TCP-Friendly Marking for Scalable Best-Effort Services on the Internet

Giampiero Lo Monaco, Azeem Feroz[1], Shivkumar Kalyanaraman, Yong Xia
ECSE Department, Rensselaer Polytechnic Institute
110 8th Street, Troy, New York, 12180
{lomong, feroza, shivkuma, xiay}@networks.ecse.rpi.edu

Abstract

This paper proposes the use of TCP-aware network-based packet marking, which in conjunction with differential packet dropping, is a powerful way to improve the performance of buffer management for best-effort services on the Internet. In particular, we extend the notion of "TCP-Friendly" packet marking, proposed recently by us [6], and apply it to improve the performance of traditional best-effort services. The TCP-aware use of deterministic packet marking at the enterprise edge allows us to protect selected TCP packets from suffering loss, thus significantly reduce the total number of timeouts and avoid the resulting service degradation. In particular, we protect TCP sessions with very small windows, disperse packet loss across a given window, and protect retransmitted packets from encountering loss. We show baseline results which illustrate that the performance gains are considerable (orders of magnitude) when compared to stateful packet dropping algorithms like FRED [15], and even when TCP SACK [18] implementations are employed. The scheme has been implemented in Linux 2.2.10, and all the results are based upon experimental data.

1 Introduction

As the Internet grows, both in terms of size and heterogeneity, it becomes necessary to have traffic management schemes that keep up with this growth by leveraging the new architectural facilities available through IETF standards. The differentiated services (diffserv) architecture [2] is an example of one such IETF standard that can be leveraged for scalable buffer management. The diffserv architecture proposes two distinct components, the "edge" nodes and the "core" nodes, as part of a diffserv domain. The edge nodes perform stateful control- and data-plane functions like policy and traffic conditioning whereas the core nodes concentrate on stateless hardware-implementable data-plane functions such as forwarding, buffer management and scheduling. The coordination of data-plane functions between the edge and the core nodes is achieved through the DS-field in the IP header, which the edge can "mark" and the core can read and optionally "re-mark". This architectural model allows us to use "marking" at the edge nodes as a way to complement and enhance the buffer management capabilities of core nodes. We refer to this marking as "network-based" marking (which has semantics only within the network) to differentiate it from end-to-end ECN marking [22], (which is conveyed to end systems).

In this paper, we develop a TCP-aware proxy that performs deterministic, TCP-conscious, per-flow marking at the edge. This component in conjunction with random, stateless, differential packet-

[1]Azeem Feroz is currently with Packeteer Inc.

dropping in the core can lead to substantial performance improvements. We also develop a stateful differential dropping algorithm for our comparative experimental evaluation. Our results indicate substantial improvements in both provider-metrics of best-effort performance (such as aggregate goodput and packet loss rate), as well as user-perceived metrics (such as TCP timeouts and per-flow goodput variance) [13]. More importantly, *these performance gains apply both to TCP Reno and SACK, stateless and stateful core buffer management schemes, and even when the parameters of the buffer management schemes are mistuned.* Furthermore, this work is *partially deployable* and can allow the *network to be safely operated at higher average utilization levels without any user-perceived degradation in service.* This scheme however requires a diffserv context, but at the same time provides positive incentives for deploying active queue management and differential dropping schemes.

The paper is organized as follows. Section 2 surveys related work to position our scheme. Section 3 presents the concepts of TCP-Friendly marking and better best-effort services. We outline the key ideas and the algorithm pseudo code in section 4, and then present a new stateful differential dropping scheme in section 5. Section 6 presents our experimental evaluation, and section 7 summarizes the paper.

2 Related Work

For the last decade, researchers have studied and proposed improvements to TCP/IP performance. These improvements include changes to TCP host implementations (e.g. Reno, SACK, etc [1, 7, 8, 9, 18, 25]), better active queue management algorithms (e.g. RED, FRED, etc [3, 10, 14, 15, 20, 21]) or schemes which require both end-system and bottleneck support (e.g. ECN, adaptive packet marking [5, 22, 24]). Packeteer's rate control [13] is a buffer management scheme which manipulates TCP headers and acknowledgment (ack) rates to perform explicit, transparent control of TCP flows in certain niche deployment scenarios. Performance improvements of all these approaches have been measured in terms of timeout avoidance, better filtering of burst loss to determine congested periods and window reductions, optimization of retransmission, and improved fairness across competing TCP sessions (including both small, http-like and large, ftp-like sessions).

