A TCP-Friendly Traffic Marker for IP Differentiated Services *

Azeem Feroz † Shivkumar Kalyanaraman Amit Rao‡
Department of ECSE
Rensselaer Polytechnic Institute
Troy, NY 12180
feroza@rpi.edu shivkuma@ecse.rpi.edu raopa@rpi.edu

Abstract

The differentiated services architecture allows the provision of "better-than-best-effort" services in the Internet. However, the performance of legacy TCP applications over differentiated services is still influenced by bursty packet loss behavior. This paper proposes the use of TCP-friendly building blocks in the diff-serv architecture, and in particular, a TCP-friendly traffic marker to enhance TCP performance over assured service. We present the marker design and resulting performance improvements in terms of improved predictability of service, reduced provisioning requirements, better fairness and fewer TCP timeouts. Though the marker improves assured service performance predictability compared to a simple token bucket marker, the performance is still dependent to some extent on TCP dynamics.

1 INTRODUCTION

There is a distinct need to move towards providing service differentiation (diff-serv) in the Internet. The current best-effort model does not provide any means of preferentially treating traffic from customers who are willing to pay more. The differentiated service model uses a combination of network edge elements (traffic conditioners) and network core elements (per-hop behaviors or PHBs) to achieve service differentiation [2, 3].

In this paper, we consider a simplified form of a better-than-best-effort service called the "assured service"[1]. The building blocks of this service include a traffic marker at the edge of the domain, and a differentiated dropping algorithm in the network interior. The traffic marker marks packets as "IN" or "OUT" (corresponding to the two "colors") depending upon the service level agreement (SLA). An example of a differentiated dropping algorithm is "RIO" - a variant of the RED (Random Early Gateway) algorithm[4, 5]. The RIO algorithm ("RED with In and Out") uses the same RED algorithm for "IN" packets and "OUT" packets, albeit with a different set of parameters for each. In particular, the OUT packets are preferentially dropped upon evidence of congestion at the bottleneck before the IN packets. For example, a minimum rate of packets could be marked as "IN" in expectation of a minimum "assured rate" because the "IN" packets would have high probability of delivery.

Ibanez et al [8] showed that the use of a simple token bucket marker for the above assured service results in TCP not realizing the minimum "assured rate." The authors attributed the cause of such behavior to TCP's complex response primarily to packet losses. Specifically, part of the TCP flow was marked "IN" and a part marked "OUT." The overall performance was affected by the bursty losses being experienced by the "OUT" packets. The authors concluded that it was unclear under such circumstances how the "assured service" can be characterized quantitatively for TCP applications. Such problems can reappear in other types of better-than-best-effort services being designed by the diff-serv community.

This paper proposes a general approach to such problems in better-than-best-effort differentiated services - the use of "TCP-friendly building blocks". Specifically, we explore the meaning of the term "TCP-friendly" and develop one example building block, a TCP-friendly marker for the simple assured service. We show that it can significantly improve the service characteristics experienced by TCP applications such as *predictability of assured rates, reduced token provisioning requirements, improved fairness, fewer overall packet losses and reduced number of TCP timeouts*. This approach can be generalized for use

* This project was supported in part by NSF contracts: ANI9806660 and ANI9819112. There is a patent pending on this work; please contact shivkuma@ecse.rpi.edu for details

† Azeem Feroz is currently with Packeteer Inc.

‡ Amit Rao is currently with Niksun Inc.

in other better-than-best-effort services.

The rest of the paper is organized as follows: Section 2 identifies the problems faced by TCP over assured service and explores the meaning of the term “TCP-friendly”. Section 3 explores the design of TCP-friendly building blocks in general and the proposed TCP-friendly traffic marker in particular. Sections 4 and 5 discuss the performance of the TCP-friendly marker in relation to other alternatives such as the token-bucket marker. Section 6 summarizes and concludes the paper.

