Design, Analysis and Experience of a Partial state router

Phani Gopal V Achanta and A.L. Narasimha Reddy
Texas A&M University, College Station

Abstract— **In this paper, we motivate the use of partial state schemes in a wide variety of network applications. Partial state can be used to arrive at a list of high bandwidth flows which can be made use of in containing non-responsive flows, providing better QoS for web traffic, and alleviate certain Denial of Service (DoS) attacks. We show the practical feasibility of partial state based schemes by implementing a novel partial state scheme, LRU-FQ, on Linux platform. The scheme makes use of an LRU cache to classify flows into high bandwidth and low bandwidth classes. A class-based fair queuing algorithm is used to obtain a policy-driven control of the proportion of link bandwidth allocated to high bandwidth flows being serviced at the router. Empirical data is presented to bring out the stated uses of partial state schemes.**

Index Terms—**Active queue management, LRU, fairness, non-responsive flows, high bandwidth flows.**

I. Introduction

A. Motivation

The adverse impact of non-responsive flows on the network as well as responsive flows have been studied before [1], [2]. Fair Queuing [3] and its variants (for example [4]) work well in containing the non-responsive flows. However, we need mechanisms which are scalable with the amount of traffic witnessed at the router. Partial state schemes are one class of approaches which try to achieve flow isolation with limited state.

Partial state schemes ([5], [6]) make use of a limited amount of state independent of the number of flows. Given the heavy tailed distribution of Internet traffic ([7], [8]) 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 elephants and a large proportion of mice. 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. Thus, partial state schemes separate the Identification and Resource Management phases. Identification phase finds the flows to populate the limited state while Resource management phase enforces the policy-driven resource control mechanism.

Further motivation in exploring the partial state approach is the possibility of containing bandwidth attacks. Certain DoS attacks are based on ICMP/UDP streams which try to consume bandwidth in order to reduce the goodput. When DoS attacks are launched from a limited set of sources, partial state mechanisms may allow identification of the high bandwidth flows. This can allow us to contain such DoS attacks through a policy-driven control of identified high bandwidth flows.

In this paper, we present a partial state router that employs a modified LRU policy [5] for state management (flow identification) and Fair Queuing for managing resources. We present the design, analysis and evaluation of an LRU-FQ router. We report on our experience in evaluating an experimental LRU-FQ router (implemented on a Linux PC) in a number of application scenarios.

B. Previous Work

The issues addressed by this paper have been approached individually by various schemes. Active Queue Management schemes usually address the resource management and fairness issues.

The various Active Queue Management schemes differ in the amount of per-flow state maintained at the router. Stateless schemes make their decisions based on overall characteristics observable at the router queue. These characteristics can be the average queue length, aggregate arrival and departure rates, packet counts etc. DropTail, Random Early Detection [9], BLUE [10] and CHOKe [11] are examples of stateless schemes.

Stateful schemes maintain a small amount of information for each flow that the router observes. The flow can be

Phani Gopal V Achanta is with the Computer Science Department. A.L. Narasimha Reddy is with the Electrical Engineering Department. `email:phani@cs.tamu.edu, reddy@ee.tamu.edu`
This work is supported by an NSF ANIR grant and by the Texas Higher Education Coordination Board

characterized as traffic between two end points defined either by the transport layer(per connection) or the network layer (per host). Longest Queue Drop (LQD) [12] and FRED [13] are examples of such schemes. Fair Queuing and its many variants employ per-flow state and scheduling mechanisms to provide different levels of service.

New mechanisms have been recently proposed that allow individual packets to carry state information to allow routers to make decisions about the service to be provided. Diffserv networks [14] and Core-Stateless Fair Queuing (CSFQ) [15] are two examples of such packet marking schemes.

Partial State schemes achieve a balance between the information to be stored and performance that can be achieved. The state required to be maintained is pruned by making use of techniques like sampling and caching. SRED [16] arrives at a zombie list of ‘misbehaving’ flows by probabilistically replacing a list entry if a random entry does not match the incoming packet. LRU-RED [5] makes use of an LRU (Least Recently Used) cache to modify the RED probability for cached flows. SACRED [6] employs random sampling and holding to maintain a cache of ‘marked’ flows which get penalized once the queue length exceeds a dropping threshold. RED-PD [17] makes use of the packet drop history at an RED router to arrive at a list of flows exceeding a target bandwidth. Traffic measurement techniques that employ fixed amount of state have also been presented recently [18]. Approximate fairness is achieved through partial state and RED mechanisms [19].