Our work was motivated by Ibanez et al's and Sahu et al's findings [12, 23], which showed performance uncertainty when TCP was mapped to the "assured service" even with relatively low degrees of connection multiplexing. This uncertainty stems from the fact that the "assured service" introduces a new dimension in best-effort buffer management: differential marking and dropping of packets which was *not being leveraged properly* by TCP/IP. Our work proposes components to brid this performance gap. Therefore, in contrast with the traditional approaches mentioned above, this paper proposes to view the buffer management problem as two parts. The first part uses network-based, TCP-friendly marking [6], which can work in combination with the second part, differential dropping [4] in a diffserv context. Note that, unlike ECN [22] our mechanisms are purely network-based and transparent to the end-systems, though the marker could conceivably be supported by end-systems in the future.

The end-goals of this work are similar to the schemes mentioned above; but the deployment environment is different. we require a diffserv context. Therefore, we view our work as being

complementary and not conflicting with the efforts of the community. In particular, we demonstrate that our performance improvements are visible even with TCP SACK and with stateful dropping algorithms (e.g. FRED). From a deployment perspective, our marker requires read-access to TCP headers and would typically be at the enterprise edge. However, our approach can potentially affect a larger scope of operation because a) our marking ensures protection of critical packets from loss even at *remote bottlenecks* (unlike TCP rate control [13]) and the marks are carried at the network layer headers and do not require TCP-level visibility or processing beyond the marker.

3 TCP-friendly Better Best-effort Services

The term "TCP-Friendly" has been recently used in literature [19] to characterize the behavior of non-TCP end-to-end congestion control schemes. Our use of this term, in contrast, is to characterize building blocks that support superior best-effort performance for TCP applications [6]. The key question then becomes: "What building blocks aspects matter for superior TCP performance?" It is well-known that timeouts are the leading source of TCP performance degradation from several perspectives such as average goodput, transfer-time and fairness. Therefore the goal of "TCP-Friendly" building blocks is to reduce the probability of timeouts. From an operational perspective, TCP times out when:

- it loses all the packets or acks corresponding to a single window;
- its window is very small (< 4*MSS) and it loses even one or two packets;
- retransmitted packets are lost.
- it runs out of duplicate acks during a fast recovery phase;
- a burst of at least three closely spaced packets within a window are lost;

The last two conditions apply only to Reno, not SACK. Active queue management schemes, such as RED [10] and FRED[15], try to address some of these issues and hence they are "TCP-Friendly" in the sense of our definition. However, we note that the RED/FRED approach is to randomize the dropping behavior, and therefore they cannot provide a high level of assurance that timeouts would be contained, especially during heavy loads and/or during high degrees of TCP multiplexing. There are also end-system proposals to reduce the TCP dupack threshold and make other TCP level implementation improvements to address this issue [1, 8, 9, 18]. This paper leverages the dimension of packet marking by extending our recent work [6] in the following ways:

- Mark packets to protect small-window (< 4*MSS) flows from packet loss;
- Mark packets to maintain maximum spacing between packets marked as "OUT" (low priority), within limits of the total number of tokens, in order to disperse the effect of aggregate bursty loss;
- Mark packets to protect retransmitted packets from encountering loss.

Of these, the idea of protecting retransmissions from loss results in a large performance improvement because it affects all TCP implementations and accounts for a large fraction of retransmission timeouts. Retransmissions are generally sent during episodes of congestion and experience a higher loss probability. Moreover, in several cases, retransmission loss leads to a timeout in TCP Reno/SACK and hence accounts for a disproportionate share of total number of timeouts. Therefore, protecting retransmissions results in remarkable reductions in the number of timeouts, especially if the window sizes are small as well. Section 6 gives an experimental validation of this argument.

Now we consider the meaning of better best-effort service from a (TCP) performance and deployment perspective. Best-effort is the service commonly available on the Internet. *Quantitative* per-flow guarantees for best-effort performance are generally not given to customers. However, quantitative per-flow statistics over the population of users can be formulated for engineering purposes. In particular, user metrics (timeouts, average per-flow goodput, and coefficient of variation of per-flow goodputs) and network operator metrics (utilization, queue length, and packet loss rates) can be used to measure best-effort TCP/IP performance (for more discussion, see [6, 13]). These two perspectives are necessary because otherwise, one could optimize user-perceived performance with aggressive control schemes at the expense of network stability and other well-behaved flows[13]. Similarly, one could optimize the network operator perceived performance while inflicting a large service variation on users. In this sense, "better best-effort" means a superior tradeoff between these multiple metrics, or an improvement in both user and operator metrics. In particular, we expect our proposed schemes to eliminate a large fraction of total timeout instances seen by TCP.

