QoS enhancement with partial state *

Deying Tong
A. L. Narasimha Reddy
Dept. of Elec. Engg.
Texas A & M University
College Station, TX 77843-3128
dytong,reddy@ee.tamu.edu

Abstract

Considerable work has been done in devising mechanisms for providing service guarantees within a network. These schemes can be broadly classified into two categories, schemes that require maintaining state for each flow and schemes that do not require maintaining state for each flow within the network. Both the approaches have their advantages and proponents. This paper looks at a scheme, that falls in between these two extremes, where a network switch may be able to maintain state for a fixed number of flows (possibly less than the number of flows it serves). This paper looks at the services that can be provided by a limited amount of state. As a first step, it presents SACRED, a method that employs Sampling and Caching in addition to RED at a router to enhance the QoS. The proposed mechanism uses caching to deal with the limited amount of state. and uses sampling to select flows. It is shown that this approach can be effective in containing non-responsive flows. It is also shown that SACRED is scalable in the sense of providing increased function with increased amount of state.

1 Introduction

Considerable work has been done in finding mechanisms for providing service guarantees over networks. Fair-queueing and its many variants [1, 2, 3, 4, 5], consider scheduling at the switch/router within the network as the basic mechanism for providing these services. These mechanisms require that state information per each flow be maintained for deciding the scheduling order at the switch. Some of the mechanisms based on buffer management also require that the switch maintain a per-flow state in order to provide guarantees [6, 7].

Recently, there has been a strong desire to find mechanisms for providing service guarantees where per-flow state need not be maintained in the network. The interest in aggregate mechanisms are based on two motivations: (a) maintaining state per each flow may not be scalable to large networks and (b) to minimize the per-packet handling cost in the network such that routers can be scaled to very high speeds. Proposals for differentiated services [8, 9] based on different drop preferences have been receiving significant interest. These mechanisms classify packets as in-profile (IN) or out-of-profile (OUT) at the ingress points of the networks based on service contracts. The OUT packets are dropped earlier than IN packets when there is congestion in the network such that throughput guarantees can be provided for the IN traffic. These mechanisms do not require that the switches/routers in the network maintain any state except at the ingress routers. Other mechanisms such as [10] do not require maintaining state within the network as well.

Thus, the current work on providing service guarantees can be classified into two categories: (a) those that require maintaining state information

*This work was supported in part by a Texas ATP grant and by an NSF Career Award.

per each flow through the network and (b) aggregate mechanisms that do not require per-flow state. This paper looks at the possibilities in the middle. What kind of services can be provided if the network can only maintain partial state i.e., maintain state for only limited number of flows at any given time? Limited or partial state here means that we can maintain state only for a partial or limited number of flows and not to the amount of state information maintained per each flow.

Stochastic Fairness Queueing [11, 12] addresses a somewhat similar problem of fair queueing with a limited number of queues. SFQ is shown to provide increased fairness over FCFS [12]. Our approach is different as we point out later in section 6.

Our motivation for this work is based on a need to find mechanisms for containing non-responsive flows in an internet. It has been shown that a non-responsive UDP flow can take disproportionate amount of the bandwidth at a congested link when all the other flows are TCP flows [13]. A non-responsive flow can claim a considerable amount of bandwidth when the network employs aggregate mechanisms.Mechanisms that maintain per-flow state can isolate such flows so as to minimize their impact on the other flows at the switch. Is it possible to maintain a limited amount of state to contain such flows if such non-responsive flows constitute only a limited fraction of the total workload at a switch?

This paper studies the question of what kind of services can be provided by a limited amount of state in the switch. Is it possible to contain non-responsive flows by maintaining fixed/limited amount of state at the switch? Is it possible to provide service guarantees with limited amount of state? Specifically, this paper proposes SACRED, a mechanism for providing services with limited amount of state. We provide an analysis of SACRED and evaluate it through simulations. We also discuss other possible enhancements that can be made possible though limited amount of state. SACRED is a first step in our efforts at identifying QoS-enhancement mechanisms that can work with a limited amount of state.