DoS attack prevention has been addressed at Network, Operating System and Middleware level. Approaches like SYN cookies [20] are specific to the type of DoS being addressed. Other examples of such approaches are given in [21] and [22]. Other approaches to DoS attacks include Network Ingress filtering [23] to filter spoofed addresses, Traceback algorithms [24] to throttle the attacker at the source network, and bandwidth attack detection schemes like MULTOPS [25].

Stateful active queue management schemes prove to be quite effective in addressing the issue of network resource control. However, they are not easily scalable to be implemented on high-speed routers where the amount of traffic is too large to maintain a per-flow state. Stateless techniques fail to protect TCP flows from aggressive UDP flows.

Most of the schemes mentioned provide a qualitative improvement in performance seen by short term flows and responsive flows. The ideal case would be to isolate the non-responsive flows from the rest of the traffic and provide a mechanism for the router to decide the proportion of non-responsive traffic it wants to handle. This paper presents such a scheme that allows policy driven regulation of non-responsive high bandwidth flows.

II. LRU-FQ: Design and Implementation

The previous section brought out the various issues in network buffer management and the previous schemes suggested for addressing the problem. This section lays down the requirements of an active queue management scheme and arrives at a design given the constraints of a software based router.

A. Requirements

The following goals dictated the design of our scheme:

- Protect web mice from non-responsive elephants.
- Protect responsive elephants from non-responsive elephants.
- Keep the per-flow state minimal in order to make it feasible to implement the scheme at high speeds.
- Keep the per-packet handling cost minimal so that packet forwarding rates on the modified routers remain same as on normal routers.
- Provide decoupled mechanisms for identifying non-responsive elephants and for managing resource consumption of such flows: this enables modular analysis of the scheme as well as independent adaptation/improvement of identification and resource management algorithms.

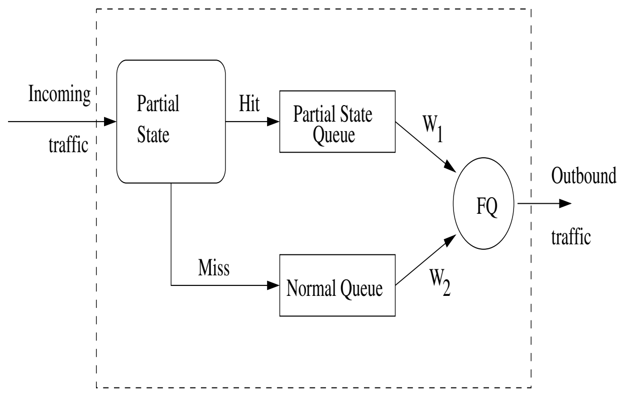

Fig. 1. Partial state based router

This section describes the LRU-FQ scheme in detail as well as the design tradeoffs involved in implementing a partial state router on a Linux router.

B. LRU-FQ: the scheme

LRU-FQ scheme employs partial state to identify high bandwidth flows. The identified flows are enqueued separately from the stateless flows as shown in Figure 1.

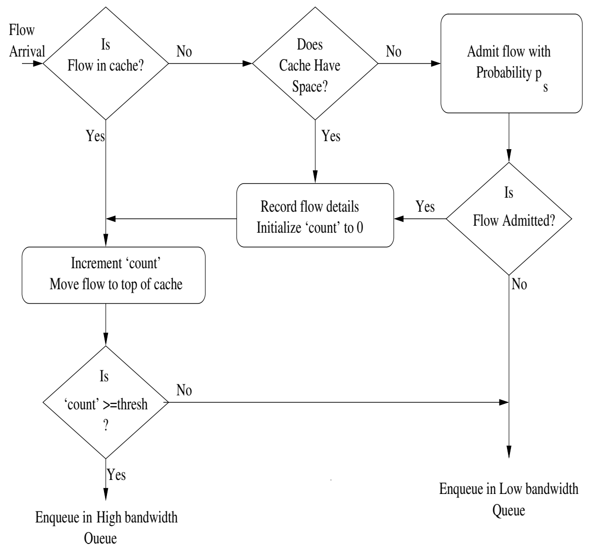
Flow Arrival
Is Flow in cache?
No
Does Cache Have Space?
No
Admit flow with Probability p s
Yes
Yes
Record flow details Initialize 'count' to 0
Yes
Is Flow Admitted?
Increment 'count' Move flow to top of cache
No
Is 'count' >=thresh ?
No
Yes
Enqueue in High bandwidth Oueue
Enqueue in Low bandwidth Queue

Fig. 2. LRU-FQ : Flow chart

By employing two separate queues and appropriate resource management algorithms, the resource consumption of high bandwidth flows can be regulated. In this paper, we employ Fair queuing between the two queues to regulate the high bandwidth flows.