Architecturally, better best-effort service requires the deployment of new components in the network, or the upgrade of end-systems. Our premise is that the incremental deployment of differential dropping and transport-aware marking components (e.g. through diffserv) can lead to such services. In this context, assuming that the network supports differential dropping mechanisms (e.g. RIO [4]), our TCP-friendly marker will provide better best-effort or enhanced assured services [4, 12, 23] for the subset of users who deploy these markers.

4 TCP-Friendly Packet Marker

A packet marker is one of the traffic conditioners in the diffserv architecture. The general problem in a marker is to optimally allocate an available pool of tokens to a set of incoming packets in a given time interval. Packets that get a token are said to be marked as "IN" and those that do not get tokens are said to be marked as "OUT". The TCP friendly packet marker implements the objectives laid out in section 3. We present the pseudo code in algorithm 1 and 2. The token allocation process is called every marking time interval (400ms in our experiments), whereas the packet marking process is called whenever there is an arriving packet. The algorithm itself is very simple and has very low time complexity (*O(1)* for each arriving packet). The detailed algorithm and explanation including Linux implementation issues are presented in an online technical report[16].

A key per-flow variable required by the marker is an approximation of the TCP window size. This can be obtained by considering the difference between the last sequence number sent and the last

Algorithm 1 Token Allocation

T = the marking interval
N = the number of flows
M = the 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 packet sequence number in one direction and the ack sequence number in the other
E(i) = smoothed estimate of the number of packets flow i sends in one marking interval T
S(i) = number of consecutive "OUT" packets between two "IN" packets of flow i
in_count(i) = allocation of "IN" tokens for flow i in one marking interval
tiny_flows = number of flows with window size less than K (defaults to 4) MSS
available = number of available tokens per interval after small window flows get their share
max_min_alloc(i) = max-min fair allocation of tokens per interval for flow i

Step I, Pre-allocate tokens for tiny flows
(a) *Available = M*;
(b) For each flow i, if *W(i) < K*MSS* then
{ *tiny_flows ++; available -= E(i);* }

Step II, Divide the available tokens among all flows
(c) If no token left for non-tiny flows, i.e. if *available <= 0*, then
(c.1) For tiny window flows, divide M tokens among them with optimal spacing:
{ *in_count(i) = max_min_alloc(i); S(i) = E(i) / in_count(i) - 1;* }
(c.2) For all other flows, just set *in_count(i)=0*;
(d) Else if available > 0, then
(d.1) For tiny window flows
{ *in_count(i) = E(i); S(i) = 0;* }
(d.2) For all other flows, divide remaining tokens among them with optimal spacing:
{ *in_count(i) = max_min_alloc(i); S(i) = E(i) / in_count(i) - 1;* }

Algorithm 2 Packet Marking

count(i) = variable to keep count of the number of packets of flow i, initialized to 0

(a) Whenever a new packet comes in, identify its flow index i;
(b) If it's retransmission, mark as "IN" then return;
(c) Else for flow i, check if tokens are available, i.e. if *in_count(i) > 0;*
(d) If yes, then
(d.1) if *count(i) >= S(i)*
{ Mark as "IN"; *count(i) -= S(i);* decrease *in_count(i);* }
(d.2) else
{ Mark as "OUT"; *count(i) ++;* }
(e) Else mark as "OUT";

byte acknowledged. This allows us to classify the packets as belonging to a "tiny" flow (window < 4*MSS) or otherwise. Tiny flows get absolute priority to tokens. The remaining tokens are fairly divided between the other flows and each flow's share is optimally spread between its packets via the spacing between the "IN" packets in the flow. Retransmitted packets can be identified by comparing the sequence number of the packet with the largest sequence number sent by that flow. Retransmitted packets also get absolute priority in the token pool. Token pool mismatches, new flows, etc. are handled across the interval of the token allocation process.

The marker is expected to be implemented at the enterprise or Internet Service Provider's (ISP) network edge and operated in cooperation with an ISP which implements differential packet dropping. This scenario is suggested because the marker as presented here requires visibility into the TCP header for read operations only. An alternative would be to split the marker into two pieces, one in the enterprise side and one on the ISP side. The enterprise side piece uses the visibility of TCP headers and marks the DS-byte [2] which can then be translated into an ISP-specific codepoint (i.e. re-marked) by the ISP-side component.

