Design and Evaluation of a Partial state router

Phani Gopal V Achanta and A.L. Narasimha Reddy
Texas A&M University, College Station, Texas 77843-3128
Email: phani@cs.tamu.edu, reddy@ee.tamu.edu

***Abstract*— In this paper, we present the design and evaluation of a partial state router. A partial state router maintains a fixed amount of state irrespective of the number of flows served at the router. We show the practical feasibility of partial state routers by implementing a novel partial state scheme, LRU-FQ, on the Linux platform. We report on our experience in employing the developed LRU-FQ router in several realistic experiments. Our results show the effectiveness of LRU-FQ in controlling high-bandwidth traffic and providing better response times for web traffic. We also present a detailed evaluation of the developed router to demonstrate the feasibility and scalability of partial state schemes.**

I. Introduction

A. Motivation

Non-responsive flows (flows that do not respond to congestion signals) have been shown to have an adverse impact on the network and the responsive flows [1].The need for identifying high bandwidth flows has been recently studied [2], [3], motivated by fair sharing of bandwidth [2]–[4].

Partial state schemes [2], [3], [6] make use of a limited amount of state independent of the number of flows. Given the heavy tailed distribution of Internet traffic [7] between mice (short-lived data transfers, eg. web transfers) and elephants (long term flows), the traffic at a router within the network will have a small proportion of elephant flows and a large proportion of mice flows. Partial state schemes try to utilize the limited amount of state to track the non-responsive flows, or bandwidth hogs, or flows above a certain target rate. The flows thus identified can be separately managed while the stateless flows are managed in an aggregate fashion. Further motivation in exploring the partial state approach is the possibility of containing bandwidth attacks.

The various Active Queue Management schemes differ in the amount of per-flow state maintained at the router [8]–[10], [12]. Core-Stateless Fair Queuing (CSFQ) [4] employs state within each packet. SRED [14] arrives at a list of 'misbehaving' flows by probabilistically replacing a list entry if it does not match the incoming packet. LRU-RED [6] makes use of an LRU (Least Recently Used) cache to modify the RED probability for cached flows. RED-PD [3] makes use of the packet drop history to arrive at a list of flows exceeding a target bandwidth.

This paper presents a scheme, LRU-FQ, that allows a *quantitative* policy-driven control of the link bandwidth allocated to the high-bandwidth, non-responsive flows.

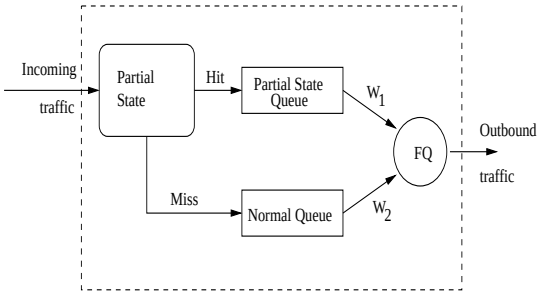


Fig. 1. Partial state based router

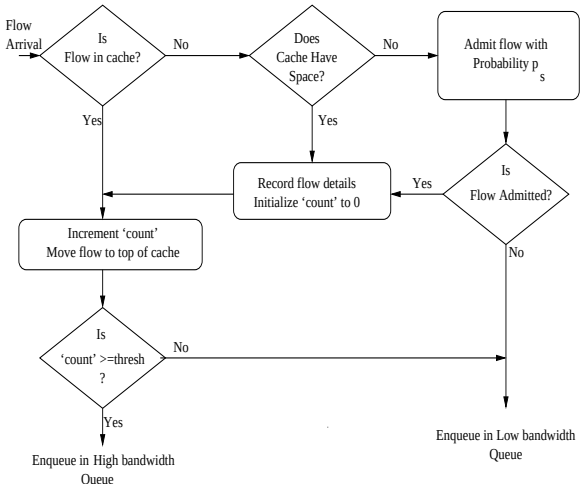


Fig. 2. LRU-FQ : Flow chart

II. LRU-FQ: Design and Implementation

A. LRU-FQ: the scheme

LRU-FQ scheme aims to protect responsive as well as low bandwidth traffic from non-responsive, high bandwidth traffic. LRU-FQ employs an LRU cache to identify and separate the non-responsive high bandwidth traffic from the rest of the traffic. The partial state (LRU cache) is managed to maintain state for these high bandwidth, non-responsive flows. The separation of these flows is done by queuing them in separate queues. Fair Queuing is employed for fair sharing of the output link between the two queues (Figure 1). The sharing of the bandwidth among the two classes can be determined by a policy and controlled by the weights on FQ queues. We discuss the identification and containment of high bandwidth, non-responsive flows in more detail.