Our mechanism treats cached high bandwidth flows differently from stateless low bandwidth flows. Our mechanism separates the identification of high bandwidth flows from resource management algorithms allowing both of them to be optimized independently. In this paper, we employ a modified LRU scheme for identifying the high bandwidth flows. Figure 2 illustrates the working of the identificaton algorithm with a flow chart.

An LRU cache is maintained for a fixed number of flows. 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 [5]. Thus, flows need to be sending at consistently high rates in order to have an entry in the cache.

Web traffic typically consists of many connections which last for a small amount of time. A low-bandwidth flow may still enter the cache due to the probabilistic admission process. In order to minimize the possibility of classifying such flows incorrectly, we require a flow to accumulate a packet count that exceeds a threshold before it can be classified as a high-bandwidth flow.

'Hit' traffic usually corresponds to high bandwidth flows and the 'miss' traffic mainly constitutes of short-term flows.

The scheme till now described the mechanism for identifying the high bandwidth flows. We use Start-Time Fair queuing [4] for regulating the traffic from both queues. Since any arbitrary positive weights can be assigned to the queues, the router can control the traffic mix it wants to support. STFQ simulates Generalized Processor Sharing [3]. STFQ algorithm requires the calculation of a start tag and a finish tag for each packet, the start tag being dependent on the previous packet's finish tag and the start tag of the packet at the head of the queue. The packets are scheduled in the increasing order of the start tags of the packets. Since, STFQ works without involving all queue states, it has a lower complexity.

The generic setup shown in Figure 1 and the LRU-FQ scheme allows for the various applications suggested in the next section.

C. Possible applications of LRU-FQ scheme

The LRU-FQ setup shown can be used for a number of possible applications. By choosing appropriate state management policies [5] [6], the cache can be managed to contain mostly non-responsive flows. Such an identification enables appropriate resource management techniques where non-responsive flows can be controlled to consume only a certain fraction of the link bandwidth.

It is also possible to manage the state in such a way to identify top bandwidth hogs. Such an identification enables DoS attacks staged with fewer flows that the amount of the state in the cache can be contained by appropriately containing the identified top flows.

The setup may find use as a scheme for Web mice to see better service in terms of lower delay bounds and larger connection rates. It is possible to control the delay observed by the stateless flows by assigning the weight of the normal queue to be higher than the weight of the partial state queue. Bandwidth allocated to each queue can be controlled through proper assignment of buffers while delays can be affected through the assignment of proper weights for the weighted fair queuing mechanism.

Another possible reason for considering partial state mechanisms could be to employ separate queuing mechanisms for high bandwidth and low bandwidth flows. Since high bandwidth applications like FTP are impervious to delay jitters, we could employ a partial state queue which provides better bandwidth performance with coarser delay performance.

D. LRU Analysis

In this section, we will provide a brief analysis of the LRU identification scheme presented earlier. Consider a flow at a rate r_i. Let use denote p_j as the probability of

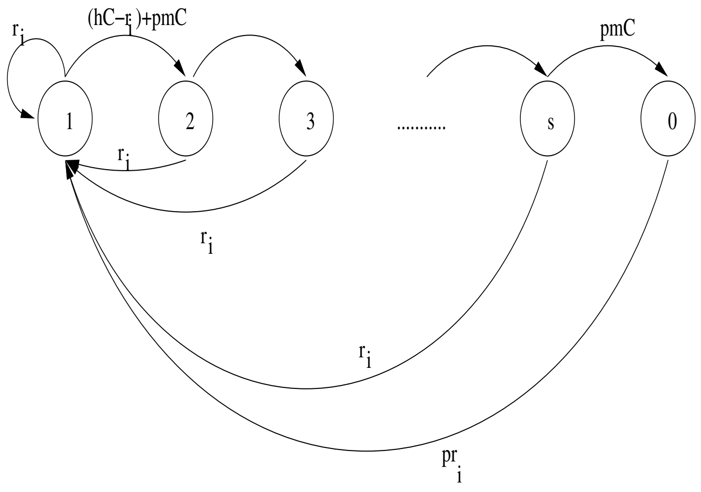

Fig. 3. Markov chain for a flow location

finding this flow in the j^{th} location in the cache and p_0 as the probability of not finding the flow in the cache. The flow's location can be modeled as a Markov chain as shown in Figure 3. Let h be the fraction of the traffic that has hits in the cache and m be the miss fraction. If p is the probability of admitting a flow on a cache miss, then the flow will move from location j to $j+1$ at a rate given by $(hC - r_i) + pmC$, C being the capacity of the link. Once the flow is in the last location, s, of the cache, it may be evicted at a rate of pmC. From any given location j in the cache, the flow can be arrive at location 1 in the cache at a rate r_i and can find entry in the cache at a rate of pr_i.