5 FRIO: A New Differential Drop Scheme

We also propose to optionally extend FRED [15] into a differential dropping scheme called FRIO in a manner similar to how RED was extended to RIO [4]. In particular, a separate set of parameters and queue measures are used for packets marked "IN" and for packets marked "OUT". The original FRED algorithm was developed to improve fairness for fragile (small window or low rate) connections. FRED preferentially drops packets from a flow that has a large number of packets buffered at the congested gateway. The implementation issues of the FRIO algorithm are detailed in the technical report [16]. The key modifications to basic FRED we made include:

- Use of instantaneous per-flow queue size instead of average per-flow queue size to approximate the per-flow number of packets in the buffer. This improved the performance seen by slow TCP connections;
- Calculation of average queue length exclusively on arrival of packets. The original FRED recalculates average queue length, both on arrival and departure of packets. They argue that the omission of the calculation during packet departure results into low link utilization due to unnecessary packet drops. We found that Linux RED implementation avoided this problem by considering the idel time during the inter-arrival of packets and used this model in computing the average queue length, done exclusively on packet arrival. We exploited this approach and do not recalculate average queue length on departure.

This FRIO extension is optional because our TCP-friendly marking component can achieve baseline gains even without this extension (see section 6). Also since FRIO and RIO are fundamentally variants of the basic RED algorithm, they inherit some of its parameter-sensitivity issues, which may degrade performance under certain conditions [17]. However, we shall see later in this paper that the determinism in the marking process effectively masks such parameter-sensitivities.

6 Performance Analysis and Experimentation

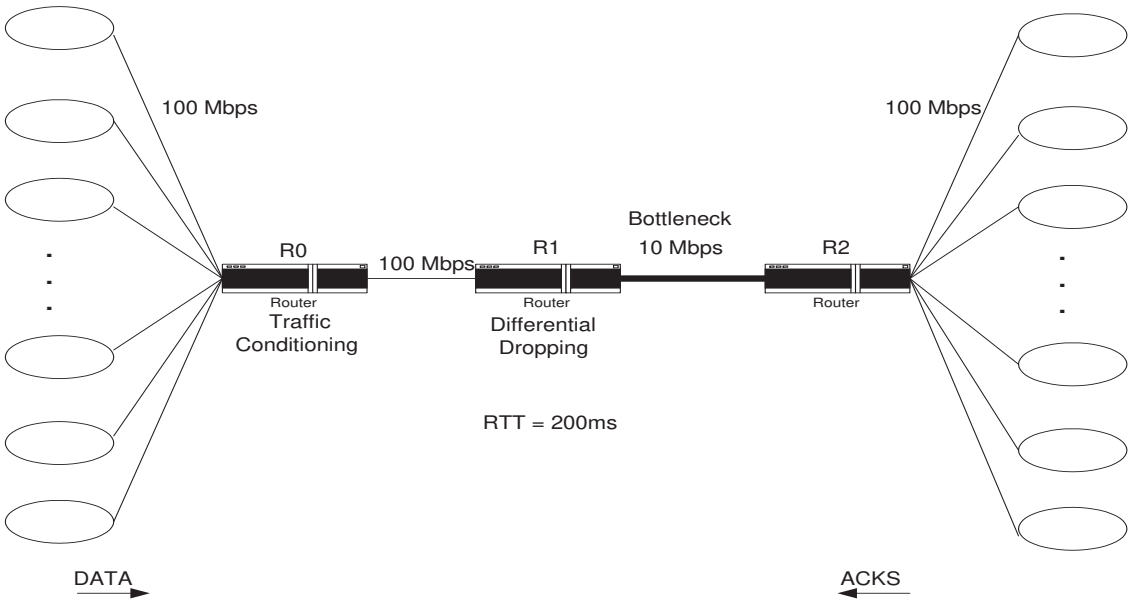

Figure 1: Logical Topology Used in Experimental Evaluation

6.1 Experiment Description

Figure 1 shows the logical topology of the network used in our experiments. Note that these experiments were conducted on a Linux-based experimental testbed, unlike our earlier simulation work [6]. We have multiple TCP flows destined to the other end of the network through a 10Mbps bottleneck. The access and egress links are 100Mbps. We implement the differential dropping algorithms at the interface of R1 connecting to R2, and the marking at the entry to access links. We implement a delay component on the outgoing interface of the destination with the intention of emulating a WAN round trip time of 200ms. The metrics used in our work have been carefully chosen to characterize best-effort performance, both from user and operator perspectives [6, 13] :

- Timeouts: a proxy measure for the increase in transmission delays as seen by the end user;
- Packet Loss Probability: a packet-level measure of the loss probability at the bottleneck;
- Average Per-flow Goodput: measures the average per-flow throughput minus retransmissions across a set of flows;
- Coefficient of Variation of Per-flow Goodputs: this metric measures the spread of per-flow goodputs, i.e., the unpredictability of resulting best-effort service.