The rest of the paper is organized as follows. Section 2 proposes an approach, SACRED, where the limited state is used to maintain state for different flows at different times i.e., each flow's state is maintained over a fraction of the time. Section 3 discusses the details of a SACRED implementation and section 4 shows simulation results of SACRED's performance.Section 5 presents a simple analysis of SACRED, section 6 briefly discusses some related work and section 7 concludes this paper.

2 SACRED: An approach to use limited state

There are a number of ways to use the partial state space. In this section, we propose one approach to utilize limited amount of state. In this approach, the state space is used to maintain state for different flows at different times. Depending on time, the same space may be utilized by multiple flows. In other words, we use the limited state space to maintain state intermittently for all the flows. This raises a number of questions: (a) how to identify flows that are using the state space? (b) how to deal with the fact that we have multiple flows that can map into the same space in the state space? (c) what kind of state information can we maintain if we can only maintain this information intermittently? (d) what kind of service can we provide with such intermittent maintenance of state? (e) does the provided service scale as we increase the amount of state? We answer these questions below.

With partial state at the switch, it is necessary to quickly identify if state is being maintained at the switch for a flow at any given time. To do this quickly, we use caching. We will assume a simple function based on the source, destination addresses (and if necessary, ports) will be used to index into the cache. For example, (source IP address XOR destination IP address) *mod* number of cache entries can be used. If the number of entries in the cache is a power of 2, this function is very easy to compute. Since there is a possibility that many flows can map into the same index, we have to cache the source IP address and destination IP address to make sure that we are dealing with the correct flow. This is very similar to how caches are used in processors. When a packet arrives at the switch, the cache index is computed and the entries (source and destination addresses) in the cache at that index are compared with that of the arriving flow. No action need be taken if the flow is not cached.

Since multiple flows can map into a cache location, we need a mechanism for determining the flow whose state will be maintained in the cache. We use sampling for this purpose. Arriving packets are sampled at random. The first packet that maps into a cache location determines the flow whose state will be maintained in the cache. The state for this flow will be maintained for a duration. After some time, this cache location may be used by other flows. At that time, random sampling is used again to determine the next flow for which state will be maintained in the cache.

With these two techniques, sampling and caching, it is possible to maintain state for a limited number of flows at any time. What kind of state information can we maintain if we observe a flow intermittently? This state information cannot depend on the history of the flow since we are not observing the flow at all times. The state information has to rely on the current (or instantaneous) characteristics of the flow. For example, we could answer the sending rate of a flow by observing the packets passing through the switch. But, we wouldn't be able to say how many buffers at the switch are exactly occupied by this flow. Since our initial motivation was to contain non-responsive sources, we will use flow rate as the information maintained in the cache.

How do we deal with a larger number of flows with only a limited number of entries at the switch. After observing a flow for some time and obtaining its state at the switch, we take some action on the flow if it is necessary. If it is not necessary to take any action on the flow, this entry can be used to maintain state for another flow. We can divide the operation of SACRED into three phases: observation, action and benefit phases. During the observation phase, a flow is observed to see if we need to take any further action during the action phase. The benefit phase corresponds to the period during which this flow need not be observed. The benefit phase depends on the action taken and the QoS we are trying to provide. The scalability of the approach depends on the relative length of the benefit phase compared to that of the other two phases. If the switch can maintain state for n flows at any time, then we can provide service for at most $n * \beta/\tau$ flows, where β is the length of the benefit phase and τ is the length of the other two phases. We will discuss more about this in the analysis section.

We use RED (Random Early Detection) as a way of controlling the service at the switch. However, the drop probabilities may be different for different flows. Specifically, if a flow is being observed/monitored, it may be treated differently and may have higher drop rates than other flows that are not being observed. Since our approach uses Sampling and Caching in addition to RED as a basis for providing service, it is termed SACRED.

3 Implementation details

In this section, we will provide a detailed discussion of possible implementation of SACRED. An ns-2 [14] based SACRED simulator is implemented following the details described in this section. The results from the simulation will be discussed in the next section.