Now, from the Markov chain, we get the following:

$$pr_i \times p_0 = pmC \times p_s$$

$$\Rightarrow p_0 = \frac{mC}{r_i} p_s \tag{1}$$

$$(hC - r_i + pmC) \times p_j = p_{j+1}(hC - r_i + pmC + r_i)$$

$$for \quad j = 1, 2, .., s-2$$

$$\Rightarrow p_{j+1} = \frac{hC + pmC - r_i}{hC + pmC} p_j \tag{2}$$

$$for \quad j = 1, 2, .., s-2$$

$$(r_i + pmC) \times p_s = (hC + pmC - r_i) p_{s-1}$$

$$\Rightarrow p_s = \frac{hC + pmC - r_i}{r_i + pmC} p_{s-1} \tag{3}$$

From Equation 1,

$$p_s = \frac{hC + pmC - r_i}{r_i + pmC} \times k^{s-2} p_1 \tag{4}$$

where

$$k = \frac{hC + pmC - r_i}{hC + pmC} \tag{5}$$

TABLE I

CACHE SIZE ANALYSIS

hit	$\frac{r_i}{C}$				
fraction(h)	0.01%	0.05%	0.1%	0.5%	1%
0.01%	1705.5				
0.05%	1738.9	344.36			
0.1%	1780.5	352.67	174.20		
0.5%	2113.8	419.17	207.35	37.95	
1%	2530.4	502.30	248.80	46.09	20.81
5%	5863.2	1167.20	580.23	110.99	52.64
10%	10029.0	1998.00	994.25	191.83	92.08

Given $\sum_{i=0}^{s} p_i = 1$, we have

$$\frac{mC}{r_i} p_s + p_1 + kp_1 + k^2 p_1 + .. + k^{s-2} p_1 + p_s = 1$$

$$\Rightarrow \frac{mC}{r_i} p_s + (1 + k + k^2 + .. + k^{s-2}) p_1 + p_s = 1$$

After some simplification, we can get p_s and then p_0 from Equations 1 and 4 as

$$p_0 = \frac{mC \times k^{s-1}}{r_i + pmC + mCk^{s-1}(1-p)}$$

$$k^{s-1} = p_0 \frac{(r_i + pmC)}{mC\Big(1 - p_0(1-p)\Big)} \tag{6}$$

From Equation 6, we can find the cache size s required to hold a flow of rate r_i with any target probability p_0 of not finding it in the cache. As can be seen from above, the required cache size is a function of traffic distribution (represented by h and m in the equations above). Table I shows the required cache size s for various target flow rates ($\frac{r_i}{C}$) and different hit/miss rates. The miss probability is kept constant at 0.01 and the admittance probability at 1/50.

We find that in most cases, the required cache size is only a small factor larger than the number of flows that we could find ($\frac{r_i}{C}$) at a given target rate (r_i). This shows that LRU identification scheme is quite efficient in meeting stated requirements.

E. The Linux Design space

Linux offers a layer-based IPv4 network stack [26], [27]. Routers are mainly involved in the task of packet forwarding. Consequently, three layers are of importance - physical layer, link layer and network layer. Figure 4 shows the various steps involved in forwarding a packet.

Since most decisions are taken at the output queue, the design space for our scheme includes the queue enqueue and dequeue events.

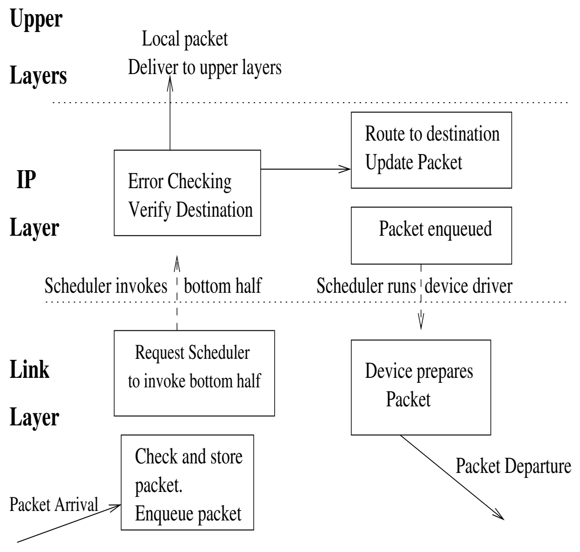

Fig. 4. IP Forwarding in Linux

Traffic control is a phase which comes immediately after a packet has been prepared for transmission by the network layer . Traffic control makes decisions on whether to queue the packet, the order(priority) of packet transmission, etc. Packets released by the traffic control are directly picked by the network device driver for transmission.