2 PERFORMANCE PROBLEMS OF TCP OVER ASSURED SERVICE

It is well known that TCP Reno (the large installed base of TCP implementations) has performance problems if a connection encounters a *burst loss of packets* i.e., if a connection sees a number of packet losses with nearby sequence numbers [9]. Specifically, three or more packets dropped in a window can lead to a *timeout* plus multiple SSTHRESH reductions. The distribution of these losses in the window is irrelevant for TCP Reno, i.e., the performance problems can occur with or without RED-type gateways as long as multiple losses are experienced by the same flow. Moreover, TCP transmission is bursty in the sense that blocks of packets are transmitted back-to-back followed by idle times. Even after multiplexing at queuing points, packets of multiple flows generally tend to exhibit a low degree of interleaving. As a result when they encounter a bottleneck, multiple successive packets of a single flow have a high probability of experiencing similar behavior, for example, get dropped.

Newer TCP Implementations like NewReno [7] and TCP SACK [6] address some of these problems by use of better end-to-end filtering, retransmission and feedback algorithms. NewReno requires support only at the TCP source (i.e. server-side upgrades), whereas TCP SACK requires support at both the source and the receiver. Therefore, in spite of aggressive deployment of these newer versions, the vast majority of TCP connections today are either TCP Reno or NewReno. Now both these TCP versions (Reno and NewReno) *timeout* when packet losses occur *and* the source window size is small. Such timeouts lead to performance degradation irrespective of the upgrade to NewReno. In other words, the definition of “burst loss” varies depending upon per-connection window size. Due to the short size of web-transfers and the reasonably large maximum segment sizes, the window size of a majority of transfers is small and are hence vulnerable to timeouts under such circumstances.

The effect of such packet losses and timeouts include service degradation as characterized by transfer times or throughputs, unfairness as characterized by spread of per-flow goodputs and a large number of packet losses and timeouts. *The goal of “TCP-friendly” building blocks is to reduce such negative performance effects*. Based upon the above discussion, the meaning of “TCP-friendly” is to provide one or more of the following features in the building blocks :

- promote the possibility of packet interleaving in the network interior,
- protect small-window connections from packet loss,
- convert aggregate burst loss at a bottleneck into widely-spaced non-bursty per-connection loss,
- reduce packet burstiness created by TCP in general

As characterized above, several well-known algorithms can be classified as “TCP-friendly.” For example, the RED packet dropping algorithm randomizes packet loss and provides a minimum average spacing between packet losses (though it operates on an aggregate of TCP flows). Packeteer’s TCP Rate control and other simple shaping algorithms can reduce packet burstiness caused by TCP [11]. The set of features mentioned above is not necessarily complete and can be extended. The term ”TCP-friendly” is also used by researchers in developing non-TCP traffic congestion control. But there, the emphasis is on fairness of bandwidth allocation between TCP and non-TCP traffic. Our emphasis here is to positively affect existing TCP performance. In this paper, we present a TCP-friendly traffic marker which satisfies some of the above criteria and results in significant performance improvement of TCP receiving assured service.

3 TCP-FRIENDLY PACKET MARKER

A packet marker is one of the traffic conditioners in diff-serv. The general problem in a marker is to optimally allocate an available pool of tokens to a set of incoming packets in a given interval of time. The available pool of tokens may depend upon service parameters such as the contracted rate and a measure of burstiness. Packets

Algorithm 1 Token Allocation

T = The Marking Interval
N = The number of flows
M = Number of Tokens available for the N flows in T seconds
$W(i)$ = Window Estimate for Flow i calculated as the running average of the difference between the sequence number in one direction and the acknowledgment number in the other
$E(i)$ = Smoothed estimate of the number of packets flow i sends in the marking interval T.
$S(i)$ = Number of consecutive OUT packets between every two IN packets of flow i.
$in\_count(i)$ = Allocation of IN tokens for flow i in time T seconds.
$tiny\_windows$ = Number of flows having windows less than k. (k defaults to 4)
$available$ = Number of Available tokens per interval after small window flows get their share
$max\_min\_alloc(i)$= Max-min fair allocation [10] of tokens per interval for flow i

