TCP-Friendly Traffic Conditioners for Differentiated Services

Feroz Azeem, Amit Rao, Shivkumar Kalyanaraman
Contact email: shivkuma@ecse.rpi.edu
Department of ECSE, Rensselaer Polytechnic Institute,
Troy NY 12180-3590

Abstract

This paper examines the performance problems associated with TCP flows running over assured service with one bit drop preference (one of the proposed services in the differentiated services architecture). The use of TCP-friendly differentiated services building blocks (specifically traffic conditioners) are investigated to alleviate these problems. The paper examines test configurations and evaluates the behaviour and performance of TCP flows with and without the proposed conditioners.

1 Introduction

There is a distinct need to move towards providing service differentiation (diff-serv) in the Internet. Today's best-effort service model is not geared to support various levels of service to different users and the pricing of such services. The differentiated services architecture being developed in the IETF [2, 3] provides a framework within IPv4 to enable development of these features. The architecture distinguishes boundary nodes from interior nodes. Boundary nodes perform control plane functions such as policy, contracting, accounting and traffic conditioning (including metering, shaping, marking) whereas interior nodes need only implement a set of forwarding behaviors known as "per-hop behaviors" (PHBs). The PHBs at interior nodes are applied to traffic aggregates which are created at the boundary nodes and labeled using the DS field in IPv4 or IPv6 headers [4]. Observe that per-flow state information is no longer needed at interior routers to provide service differentiation – it can be (partially) inferred from the DS-field (or labels in MPLS [16]) which is used as an index to a table of PHBs.

A small set of PHBs are being standardized at the IETF to allow development of end-to-end differentiated services. One such PHB is the assured forwarding [1] (AF) PHB which is used to build the "assured service" (AS) which is roughly an emulation of the frame-relay CIR service. In this paper we look at TCP performance over a simplified form of the assured service (which corresponds to the original model suggested by Clark and Fang [5]). In particular, we assume that the interior routers have a single queue (class) and there are two levels of drop precedence within that class. The two levels are distinguished by two codepoints in the DS-field and we refer to packets as being either "in-profile" or "out-of-profile." The two-level drop precedence is implemented at interior nodes using the RIO (RED with In/Out) algorithm also proposed by Clark and Fang. This algorithm simply uses the RED drop scheme, albeit with different parameters for IN and OUT traffic.

Our focus in this paper is to examine the performance of TCP over this service and propose design of TCP-friendly building blocks as a generic technique to boost performance. The rest of the paper is organized as follows: Section 2 discusses issues involving TCP performance over the assured service. Section 3 contains descriptions of some proposed TCP-friendly building blocks. Sections 4 and 5 present performance analyses using these proposed building blocks. Section 6 presents a summary and conclusion.

2 Performance problems of TCP over assured service

It is well known that TCP Reno (the large installed base of TCP implementations) has serious performance problems if it encounters "per-connection burst drop" of packets, i.e., if a connection sees a number of packet drops with nearby sequence numbers. Even three consecutive packets dropped can lead to a timeout plus multiple SSTHRESH reductions [7, 12]. Also the definition of "burst" varies depending upon per-connection window size. For example a new connection with a small window may be hurt with a single packet drop itself [7].

At bottlenecks, packets may be dropped in bursts during congestion events. We distinguish these drops by calling them "aggregate burst drops" since they may span a number of connections. The number of packets dropped in a burst in such cases depends upon the time-scale of the congestion event at the bottleneck. Many congestion events last for a short time-scale. But even this results in unfairness (as measured by variance in per-connection goodput) because the aggregate burst drop is not spread fairly over all the congestion-causing connections. Such behavior is observable even in symmetric configurations where RTT does not vary [12]. The unfairness problem is exacerbated for high speed links, large number of sources and heterogeneous RTT cases [10, 11, 12].

We also find that the probability of aggregate burst drop is high when a majority of TCP connections are in the slow start phase (due to the super-linear nature of window increase). This is likely for sources performing short transfers (as in WWW applications). Further, the probability of per-connection burst drop increases since packets from each source tend to arrive in batches rather than being interleaved with other connection packets. Finally we find that optimization of what we call "service provider metrics" (utilization, queue length, drop rate) does not necessarily lead to optimization of "user metrics" (average goodput and variance in per-connection goodput).

