Analysis of a Method for Differential TCP Service

Panos Gevros, Fulvio Risso and Peter Kirstein
Department of Computer Science,
University College London,
Gower Str. WC1E 6BT,
London, U.K.
phone +44 (0)20 7679 3666, fax +44 (0)20 7387 1397
{p.gevros, f.risso, kirstein}@cs.ucl.ac.uk

Abstract—**Recently we have witnessed an increasing interest in providing differentiated Internet services, departing from the traditional notion of fairness in the best effort service model. However research efforts have almost exclusively focused on routers, by enhancing their scheduling and queue management capabilities in order to treat flows according to policies. There has been much less work on transport level approaches to differentiated services. MulTCP [1], is the only piece of work in this direction known to the authors. In this paper we briefly describe the MulTCP modifications in TCP's congestion control mechanism, its implementation in a BSD networking stack and present some experiences from a series of experiments over real networks comparing its performance when implemented on different TCP variants. Our results were particularly interesting in the case of RED gateways. We comment on the effectiveness and scalability of the differentiation mechanism and conclude that in certain popular environments the proposed method for transport level differentiation can be both feasible and effective.**

I. Overview

The Internet has long been changed from the network used only by academics and researchers to an environment with large and rapidly growing commercial interest. Users do real business on it, consider it vital and are willing to pay more for a better service. It is evident that a datagram network where anyone is treated like anyone else does not scale from an economic viewpoint; even if people value their traffic differently their data are still treated the same way.

Although the bandwidth available is increasing and prices are dropping, the traffic on the network is increasing at least as fast; in the near future there will not be enough bandwidth to satisfy everyone's needs. It is expected, therefore, that the Internet will have to live with congestion for some time. In order to avoid ending up being what is called "the tragedy of the commons', some sort of control on the performance it delivers to its various users is clearly needed. The major challenge is finding scalable solutions, which can be deployed incrementally without major changes to the current infrastructure.

The commercialisation of the Internet is changing the premise of resource allocation from the traditional "best effort" to one based on user profiles. Most customers want and are willing to pay for preferential treatment of their traffic. Thus the network providers could satisfy this need and generate additional revenue from the same investment in network resources if the appropriate mechanisms and policies were in place. Moreover part of the revenue would finance the deployment of additional resources.

The "integrated services" was a first attempt in the IETF to offer guaranteed end-to-end service based on reservations using RSVP signalling and appropriate traffic control modules in the routers like Class Based Queueing and Weighted Fair Queueing. However it soon became evident that this approach does not scale due to the per flow state required in the routers in the core of the network. RSVP will not be an answer to all the problems as originally it had been thought.

The problem of providing consistent service differentiation with strictly quantifiable service levels across the Internet is a very hard one. However it seems widely accepted that even statistically guaranteed service levels for "premium" customers would suffice for differential pricing.

TCP is by far the most widely used transport protocol accounting for over 80% of the total Internet traffic which appears to be originating from large servers to the periphery. TCP has window based congestion avoidance and control [2] based on the well proven principles of additive increase and multiplicative decrease (AIMD) [3] ; it interprets packet drops as congestion signals, ensuring stability and efficient resource utilisation. However as discussed in [4] the TCP window mechanism does not generally achieve fairness. Results published recently in [5], showed that rate control based on additive increase and multiplicative decrease achieves proportional fairness. Moreover in a weight proportional fair system where weights are the prices the users pay per time unit, both the utility perceived by each user and the total utility perceived by the network provider are maximised. This was acknowledged in MulTCP [1] which modifies the standard congestion control behaviour of TCP in an attempt to emulate the behaviour of multiple concurrent TCP connections using the same control loop.

The main contribution of this work is the performance evaluation of a modified, intentionally more aggressive TCP

(MulTCP) in terms of average throughput in real networks with both drop tail and Random Early Detection gateways. For our study we used real TCP stacks and carried out experiments both in the field and in a controlled testbed.