Step I

1. For each flow i if $W(i) < k$ then

$$tiny\_windows{+}{+}; \quad /{*}Increment\ tiny\_windows{*}/ \tag{1}$$

$$available = (M - \Sigma(E(i)\ for\ small\ window\ flows)) \tag{2}$$

Step II :Divide the available tokens among the flows

2. If *available* $==0$ then /* If no tokens left after protecting small window flows */
   - **For Small Window Flows**
     Divide M among the small window flows
     $$in\_count(i) = \frac{M}{tiny\_windows} \tag{3}$$
     $$S(i) = \left(\frac{E(i)}{in\_count(i)}\right) - 1 \tag{4}$$
   - **For All Other Flows**
     $$in\_count(i) = 0 \tag{5}$$
3. Else if *available* > 0 then
   - **For Small Window Flows**
     $$in\_count(i) = E(i) \tag{6}$$
     $$S(i) = 0 \tag{7}$$
   - **For All Other Flows**
     Divide remaining tokens (available) among the other flows with optimal spacing
     $$in\_count(i) = max\_min\_alloc(i) \tag{8}$$
     $$S(i) = \left(\frac{E(i)}{in\_count(i)}\right) - 1 \tag{9}$$

which get a token are said to be marked "IN" and those which do not get tokens are said to be marked "OUT." As mentioned earlier, this can be used as the basis of a simplified form of the "assured service" [1]. Our algorithm can be generalized to the full assured service specification as well. Ongoing work involves implementation of our algorithm in linux-diffserv.

The Packet Marker is designed to :

1. **Protect small-window flows from packet losses:** Small-window flows are "protected" from packet losses by allocating only "IN" tokens to them (subject to the availability of tokens). A "Max-Min"' fair [10] allocation of the rest of the tokens is made among the remaining flows. If there are not enough "IN" tokens to provide all small window flows with their demand, the number of available tokens is equally divided among the small window flows and the remaining packets are marked as "OUT".

2. **Maintain optimum spacing between "IN" and "OUT" tokens allocated for a flow:** There could be scenarios wherein a flow gets a burst of "OUT" tokens followed by a burst of "IN" tokens. This could lead to burst loss of "OUT" packets resulting in a timeout with TCP Reno. The probability of timeout could be reduced if an optimal spacing is maintained between "IN" and "OUT" packets for each flow. This allocation of tokens is done once every T seconds. This is illustrated in the Token Allocation algorithm (Algorithm 1).

3. **Mark packets according to the allocations** : Packet marking is done according to the per-flow allocations of tokens made in the previous two steps. In the marking algorithm, a packet is identified as belonging to a flow. All packets from small-window flows are marked as "IN". Also while marking, a spacing variable (number of consecutive "OUT" packets between every two "IN" packets of flow) is maintained on a per-flow basis. If "IN" tokens run out for any flow (either due to a prediction error or a sudden burst), then all successive packets for that flow are marked as "OUT". Figure 1 further illustrates the marking scheme. Note that inspite of the packet "interleaving" introduced by this marking scheme, there is a residual risk that a burst loss could result in multiple (though not consecutive) packet losses within a TCP window. The Packet Marking Algorithm (Algorithm 2) illustrates this mechanism.

Algorithm 2 Packet Marking

$count(i)$ = flow variable to keep count of number of packets

(a) Save the spacing variable for each flow i in a temporary variable

$$default\_spacing(i) = S(i) \quad (10)$$

(b) Whenever a new packet comes in, identify its flow index i.

(c) For flow i, check if "IN" tokens are available i.e if *in_count(i)* > 0.

(d) If yes then do the following :