Ibanez et al [11] observe similar problems when TCP runs over the assured service. Kim et al [15] use an improved version of TCP and see fewer problems. Feng et al propose an adaptive marker, TCP modifications and an improved PHB implementation using the ERED algorithm [8]. In contrast, we postulate that one important reason for these performance problems is the lack of TCP-friendly building blocks (droppers, markers, shapers, PHBs) in the network. By "TCP-friendly" we mean the capability of these building blocks to:

- provide for packet interleaving,
- protect small-window connections from drop,
- convert aggregate burst drop into interleaved non-bursty per-connection drop and
- reduce packet burstiness created by TCP.

We present results of our preliminary work on developing such components and find that TCP-friendly components allow a marked improvement in performance. Though there is room for improvement in several of our components, we find that the single most critical component is the shaper which directly provides packet interleaving and reduces TCP burstiness.

3 TCP-Friendly Building Blocks

In general each and every building block along the connection path could be made TCP-friendly or the TCP friendly functions can be aggregated at the edge (as done in a product by Packeteer Inc. [12] currently for best-effort traffic). We identify some components which particularly relate to diff-serv.

3.1 Packet Marker

A packet marker is one of the components of the traffic conditioner. The general problem in a marker is to optimally allocate a currently available pool of tokens (which depends upon contracted rate and burstiness parameters) to a set of incoming packets in a given interval of time. Packets which are allocated tokens are said to be marked "IN" and those which do not get tokens are said to be marked "OUT" in our service.

Our simple TCP-friendly marker is an extension to a token bucket marker. The idea is to try to avoid marking packets as "OUT" in a bunch because these packets will then have a high probability of (burst) drop. Given a set of tokens in an interval, it marks consecutive packets in the beginning of the interval as "IN profile" until it hits either the middle of the interval or finishes half the available tokens. In the latter case, it marks every other subsequent packet as "OUT of profile". Excess tokens (upto a limit) are transferred to the shaper (or to the next interval if a shaper is not available) so that it can re-mark some OUT packets as IN if necessary. If we run out of tokens, we mark new packets as OUT. The rough pseudo code is given below.

num_tokens = number of tokens handed out so far. Set to zero at the beginning of each interval.
max_tokens = maximum number of tokens that can be handed out in any interval. This corresponds to the contracted rate (profile).

Marker Implementation

```
On packet arrival
if num_tokens < max_tokens/2 then
  Mark packet as IN
else if num_tokens ≥ max_tokens/2 before middle of interval is crossed then
  Mark alternate packets IN or OUT {threshold crossed}
else if num_tokens < max_tokens then
  if threshold crossed in the first half interval then
    Mark alternate packets IN or OUT
  else
    Mark packet as IN
  end if
else
```

Mark packet OUT { No More Tokens Left}
end if

While the goal is to convert an "aggregate burst drop" into an "interleaved per-connection drop," the scheme suffers from the fact that packets with nearby sequence numbers can be marked "OUT" and hence dropped with high probability. It also does not provide complete protection for small window connections. There is room for significant improvement of this scheme and we are working on several variants. Accordingly, our performance analysis indicated that the marking strategy did not significantly explain variation in metrics, except for the fact that the total number of packets dropped was usually smaller when this marker was used.

3.2 Packet Shaper

A packet shaper is a component of the traffic conditioner which delays some or all of the packets in a traffic stream to force compliance with its traffic profile. We develop a simple TCP-friendly shaper which shapes output rate to be a scaled factor of the measured, smoothed input rate. The shaper calculates the average inpute rate (the number of packets that arrive into the node) using a low-pass filter with an exponential weighted moving average. The output rate is set to twice the average input rate. Excess tokens received from the packet marker are used to re-mark some OUT packets as IN when the packet is being transmitted. However, to avoid infinite overflow of tokens from the marker to the shaper (and the resulting bustiness on the network in terms of marked packets), the total number of tokens which can be carried over is limited by a configurable parameter.