LRU-FQ scheme employs an LRU cache as the partial state to identify non-responsive high bandwidth flows. Figure 2 il-

lustrates the identification algorithm with a flow chart. An LRU cache is maintained for a fixed number of flows, irrespective of the number of flows at the router. The LRU cache is searched for an entry corresponding to the incoming flow. A match results in the flow entry being updated and the corresponding cache entry being moved to the topmost position in the cache. A miss results in the bottom most entry in the cache being replaced with the incoming flow with a fixed probability p [6]. Thus, flows need to be sending at consistently high rates in order to have an entry in the cache. An entry in the cache can keep track of the flow state such as its sending rate or number of packets/bytes seen since the flow is admitted into the cache or other related information.

The LRU cache enables in identifying high bandwidth flows [6]. The probabilistic admission policy helps in keeping the short-term flows out of the cache. However, a short-term flow may still get admitted into the cache. To avoid treating such flows as high bandwidth flows, the cache state is utilized. A flow has to be observed to be sending packets above a certain threshold (rate) before being treated as a high bandwidth flow. The probabilistic admission and the threshold prunes the cache list to contain mostly high bandwidth flows. The identified flows are enqueued separately from the stateless flows as shown in Figure 1. Hit traffic usually corresponds to high bandwidth flows and the miss traffic mainly constitutes of short-term flows.

LRU-FQ employs fair queuing between the two queues of cached and non-cached flows to regulate the bandwidth consumption of non-responsive high bandwidth flows. LRU-FQ treats cached high bandwidth flows differently from stateless low bandwidth or responsive flows. While it is possible to employ the state of the cached flows in regulating the resource consumption of these flows individually, in this paper, we focus on the aggregate resource consumption of the cached flows. The non-cached flows are treated in an aggregate manner. to control the traffic mix the router wants to support. Packet scheduling is O(1) and does not require any sorting since we are only considering two queues.

The LRU-FQ setup shown can be used for a number of possible applications. If the cache can be managed to contain mostly high-bandwidth non-responsive flows, the non-responsive flows can be controlled to consume only a certain fraction of the link bandwidth by setting appropriate weights on the queues.

LRU-FQ could be used to provide Web mice better delay network service. It is possible to control the delay observed by the stateless flows by assigning the weight of the “miss” queue to be higher than the weight of the “hit” queue. Bandwidth allocated to each queue can be controlled through rate control while delays can be affected through the assignment of proper weights for the weighted fair queuing mechanism.

An analysis [13] has shown that our scheme identifies malicious DoS flows with a high probability and performs signficantly better than Stochastic Fair queueing and Class Based Queuing.

B. The Linux Design Space

Linux offers a layer-based IPv4 network stack [15]. Routers are mainly involved in the task of packet forwarding. Consequently, three layers are of importance - physical layer, link layer and network layer.

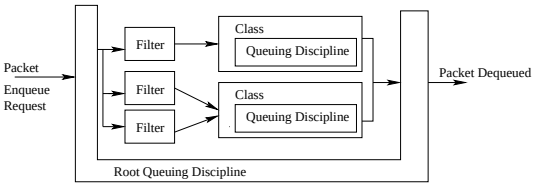


Fig. 3. Linux Traffic Control Architecture

Linux provides a modular architecture for QoS. Figure 3 shows the typical layout of the components of the Linux QoS, namely, queuing disciplines, classes and filters. *Filters* are classifiers which divide the outbound traffic into *classes*.

1) Design: Challenges and choices: Given the modular nature of Linux QoS architecture, we distributed the LRU-FQ scheme implementation among the various QoS components. Maintenance of the LRU cache and taking decisions about enqueuing the outbound packets on the two queues is implemented as a filter. The LRU filter needs weights of the high-bandwidth and low-bandwidth queues as parameters along with the scheme specific parameters like probability, threshold and cache size. Since none of the existing filters allow for state maintenance, we implemented a new LRU filter. The state is maintained in a doubly linked list and a hash-based data structure. Hashing enables faster detection of existing cache entries and the linked list allows for faster modification of the LRU cache contents.