Packets of cached flows only need to be processed by SACRED and with limited state, packets of only a fraction of the flows get processed. SACRED uses the following parameters *sampling threshold*, *dropping threshold*, and *observation period*. Packets flowing through the router are sampled when the router queue length is above the *sampling threshold*. When the router is not very busy and the queue lengths are below sampling threshold, no extra actions are taken by SACRED. A selected flow is observed at least for a time of *observation period*. During this time, the flow characteristics are observed. In this paper, we report on our experiments where we observed the flow's sending rate. To observe the flow's sending rate, we keep a count of the number of packets of this flow that are sampled during this observation period. This count is used to compute the flow's rate. If a flow is observed to exceed a *limit-share threshold*, this flow is marked for further observation. If not, this flow is dropped to make room for the observation of other flows. When the queue length exceeds the *dropping threshold*, the packets of marked flows are dropped at a higher rate than the other flows. We will explain below how the drop rates for marked flows are determined.

For our simulation experiments, we set sampling threshold to be below the min threshold of RED so that aggressive flows can be identified before packets need to be dropped. We set the dropping threshold equal to the min threshold such that when packets need to be dropped, more packets of aggressive flows can be dropped. Sampling is based on packet arrival and not based on a timer. It has been shown earlier that packet arrival based sampling gives better results than time-based sampling [15].

Each cache location contains the following information: the source address, the destination address to identify the flow (port numbers may be needed as well), a *count* field indicating the number of packets of this flow observed so far, and the time of expiry for this entry. Each cache location also has some status information: *valid* bit indicates if this entry is being currently used and *pinned* bit indicates that the cached flow is selected for further observation. Each cache location also has a field called *drop weight* which is used to control the drop rate of pinned flows as explained below. When a packet is sampled, the cache index of the sampled flow is computed as a function of its header (source address XOR destination address % number of cache entries may suffice). The cache is accessed using this index. If the entry in the cache at that location is for this flow, the count field is incremented. If the current time is past the time of expiry of this entry, the sending rate of this flow is calculated by flow's rate = packet count/(observation time) The observation time may be slightly larger than the observation period because of the randomness involved in packet arrivals. If the computed flow rate is above the limit-share, this cache entry is pinned and the drop weight is increased. If it is not, the entry is marked invalid.

If an invalid cached entry does not belong to an arriving packet, it is replaced by the new flow and marked valid. The cache entries are initialized by marking the count to 1, the expiry time to clock + observation period and drop weight to 1. Otherwise, SACRED takes no action on the sampled packet. When a sampled packet results in marking a flow for further observation, that packet is dropped. The count field and the expiry time fields are initialized to observe this flow for another observation period. If it is observed that this flow's rate is below limit-share during the next observation period, this flow is removed from the cache by marking its entry invalid. If it is observed that this flow's rate remained above the limit-share during the next observation period, the drop weight is increased. This is continued as long as the flow's rate remains above the limit-share threshold.

Packets for individual flows (if cached) are dropped at a higher rate than the other flows. The drop rate for cached flows is calculated by multiplying the RED drop probability with the drop weight of that flow. Thus, flows observed to be sending at a high rate are dropped at a higher rate.

A number of factors influence the effectiveness of SACRED. First, sampling results in picking flows at random for observation. It may take some time for aggressive flows to be picked for observation. Second, when aggressive flows are picked for observation, they may collide in the cache with another flow already being monitored. Aggressive flows are sending more packets than other flows and hence they will be picked for observation with higher probability than other flows. To reduce the problem of cache collisions, we could employ higher (2-way or 4-way) set-associative caches. When a non-aggressive flow is picked for observation, the cache entry becomes vacant within an observation period making it possible to observe other flows quickly. Because of the randomness involved, the "gestation time" (time to pick the right flows for observation) will be measured. The gestation time will give an indication of how long the flows have to exist for SACRED to be effective. We also give an analytical expression in the analysis section. Our analysis shows that with a state to monitor 10% of the flows, an aggressive flow will be picked with a very high probability (95%) within 25 observation periods. With an observation period of 100 ms, a flow will be picked within 2.5 seconds with a very high probability.

All the actions in SACRED are triggered by packet arrivals. Actions for a flow are initiated only on an arrival of that flow's packet. The work done per sampled packet is constant and does not depend on the number of flows or the number of cache entries. If a sampled packet is not cached, only a cache lookup is needed. A cache lookup can be implemented within a few nano seconds. If the sampled packet is cached, we do an addition (count), subtraction (to check if observation period is over) and a division at the end of the observation period. These actions can be implemented fairly quickly and hence the per-packet handling cost is small.