The paper is organised as follows; Section II provides an essential brief overview of MulTCP (details can be found in [1]) and discusses implementation issues. Section III describes the environment, the setup and the methodology of our experiments. Section IV examines the impact of different TCP flavours (Reno TCP and TCP SACK). Section V presents the results obtained with router mechanisms like WFQ, CBQ and RED. MulTCP problems and limitations are reported in Section VI. Section VII suggests potential environments where such a mechanism could be successfully deployed and finally Section VIII has discussion and conclusions.

II. The MulTCP Algorithm

Any reasonably long-lived TCP connection goes through different phases; start up, experiencing losses and reaching steady state. During *slow start* the congestion window (`cwnd`) of standard TCP is opened exponentially by sending two packets for every received acknowledgement (ACK): *one* for the ACK which says that a packet has left the pipe (principle of conservation of packets) and *one* in order to open the congestion window by one more packet in an attempt to quickly find and utilise all the available bandwidth [2].

Behaving like an aggregate of N virtual TCP connection in the same control loop requires the connection to start by sending N packets, N acknowledgements being received and $2N$ packets being sent out after one round trip time. This clearly results in large bursts that lead to losses and prevent the connection from rapidly reaching steady state. To deal with this problem, MulTCP sends *3 packets* (instead of N) for each received ACK until it opens its congestion window as far as N TCP connections would have done cumulatively[1]. Assuming that this happens after k_N RTTs, MulTCP will have a congestion window of 3^{k_N} and N TCPs will have (in total) a congestion window of $N \cdot 2^{k_N}$ so:

$$k_N = \frac{\log N}{\log 3 - \log 2}$$

MulTCP introduces a new threshold value during the *slow start* phase that is 3^{k_N} (variables `multcp_cwthresh` and `wN` in Figure 1). Below this value MulTCP's congestion window is in rapid exponential increase (3 packets sent for each ACK received); above this value it increases exponentially as standard TCP does during the slow start phase (2 packets sent for each ACK received) until it reaches `ssthresh`. Above `ssthresh` the congestion window increases by *one packet per window of data* (or $\frac{1}{cwnd}$ per packet) probing the network for available bandwidth. MulTCP on the other hand attempts to emulate N TCPs during its linear increase phase by opening `cwnd` by $\frac{N}{cwnd}$ per packet, as shown in Figure 1.

A packet drop in a window of data is interpreted by any conformant TCP as congestion indication and it normally halves its congestion window and sets `ssthresh` to the new `cwnd` value. In the N TCP connections case, only one of the connections will halve its congestion window when a packet is lost, so MulTCP will set `cwnd` and `ssthresh` to $\frac{N-0.5}{N}$ of the initial `cwnd` value. When there are multiple losses in the same window of data and a *timeout* occurs, standard TCP sets `ssthresh` to half the value of the `cwnd` and then reduces `cwnd` to one segment and goes in slow start.

A single MulTCP connection is more prone to timeout when compared with N distinct TCP connections because with the same number of losses distributed over N connections, the probability that one of them will experience enough of the losses to cause a timeout is smaller. After a timeout MulTCP sets `ssthresh` to $\frac{N-0.5}{N}$ of `cwnd` value instead of half, `cwnd` is set to 1 segment and performs its own version of slow start. This is the only thing one can possibly do to make MulTCP resemble N TCPs[2].

A. Implementation issues

We modified TCP in the FreeBSD kernel[3] to allow assigning a multiplicative factor to a TCP connection from user space. We defined the `TCP_MULT` socket option in the sockets API and used the `setsockopt()` system call to set it. An application can select the multiplicative factor from the supported preconfigured range of values. The standard TCP behaviour is used when the factor is not specified explicitly.

The MulTCP factor (the unsigned integer variable `mult`) becomes an important attribute of the TCP connection and is therefore kept in the TCP control block (`struct tcpcb`) which also holds information about the connection state, the associated local process and transmission-related parameters like *cwnd* and *ssthresh*.