The fair queuing component of the scheme requires a queuing discipline which does class based fairness. We implemented a class based fair queuing discipline based on STFQ [5]. We extend the sk_buff data structure to allow the the maintenance of start tag and finish tags of STFQ.

The user level application, tc, has been modified to allow for usage of the new filter and queuing discipline.

III. Experimental Setup and Results

A. Experimental Test bed

All experiments were conducted on a Linux router based on 2.4 version of the Linux kernel[1]. The machines used to test the LRU-FQ implementation on Linux were connected on a private network as shown in Figure 4. The 100Mbps private network ensured that the experiments were isolated from any spurious traffic spikes from the university network. The server side machine was chosen to be the fastest so that there is no bottleneck at the receiving end. This ensures that any performance characteristics observed are due to the router and not due to the end-hosts’ capabilities.

The only external network port is connected to the server side and was used for regular maintenance purposes. The UDP

[1]As of June 2002, the latest Linux kernel version is 2.4.17

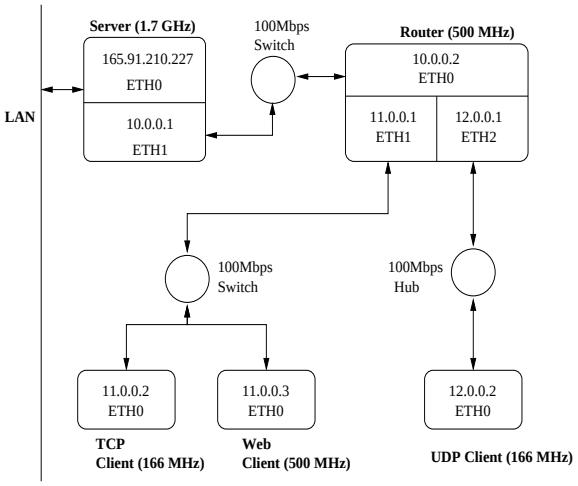


Fig. 4. Experimental Setup

and TCP traffic was separated into two networks as there is a possibility of UDP traffic adversely affecting the TCP traffic when using the same hub/switch.

B. Containing non-responsive flows

Our scheme attempts to distinguish responsive flows from non-responsive flows. This section shows the experiments conducted to show the effectiveness of LRU-FQ in differentiating long term flows based on their responsiveness.

The initial experiment was run with the the LRU-FQ parameters of cache size = 12, threshold = 125 and probability p = 1/50. The scenario has 20 TCP long term flows and the effect of varying the queue weights is studied for various number of UDP flows. The UDP flows are pumping traffic to the full capacity of the link i.e. 100 Mbps. Packet sizes were chosen to be 1472 bytes, the maximum data segment possible, in order to avoid data fragments from skewing the throughput results.

In these experiments, we set a goal to limit the bandwidth consumption of 'hit' traffic to a set fraction of the link capacity. Ideally, the non-responsive traffic is limited to the set limit. As we can observe in Figure 5(a), our results reach the ideal value until a set limit of 30% when we have 2 or 3 UDP flows. At lower limits, the work-conserving nature of fair queuing algorithm favors the constantly backlogged high bandwidth queue and hence we do not reach the ideal limits of 20% and 10%. With larger number of UDP flows, the performance degrades as the UDP flows occasionally get replaced from the cache by the TCP flows thus reducing the effectiveness of containing non-responsive flows. The effectiveness of containing the non-responsive flows can be improved by either increasing the cache size or by pinning the flows in the cache once identified.

Our results are compared with the normal router which allows only about 7% of the link bandwidth for TCP applications in the presence of UDP flows (Presence of high bandwidth UDP flows can be considered as a DoS attack scenario also). The results clearly indicate the effectiveness of LRU-FQ in improving the fairness among the flows by consistently providing higher bandwidths to responsive flows.

In order to show that the LRU-FQ scheme is effective even with reduced non-responsive flow rates, we conducted an experiment to study the effect of UDP sending rates on the requested traffic mix. It can be seen in Figure 5(b) that the LRU-FQ scheme is effective at various non-responsive loads. The results also show that at reduced non-responsive loads, the Fair Queuing mechanism does better at reaching the policy goals.