4 Simulation Results for SACRED

In this section, we present the results obtained through simulations based on an implementation of SACRED in ns-2. The simulations used a network setup as shown in Fig. 1. We used 40 TCP sources with 5 different RTTs and 2 UDP sources. The UDP sources transmitted data at 6Mbs and didn't respond to congestion. The link bandwidth was set at 50 Mbs and router R1 employed SACRED. We set the SACRED parameters (# of cache entries, sampling threshold, dropping threshold, observation period, limit-share bandwidth) to (4, 5, 10, 200ms, 1.5Mbs). This corresponds to maintaining state for about 10% of the flows. The RED parameters of (min threshold, max threshold, p_{max}) are set to (10, 30, 0.02). RED randomly drops packets when the average queue length is between the min and max thresholds with a probability given by $p_{max}(qlen - min)/(max - min)$, where $min \leq qlen \leq max$. The results of the simulation are shown in Fig. 2. It is observed that the 2 UDP flows are contained to be below the limit-share bandwidth. The 2 UDP flows achieve considerably more bandwidth in DropTail and RED routers. The TCP flows achieve better throughput with SACRED than in the other schemes.

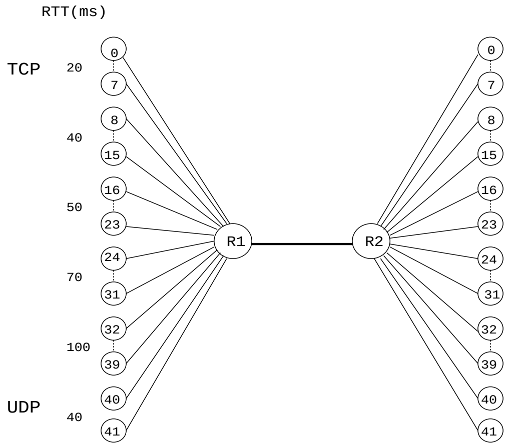

Figure 1: Network setup for simulations.

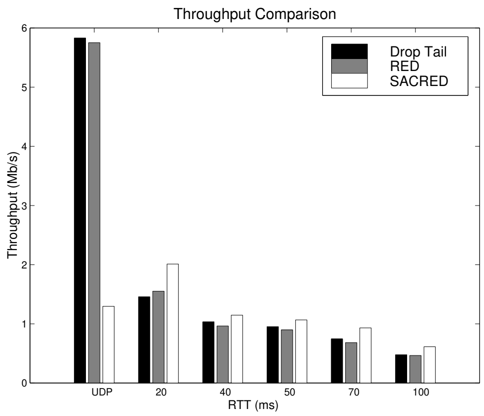

Figure 2: Containing UDP flows with 10% state.

Fig. 3 shows the throughput of the flows as a function of time. It is observed that the UDP flows are picked for observation very shortly after the start of simulation. It is also observed that the drop rates are progressively increased until the UDP flows are below the limit-share bandwidth. The UDP flows are below 3 Mbs within 3 seconds (i.e., a drop rate of 50%) and below the limit-share bandwidth within 10 seconds.

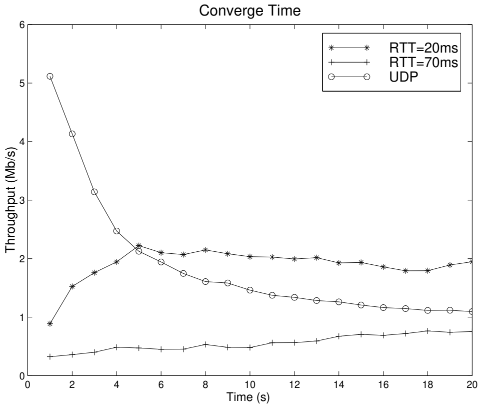

Figure 3: Time to converge to limit-share rate.