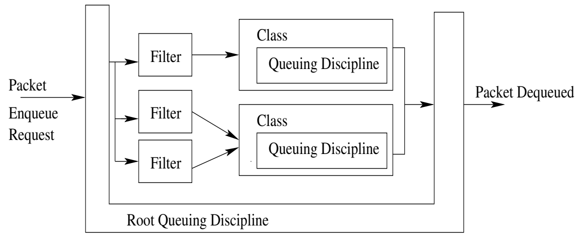

Fig. 5. Linux Traffic Control Architecture

Linux provides a modular architecture for QoS [28], [29]. Figure 5 shows the typical layout of the components of the Linux QoS, namely, queuing disciplines, classes and filters.

Queuing discipline controls how the packets are handled on a particular output interface. A queuing discipline can be a rudimentary FIFO (First In First Out) queue or can have multiple queues with packets being classified by filters. *Filters* are classifiers which divide the outbound traffic into *classes*. The classification can be based on type of traffic, destination or source addresses, protocol information etc. Availability of complex filters like u32 allow the router to be set up with complex parsing rules.

The Linux diffserv architecture is scalable to an extent that each queuing discipline can further classify its set of packets. Parameters of individual components of the traffic control are controlled using a user level application name *tc*.

F. Design: Challenges and choices

Given the modular nature of Linux diffserv architecture, we distribute the LRU-FQ scheme implementation among the various QoS components. Maintenance of the LRU cache and taking decisions on which queue to enqueue the outbound packets on 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. The existing fair queuing disciplines offered by Linux diffserv operate on flows within a queue. Since we need the fair queuing to operate across multiple queues, we implemented a class based fairness queuing discipline.

Queues belonging to this discipline are served using the Start-Time Fair Queuing [4] algorithm (STFQ). Since the fairness algorithm requires the maintenance of start tag and finish tag, we extend the skbuff data structure[1] to include this information.

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

G. Validation

In order to ensure that the scheme has been implemented as per design, we have run a set of tests to validate our implementation.

The issue of per-packet handling delay at the router can be studied by observing the throughput offered to a regular UDP flow. We introduced a controllable amount of delay in the forwarding path of the router code to study the amount of delay that can be introduced without causing inordinate loss of throughput. The delay code was introduced in the function *ip_route_input* function located in the file */linux/net/ipv4/route.c*.

[1] *skbuffs* are the Linux implementation of Socket buffers which are used to carry Protocol Data Units (PDUs)

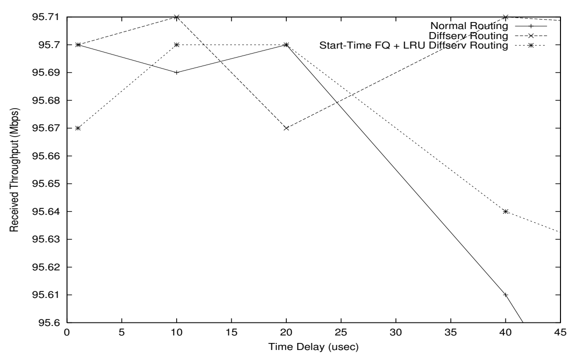

Fig. 6. Timing Study of LRU-FQ

Figure 6 shows the results obtained by running the experiment on a 100Mbps network with no cross traffic. As can be seen from the graph, the addition of our scheme does not cause a significant difference in the offered throughput compared to the existing schemes. Thus, we meet the initial goal of not reducing the throughput.

III. Experimental Setup and Results

Given the aim of containing high bandwidth non-responsive flows, the experiments conducted in this paper are focussed on studying the effects of LRU-FQ parameters on its performance, and the amount of control achieved on the non-responsive component in presence of both long term and short term flows.

A. *Experimental Test bed*

All experiments were conducted on a Linux router based on 2.4 version of the Linux kernel[2](Table II).The machines used to test the LRU-FQ implementation on Linux were connected on a private network as shown in Figure 7. 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 and TCP traffic was seperated 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.

