten most popular flows of each trace. This data motivates the load balancing scheme developed in this paper.

Implications for Load Balancing

Let m be the number of identical FE's in a parallel forwarding system and let K be the number of distinctive addresses. 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 addresses distributed to FE j.

Any hash function that generates uniformly distributed random numbers over its hash key space is said to distribute workloads in a balanced way only when the the load imbalance of the system, expressed as the coefficient of variation (CV) of q_j:

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

approaches zero as K and the number of packets approach infinity.

However, because of the Zipf-like distribution of flows, no hashing function exists that can produce this result. That is, no hash-based scheduler can balance a Zipf-like load. We prove this as follows.

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

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

where Z is a normalizing constant:

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

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

$$CV[p_i]^2 \quad = \quad \frac{Var(p_i)}{E[p_i]^2} \tag{5}$$

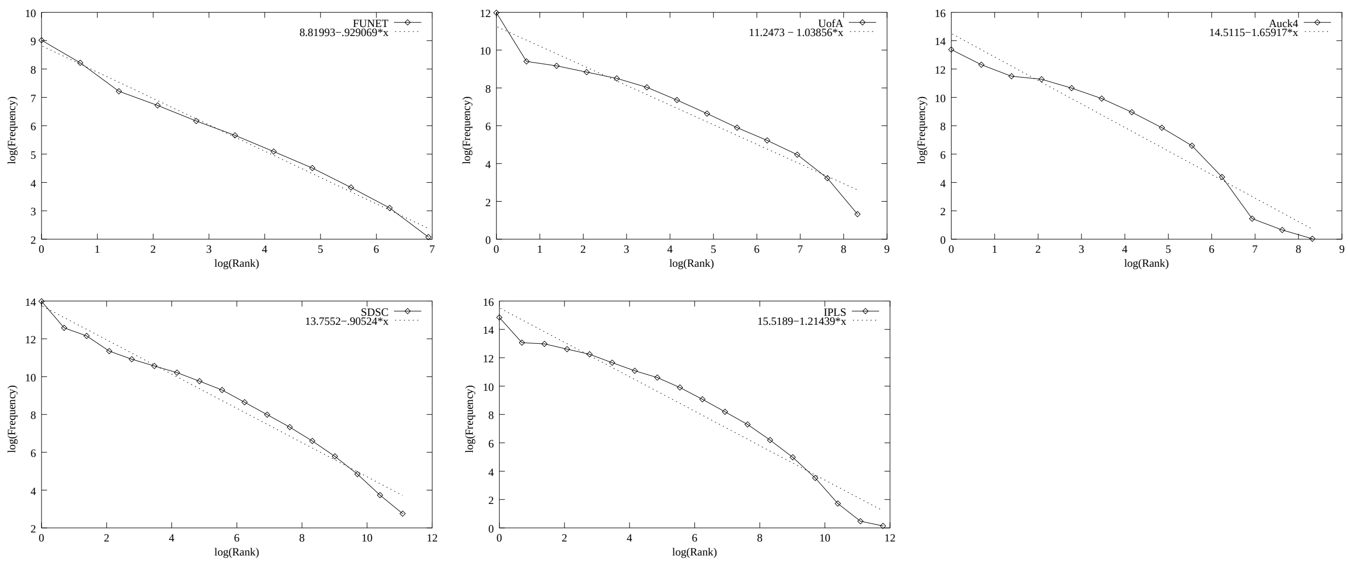

Figure 3: Fitting IP Address Popularity Distributions with Zipf-like Functions

$$
\begin{aligned}
&= \frac{E[p_i^2] - E[p_i]^2}{E[p_i]^2} \\
&= \frac{\frac{1}{K}\sum_{i=1}^{K}\frac{1}{Z^2}i^{-2\alpha}}{\frac{1}{K^2}} - 1 \\
&= \frac{K}{Z^2}\sum_{i=1}^{K} i^{-2\alpha} - 1
\end{aligned}
$$

Substituting $CV[p_i]^2$ from Eq. 5 in Eq. 2, we have

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

As $\alpha > 1$ and $K \to \infty$, items Z and $\sum_{i=1}^{K} i^{-2\alpha}$ converge, and thus $CV[q_j]^2$ is *non-zero*. This is the proof that a hash based scheme, such as HRW [10], is not able to achieve load balancing in parallel forwarding systems when the population distribution of objects in its input space is Zipf-like.