Observe that the shaper described above is different from conventional token bucket shapers (which are based upon a peak/average rate and a burstiness parameter). This shaper requires just a measurement interval and an exponential averaging parameter to be configured. This shaping can be done at a per-flow or per-behavior aggregate level.

The idea in this TCP-friendly shaper is to reduce packet burstiness by smoothing out the input burst while roughly preserving the input rate. If done at a per-flow level or over an aggregate of a small number of flows (which we call "hierarchical shaping"), the shaping mechanism also effectively interleaves packets of multiple flows. The cost of shaping is delay at the shaper. We find that even this simple shaping mechanism provides a substantial improvement in performance and we are working on new versions of the shaper which counter the delay penalty using adaptive parameter settings.

3.3 TCP Rate Increase Damper

One reason for overload, burstiness and packet drop is the rate of increase of TCP window itself. Therefore one possible way to counter this issue is through limiting the rate of increase of TCP windows. This can be achieved at an edge node by limiting the rate of release of acks [12]. For our simulation purposes, we approximate this functionality by implementing it in the source TCP simulation code as described below.

Normally, in TCP *slow start* phase, when each ack is received, the congestion window is increased by one segment. The sender can transmit upto the minimum of the congestion window (*cwnd*) and the advertised window [9]. The congestion window thus increases exponentially as a function of round trip time. In our modified TCP simulation code, we allow *cwnd* to increase by 1 segment for every 2 acks received. This is achieved through the use of a single bit control variable which is flipped and tested each time an ack is received. The rise of *cwnd* is thus limited to a factor of 1.5 rather than 2. Since a long-lived TCP connection may reside only temporarily in the slow-start phase, we use same rate-increase-damping policy during congestion avoidance, i.e., we limit window increase to $1/cwnd$ for every two acks received. To avoid negative interactions with small window connections, we trigger these policies only when the window size is at least 8 segments.

We find that the damping of TCP rate increase is most effective in smaller RTT configurations (small WANs or MANs) which also have high bandwidth and a large number of flows. In such configurations, the TCP window-increase cycles (which are proportional to RTT) are very small. This leads to faster TCP window increases, leading to overload and bursty packet losses at the bottleneck. The damper is useful here because it slows down TCP's window increase and allows better buffer management at the bottleneck (because of the reduced average overload).

Note that the TCP rate-increase damper outlined above is an example of a crude TCP rate damper. Packeteer Inc. [12] builds sophisticated TCP rate-control components. These components can control TCP performance more tightly and over a wider range of RTTs through control of the left, right edges of the TCP window in addition to control of the ack rate as suggested here.

3.4 Drop policies and Edge-to-edge feedback control

Drop policies are part of the per-hop behavior (PHB). RED is the first example of a TCP-friendly building block (as compared to drop-tail). Examples of even more TCP-friendly drop policies are FRED and ERED [7, 8] which can allocate loss probabilities during a congestion event in a more controlled manner. While we have work-in-progress in this area, we do not present results about this dimension in this paper.

Edge-to-edge feedback control introduced in an earlier work by our group [13] involves a simple protocol between ingress and egress conditioners. Though our initial work in the mentioned reference involves participation of interior router, our current work involves only the two edge conditioners. In this sense, edge-to-edge control is very similar to end-to-end control and it does not require standardization or interior router participation. Edge-to-edge control is TCP-friendly in the sense that it allows better use of distributed buffer resources in the network to reduce packet drop rate and probability of burst drops. It also provides limited protection against denial of service attacks and improves fairness in resource allocation. We are developing it to provide a basis for some forms of traffic engineering and congestion-sensitive pricing. The key issue is the control overhead required for achieving the different goals just mentioned. This is work-in-progress and simulation results are not reported in this paper. Our focus in this paper is on TCP-friendly traffic conditioners alone.

Observe that these solutions can work at different levels of granularity: from per-microflow to per-behavior-aggregate. The former solutions tradeoff increased complexity for tighter control over performance. These are just examples of components which can be adapted to accomodate TCP especially and non-TCP type of flows.