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

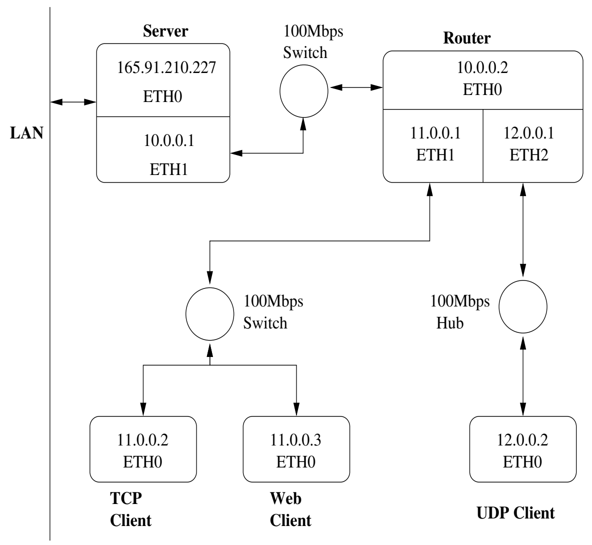

Fig. 7. Experimental Setup

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.

Given that the link load is constantly high, it is expected that the UDP flows occupy the top entries in the LRU cache and hence consistently remain in high-bandwidth queue. This hypothesis is substantiated by the results seen in Figure 8. The Partial State Queue is predominantly occupied by non-responsive flows: this is because the top cache entries are usually occupied by UDP and the bottom cache entries are constantly modified by the responsive flows without crossing the threshold.

Ideally, the non-responsive traffic is limited to the set limit. As we can observe, 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.

TABLE II
MACHINE CONFIGURATIONS

	CPU	RAM	Swap space	Kernel
Server	Pentium 4 1.70GHz	256MB	1 GB	2.4.17
Router	AMD K-6 500MHz	128MB	500 MB	2.4.17
Web Client	AMD K-6 500MHz	128MB	176 MB	2.2.16-22
TCP Client	Pentium 166MHz	48MB	256 MB	2.4.17
UDP Client	Pentium 166MHz	48MB	128 MB	2.4.17

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.

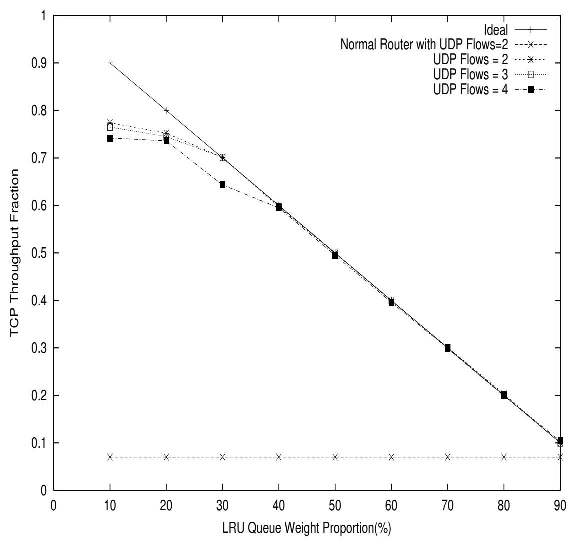

Fig. 8. Effect of non-responsive long term flows

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 9 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

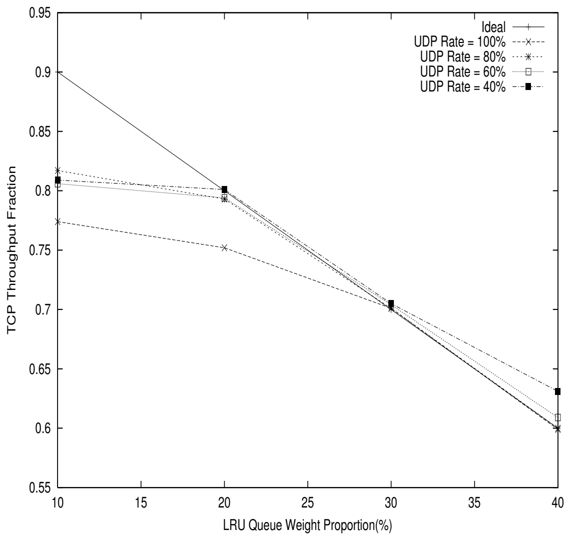

Fig. 9. Effect of varying non-responsive flow rates

use of the Webstone[30] 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 web traffic was governed by the frequency histogram shown in Figure 10. 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 III. 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 in the observation period. The long term TCP flows are still getting the assigned proportion of the bandwidth.