Table 2: No. 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

Adaptive Load Balancing

Typically, when a system is unbalanced to some degree, an adaptive mechanism in the scheduler would be invoked to manage the mapping from the system's input space to its output space [2] [3]. The result is that some flows will be shifted from the most loaded (source) FE's to less loaded (target) ones. To some extent, the benefit of shifting flows is offset by its costs. For example, for target FE's, the shifted flows usually cause cache cold start and for source FE's, flow emigration renders established cache entries useless. For these reasons, during re-mappings, it is desirable that the number of flows shifted is small to achieve minimum *adaptation disruption.*

It is worth noting that the disruption here is not the same as that in HRW. The latter tries to migrate as few flows as possible during system configuration changes when removing or adding servers. On the other hand, here we are concerned with disruptions caused by migrating flows amongst FE's in order to re-balance workloads.

Most state-of-the-art schedulers migrate flows without considering their rates, but this is ineffective. The probability of shifting low-rate flows is high because there are many of them. Shifting these flows does not help balance the system much, but causes unnecessary dis-

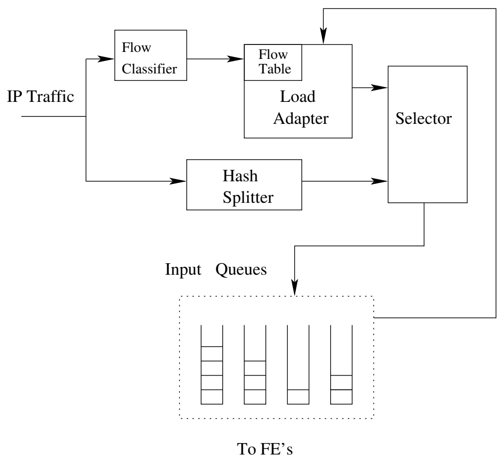

Figure 4: Load Balancing Packet Scheduler

ruption to FE states. The high rate flows are few so it is unlikely that they would be shifted, but it is these flows that cause trouble. While the scheduler is busy shifting low-rate flows, the high-rate ones keep swamping the overloaded processor(s).

It is worth noting that the *packet spraying* in [2] was proposed to deal with rare "emergency" situations when a flow by itself exceeds the processing power of one FE. Packet spraying operates on bundles instead of individual flows and the scheme does *not* actively spray to achieve load balance. In contrast, our goal is specifically to balance load with minimum disruption by identifying and *actively* spraying the high rate flows.

LOAD BALANCER

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

Fig. 4 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. Obviously, there are many choices for the hash function. For example, the function could use the low order bits of the address and calculate the FE as the modulus of the number of FE's. Alternatively, HRW could be used to minimize disruption in the case of FE failures.

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

An important design parameter is F, the size of the balancer's flow table. Generally, shifting more high-rate flows by having more flows in the table is more effective as far as load balancing is concerned. Nevertheless, to reduce cost, accelerate 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. 4). The flow classifier monitors the incoming traffic to decide which flows are high-rate and should be put in the balancer's flow table. In the next section, we discuss the flow identification procedure in detail.

IDENTIFYING HIGH-RATE FLOWS

We define *window size*, W, as the number of packets over which flow information is collected. Therefore, the incoming IP traffic is a sequence of windows: $W_1, W_2, \ldots, W_n,\ n \rightarrow \infty$, each containing W packets. Suppose we are receiving packets in W_i. We find the set F_i that contains the largest flows in W_i. The number of flows in F_i equals the size of the flow table, F, $|F_i| = F$. $F_0 = \{\}$. At the end of W_i, we replace the flows in the flow table by those in F_i. This mechanism benefits from the phenomenon of *temporal locality* in network traffic. Due to the *packet train* [17] behavior of network flows, it is likely 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

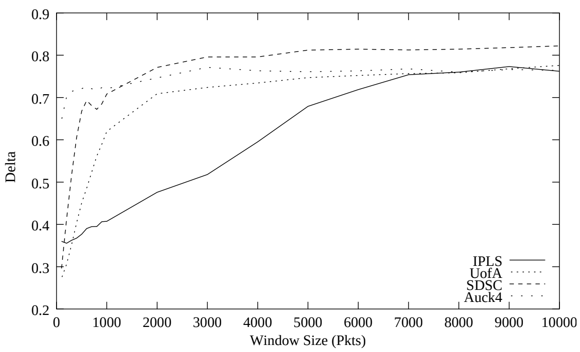

Figure 5: Effects of W on Δ ($F = 5$)

temporal locality, we define

$$\Delta = \frac{\sum_{i=1}^{n} \delta_i / F}{n}, \quad n = \frac{N_F}{W}, \qquad (7)$$

where N_F is the number of packets forwarded during the measurement. Thus, $0 \leq \Delta \leq 1$. The larger the value of Δ, the better the flow information collected in the current window predicts high-rate flows for the next window.