i. If $count(i) \geq S(i)$

$$Mark\ as\ \text{``}IN'' \quad (11)$$

$$S(i) = default\_spacing(i) - (count(i) - S(i)) \quad (12)$$

$$Reset\ count(i) = 0 \quad (13)$$

$$Decrement\ in\_count(i) \quad (14)$$

ii. Else if $count(i) < S(i)$

$$Mark\ as\ \text{``}OUT'' \quad (15)$$

$$Increment\ count(i) \quad (16)$$

(e) Else

$$Mark\ as\ \text{``}OUT'' \quad (17)$$

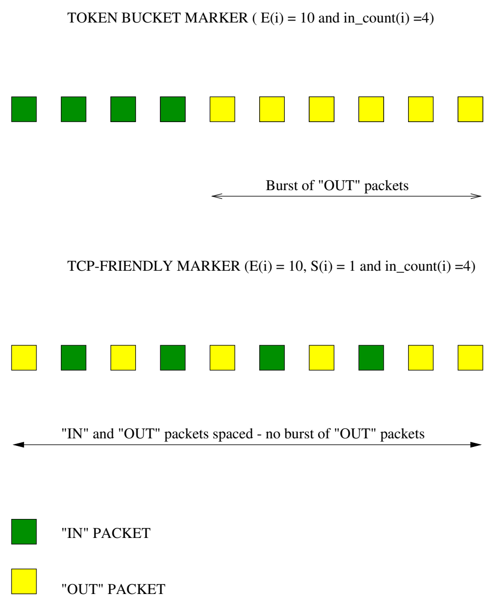

Figure 1: Illustration of marking scheme

4 SIMULATION AND PERFORMANCE ANALYSIS

For performance analysis, we consider the system (of TCP flows in this case) as a black box to which is input a set of *parameters* and workloads (parameters include the choice of the scheme, configurations etc) and the output is a set of *metrics* which evaluate the tradeoff among various resource constraints in the system. The next two sections explain the choice of metrics and parameter dimensions explored in this study.

4.1 Metrics

Our goal in performance analysis is to quantify "best effort" performance and "better-than-best-effort" performance. Best effort performance hitherto has been ill-quantified at best. The following set of metrics divided into two groups, operator and user metrics, represents the view of best-effort performance from the operator and user's perspectives. The overall performance of best effort is a combination of these two groups of metrics. These metrics were first introduced in our earlier work [11].

4.1.1 Service Provider (or operator) metrics

The operator's key resources are bandwidth and buffers. The operator is willing to tradeoff buffer resources to ensure high utilization of bandwidth. But high queuing delays or drop rates are undesired for supporting customers' interactive applications such as telnet or WWW. The metrics which measure the tradeoffs among these resources are:

1. **Average link Utilization**: Low link utilization, given adequate load is unacceptable. (We use the average goodput metric as a partial proxy for this metric).
2. **Packet loss rate**: Packet loss represents wasted bandwidth and buffer resources on upstream links.

4.1.2 User Metrics

The user is interested in per-flow goodput (assuming infinite flows). This requires us to use N metrics (where N = number of flows). But for brevity, we use:

1. **Average (per-flow) Goodput**: This quantity which excludes retransmitted packets should be as high as possible.
2. **Coefficient of Variation of (per-flow) Goodput**: This quantity is a rough measure of fairness. The coefficient of variation (denoted CoV) is the ratio of the standard deviation in goodput to the average goodput. Ideally, for a single bottleneck with infinite transfers, this metric should be close to zero.

4.2 Baseline Configuration

Figure 2 shows our baseline configuration. We have 10 sources which are divided into groups of five each. R0 and R1 incorporate the marker building blocks and handle traffic from five flows each. R2 is the bottleneck router and both R0 and R1 feed into R2. The R2-R3 link is the bottleneck. R3 connects the traffic to the respective destinations.

Each source is connected to its respective marker through links of 1.5 Mbps. The marker feeds this into the bottleneck through 10 Mbps links. The bottleneck link has a 1.5 Mbps capacity. All the destination links have 1.5 Mbps capacity. All links have a length of 1000 km. The default packet size is 1024 bytes. The default marking interval (T in algorithm 1) is 250ms.

We use a simulator called "netsim" developed at Ohio State University.

The queuing discipline used at the bottleneck router R2 is RIO [4]. As mentioned earlier RIO is a simple variant of the RED (Random Early Gateway) algorithm[5]. The RIO algorithm ("RED with In and Out") uses the same RED algorithm for "IN" packets and "OUT" packets, albeit with a different set of parameters for each. We use the following RIO parameters :

- min threshold for "OUT" packets(min_th_out) = 1
- max threshold for "OUT" packets(max_th_out) = 20
- max probability of packet loss for "OUT" packets(max_p_out) = 0.2
- min threshold for "IN" packets(min_th_in) = 150
- max threshold for "IN" packets(max_th_in) = 200
- max probability of packet loss for "IN" packets(max_p_in) = 0.005
- queue weight(w_q) = 0.002

All our simulations used "long TCP flows" simulating infinite FTP transfers. All flows start simultaneously and therefore we did not repeat simulations multiple times to get an average. Given the different variants of TCP implementations in practice, these simulations should only be interpreted as representative cases of absolute performance improvements, and not as exact percentage improvements.