The premise of these multiple metrics is that best-effort performance looked at from the perspective of all these metrics is very different from looking at it just from the perspective of average per-flow

goodput. We use four configurations in our experiments to capture: a) the two most important and commonly deployed TCP implementations (Reno and SACK), as well as, b) good or mistuned setting of RED-equivalent parameters. We consider "good" RED parameters as those in which the max and min thresholds are relatively far apart [17]. Mistuned parameters are those where the thresholds are set too close to each other, which in effect defaults to a taildrop policy on a smaller buffer limit. Parameter mistuning risk is one of the leading reasons why RED is not widely deployed today and our marking scheme can help mitigate this risk in a transparent manner. The specific RED parameters for each of these cases are presented in Table 1. The total bottleneck buffer size in these experiments is 500Kbytes (approx. 500 packets).

	Good RED Parameters		**Mistuned RED Parameters**	
	IN Packets	OUT Packets	IN Packets	OUT Packets
Min threshold (Kbytes)	50	10	50	10
Max threshold (Kbytes)	490	200	490	30
Max drop probability	0.005	0.25	0.005	0.25

Table 1: Good and Mistuned RED Parameters

Based upon the above settings, the four configurations we use are shown in Table 2. Further, if we consider combinations with dropping schemes and markers, we get a set of cases shown in Table 3. The "old" TCP-friendly marker refers to the marker reported in [6] which does not protect retransmissions. The token bucket and TCP-friendly markers deal with a pool of tokens refreshed every update interval. The number of token bytes per interval is 296K and the update interval is 400ms. In addition, each case was tested with 50 and 100 simultaneous TCP flows to get a moderate level of multiplexing.

	Config A	**Config B**	**Config C**	**Config D**
TCP Version	SACK	Reno	SACK	Reno
RED Params	Good	Good	Mistuned	Mistuned

Table 2: Experimental Configurations

	Marker Option	Dropper Option	Abbreviation
Case 1	TCP Friendly	FRIO	TCPF+FRIO
Case 2	TCP Friendly	RIO	TCP+RIO
Case 3	TCP Friendly (old)	RIO	TCPF(O)+RIO
Case 4	Token Bucket	RIO	TB+RIO
Case 5	No Marker	FRED	NoMark+FRED
Case 6	No Marker	Tail Drop	NoMark+TailDrop
Case 7	No Marker	RED	NoMark+RED

Table 3: Marker and Dropper Choices in Each Configuration

We acknowledge that this experimental setup captures the behavior only at a canonical bottleneck with a simple workload. We have chosen to focus on a variety of schemes (Reno, SACK, TCPF,

TB, RED, FRED, RIO, FRIO, RED parameters etc) rather than a variety of configurations and workloads. However, we would argue that our basic observations on the effects of timeout reductions are valid more generally, even though the particular performance gains may vary. Our earlier work [13] has shown that factors like heterogeneous RTTs, multiple bottlenecks are not essential in verifying basic timeout mitigation effects, even though they are important in determining issues like unfairness (because timeout is only one of the causes of the unfairness problem). As an aside, we have performed some preliminary experiments with short flows and see substantial performance gains there as well. We do not report them for reasons of space and because they represent an incomplete set.

6.2 Experiment Results

We choose to present our results as tables to allow selective row-by-row or column-by-column comparison. The following abbreviations are used in addition to the ones mentioned in Table 3: "Ploss" for total packet loss percentage, and "CoV" for coefficient of variation of per-flow goodputs. Table entries which are referenced in the text are bold-faced.

6.2.1 Configuration A: SACK and Good RED parameters

Table 4 shows significant improvements achieved by the TCP-Friendly marker across all metrics. On the case of 100 flows we see a reduction of timeouts from 911 (TB+RIO case) to 41 (TCP+FRIO case), a factor of 22 improvement. In the 50 flows case, timeouts are virtually eliminated; a reduction from 491 to 2 timeouts. These results show that SACK by itself does not eliminate timeout behavior (e.g. row #7, column # 1 has 1322 timeouts), and that the TCP-Friendly marker provides performance improvement even with TCP SACK.