C. Web mice versus Elephants

Since the common web traffic has to compete with both responsive and non-responsive long term traffic, we conducted an experiment to study the effect of long term flows on the web traffic. For generating the web traffic, we made use of the Webstone [16] benchmark software. Webstone emulates realistic web traffic by allowing the user to specify a frequency histogram of the web files requested from the web server.

In our experiment, we chose the long term responsive(TCP) flows to be 20 and studied the effect of varying non-responsive(UDP) flows on the web traffic. The experiment was conducted by setting the LRU-FQ parameters probability p = 1/50, threshold = 125, cache size = 12 and LRU to normal queue weights = 1:1.

The throughput results observed are shown in Table I. The results show that web traffic is given a higher degree of isolation from the long term flows by the LRU-FQ scheme. This is reflected by the higher number of successful web fetches (almost ten times larger) in the observation period. The long term TCP flows are still getting the assigned proportion (50%) of the bandwidth.

TABLE I
BANDWIDTH RESULTS FOR WEB MICE

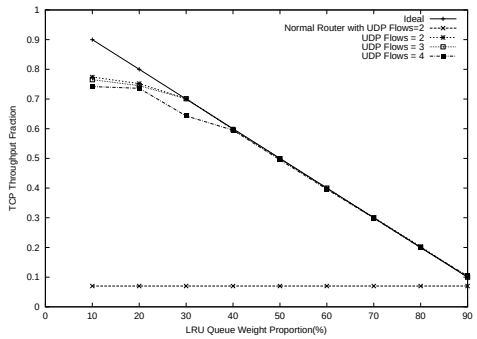
	UDP Flows	UDP Throughput	# of Web Requests	TCP Throughput	TCP Fraction
Normal DropTail Router	2	89.450	1313	5.883	0.0617
	3	89.796	1284	5.548	0.0581
	4	89.125	927	6.212	0.0651
LRU-FQ Router	2	45.727	13915	44.916	0.4955
	3	45.733	13828	44.830	0.4950
	4	46.237	13632	44.514	0.4905

The corresponding results for the delays observed are shown in Table II[2]. Connection time is the time taken by the TCP to establish a connection while response time is the actual time taken to transfer the data file. As can be seen from the table, web traffic encounters lower delays with the LRU-FQ router. The lower delays are both in terms of average delay as well as the jitter thus providing a better service than normal routers. It is observed that LRU-FQ provides average delays that are roughly one-tenth of the delays observed in a normal router.

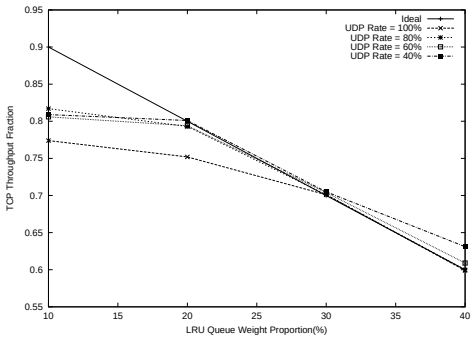
D. Effect of varying cache size

In order to study the impact of cache size on the performance, we ran several experiments with different cache sizes.

[2]Avg, Dev, Min and Max imply Average, Standard Deviation, Minimum and Maximum respectively



(a) Effect of non-responsive long term flows



(b) Effect of varying non-responsive flow rates

Fig. 5. Containing non-responsive flows

TABLE II

DELAY RESULTS FOR WEB MICE

	UDP Flows	Response Time sec				Connection Time sec			
		Avg	Dev	Min	Max	Avg	Dev	Min	Max
Normal DropTail Router	2	2.540	4.429	0.026	45.080	1.952	3.074	0.0118	45.007
	3	2.696	4.923	0.026	93.017	1.935	3.110	0.0115	45.013
	4	3.064	4.826	0.026	45.028	2.112	3.415	0.0122	45.005
LRU-FQ Router	2	0.255	0.849	0.012	21.149	0.136	0.661	0.0014	21.014
	3	0.258	0.854	0.013	22.265	0.131	0.589	0.0017	9.034
	4	0.260	0.875	0.013	21.054	0.134	0.614	0.0020	9.026

The graph in Figure 6 brings out this aspect in detail. The parameters used for this experiment were probability p = 1/55, threshold = 125, number of TCP flows = 20, and the fairness weights for both queues were equal.