5 SIMULATION RESULTS

The TCP-friendly traffic marker is designed to protect small windows from drops and maintain optimal spacing between "IN" and "OUT" packets. The effects of the marker are evaluated under various scenarios and described in detail. In all the scenarios we compare the performance of the TCP-friendly marker to a token bucket marker. We also present results with no marking i.e. best-effort service.

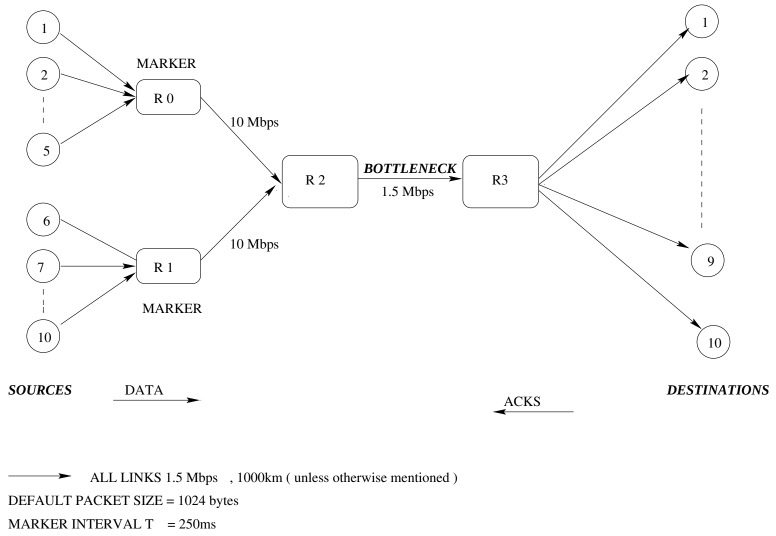

Figure 2: Base Configuration

Table 1: COMPARISON OF MARKERS FOR THE BASE CONFIGURATION : 10 FLOWS, SIMULATION TIME = 100s, BOTTLENECK BANDWIDTH = 1.5Mbps

	Operator Metrics		*User Metrics*	
Configuration	**Timeouts**	**Packet Losses** packets/sec	**Avg Goodput** bps	**CoV**
TCP Friendly Marker	**11**	**1.46**	18503	0.12
Token Bucket Marker	18	2.28	17858	0.09
No Marker	18	4.24	18257	0.15

5.1 Simulation results with the Baseline Configuration

We use the baseline configuration, and compare the metrics obtained with the TCP-friendly marker with those obtained using a simple token bucket marker. Table 1 summarizes the results. The TCP-friendly marker has a beneficial effect on all the metrics as compared to a simple token bucket marker. We observe a smaller packet loss rate (1.46 as compared to 2.28) and a significantly fewer number of timeouts (11 as compared to 18). Changes in the other metrics are insignificant. Though the gains are small in this baseline configuration, the TCP-friendly marker still does better. A smaller coefficient of variance for the token bucket marker can be attributed to the lower average goodput obtained.

5.2 Simulations with large number of flows

In this section, we test the performance of the TCP-friendly marker with a large number of flows. We modify the baseline configuration to have 100 flows. We still have a TCP-friendly marker for a set of five flows. Bottleneck speed is 15 Mbps and link lengths 1000 km. The simulation results are summarized in Table 2. We observe that deployment of the TCP-friendly marker causes a dramatic improvement in most of our metrics. Specifically we observe a smaller packet loss rate (23.26 as compared to 31.96) and fewer timeouts (348 as compared to 399) when compared to the token bucket marker. We also observe improved fairness and hence improved predictability of service (Coefficient of Variation in goodput of 0.09 as compared to 0.16). There is also a marginal decrease in average goodput (20377 as compared to 20449).