It is clear from these experiments, that UDP flows can be quickly identified and contained to within a limit-share bandwidth with a limited amount of state at a router. We used 40% state here. We experimented with several other situations. Specifically, we chose the ids of UDP flows such that they would collide in the cache table of SACRED. The results (throughput in Mb/s) of this simulation are shown in Table. 1. SACRED can only monitor one of the UDP flows at a time and as a result, the effective throughput of the other UDP flow is much higher than earlier.

RTT(ms)	Flow ID	Direct Map	2 Set Assoc.
20	0	3.381	3.5718
20	1	3.849	5.9406
40	2	2.6682	3.204
40	3	2.568	3.0342
50	4	2.6178	2.7228
50	5	2.0946	2.2668
70	6	1.6266	2.2428
70	7	1.746	1.8684
100	8	1.1748	1.7322
100	9	1.6098	1.3272
UDP	11	5.94048	2.3568
UDP	15	2.61648	2.1096

Table 1: Direct Map vs 2 Set Associative

To address such situations, we could use higher associativity caches such that cache collisions will be rare. The results of simulation with a 2-way set-associative cache are shown in Table. 1. With a direct-mapped cache of 4 entries, SACRED maps two UDP flows into two different entries and they both will be contained.

Even without a 2-way set-associative cache, it is likely that the UDP flow will be contained at a second router as it passes through the network. To test this hypothesis, we used a network setup as shown in Fig. 4. The network consists of 3 routers where traffic from two routers converges into a third router. Our simulation consisted of 40 flows: 8 UDP flows, 8 TCP flows each at different RTTs of 20ms, 40ms, 50ms, and 70ms. The parameters are set to be the same as before while the bandwidth between R1/R2 and R3 is 20 Mbs and outgoing link of R3 is 33Mbs. The limit share for each flow is 1 Mbs and the cache space allows R1/R2 to monitor 2 flows and R3 to monitor 4 flows. The simulation results are shown in Table. 2. It is observed that the TCP flows achieve almost no bandwidth with RED and DropTail routers. With SACRED, the UDP flows are contained to be within their limit-share bandwidths. The TCP flows achieve considerably more bandwidth with SACRED than with RED and Droptail.

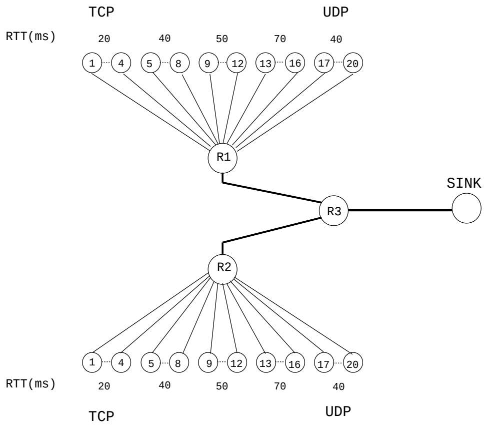

Figure 4: Three routers example.

These experiments show that it is possible to use limited amount of state to effectively curtail non-responsive flows while allowing responsive flows to achieve higher throughput.

What kind of service can SACRED provide with increased amount of state? To explore this question, we kept increasing the amount of state at

RTT(ms)	DropTail	RED	SACRED
UDP	3.64	3.70	1.73
TCP/20	0.03	0.01	0.63
40	0.02	0.02	0.47
50	0.04	0.01	0.50
70	0.01	0.01	0.42
Total	29.99	29.99	29.96

Table 2: Throughputs with a network of routers(Mb/s)

the router to see if we could achieve fair sharing of bandwidth across all the flows. First, we experimented with 100% state. Fig. 5 shows the throughput of 10 TCP flows and 1 UDP flow with 100% state. Again, the results show that the SACRED contains the UDP flow. With RED and DropTail, the flows with 20ms RTT achieve substantially more bandwidth than the other flows with larger RTTs. SACRED realizes a more fairer distribution of bandwidth. This is achieved by monitoring TCP flows with short RTT as well when cache space is available.