The performance improvement is also seen in metrics other than timeouts. Observe a reduction of 30-50% in the packet loss rate with the TCP-friendly marker. For example, in 100-flow case, Ploss reduces from 9% in the last row to 4% in the first row. The average per-flow goodput is always higher by about 10-15% when using our scheme; e.g. in column #3, with 100 flows, the average per-flow goodput increases from about 9.0kB/s to 11.4kB/s. The coefficient of variation is also better (e.g. column #4, 0.06 vs 0.38), though this CoV analysis is incomplete in general because factors such as RTT and multiple bottlenecks have a non-trivial effect. Also note that the *relative* performance improvement in the first 4 columns (100-flow case) in order-of-magnitude terms is similar to that seen in the last four columns (50-flow case), i.e., an indication of scalability performance gains using our scheme.

Though the use of FRIO provides better results than RIO, the relative improvement achieved by TCP-Friendly marker with RIO itself is substantial. For example, compare results in row #2 (TCPF+RIO) with later rows; e.g. 120 timeouts vs 911 timeouts. This means that, even though we propose and study FRIO, we are not necessarily advocating complex per-flow schemes at the interior bottlenecks. In other words, the combination of per-flow, deterministic marking at the edge with stateless, random, differential dropping is a powerful one by itself. As an aside, note that this table also validates aspects noted in other papers including:

	100 flows				50 flows			
	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV
TCPF +FRIO	**41**	**4.00**	**11.421**	**0.06**	**2**	2.14	22.236	0.03
TCPF +RIO	**120**	5.46	10.575	0.12	9	2.67	20.742	0.04
TCPF(O) +RIO	390	6.38	10.020	0.20	49	3.27	19.514	0.09
TB +RIO	**911**	7.75	9.705	0.33	253	4.13	18.892	0.12
NoMark +FRED	1030	8.02	9.460	0.22	323	4.76	18.370	0.10
NoMark +TailDrop	**1140**	8.43	8.970	0.24	415	5.01	17.232	0.12
NoMark +RED	**1322**	**9.01**	**9.015**	**0.38**	**491**	5.25	17.865	0.17

Table 4: Results for Config A: TCP SACK and Good RED parameters

- The performance improvement using FRED compared to RED. Compare row #5 (NoMark +FRED) with row #7 (NoMark+RED); e.g. 1030 timeouts with FRED vs 1322 timeouts with RED.
- The good performance of tail-drop compared to RED under reasonably high degrees of multiplexing [17]. Compare row #6 (NoMark+TailDrop) with row #7 (NoMark+RED); e.g. 1140 timeouts in TailDrop vs 1322 timeouts in RED;
- Ibanez et al's and Sahu et al's [12, 23] observations on the fact that assured services with simple token bucket marking does not lead to considerable performance improvement. Compare row #4 (TB+RIO) with later rows; e.g. 911 timeouts with token bucket which is a marginal improvement over 1140 timeouts achieved with NoMark plus TailDrop;
- Our earlier simulation work illustrating the performance benefits of the old TCP-friendly marker which just focussed on small window flows [6]. Compare row #3 (TCP(O)+RIO) with later rows; e.g. 390 timeouts achieved with the old marker is a good improvement over 911 timeouts achieved with token bucket marker.

6.2.2 Configuration B: Reno and Good RED Parameters

Table 5 reiterates the basic performance benefits noted in the earlier section with TCP Reno instead of TCP SACK. We see the number of timeouts go from 1127 (row #4) down to 62 (row #1) for 100 flows and from 610 to 8 for 50 flows. This represents orders-of-magnitude gain in the total number of timeouts, clearly decoupling the benefits of our marker to the use of either Reno or SACK. The other metrics show similar relative performance gains across the board. As an aside note that

	100 flows				50 flows			
	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV
TCPF +FRIO	**62**	4.33	11.391	0.07	**8**	2.49	22.079	0.03
TCPF +RIO	176	5.47	10.472	0.14	16	2.83	20.635	0.04
TCPF(O) +RIO	578	9.24	9.992	0.22	129	4.78	19.034	0.14
TB +RIO	**1127**	12.49	9.660	0.31	350	6.66	18.486	0.19
NoMark +FRED	1209	13.97	9.235	0.24	398	7.13	18.125	0.17
NoMark +TailDrop	1293	14.63	8.528	0.26	435	8.08	17.004	0.22
NoMark +RED	1405	16.47	8.807	0.37	**610**	9.63	17.319	0.25

Table 5: Results for Config B: Reno and Good RED Parameters

the absolute magnitudes of the Reno metrics are not as good as that of the corresponding SACK metrics presented in the previous table, reaffirming the general performance benefits of SACK.