Small W values are preferred when the input buffer size is small and load adjustment must be made to reflect the existence of smaller scale, short-term bursty flows. Larger W values can be used for larger buffers where the system can tolerate the load imbalance caused by bursts of small flows. Fig. 5 shows the effects of W on Δ for the first one million entries of the four larger traces in Table 1 with $F = 5$. The larger the value of W, the better the current high-rate flows predict the future. This high predictability is critical to the success of the flow classifier. Our experiments show that the largest flow of the entire trace is almost always identified as the largest flow of every window (the smallest W experimented with is 100). And we will see that shifting even only the one largest flow is very effective in balancing workloads.

To implement high-rate flow detection, another traffic model, the hyperbolic *footprint* curve [18]:

$$u(W) = AW^{1/\theta}, \ A > 0, \ \theta > 1, \qquad (8)$$

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

SIMULATIONS

To validate the load balancer design, we conduct trace-driven simulations and compare packet drop rates under simple and adaptive hashing schemes, where simple hashing implements a modulo operation, $FE\_ID = (IPAddress)\%N$ where N is the number of FE's.

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 achieve balanced load with minimum adaptation disruption, the adapter only schedules packets in the high-rate flows. Lower-rate flows are mapped to FE's by the hash splitter.

There are multiple choices for deciding when the adapter should be activated to redirect packets. For example, the adapter can be triggered periodically. This scheme is easy to implement, as it does not require any load information from the system. Periodic remapping may not be efficient, however, as far as minimizing adaptation disruption is concerned because it could shift load unnecessarily, such as when the system is not unbalanced. As an alternative, the adapter can monitor the lengths of the input queues and remapping can be triggered by events indicating that the system is unbalanced to some degree. For example, remapping could be based on the input buffer occupancy, the largest queue length, or the CV of the queue length growing above some predefined threshold. The system load condition could be checked at every packet arrival for fine-grained control or periodically to reduce the overhead of checking.

As another design dimension, the remapping policy decides to which processor(s) the high-rate flows should be migrated. The solution in our simulations is to redirect all the high-rate flows to the FE with the shortest queue.

Simulations with Periodic Remapping

Parameters

For the packet scheduler shown in Fig. 4, the major design parameters include the number of entries in the flow table (E), the number of FE's (N), and the rescheduling interval (T) for periodic invocation.

Results

Fig. 6 shows the simulation results for the UofA trace and the first one million entries of the IPLS trace. CV (Eq. 2) is the load balance metric. For the UofA trace, scheduling the one largest flow is all that is needed to reduce the CV by orders of magnitude. This is especially true when the number of FE's is small. For the IPLS trace, we have similar results although the scale differs. Two trends are obvious from the figures. First, as the number of FE's, N, increases, more high-rate flows need to be scheduled to achieve the same CV as with smaller values of N. On the other hand, when more high-rate flows are scheduled, the load balancing results are bet-

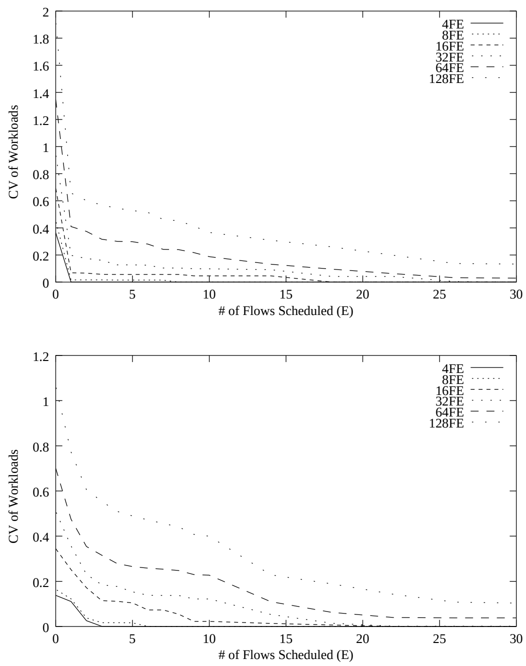

Figure 6: CV for the scheduled UofA Trace (top) and the First 1 Million Entries of the IPLS Trace (T=50)