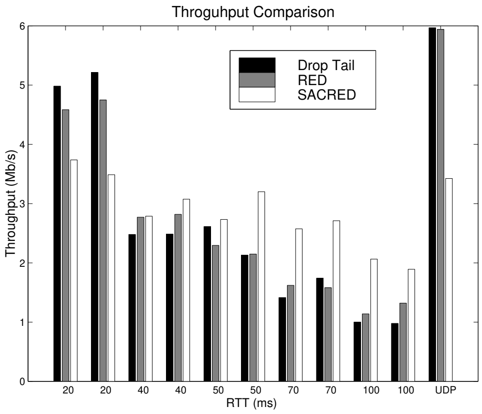

Figure 5: Full state simulation.

To study the performance of SACRED under pure TCP traffic, we compare the throughputs of 10 TCP flows with 60% and 20% state to the full state case in Fig. 6 We used a network similar to Fig. 4 with the link bandwidth of 33Mbs and 10 TCP flows at different RTTs(20,40,40,70 and 100ms). Limit-share bandwidth was set to 3.5Mbs. The results show that with about 20% state, SACRED doesn't perform significantly different from RED. As the state is increased to 60% and 100%, SACRED results in improved fair distribution of bandwidth.

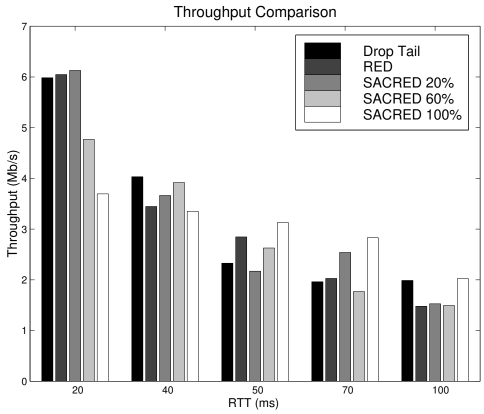

Figure 6: TCP only simulation.

However, when multiple SACRED gateways are present in the network, we observe additive effect for balancing TCP traffic. To show this, we use a network topology shown in Fig. 4. In addition to using Drop Tail and RED as references, we first set R3 to be SACRED with 25% state and R1, R2 as RED. In another experiment, we employ SACRED with 25% state at all the routers R1, R2 and R3. Results are shown in Fig. 7. From the chart, we observe that 3 SACRED routers cooperate with each other and get much fairer distribution of bandwidth. This shows that even with partial state, when employed across a number of routers, better QoS can be obtained.

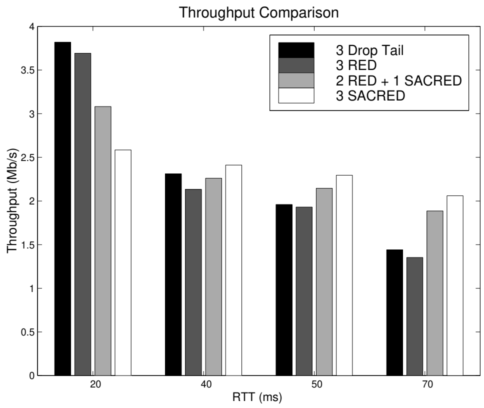

Figure 7: TCP only simulation with 3 routers.

5 Simple analysis of SACRED

In this section, we provide a simple analysis of SACRED to understand its effectiveness in monitoring a larger number of flows than the limited amount of state at the switch. The simulations have clearly shown that it is possible to identify non-responsive UDP flows and that SACRED is effective in containing those flows to a limited share of the bandwidth at the switch. Since non-responsive flows continue to retain an entry in the cache, the number of flows that can be contained is limited to the number of entries in the cache. When there are more flows, as we have shown earlier through simulations, the non-responsive flows will be identified and contained at other switches in the network. The work done in identifying a non-responsive flow remains useful through the lifetime of that flow. The service provided by different routers is additive i.e., more number of routers can contain more number of non-responsive flows. This is an important attribute of SACRED. We will point out later that this requirement of additive property may impact the service that can be provided by the use of limited state.

How long does it take to find a non-responsive source? For a configuration of s entries in cache and n flows in total, the probability that a non-responsive flow was not monitored in one observation period is $1 - s/n$. So the probability of being sampled during xth observation period is given by

$$P(t = x) = (1 - s/n)^{x-1} s/n$$

which indicates that a flow can be detected before xth observation period by the probability of

$$P(t <= x) = \sum_{i=1}^{x} (1 - s/n)^{x-1} s/n = 1 - (1 - s/n)^{x}$$

Assuming the gateway can keep 10% state, it will catch an unresponsive flow in 25 observation periods with a probability higher than 93%. With an observation period of 100ms, this corresponds to a gestation time of 2.5 seconds.