4 Simulation and Performance Analysis

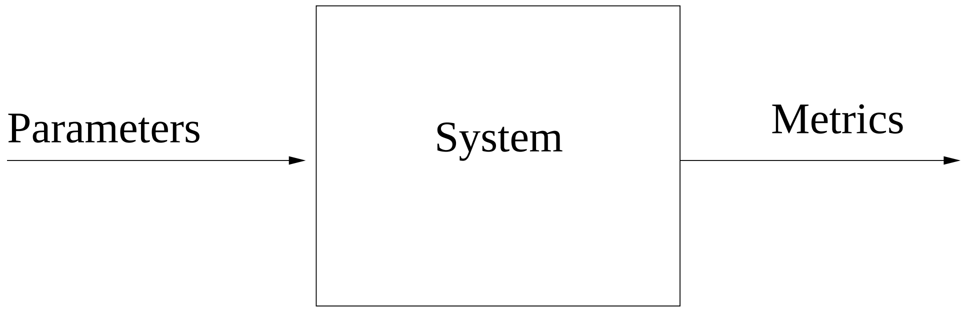

Figure 1: Model for performance evaluation

We use the high level model shown in Figure 1 to guide our evaluation. Specifically, 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 following sections explain the choice of metrics and parameter dimensions explored in this study.

4.1 Parameters and configurations

The configurations for the simulations involve changes to one or more of the following parameters:

- TCP-friendly component (or combination of components) used
- Granularity of shaping and marking (per-flow or hierarchical). For eg, in "hierarchical" shaping, the shaper deals with an aggregate of a set of flows.
- Bottleneck link speed: 1.5 Mbps or 150 Mbps
- Round trip time: 10 ms or 30 ms
- Number of connections: 10 or 100 flows
- Staggered connection start times

4.2 Base Configuration

Figure 2 shows our base configuration – later configurations are derived based upon this. We have 10 sources which are divided into groups of five each. R0 and R1 incorporate the shaper and marker building blocks building blocks and handle traffic from five flows each. R2 is the bottleneck router

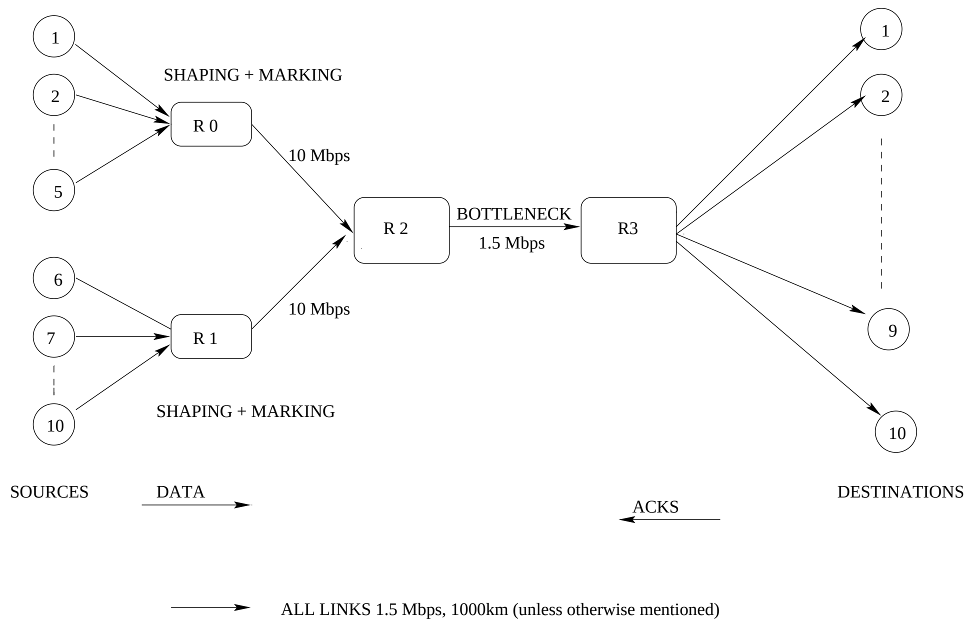

Figure 2: Base Configuration

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 shaper/marker through links of 1.5 Mbps. The shaper/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.