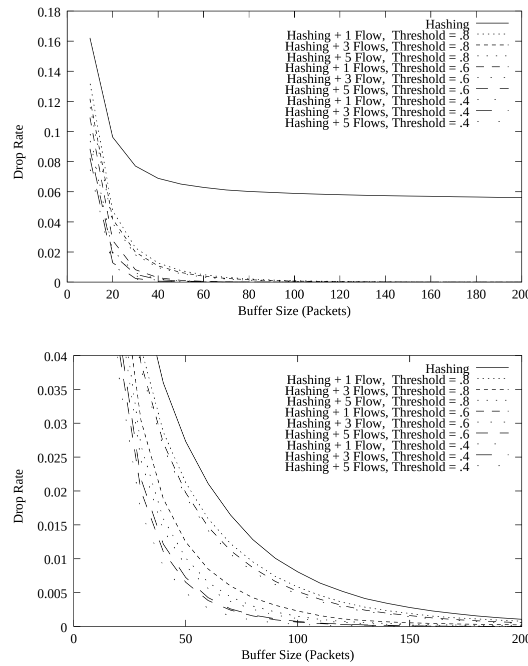

Figure 7: Packet Drop Rates in a 4-FE System for the UofA Trace (top) and the First 1 Million Entries of the IPLS Trace

ter. For most configurations, re-scheduling only the one largest flow reduces the CV by one or more orders of magnitude. This is particularly true for the UofA trace, where the largest flow is especially dominant.

The results for larger values of T have similar trends. Larger T values reduce the demand for computing resources by the scheduler and reduce disruption. On the other hand, larger values of T might lead to temporarily undetected load imbalance, or even packet loss. Our simulation results show that values of T from 50 to 6400 packets do not cause significant differences in the CV of FE workloads, so the system appears relatively insensitive to this parameter.

Simulation of an Occupancy-driven Adapter

Instead of simple periodic remapping we can use some indication of load imbalance from the FE's. A combination of a measure of system load or load imbalance and a threshold can be effective. When the metric is over the threshold, the adapter is invoked.

We present simulation results for a 4-FE system. The event invoking the adapter is the input buffer occupancy: the adapter is triggered when the buffer is filled above a certain percentage. We are concerned with the effects of different threshold values, buffer sizes, and numbers of shifted high-rate flows on packet drop rates. In the simulations, the classifier uses a window of 1000 packets to detect high-rate flows.

For the UofA trace, we impose a fixed service time $(1/\mu)$ of 200 milliseconds for each packet at the individual FE's. With the observed mean inter-arrival time $(1/\lambda)$ of about 71 milliseconds, this assumption gives the overall utilization (ρ) of the system:

$$\rho \quad = \quad \frac{\lambda}{\mu} = \frac{1/71}{1/(200/4)} = 0.7142.$$

For the first one million entries in the IPLS trace, we selected a system utilization of 0.8.

Fig. 7 shows the results. First, scheduling high-rate flows outperforms simple hashing by a large margin, especially as buffer sizes increase. This is desirable because large buffers are usually needed for today's high-speed links to relax the demand for processing power of the FE's. This is necessary due to the gap between optical transmission capacity and electronic processing power in store-and-forward devices. Second, scheduling more than one of the high-rate flows using the same threshold value only modestly improves performance, compared with scheduling just the largest flow. Lowering the value

Table 3: Comparison between Shifting Only the Most Dominating Flow and Shifting Only Less Dominating Ones

Simulation	Auck4 MDF	Auck4 LDF	IPLS MDF	IPLS LDF
No. of Flows	1	37	1	439
$CV[q_i]$	.172	.265	.0782	.137
η	.133	.133	.0103	.0121
ζ	.0413	1.84	.0357	22.2
Rr	.0612	.146	.00965	.0626

Simulation	SDSC MDF	SDSC LDF	UofA MDF	UofA LDF
No. of Flows	1	2	1	500
$CV[q_i]$	.181	.164	.143	.288
η	.0904	.0749	0	.103
ζ	.0342	.0714	.0357	25.2
Rr	.00546	.00740	.00965	.0511

of the threshold is more effective. This comes, however, at the cost of increasing the frequency of remapping, which causes more disruption. At the extreme, the adapter is invoked at every packet arrival when the threshold value is 0.

For the IPLS trace, the performance of hashing does not lag as much behind as in the case of the UofA trace. This is due to peculiarities of the traffic used in the experiments. For example, the largest flow in the UofA trace is responsible for a much larger fraction of the total number of packets in the trace.