For responsive flows, the benefit from identifying a responsive flow taking more than the limit-share doesn't last long. SACRED drops a packet of the flow once identified as taking more than limited share of the bandwidth. If the flow responds to a drop, it becomes a candidate for replacement to allow monitoring another flow. Hence, the benefit in identifying this responsive flow is the accompanying reduction of the sending rate of this flow. TCP flows slowly build up the sending rate after a packet drop until the next packet drop. Hence, the duration over which SACRED controls the rate of this flow is limited by the TCP's fast recovery mechanism. Let *cwnd* be the ideal congestion window of a flow for its share of bandwidth. Ideally, the flow is identified when its window reaches $4/3cwnd$ and as a result of the packet drop, its window is reduced to $2/3cwnd$ such that the flow rate oscillates around the ideal rate. In such a situation, the benefit of identifying a flow is then limited to the time it takes for the flow to build its window from $2/3cwnd$ to $4/3cwnd$. In fast recovery, TCP increases the window by 1 for every RTT. Hence, the time for recovery equals $RTT * 2/3 * cwnd$. Since it takes τ seconds to identify a flow, the maximum number of flows that can be monitored by SACRED is given by

$$n * 2/3 * cwnd * RTT/\tau$$

where n is the amount of state that can be maintained in number of flows. If a flow is sending at a rate of 1Mbs with a packet size of 512 bytes and its RTT is 50ms, its ideal cwnd can be calculated by cwnd/RTT = 1 Mbs/(512*8), cwnd = 12.8. With a τ of 2*RTT, then the number of flows SACRED can monitor is bounded by $12.8 * n/3 = 4.3 * n$. This indicates that SACRED can monitor about 4 times as many flows as the state allows.

Clearly, the above equation gives an upper bound on the effectiveness of SACRED. The effectiveness can be lowered due to: (a) an identified flow not sending above the limit-rate, (b) collisions within the cache for monitoring flows and (c) the flows not being identified at the ideal times. The above equation shows that the effectiveness decreases for flows that have shorter RTTs and that use larger packets (resulting in smaller cwnd). The effectiveness of SACRED can be increased by the following techniques.(1) More than one packet can be dropped at a time such that the benefit of monitored flow can be extended. This, however, leads to larger variations in the instantaneous bandwidth realized by a flow. For example, dropping two packets at $8/5 * cwnd$ reduces the flow rate to $2/5 * cwnd$ increases the recovery time to $6/5 * cwnd * RTT$ and hence increases the effectiveness bound of SACRED to $n * 6/5 * cwnd * RTT/\tau$. (2) Reducing τ and allowing more impreciseness in the measured state. As mentioned earlier, the burstiness of TCP flows limits how small a τ we can use. With smaller values of τ, we may identify a flow to be incorrectly sending above the limit rate. (3) Using smaller packets such that TCP builds up the sending rate more slowly after a packet drop. Techniques (1) and (2) can be employed at a switch whereas (3) requires cooperation from the sender.

6 Related work

Considerable amount of work has been done in related areas. As mentioned in the introduction, a number of scheduling strategies [1, 2, 3, 4, 5], based on maintaining some form of state for each flow have been proposed. Buffer management policies that keep state for each flow have also been studied [6, 7]. Differentiated services framework proposes using stateless routers [8, 9] in the core of the network while limiting state maintenance to the edge routers of the network.

SFQ utilizes limited number of queues to improve fairness compared to FCFS scheduling. SFQ dynamically aggregates flows(when collisions are unavoidable) in a queue. SFQ uses a large number of queues, 5-6 times the number of flows, to achieve fast queueing of packets from different flows. Main motivation of SFQ is fast execution of fair scheduling. Our work here deals with utilizing state that is a fraction of the number of flows through the router. Recently, a scheme for fair sharing of bandwidth has been proposed that utilizes rate measurement at the edge and flow-based drop rates within the core [16]. This scheme limits state maintenance to the edge routers to mark packets with the sending rates of the flows. These rate labels are used to drop packets within the core. It is shown that this scheme works well with UDP flows [16]. Our work has a similar motivation, but reaches the same goal through a different mechanism of employing partial state within a switch.