Table 2: COMPARISON OF MARKERS : 100 FLOWS, SIMULATION TIME = 100s, BOTTLENECK BANDWIDTH = 15Mbps

	Operator Metrics		*User Metrics*	
Configuration	**Timeouts**	**Packet Losses** packets/sec	**Avg Goodput** bps	**CoV**
TCP Friendly Marker	**348**	**23.26**	20377	**0.09**
Token Bucket Marker	399	31.96	20449	0.16
No Marker	2145	100.27	20213	0.07

Table 3: COMPARISON OF MARKERS : 100 FLOWS, SIMULATION TIME = 100s, BOTTLENECK BANDWIDTH = 150Mbps

	Operator Metrics		*User Metrics*	
Configuration	**Timeouts**	**Packet Losses** packets/sec	**Avg Goodput** bps	**CoV**
TCP Friendly Marker	**1261**	**240.48**	**194795**	**0.03**
Token Bucket Marker	4254	311.65	134952	0.05
No Marker	2954	322.07	151162	0.04

5.3 Simulations with large number of flows and high speed links

In this section, we test the performance of the TCP-friendly marker with a large number of flows. We modify the baseline configuration to have 100 flows. Also all source to marker links have a 15 Mbps bandwidth. Bottleneck speed is 150 Mbps. We still have a TCP-friendly marker for a set of five flows. Link lengths are 1000 km. The simulation results are summarized in Table 3. We observe that deployment of the TCP-friendly marker causes a dramatic improvement in most of our metrics. Specifically we observe a smaller packet loss rate (240.48 as compared to 311.65), fewer timeouts (1261 as compared to 4254) and higher average goodput (19475 as compared to 134952) when compared to the token bucket marker. We also observe improved fairness and hence improved predictability of service (Coefficient of Variation in goodput of 0.03 as compared to 0.05).

5.4 Simulations with Different TCP Implementations

In this section, we examine the effect of the TCP-friendly marker with different TCP implementations. In the earlier sections we demonstrated the usefulness of the TCP marker with TCP Reno, the largest currently installed base of TCP implementations. However new TCP implementations aimed at solving Reno's problems have been proposed and investigated. The most popular among the newer TCP implementations are New Reno [7] and SACK [6].

- **SACK** : The main weakness of Reno is that multiple (three or more) packet drops from window of data may result in the source incurring a timeout. With the limited information that cumulative acknowledgments provide the source is unable to detect multiple packet drops from the same window. SACK avoids this problem by using Reno with selective acknowledgments and selective retransmission. The receiving TCP sends back SACK packets to the sender "selectively" acknowledging only received data. The sender then need only send the missing data segments.

  The results of using a TCP-friendly marker as compared to a simple token bucket marker in conjunction with SACK as the TCP implementation are shown in Table 4. Comparison of Tables 4 and 1 shows that in general SACK reduces the total number of timeouts. We also note that the TCP-friendly marker reduces the total number of timeouts to a greater degree (1 as opposed to 5) as compared to a token bucket marker. We also observe improved fairness (as indicated by a smaller coefficient of Variation in goodput) and hence greater predictability of service (0.10 as opposed to 0.16). We again observe a smaller packet loss rate (1.76 as compared to 1.82), and a marginally higher average goodput (18421 vs 18329).

- **NewReno** : The NewReno algorithm proposes a modification to Reno's fast recovery algorithm. In the absence of SACK, there is little information available to the TCP sender in making retransmission decisions during Fast Recovery. In NewReno, during Fast Recovery an acknowledgment that acknowledges some but not all of the packets outstanding is called a "partial ack". With NewReno, partial acks do not take the source out of fast recovery. Instead a partial ack is understood to indicate a packet loss immediately after the last acknowledged packet. This packet is then retransmitted.

  As opposed to SACK, NewReno, does not require both the sender and receiver TCP support. The results of using a TCP-friendly marker as compared to a simple token bucket marker in conjunction with NewReno as the TCP implementation are shown in Table 5. We again observe improved fairness and hence improved predictability of service (coefficient of variation of goodput of 0.11 as compared to 0.14). We also observe a smaller packet loss rate (1.49 as compared to 1.77), fewer timeouts (17 as compared to 26) when compared to the token bucket marker. There is also a marginal increase in average goodput (18513 as compared to 18472).