4.3 Metrics

We classify metrics into two major categories: user metrics and operator metrics. Although the two are related, an end user is typically concerned about response (which is measured as average and variance in per-user goodput), and the provider is interested in optimally utilizing his costly resources.

4.3.1 Operator metrics

The operator's key resources are bandwidth and buffers, and is willing to tradeoff (cheaper) buffer resources to ensure high utilization of bandwidth. But high queueing 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. **Average Queue Length**: Low average queue lengths imply lower average queueing delay experienced by participating connections. The operator prefers low queue lengths combined with high utilization.
3. **Maximum Queue Length**: Very high maximum queue lengths indicate high buffer requirements.
4. **Packet drop rate**: Packets dropped represent wasted bandwidth and buffer resources on upstream links.

4.3.2 User Metrics

The user is interested in response (per-flow) which can be approximated using the following metrics:

1. **Average (per-flow) Goodput**: This quantity which excludes retransmitted packets should be as high as possible.
2. **Standard deviation in (per-flow) goodput**: This quantity is a rough measure of fairness. Ideally, for a single bottleneck with infinite transfers, this metric should be close to zero.

We conduct a full factorial simulation and use linear regression modeling as described in [14]. In this study, we have not examined the effects of dimensions such as:

- Multiple classes
- Multiple drop levels
- Variable capacity bottlenecks
- Heterogeneous RTT bottlenecks
- Sharing of assured service class by TCP and non-TCP traffic

5 Simulation Results

In the following sub-sections we present key results from our simulations. The TCP-friendly shaper used accounted for the variation in a majority of metrics in a majority of the parameter dimensions explored. The user metrics (average and standard deviation in goodput) could be reasonably optimized. The cost of this optimization was an increased average queue length and drop rate in certain cases. The preliminary TCP-friendly marker design reduced the total number of dropped packets (a provider metric), but did not significantly affect user metrics indicating room for further refinement. The TCP rate damper positively affected user metrics namely average goodput, standard deviation in goodput in small and medium RTT configurations. However with higher RTT configurations TCP rate damping did not show a perceptible improvement.

Each of the following subsections has a table wherein the numbers represent the percentage of variation in the metric (the columns). These numbers are generated by a linear regression model fit to the results obtained [14]. Specifically, a larger percentage denotes profound dependency of the metric on that parameter. A "-" denotes no perceptible dependency.

5.1 Base configuration: low speed, large RTT, hierarchical conditioners

	Operator Metrics				*User Metrics*	
Parameters	**Avg Q length** percent	**Max Q length** percent	**Timeouts** percent	**Drops** percent	**Avg Goodput** percent	**SD in Goodput** percent
T	-	-	-	-	-	-
S	**62**	**63**	-	-	-	**26**
M	16	-	-	**39**	8	-
U	-	9	7	-	7	10
MU	-	-	**32**	-	7	-
TS	-	-	-	-	7	11
SM	18	8	-	**48**	7	7
TMU	-	-	**21**	-	8	-
TSMU	-	-	8	-	7	7

Table 1: 10 Sources ; Hierarchical shaping/marking, 1.5 Mbps link speeds
LEGENDS T = TCP Rate Damper S = Shaper M= Marker U = Staggered sources

Scanning through the rows of Table 1, we observe that shaping has the maximum effect on metrics such as queue lengths and standard deviation in goodput (observe the larger numbers in the highlighted row). The larger numbers simply mean that the particular parameter (in this case, shaping) has a profound effect (possibly good or bad) on that single metric – we desire good performance in terms of a number of metrics.

Specifically, we observed that we could not simultaneously achieve good performance in user metrics and optimize queue lengths. So, we note that with shaping we can tradeoff queue lengths for better average and standard deviation in per-user goodput. For example, the larger queue sizes were the result of shaping parameters. The reason for this performance is that shaping reduces the probability of TCP synchronization by spreading out bursts in time (and also introducing some interleaving). So a single congestion event does not result in burst drop for a single connection (or a small subset of connections). Also, the smoothing of bursts meant that the first packet dropped occurs at a higher window value. This in conjunction to the fact that average queues are high means that the flows get higher average throughput.