We have also introduced two tables of precomputed values needed for the MulTCP operation, indexed by the multiplicative factor N:

- `multcp_cwthresh[N]` holds the number of segments up to which the congestion window can grow increasing by 3 segments each time an ACK is received.
- `multcp_cwdecr[N]` holds the ratio by which the congestion window is decreased each time loss occurs.

[1] We assume that the N TCPs perform slow start for the first time (i.e. not after a timeout), so that the exponential increase phase is not limited by the `ssthresh` value.

[2] Indeed after exiting slow start MulTCP will have the same congestion window as N TCPs would have if one of them had performed slow start.

[3] FreeBSD-2.2.5-RELEASE, a 4.4 BSD derived Unix.

```
{
 register u_int cw = tp->snd_cwnd;
 register u_int N = tp->mult;
 register u_int wN =
multcp_cwthresh[N] * tp->t_maxseg;
 register u_int incr = tp->t_maxseg;

 if (cw >= tp->snd_ssthresh)
    /* linear increase,
       N maxseg per window */
    incr = N * incr * incr / cw;
 else
    /* congestion window < ss_thresh,
       slow start */
    if (cw < wN)
        incr = 2 * incr;
    tp->snd_cwnd = min(cw + incr, \
              TCP_MAXWIN<<tp->snd_scale);
}
```

Fig. 1. Modified TCP slow start/congestion avoidance, from BSD TCP implementation.

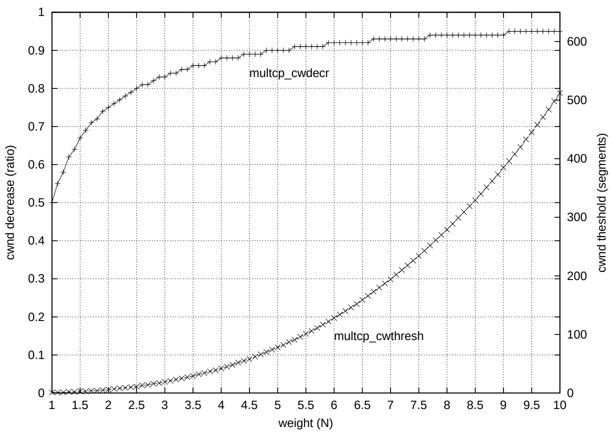

Fig. 2. Values of the cwnd decrease ratio and its threshold for various MulTCP factors.

These values (Figure 2) can be viewed as a measure of *how responsive* to congestion a MulTCP connection is and they have been previously calculated to avoid expensive arithmetic in the kernel.

III. Experiment Setup

Experiments were done in a controlled network environment and over the wide area Internet. The controlled environment experiments were done on the CAIRN [6] network using a transatlantic ATM PVC between UCL and the NASA Goddard Space Flight Center over which we had complete

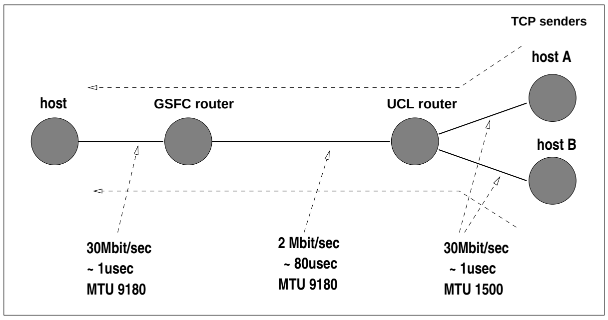

Fig. 3. Topology in the experiments between UCL and NASA.

control. The Internet experiments were done between UCL and the University of Pisa over the European academic IP network.

Our goal was to determine to what extent an approach based on modified TCP congestion control algorithm can provide weighted proportional performance differentiation between TCP flows. For our evaluation we used different TCP implementations (Reno, SACK) and routers that implemented drop-tail and Random Early Detection (RED) queue management (in the controlled environment only). This *soft* transport level approach to differentiated service has also been compared with network level mechanisms (WFQ, CBQ) for per flow guaranteed service (delay and throughput), when these mechanisms were applied on a single hop in the controlled environment case.