Class based queuing can be used to contain the effects of UDP flows on TCP flows. This however penalizes all the UDP flows whether they are responsive or non-responsive [17]. To isolate responsive UDP flows from non-responsive UDP flows requires further mechanisms such as tagging or packet marking [17]. Per-flow RED or FRED [18] can be used to contain non-responsive sources while achieving fair sharing of bandwidth. FRED however requires 100% state. TCP-friendly test [13] is suggested as a mechanism to contain non-responsive flows. This approach requires RTT measurements of flows and also requires that all the flows follow a TCP-style exponential decrease of rates when a packet is dropped.

7 Conclusion

In this paper, we have addressed the problem of providing QoS with limited or partial state at a switch. As a first step in this direction, we presented SACRED. SACRED uses sampling, caching and RED to provide QoS with limited state. SACRED uses partial state to provide state intermittently for a larger number of flows. It was shown that SACRED can be effective in containing non-responsive flows and allowing a fairer distribution of bandwidth among the flows. We have shown that SACRED is scalable in the sense that with increased amount of state, it provides improved QoS. We have also provided a simple analysis for the scalability of SACRED. We have also shown that SACRED is additive i.e., improved service can be provided as more switches/routers employ SACRED in the network. We are in the process of identifying other approaches to utilize partial state.

References

[1] A. Demers, S. Keshav, and S.Shenker, "Analysis and simulations of a fair queuing algorithm," *Proc.*

of ACM SIGCOMM, pp. 3–12, 1989.

[2] S. Golestani, "Duration limited statistical multiplexing of delay-sensitive traffic," *Proc. of INFOCOMM*, pp. 12–20, 1991.

[3] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: The multiple node case," *Proc. IEEE INFOCOMM'93*, pp. 521–530, Mar. 1993.

[4] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *Proc. of ACM SIGCOMM '95*, Aug. 1995.

[5] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switched networks," *Proc. ACM Trans. on Comp. Systems*, pp. 101–125, May 1991.

[6] R. Guerin, S. Kamat, V. Peris, and R. Rajan, "Scalable QOS provision through buffer management," *Proc. of ACM SIGCOMM '98*, Oct. 1998.

[7] B. Suter, T.V. Lakshman, D.Stiliadis, and A. Choudhury, "Design considerations for supporting TCP with per-flow queuing," *Proc. of INFOCOMM '98*, Apr. 1998.

[8] David D. Clark and Wenjia Fang, "Explicit allocation of best-effort packet delivery service," *IEEE/ACM Trans. on Networking*, pp. 362–373, Aug. 1998.

[9] 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*, Dec. 1997.

[10] H. Zhang and D. Ferrari, "Rate-controlled static-priority queueing," *Proc. IEEE INFOCOM*, Apr. 1993.

[11] A. Mankin, K.Ramakrishnan, "ateway Congestion Control Survey," *Request for Comments: 1254*, Aug. 1991.

[12] P. McKenny, "Stochastic Fairness Queueing," *Proceddings of SIGCOMM '90*, 1990.

[13] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *Under submission, http://www-nrg.ee.lbl.gov/floyd/end2end-paper.html*, Feb. 1998.

[14] University of California at Berkeley, "Network simulator (ns)," *Available via http://www-nrg.ee.lbl.gov/ns/*, 1997.

[15] K. C. Claffy, G. C. Polyzos, and H-W. Braun, "Application of sampling methodologies to network traffic characterization," *Proc. of ACM SIGCOMM '94*, 1994.

[16] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," *Proc. of ACM SIGCOMM'98*, Oct. 1998.

[17] M. Parris, K. Jeffay, and F. D. Smith, "Lightweight active router-queue management for multimedia networking," *Proc. of SPIE Conf. on Multimedia Computing and Networking*, Jan. 1999.

[18] D. Lin and R. Morris, "Dynamcis of random early detection," *Proc. of ACM SIGCOMM '97*, 1997.