6.2.3 Configuration C: SACK and Mistuned RED Parameters

Table 6 presents results with mistuned RED-equivalent parameters. In comparison with Table 4, note that every timeout observation almost doubles (e.g. row #1, column #1 shows 92 timeouts in table 6 vs 41 timeouts in table 4) validating the negative effect of parameter mistuning. However, the relative performance benefits in terms of all the metrics of TCP-friendly marking are clearly preserved. Compare the first two rows with later rows in tables 4 and 6; e.g. we see 92 or 152 timeouts with TCPF marker (rows #1-2, column #1) vs 1200+ timeouts without the TCPF marker (rows #4-7, column#1). In other words, the orders-of-magnitude performance gains especially in timeouts due to TCP-friendly marking overshadow the factor-of-2 performance degradation due to RED parameter mistuning. This provides some positive incentive to deploying active-queue-management-based [3] differential dropping schemes.

6.2.4 Configuration D: Reno and Mistuned RED Parameters

Table 7 replicates the results of mistuned parameters with TCP Reno instead of TCP SACK. The observations regarding relative performance and compensation of mistuning effects made in the last section hold in this case too. Since TCP Reno is the dominant deployed protocol and since parameter mistuning is a common occurrence, this experiment is more in tune with reality. Note that the loss rates during persistent congestion with Tail drop or RED (last two rows) can be as

	100 flows				50 flows			
	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV
TCPF +FRIO	**92**	5.02	10.854	0.07	7	2.79	21.687	0.05
TCPF +RIO	**152**	6.17	10.168	0.13	10	3.63	20.104	0.06
TCPF(O) +RIO	529	8.34	9.815	0.25	78	4.65	18.914	0.13
TB +RIO	**1200**	9.16	9.092	0.40	320	5.71	17.277	0.22
NoMark +FRED	**1315**	9.45	8.873	0.29	412	6.02	17.010	0.15
NoMark +TailDrop	**1518**	10.12	8.024	0.34	518	6.61	16.291	0.18
NoMark +RED	**1864**	10.75	8.316	0.45	689	7.03	16.825	0.25

Table 6: Results for Config C: SACK and Mistuned RED Parameters

large as 17-20% even with moderate levels of flow multiplexing (100 flows).

7 Summary

Our goal in this paper is to leverage the tools provided by diffserv: packet marking and differential dropping. TCP-friendly marking is an interesting and powerful alternative especially due to its capability to deterministically protect small window flows and retransmitted packets. The methodology of leveraging the marking and preferential dropping capabilities is clear: provision enough tokens, deterministically identify the packets which, when dropped, can disproportionately affect user performance and protect them, and then randomize or spread the remaining tokens over the rest of the packets. This type of transport-aware marking can be done by the end systems or by edge proxy devices. In the particular case of TCP, we have observed the following benefits:

- Reductions of up to two orders of magnitude in the number of timeouts;
- Consistently larger average data goodput (10-15%) for all TCP flows;
- 30-50% reduction in the total loss probability across all instances of implementation;
- Considerable improvements to the predictability (variance) of best-effort service. Service variance of course depends upon factors such as heterogeneous RTTs and multiple bottlenecks;
- Masking of the negative performance effects of RED parameter misconfiguration;
- Improvements irrespective of TCP implementation (Reno or SACK);

	100 flows				50 flows			
	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV	Time -outs	Ploss (%)	Average Goodput(kB/s)	CoV
TCPF +FRIO	**130**	6.33	10.668	0.08	13	3.43	21.315	0.05
TCPF +RIO	**239**	6.97	10.214	0.15	19	4.01	20.168	0.06
TCPF(O) +RIO	715	11.62	9892	0.28	175	6.04	18.231	0.18
TB +RIO	1410	13.98	8.954	0.43	413	7.8	17.112	0.33
NoMark +FRED	1586	14.78	8.345	0.38	512	8.32	16.993	0.22
NoMark +TailDrop	**1794**	**17.11**	7.810	0.41	646	9.55	15.212	0.27
NoMark +RED	**2163**	**20.58**	8.046	0.5	844	11.2	16.152	0.35

Table 7: Results for Config D: Reno and Mistuned RED Parameters

- All stateful and TCP-aware activity can be done at the network edge and not in the core;
- Can be incrementally deployed on a customer-by-customer basis, provided bottlenecks are upgraded to support differential dropping;
- Could be applied to peering points and international links (major congestion points) on the longer term to alleviate the effects of packet loss on TCP-flows.