A. Controlled Environment and Internet Path

The topology used in the controlled environment experiments is shown in Figure 3. Both the hosts and the routers were PCs running the FreeBSD operating system. The hosts had modified kernels for MulTCP (needed only at the senders) and SACK (needed at both senders and receivers); see [7] for SACK implementation details. The routers were running ALTQ [8] kernels capable of Class Based Queueing, Weighted Fair Queueing and RED. The links were all ATM PVCs, rate limited to the appropriate bandwidth. The link MTU on the senders was set to 1500 bytes for several reasons; first to avoid influencing TCP performance by using large MSS size, because we wanted the results to be applicable to usual Internet paths where MSS is dominated by the Ethernet MTU size (1500 bytes) and in order to make the windowing behaviour more obvious than with the 9180 bytes MTU of the ATM links (since the window in segments will grow larger for the same size of the end-to-end pipe when the segments are smaller).

The aim in the Internet experiments was to evaluate the robustness of the MulTCP differentiation mechanism under real world congestion conditions and to see how it compared with standard TCP flows [9]. The routers in the experiments between UCL and University of Pisa implemented drop-tail queue management and the maximum segment size was 1460 bytes; we had no control over the 16 hops path which was verified to be symmetric.

B. Methodology

Each experiment involved initiating a number of TCP connections between the senders and the receiver using `ttcp`. The number of connections should be sufficiently large for creating contention for buffer space and subsequently packet drops at the bottleneck router. In the controlled environment case usually there were twelve flows; in the Internet experiments the number of connections was typically smaller (usually two to five), since creating congestion in the wide area Internet path was not really an issue. The traffic was captured using `tcpdump` near the receiver and the traces were analysed off-line to obtain the TCP sequence number over time plots. Internet tests were performed several times during different times of day to eliminate bias due to specific network conditions at the time of an experiment.

The duration of an experiment usually was in the order of seconds so that it can be safely assumed that traffic conditions on the path were relatively constant for the duration of an experiment. However traffic conditions may vary across different experiments; this was particularly true in the Internet tests where different TCP throughput was observed in different experiments. Therefore we argue that only qualitative comparisons of weighted differentiation should be made across different experiments, without comparing absolute throughput values which could vary widely. However almost identical behaviour was observed when the same experiment was repeated several times in the controlled environment.

C. MulTCP and Differentiation Criteria

We explored the entire parameter space (1 to 10) for the weight assigned to each TCP flow (multiplicative factor N) but for reasons described in Section VI we focused on $N = 2$ aiming at a TCP with 100% gain in throughput, one which would perform as the aggregate of two TCPs. From this point on when the term MulTCP connection is used we will assume a weight of two unless otherwise stated.

For TCP the primary differentiation metric is *throughput*. Differentiation among various TCP flows needs to be observed when all TCPs in the test are simultaneously active, because when the first connection terminates the rest compete for the freed up bandwidth and usually manage to increase their throughput. This is not so important in the Internet experiments because the bottleneck is shared by a much larger number of TCP connections and the effects of re-distributing freed-up bandwidth were not noticeable. In a ideal proportionally fair system the bandwidth that becomes available should be distributed between the remaining connections according to their relative weights.

IV. TCP IMPLEMENTATION ISSUES

This section discusses MulTCP performance when implemented on different TCP variants. Although MulTCP needs to be implemented only on the sender host, its performance is strongly influenced by the TCP implementations of both sender and receiver. We examine TCP Reno and TCP with Selective Acknowledgements (TCP SACK), since they use quite different mechanisms for detecting and responding to packet loss. Packet loss is the primary and usually the only congestion signal available to TCP and this is the reason for the coupling between error and congestion control. Finally, MulTCP performance with different kinds of receivers (FreeBSD and Windows 95) is examined.