We also observed that the TCP-friendly marker typically reduces the number of drops, though its influence on user metrics (average and standard deviation in goodput) is minimal.

5.2 Simulation results with Small RTTs

In these simulations, we use the base configuration, but make all links as 1km. Table 2 summarizes the results. Again scanning through the rows of Table 2, we observe that shaping still has a significant effect (larger numbers in the rows). However, the effect TCP rate increase damping has increased significantly in this situation. Specifically, TCP rate increase damping accounts for nearly 94 percent of variation in fairness.

The reason for the increased effect of the TCP rate damping is that smaller RTT's imply faster window increases, which in turn leads to burstiness and variance in throughput. Now, with rate increase damping, the aggregate rate increase (i.e. overload seen at the bottleneck) is reduced. This reduces probability of bursty losses, leading to better fairness.

The marker in this case also leads to reduced packet drop rate, though the effect on user metrics is not appreciable. The shaper affects timeouts significantly due to increased queueing delays caused by it.

	Operator Metrics				*User Metrics*	
Parameters	**Avg Q length** percent	**Max Q length** percent	**Timeouts** percent	**Drops** percent	**Avg Goodput** percent	**SD in Goodput** percent
T	-	-	-	-	**94**	6
S	**49**	**39**	**37**	**27**	-	8
M	8	-	-	9	-	12
U	-	-	15	-	-	**22**
SU	-	-	**25**	-	-	-
TM	-	-	-	-	-	**23**
SM	9	9	-	10	-	-

Table 2: 10 Sources ; Hierarchical shaping/marking, 1.5 Mbps link speeds, All links of 1km - small RTT's

LEGENDS T = TCP Rate Damper S = Shaper M= Marker U = Staggered sources

5.3 Simulations with Per-flow traffic conditioning

In this section, we examine results with traffic conditioning (shaping and marking) done at a per-flow granularity. We approximate it by assuming that the conditioning elements are implemented in the source itself. The simulation results are summarized in Table 3. We note again that the shaper has maximum effect on most of the metrics - it tends to increase the average and maximum queue lengths, number of drops, average goodput and number of timeouts. The marker has a slight effect on a few metrics.

The profound effect of shaping is due to the increased interleaving and control of burstiness possible by doing shaping at a per-flow granularity. As explained in the earlier sections, shaping also results in a stable larger average queue which in turn accounts for a higher drop rate (due to the RED algorithm).

	Operator Metrics				*User Metrics*	
Parameters	**Avg Q length** percent	**Max Q length** percent	**Timeouts** percent	**Drops** percent	**Avg Goodput** percent	**SD in Goodput** percent
T	-	-	-	-	-	-
S	**90**	**85**	**86**	**92**	**91**	7
M	8	6	7	-	-	12
U	-	-	-	-	-	**32**
TM	-	-	-	-	-	**30**

Table 3: 10 Sources ; Per-flow shaping/marking, 1.5 Mbps link speeds, All links of 1000km
LEGENDS T = TCP Rate Damper S = Shaper M= Marker U = Staggered sources

5.4 Simulation results with high speed links

In this section we consider simulation results of the base configuration, with the change that all links run at 150Mbps. The results are summarized in Table 4. We observe again that the shaper has maximum effect on most of the metrics, achieving better user metric performance at the expense of increased queue lengths and packet drop rate. Effects of other parameters are nominal and scattered in terms of metrics affected.