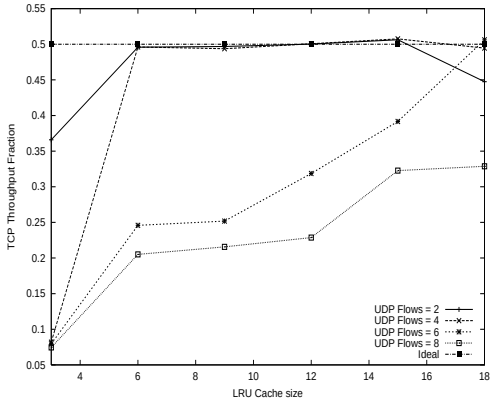


Fig. 6. Effect of cache size on LRU-FQ scheme

For smaller cache sizes, the observed throughput fraction for TCP flows falls much below the ideal case. This can be explained by the fact that choosing a cache size to be exactly equal to the number of non-responsive flows does not take into account the possibility of a non-responsive flow being probabilistically replaced in the LRU. Thus, keeping the cache size larger than the non-responsive flows reduces the probability of non-responsive elephants being the bottom entry of the LRU.

The other aspect in the cache size is that normal responsive traffic may find its way into the LRU cache. Sensitivity of the scheme to the cache size can be further reduced by employing a rate-based threshold to pin the high bandwidth flows' entries in the cache.

E. Performance of LRU-FQ under normal workloads

The LRU-FQ scheme has been shown to work well when there is a constant non-responsive load presented to the router. This section shows that the scheme does not impact flows which are either responsive or using atmost their fair share of the link bandwidth.

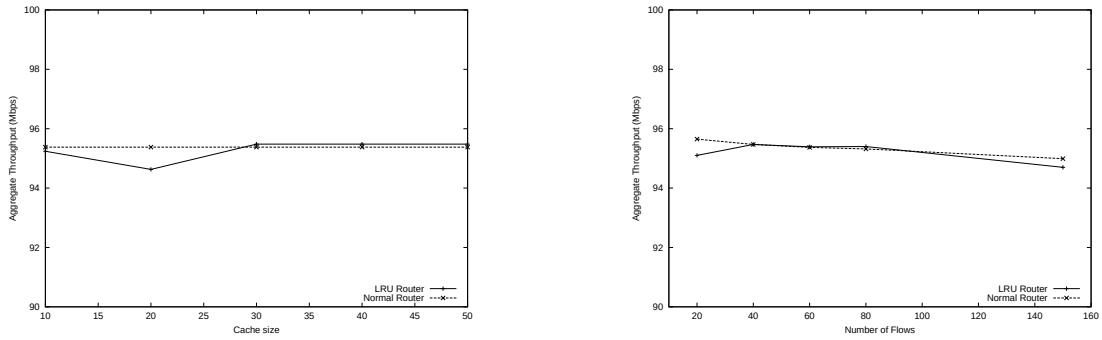
In the first set of experiments, the workload consists of only responsive TCP flows. Statistics were collected for a responsive(TCP) load with the LRU-FQ scheme activated and deactivated. When the LRU-FQ scheme is active, it is configured to have equal weights for both queues. The remaining parameters were configured to be cache size = 9, threshold = 125 and probability p = 1/55. Table III shows the corresponding results. The results substantiate the claim of responsive loads not being adversely effected by the scheme with arbitrary parameters.

TABLE III

LRU-FQ UNDER NORMAL WORKLOAD

Responsive Flows	Average	Std. dev	Total BW
Normal Router			
15	6.194959	0.273623	92.92439
20	4.637118	0.246848	92.74325
25	3.710390	0.179423	92.75976
LRU-FQ Router			
15	6.196750	0.104381	92.95125
20	4.627254	0.235561	92.54508
25	3.709822	0.285475	92.74556

The other normal scenario would be the case where non-



(a) Effect of flows on LRU-FQ scalability

(b) Effect of cache size on LRU-FQ scalability

Fig. 7. Scalability issues

responsive flows are sending at their fair share: the scheme should not penalize the non-responsive flows when they are sending at their fair rates. In order to test this scenario, we conducted an experiment where the non-responsive flows are bandwidth limited to their fair share and the effect of the scheme is studied. Table IV shows the corresponding results which show that LRU-FQ does not penalize non-responsive flows when they claim only their fair share of bandwidth.