The corresponding results for the delays observed are shown in Table IV[3]. Connection time is the time taken by

[3]Avg, Dev, Min and Max imply Average, Standard Deviation, Mini-

TABLE III
BANDWIDTH RESULTS FOR WEB MICE

	UDP Flows	UDP Throughput	# of Web Requests	TCP Throughput	TCP Fraction
Normal 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

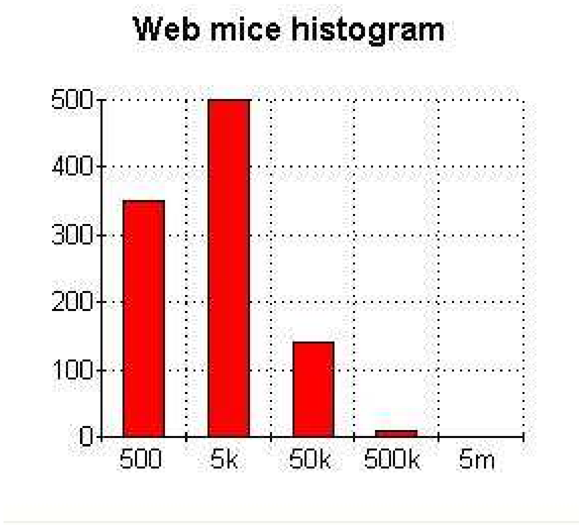

Fig. 10. Web mice file histogram

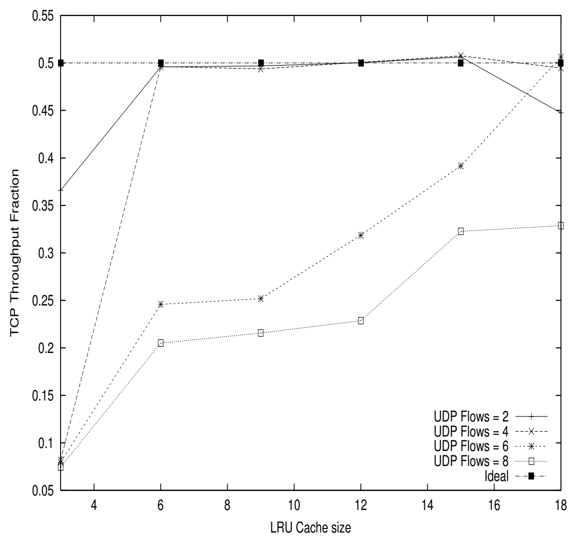

Fig. 11. Effect of cache size on LRU-FQ scheme

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 scheme for real-time data than normal routers.

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. The graph in Figure 11 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.

For lesser 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. It has been observed empirically that choosing a cache size larger than thrice the number of non-responsive flows provides a good performance.

mum and Maximum respectively

The other aspect is the choice of excessive cache size which will result in a large portion of responsive traffic also being treated as part of the LRU cache thus resulting in degraded performances. 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

TABLE IV
DELAY RESULTS FOR WEB MICE

	UDP	Response Time *sec*				Connection Time *sec*			
	Flows	Avg	Dev	Min	Max	Avg	Dev	Min	Max
Normal 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

hinder flows which are either responsive or using atmost their share of the link bandwidth. For this purpose, the following experiments were conducted.

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 fairness weights for both queues. The remaining parameters were configured to be cache size = 9, threshold = 125 and probability p = 1/55. Table V shows the corresponding results.

The results substantiate the claim of responsive loads not being adversely effected by the scheme with aribitrary parameters. The other normal scenario would be the case where non-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 VI shows the corresponding results. The results show that the scheme does take the sending rate into consideration.

IV. CONCLUSIONS AND FUTURE WORK

This paper presented the various applications of partial state based routers. A novel scheme, LRU-FQ, is presented which exploits a partial state to provide better flow isolation and better delay seen by web traffic. The scheme has been shown to be practically feasible and provides the network administrator the control of the actual amount of non-responsive traffic flowing through the router. Our Linux implementation shows that the cost of partial state based routers is negligible on packet forwarding throughput.

The scheme can be extended further to study aggregate traffic instead of flow based studies. For example, source address based aggregation can help in identifying DoS attacks from a single network. The scheme can be extended to detect the presence of non-responsive traffic in order to dynamically activate the scheme thus reducing processing overhead when no such traffic is present.

The current design of LRU-FQ makes it a policy driven scheme wherein the network administrator decides the proportion of non-responsive traffic to be contained. Detection of non-responsive fraction of the traffic could automate the parameter tuning of the scheme. Implementation of partial state mechanisms on network processors is currently being investigated.

REFERENCES

[1] Sally Floyd and Kevin Fall, “Promoting the use of end-to-end congestion control in the Internet,” in IEEE/ACM Transactions on Networking, pp. 458-472, August 1999.

[2] Inkoo Kim, “Analyzing Network Traces To Identify Long-Term High Rate Flows,” Technical report TAMU-ECE-2001-02, Texas A&M University, Computer Engineering Dept., May 2001.

[3] A.K. Parekh, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks,” PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[4] Pawan Goyal, Harrick M. Vin and Haichen Cheng, “Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks,” in IEEE/ACM Transactions on Networking, pp. 690-704, October 1997.