	Operator Metrics				*User Metrics*	
Parameters	**Avg Q length** percent	**Max Q length** percent	**Timeouts** percent	**Drops** percent	**Avg Goodput** percent	**SD in Goodput** percent
T	-	-	-	-	-	-
S	**31**	**68**	**38**	**57**	**43**	**55**
M	-	-	-	15	7	-
U	-	-	-	-	**28**	-
MU	**25**	-	-	-	-	-
TS	8	9	-	10	-	-
SM	-	-	5	7	-	-
SMU	**25**	-	-	-	-	-
SU	-	-	-	-	**28**	14

Table 4: Configuration E ; 10 Sources ; Hierarchical shaping/marking, 150 Mbps link speeds, All links of 1000km
LEGENDS T = TCP Rate Damper S = Shaper M= Marker U = Staggered sources

5.5 Simulations with large number of sources

In this section, we modify the base configuration to have 100 flows, hierarchical traffic conditioning, bottleneck speeds at 1.5 Mbps and link lengths 1000 km. The traffic conditioning (shaping and marking) is done for sets of 5 sources (just like in the base configuration). The simulation results are summarized in Table 5. Again we find that the shaper has the maximum effect on almost all the metrics, with negative effects on queue lengths and packet drop rate (as observed earlier).

	Operator Metrics				*User Metrics*	
Parameters	**Avg Q length** percent	**Max Q length** percent	**Timeouts** percent	**Drops** percent	**Avg Goodput** percent	**SD in Goodput** percent
T	-	-	-	-	-	-
S	**74**	**68**	**91**	**97**	-	**77**
M	-	-	-	-	-	-
U	7	11	-	-	**41**	-
SM	7	-	-	-	-	-
SU	-	10	-	-	**48**	7

Table 5: Configuration F ; 100 Sources ; Hierarchical shaping/marking, 1.50 Mbps link speeds, All links of 1000km

LEGENDS T = TCP Rate Damper S = Shaper M= Marker U = Staggered sources

6 Conclusion And Future Work

In summary, we looked at generic TCP performance problems which remain in differentiated services, and demonstrate that simple enhancements in some of the building blocks (traffic conditioners, PHBs, closed loop edge-to-edge control) can lead to significant performance enhancements. Specifically, in this paper we look at traffic conditioners and observe that even a simple TCP-friendly shaper has the potential to change performance characteristics significantly. While we have presented crude initial versions of these building blocks, there is scope for work in improving the design of such building blocks and hence TCP performance over diff-serv.

References

[1] J. Heinanen, et al., Assured Forwarding PHB Group. IETF Internet draft *draft-ietf-diffserv-af-05.txt*, February 1999.

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

[3] S. Blake, et al., “A Framework for Differentiated Services”, IETF Internet Draft, *draft-ietf-diffserv-framework-01.txt*, October 1998.

[4] K. Nichols, et al., "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", *Internet RFC 2474*, December 1998.

[5] 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.

[6] 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.

[7] Dong Lin and Robert Morris, "Dynamics of Random Early Detection", Proceedings of SIGCOMM'97, August 1997.

[8] W. Feng, D. Kandlur, D. Saha, K. Shin, "Techniques for Eliminating Packet Loss in Congested TCP/IP Networks," U. Michigan CSE-TR-349-97, November 1997.

[9] Stevens, W. R., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", *Internet RFC 2001*, January 1997.

[10] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," Proceedings of SIGCOMM'98, Vancouver, August 1998.

[11] J. Ibanez, K. Nichols, "Preliminary Simulation Evaluation of an Assured Service", IETF Internet Draft, *draft-ibanez-diffserv-assured-eval-00.txt*, August, 1998.

[12] Prasad Bagal, Shivkumar Kalyanaraman, Bob Packer, "Comparative study of RED, ECN and TCP Rate Control," Technical Report, March 1999. Available from http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers-rpi.html

[13] S. Kalyanaraman, D. Harrison, S. Arora, K. Wanglee, G. Guarriello, "One-bit Feedback Enhanced Differentiated Services Architecture," IETF Internet Draft, *draft-shivkuma-ecn-diffserv-00.txt*, March 1998. Available from: http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers-rpi.html

[14] Raj Jain, *The Art of Computer Systems Performance Analysis*, John Wiley and Sons Inc., 1991.

[15] Hyogon Kim and Will E. Leland and Susan E. Thomson, "Evaluation of Bandwidth Assurance Service using RED for Internet Service Differentiation," *Submitted to INFOCOM'99*, 1999.

[16] R. Callon et al, "A Framework for Multiprotocol Label Switching," *IETF Internet Draft, draft-ietf-mpls-framework-02.txt*, November 1997.