5.5 Evaluation of Assured Service

Ibanez et al.[8] pointed out some serious performance issues with assured services. In particular, the authors pointed out that in many cases assured service may not give the "guarantees" it is aiming to provide. Specifically, in configurations where the assured traffic has to compete with unbounded best-effort traffic, this effect is pronounced.

We investigate the performance of the TCP-friendly marker in such scenarios where assured traffic is competing with unbounded best-effort traffic. A performance evaluation of the TCP-friendly marker is done against the simple token bucket marker. In this configuration a single source is provided with increasing levels of assurance up until the entire bottleneck bandwidth and the assurance it actually gets from the network is observed. The performance of the TCP-friendly marker is contrasted with a simple token bucket marker.

Table 6 shows the performance improvement that the TCP-friendly marker provides as compared to the token bucket marker in terms of providing assurances. It is observed that both the token bucket and the TCP-friendly markers fall short of providing the assurance they aim to provide. For example for a assured bandwidth of 0.49 Mbps, the token bucket marker can only get a bandwidth of 0.26 Mbps as compared to 0.44 Mbps got by the TCP-friendly marker. However the TCP-friendly marker is able to provide the guarantees to a far greater extent. This means that in order to provide "assured" service for TCP, some kind of over-provisioning is required in both cases. This over-provisioning could be in the form of providing more tokens than contracted for at the marker and having excess capacity at the bottleneck. For example if we wanted to guarantee approximately 0.5Mbps of bandwidth on a 1.5Mbps bottleneck, we need to give 20 tokens per interval in the case of a TCP-friendly marker and 40 tokens per interval for a token bucket marker. However the TCP-friendly marker reduces the degree of token over-provisioning required to provide guarantees. Over and above this *token over-provisioning*, we also need *bandwidth over-provisioning* as observed from the ratio of bandwidth obtained to bandwidth assured in Table 6. Thus the service realized by an application is a function of the assured rate, the token over-provisioning, the marking scheme and TCP dynamics.

5.6 Evaluation of Assured Service over High Speed Links and Large Number of Flows

In this set of simulations we extend the configuration from section 5.5 to 100 flows and link speeds of 15Mbps. Again only one source is given increasing levels of assurance. All other flows are best-effort. The links connecting the assured source to the network have a speed of 150Mbps.

Table 7 reaffirms the results we observed in Table 6. We observe again that both markers fall short of providing the assurance they aim to provide. For example for an assured bandwidth of 49 Mbps, the token bucket marker can only get a bandwidth of 12.86 Mbps as compared to 22.92 Mbps got by the TCP-friendly marker. The TCP-friendly marker is able to provide the guarantees to a far greater extent.

Table 4: COMPARISON OF MARKERS FOR SACK : 10 FLOWS, SIMULATION TIME = 100s, BOTTLENECK BANDWIDTH = 1.5Mbps

	Operator Metrics		*User Metrics*	
Configuration	**Timeouts**	**Packet Losses** packets/sec	**Avg Goodput** bps	**CoV**
TCP Friendly Marker	**1**	1.76	18421	**0.10**
Token Bucket Marker	5	1.82	18329	0.16
No Marker	5	3.16	18309	0.17

Table 5: COMPARISON OF MARKERS FOR NEWRENO : 10 FLOWS, SIMULATION TIME = 100s, BOTTLENECK BANDWIDTH = 1.5Mbps

	Operator Metrics		*User Metrics*	
Configuration	**Timeouts**	**Packet Losses** packets/sec	**Avg Goodput** bps	**CoV**
TCP Friendly Marker	**17**	**1.49**	18513	**0.11**
Token Bucket Marker	26	1.77	18472	0.14
No Marker	30	3.31	18452	0.17

5.7 Other Simulations

The performance of the TCP-friendly marker was observed in various other configurations. Some of the other configurations we looked at were short transfers, heterogeneous rtt flows and high speed links. We do not present the results for all the above configurations because of space constraints. The results are consistent with the findings from the earlier sections. The TCP-friendly marker is seen to provide performance enhancements in all of the above configurations.