In general, this work is complementary with other edge-to-edge [11] and end-to-end work to attain superior and scalable traffic management. The approach is consistent with the philosophy of moving complexity to network edges; the marker at this point is best deployed at the enterprise edge. The reduction in timeouts can also be used as the basis of differentiated TCP services offered by ISPs offered on private networks where a single ISP or limited multi-ISP networks support differential dropping. As mentioned earlier, this approach masks parameter sensitivities of RED and provides an incentive to deploy differential dropping and active queue management techniques at key congestion points. Finally, we would like to re-emphasize that these findings are experimental and not simulation based. Extending this work under the edge-to-edge overlay framework [11] in the real WAN environment represents our future effort.

8 Acknowledgment

The authors are grateful for thier colleagues' support to this work. Thanks also go to the reviewers for their comments on improving the quality of this paper.

References

[1] M. Allman, H. Balakrishnan and S. Floyd, "Enhancing TCP's Loss Recovery Using Early Duplicate Acknowledgment Response," *IETF Internet RFC 3042*, Janurary 2000.

[2] S. Blake et al, "An Architecture for Differentiated Services," *IETF Internet RFC 2475*, December 1998.

[3] B. Braden et al, "Recommendations on Queue Management and Congestion Avoidance in the Internet," *IETF Internet RFC 2309*, April 1998.

[4] D. Clark and W. Fang, "Explicit Allocation of Best-Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking*, V.6, N.4, pp.362-373, August 1998.

[5] W. Feng, D. Kandlur, D. Saha, K. Shin, "Adaptive Packet Marking for Maintaining End-to-end Throughput in a Differentiated Services Internet," *IEEE/ACM Transactions on Networking*, V.7, N.5, pp.685-697, October 1999.

[6] A. Feroz, S. Kalyanaraman and A. Kumar, "A TCP-Friendly Traffic Marker for IP Differentiated Services," *IwQoS'2000*, Pittsburgh, June 2000.

[7] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and Sack TCP," *Computer Communication Review*, V.26, N.3, pp.5-21, July 1996.

[8] S. Floyd, "A Report on Some Recent Developments in TCP Congestion Control," *IEEE Communication Magazine*, Vl39, N.4, pp.84-90, April 2001.

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

[10] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, V.1, N.4, pp.397-413, August 1993.

[11] D. Harrison and S. Kalyanaraman, "Edge-to-edge traffic control for the Internet," *RPI ECSE Technical report*, January 2000.

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

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

[14] V. Kumar, T.V. Lakshman, D. Stiliadis, "Beyond Best Effort: Router Architectures for the differentiated services of tomorrow's Internet," *IEEE Communications Magazine*, V.36, No.5, pp. 152-164, May 1998.

[15] D. Lin and R. Morris, "Dynamics of random early detection," *Proceedings of SIGCOMM'97*, September 1997.

[16] G. Lo Monaco, "TCP-Friendly building blocks implementation: Validation of a new approach to better best-effort service on the Internet," *RPI ECSE Technical Report*, May 2000.

[17] M. May et al, "Reasons not to deploy RED," *IWQoS'99*, June 1999.

[18] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP Selective Acknowledgment Options," *IETF Internet RFC 2018*, October 1996.

[19] M. Mathis and J. Mahdavi, "Forward Acknowledgement: Refining TCP Congestion Control," *Proceedings of SIGCOMM'96*, August 1996.

[20] R. Morris, "Scalable TCP Congestion Control," *IEEE INFOCOM'00*, Tel Aviv, 2000.

[21] T. J. Ott, T. V. Lakshman, L. Wong, "SRED: Stabilized RED," *IEEE INFOCOM'99*, April 1999.

[22] K. K. Ramakrishnan, and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," *IETF Internet RFC 2481*, January 1999.

[23] S. Sahu et al, "On Achievable Service Differentiation with Token Bucket Marking for TCP," *Proceedings of ACM SIGMETRICS'00*, Santa Clara, CA, June 2000.

[24] N. Seddigh, B. Nandy, P.Pieda, "Bandwidth Assurance Issues for TCP Flows in a Differentiated Services Network," *IEEE Globecom'99*, December 1999.

[25] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit and fast recovery algorithms," *IETF Internet RFC 2001*, January 1997.