TABLE IV

LRU-FQ UNDER NORMAL MIXED WORKLOAD

UDP Flows	TCP Flows	Average UDP B/w (Mbps)	Average TCP B/w (Mbps)	Ideal Average B/w (Mbps)
2	18	4.9141	4.6327	4.6608
3	17	4.7240	4.6607	4.6702
4	16	4.5839	4.7015	4.6780

F. Scalability issues

The scalability of the LRU-FQ scheme is shown using three experiments which measure the aggregate throughput for varying flows and state respectively.

Figure 7(a) shows the impact of cache size with a fixed number(30) of flows. Irrespective of the cache size, LRU-FQ aggregate throughput matches the throughput of a normal router. Figure 7(b) shows the impact of the number of flows on the aggregate throughput at a fixed cache size of 60. It is observed that the aggregate performance is nearly the same as a normal router.

This shows that an LRU-FQ implementation's aggregate performance is scalable with the number of flows, amount of state and the packet size. The experiments have been limited to a maximum of 150 flows. This is a limitation of the testbed and not of the scheme.

IV. CONCLUSIONS AND FUTURE WORK

This paper presented a novel scheme, LRU-FQ, which exploits partial state. LRU-FQ has been shown to provide better flow isolation and significantly better delays for web traffic. The scheme has been shown to be practically feasible in providing the network administrator a control of the amount of high-bandwidth, non-responsive traffic at the router.

The scheme can be extended further to study aggregate traffic instead of individual flows. For example, flow identification based only on destination address can help in identifying DoS attacks on a single network. Implementation of partial state mechanisms on network processors is also currently being investigated.

Acknowlegements: This work was supported in part by NSF ANI-0087372 grant, State of Texas, TITF and Intel Corp.

REFERENCES

[1] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," ACM Transactions on Networking, Aug. 1999.
[2] D. Tong and A.L.N. Reddy, "QoS enhancement with partial state," in Proceedings of iWQOS, June 1999.
[3] R. Mahajan, S. Floyd and D. Wetherall, "Controlling High-Bandwidth Flows at the Congested Router," ICNP, Nov. 2001.
[4] I. Stoica, S. Shenker and H. Zhang, "Core-Stateless Fair Queuing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks," SIGCOMM'98.
[5] P. Goyal, H.M. Vin and H. Cheng, "Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," in IEEE/ACM Transactions on Networking, Oct. 1997.
[6] Smitha and A.L.N. Reddy, "LRU-RED: An active queue management scheme to contain high bandwidth flows at congested routers," Proc. of Globecomm, November 2001.
[7] S.B. Fred, T. Bonald, A. Proutiere, G. Regnie, and J.W. Roberts, "Statistical bandwidth sharing: a study of congestion at flow level," Proc. of ACM SIGCOMM, Aug. 2001.
[8] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," in IEEE Trans. on Networking, Aug. 1993.
[9] W. Feng, D. Kandlur, D. Saha and K. Shin, "A New Class of Active Queue Management Algorithms," Tech. rep. U. Michigan, Apr. 1999.
[10] R. Pan, B. Prabhakar and K. Psounis, "CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," Proc. of IEEE Infocomm, Mar. 2000.
[11] B. Suter, T.V. Lakshman, D. Stiliadis and A.K. Choudhary, "Design Considerations for supporting TCP with per-flow queuing," INFOCOMM'98.
[12] D. Lin and R. Morris, "Dynamics of Random Early Detection," in Proceedings of ACM SIGCOMM conference, Sept. 1997.
[13] S. Voorhies, H. Lee, and A. Klappenecker, "Randomized Caching, Probabilistic Queuing, and Denial of Service Attacks," Texas A&M Tech. Report, May 2003.
[14] T.J. Ott, T.V. Lakshman and L.H. Wong, "SRED: Stabilized RED," in Proceedings of IEEE Infocomm, Mar. 1999.
[15] G. Herrin, "Linux IP Networking: A Guide to Implementation and Modification of the Linux Protocol Stack," May 2000, Available online at http://www.cs.unh.edu/cnrg/gherrin
[16] Mindcraft Incorporated, "Webstone 2.5 benchmark," Available online at http://www.mindcraft.com/webstone/.