6 CONCLUSION AND FUTURE WORK

In summary, we looked at generic TCP performance problems which remain in differentiated services, and demonstrate that enhancements are needed in traffic conditioners. Specifically, in this paper we observe the performance improvements we are able to produce by the deployment of a TCP-friendly traffic marker. The service realized by an application is seen to be a function of the assured rate, the marking scheme and TCP dynamics. By using our TCP-friendly marker we can get better predictability of service, lower token provisioning requirements, improved fairness, fewer TCP timeouts and fewer packet losses. Further ongoing work includes implementing the TCP-friendly marker in linux-diffserv to experimentally validate the results and moving towards a more comprehensive TCP-friendly assured service solution. These directions include integration with TCP rate control [11], protection against parameter sensitivity issues in RIO, and stateful differentiated buffer management schemes at the core.

References

[1] J. Heinanen, F. Baker, W.Weiss and J.Wroclawski, Assured forwarding PHB group. IETF Internet draft *draft-ietf-diffserv-af-05.txt*, February 1999.

[2] S. Blake, D.Black, M.Carlson, E.Davies, Z.Wang and W.Weiss, "An architecture for differentiated services", *Internet RFC 2475*, December 1998.

[3] S.Blake, Y.Bernet, J.Binder, M.Carlson, B.Carpenter, S.Keshav, E.Davies, B.Ohlman, D.Verma, Z.Wang and W.Weiss, "A framework for differentiated services", IETF Internet Draft, *draft-ietf-diffserv-framework-01.txt*, October 1998.

[4] D.D. Clark and W. Fang, "Explicit allocation of best-effort packet delivery service", *IEEE/ACM Transactions on Networking*, 6(4):362-373, Aug. 1998.

Table 6: COMPARISON OF MARKERS FOR ASSURED SERVICE : 11 FLOWS, SIMULATION TIME = 100s, BOTTLENECK BANDWIDTH = 1.5Mbps

Tokens/Interval	**BW Assured**	**BW Obtained**		**BW Obtained/Assured**	
packets	Mbps	Mbps	Mbps	percent	percent
		Friendly	**TokenBucket**	**Friendly**	**TokenBucket**
15	0.49	0.44	0.26	90	53
20	0.66	0.52	0.34	79	52
25	0.82	0.57	0.36	70	44
30	0.98	0.7	0.42	71	43
35	1.15	0.73	0.46	63	40
40	1.31	0.89	0.49	68	37
45	1.47	0.99	0.59	68	40

Table 7: COMPARISON OF MARKERS FOR ASSURED SERVICE : 100 FLOWS, SIMULATION TIME = 100s, BOTTLENECK BANDWIDTH = 150Mbps

Tokens/Interval	**BW Assured**	**BW Obtained**		**BW Obtained/Assured**	
packets	Mbps	Mbps	Mbps	percent	percent
		Friendly	**TokenBucket**	**Friendly**	**TokenBucket**
1500	49	22.92	12.86	47	26
3000	98	45.77	30.51	47	30
4500	147	61.08	32.24	41	21

[5] S. Floyd, and V. Jacobson, “Random early detection gateways for congestion avoidance,” *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August 1993, pp.397-413.

[6] M.Mathis, J.Madhavi, S.Floyd and A.Romanov, “TCP selective acknowledgement options” *Internet RFC 2018*, October 1996.

[7] S.Floyd, and T. Henderson “The NewReno modification to TCP's fast recovery algorithm” *Internet RFC 2582*, April 1999.

[8] J. Ibanez, K. Nichols, “Preliminary simulation evaluation of an assured service”, IETF Internet Draft, *draft-ibanez-diffserv-assured-eval-00.txt*, August, 1998.

[9] Kevin Fall and Sally Floyd, “Comparisons of Tahoe, Reno and SACK TCP”, Computer Communication Review, V. 26 N. 3, July 1996,

[10] A. Charny, “An algorithm for rate allocation in a packet-switching network with feedback”, Masters thesis, MIT 1994

[11] S.Karandikar, S.Kalyanaraman, P.Bagal and B.Packer, “TCP rate control”, Computer Communication Review, V. 30 N. 1, January 2000,