[5] Smitha and A.L.N. Reddy, “LRU-RED: An active queue management scheme to contain high bandwidth flows at congested routers,” in Proceedings of Globecomm, November 2001.

[6] D. Tong and A.L.N. Reddy, “QoS enhancement with partial state,” in Proceedings of International Workshop on QoS, UCL, London, pp. 87-96, June 1999.

[7] S. Ben Fred, T. Bonald, A. Proutiere, G. Regnie, and J.W. Roberts, “Statistical bandwidth sharing: a study of congestion at flow level,” in Proceedings of ACM SIGCOMM conference, pp.111-122, August 2001.

[8] Carlos Cunha, Azer Bestavros and Mark Crovell, “Characteristics of WWW Client-based Traces,” Technical report BU-CS-95-010, Boston University, CS Dept., July 1995.

[9] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” in IEEE/ACM Transactions on Networking, pp. 397-413, August 1993.

[10] W. Feng, D. Kandlur, D. Saha and K. Shin, “A New Class of Active Queue Management Algorithms,” Technical report CSE-TR-387-99, U. Michigan, CS Dept., April 1999.

TABLE V
LRU-FQ UNDER NORMAL WORKLOAD

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

TABLE VI
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

[11] Rong Pan, Balaji Prabhakar and Konstantinos Psounis, “CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation,” in Proceedings of IEEE Infocomm, pp. 942-951, March 2000.

[12] B. Suter, T.V. Lakshman, D. Stiliadis and A.K. Choudhary, “Design Considerations for supporting TCP with per-flow queuing,” INFOCOMM’98.

[13] Dong Lin and Robert Morris, “Dynamics of Random Early Detection,” in Proceedings of ACM SIGCOMM conference, pp. 127-137, September 1997.

[14] K. Nichols, V. Jacobson and L. Zhang, “A Two-bit Differentiated Services Architecture for the Internet,” draft-nichols-diff-svc-arch-00.txt, INTERNET DRAFT, December, 1997.

[15] Ion Stoica, Scott Shenker and Hui Zhang, “Core-Stateless Fair Queuing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks,” SIGCOMM’98.

[16] T.J. Ott, T.V. Lakshman and L.H. Wong, “SRED: Stabilized RED,” in Proceedings of IEEE Infocomm, pp. 1346-1355, Vol.3, March 1999.

[17] Ratul Mahajan, Sally Floyd and David Wetherall, “Controlling High-Bandwidth Flows at the Congested Router,” in Proceedings of International Conference on Network Protocols (ICNP), November 2001.

[18] Cristian Estan and George Varghese, “New Directions in Traffic Measurement and Accounting,” in Proceedings of SIGCOMM Internet Measurement Workshop, 2001.

[19] Rong Pan, Lee Breslau, Balaji Prabhakar and Scott Shenker, “Approximate fairness through differential dropping,” in ACM SIGCOMM Computer Communication Review, Volume 32, Issue 1, January 2002.

[20] “SYN Cookies,” Available online at http://cr.yp.to/syncookies.html

[21] C. L. Schuba, I. V. Krsul, M.G. Kuhn, E.H. Spafford, A. Sundaram and D. Zamboni , “Analysis of a Denial of Service Attack on TCP,” in Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, May 1997.

[22] “Packeteer Inc. White Papers,” Available online at http://www.packeteer.com/solutions/resources.

[23] P. Ferguson and D. Senie, “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Address Spoofing,” Internet Draft, January 1998, http://www.landfield.com/rfcs/rfc2267.html.

[24] Steve Bellovin, Marcus Leech and Tom Taylor, “ICMP Traceback Messages,” Internet Draft, October 2001, http://www.ietf.org/internet-drafts/draft-ietf-itrace-01.txt.

[25] Thorner M. Gil and Massimiliano Poletto, “MULTOPS: a datastructure for bandwidth attack detection,” in Proceedings of the 10th USENIX Security Symposium, Washington, D.C., August 2001.

[26] Glenn 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

[27] David Rusling, “The Linux Kernel,” January 1998, Available online at http://www.tldp.org/LDP/tlk/tlk.html

[28] Werner Almesberger, Jamal H. Salim and Alexey Kuznetsov, “Differentiated Services on Linux,” June 1999, Internet Draft <draft-almesberger-wajhak- diffserv-linux-01.txt>.

[29] Werner Almesberger, “Linux Traffic Control - Implementation Overview,” April 1999, Available online at ftp://icaftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz

[30] Mindcraft Incorporated, “Webstone 2.5 benchmark,” Available online at http://www.mindcraft.com/webstone/.