Advantages of Shifting the Most Dominating Flow

To further illustrate the advantages of shifting the most dominating flows, we compare the results of two simulations: scheduling only the most dominating flow (MDF) and scheduling only less dominating flows (LDF) to achieve similar drop rates as with MDF. We simulated the periodic remapping policy with a 20-packet checking period. Table 3 summarizes the results for four traces. With similar packet drop rates (η), scheduling the most dominating flow always causes less adaptation disruption (ζ) and packet reorders (R_r). For the Auck4, IPLS, UofA traces, scheduling the most dominating flow also achieves a smaller $CV[q_i]$. More than one of the smaller flows are needed to achieve similar packet drop rates as scheduling the largest flow. The smallest number of less dominating flows needed is two, as in the SDSC case where scheduling less dominating flows achieves a lower miss rate and $CV[q_i]$. One reason might be that in the SDSC trace, the most dominating flows identified by the mechanism in section only accounts for a small portion of the total traffic, not significant enough for scheduling the most dominating flow to outperform scheduling less dominating flows. The other extreme is the UofA trace, where the most dominating flow by itself represents around 16 per cent of the aggregate traffic. When it is scheduled onto an FE, even if the rest of the traffic is spread evenly among the other seven FE's (each 12 per cent), the system is still not perfectly balanced.

CONCLUSIONS AND FUTURE WORK

In this paper, we first use Zipf-like distribution functions to model the flow frequency distributions in traffic collected from a wide range of network environments. We further demonstrate that under this model, hashing schemes for parallel forwarding systems cannot balance workload.

Based on the observations of flow level characteristics of Internet traffic, we propose a hash based load balancing packet scheduler for parallel forwarding systems. Our scheduler achieves effective load balancing by shifting high-rate flows. Our scheme has low complexity and thus is efficient; the storage demand is trivial; and moreover, as high-rate flows are few, the disruption to system states is small, which leads to high forwarding performance.

Minimizing *adaptation disruption* is an important goal of the scheduling scheme described in this paper. The concept of adaptation disruption can be quantified to measure the degree of disturbance caused by different load balancing schemes, or by different parameter sets. One method that is particularly targeted at cache performance evaluation is to measure the temporal locality of the traffic seen at each FE using the technique proposed in [4]. Nonetheless, more general and succinct metrics of adaptation disruption are desirable.

Comparing with other load distribution schemes for parallel forwarding, e.g., [2,3], should be interesting and is another direction of future work.

References

[1] N. Shah, "Understanding network processors," M.S. thesis, U. C. Berkeley, September 2001.

[2] G. Dittmann, A. Herkersdorf, "Network processor load balancing for high-speed links," in *2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*

(SPECTS 2002), San Diego, CA, USA, July 2002, pp. 727–735.

[3] L. Kencl, J. Le Boudec, "Adaptive load sharing for network processors," in *IEEE INFOCOM 2002*, New York, NY, USA, June 2002, pp. 545–554.

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

[5] S. Nilsson, G. Karlsson, "IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083–1092, June 1997.

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

[7] R. Jain, "A comparison of hashing schemes for address lookup in computer networks," *IEEE Transactions on Communications*, vol. 40, no. 3, pp. 1570–1573, October 1992.

[8] V. Srinivasan, G. Varghese, "Fast address lookup using controlled prefix expansion," in *ACM SIGMETRICS 1998*, Madison, WI, USA, 1998, pp. 1–10.

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

[10] D. G. Thaler, C. V. Ravishankar, "Using name-based mappings to increase hit rates," *IEEE/ACM Transactions on Networking*, vol. 6, no. 1, pp. 1–14, February 1998.

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

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

[13] W. Willinger, M. Taqqu, R. Sherman, D. Wilson, "Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 71–86, Feb 1997.

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

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

[16] L. A. Adamic, "Zipf, power-laws, and pareto - a ranking tutorial," http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html, April 2000.

[17] R. Jain, S. Routhier, "Packet trains: Measurements and a new model for computer network traffic," *IEEE Journal of Selected Areas in Communications*, vol. SAC-4, no. 6, pp. 986–995, September 1986.

[18] D. Thiebaut, J. L. Wolf, H. S. Stone, "Synthetic traces for trace-driven simulation of cache memories," *IEEE Transactions on Computers*, vol. 41, no. 4, pp. 388–410, 1992.