A. TCP Reno

TCP Reno, the de facto TCP standard, is known to have performance problems during its recovery phase when multiple losses occur in the same window. This happens because a sender can only learn about a single packet loss per round trip time due to the limited information carried by cumulative acknowledgements. An aggressive sender might decide to retransmit packets early, but such retransmitted segments may have already been received successfully. There is a trade off between unnecessary retransmissions of segments that have already being received and unnecessary delays in useful retransmissions of segments that were actually lost. Eventually multiple losses in the same window of data cannot be recovered with fast retransmits; Reno TCP often has to wait for the retransmit timer to expire and then go into slow start. The timeouts are idle periods that have severe impact on TCP throughput.

The problem is exacerbated with large windows such as the MulTCP case. Assuming intuitively the same loss probability for each packet, it is more likely to experience irrecoverable losses when the window becomes larger. It has long been realised that the performance of TCP with large windows will remain handicapped until the SACK option is added to TCP [10]. Indeed both the simulations in [1] and the real network experiments in [9] showed a clear difference when SACK is used with MulTCP.

Figure 4 shows typical traces of two competing Reno TCPs from the experiments over the Internet path. Despite the fact that in this trace MulTCP finishes first, this is obviously not what could be described as weighted proportional differentiation. Both TCPs experience a number of timeouts evident from the breaks in the continuity of the curves (*pipe breaks*).

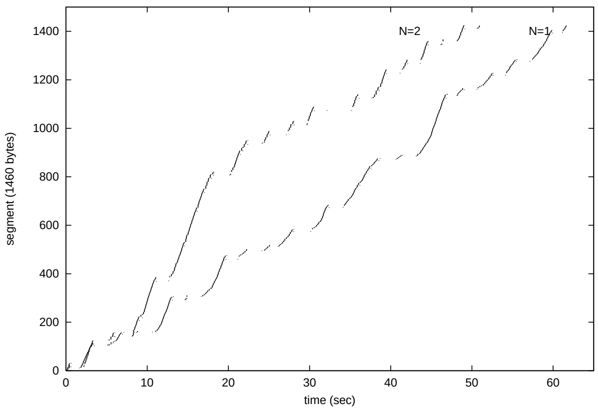

Fig. 4. TCP Reno traces from Internet experiments.

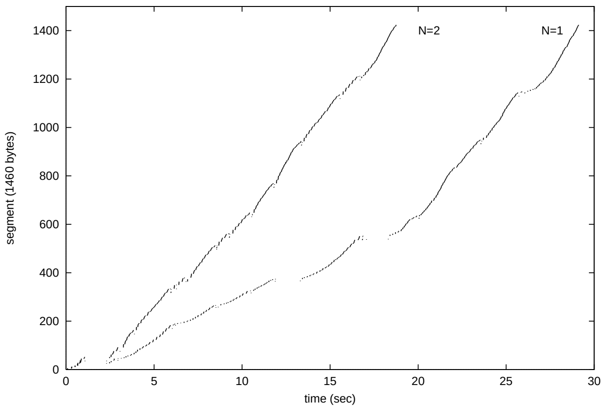

Fig. 5. TCP SACK traces from Internet experiments.

B. TCP Selective Acknowledgements (SACK)

TCP with Selective Acknowledgements (SACK) has become a Proposed Internet Standard Protocol [11]; it overcomes TCP Reno's problems with large windows. SACK requires both sender and receiver modifications; the receiver sends SACK packets to the sender reporting all segments that have arrived successfully and are currently present in its receive queue[4]. The sender implements a selective retransmission strategy based on receiver's SACK information [7].

The same experiments done with TCP Reno were repeated with TCP SACK and the results are shown in Figure 5. The number of timeouts is significantly smaller and MulTCP throughput (intuitively the gradient of the curve) is almost double.

Both Reno and SACK TCP traces shown above were obtained sufficiently close in time (within minutes), so that one can assume that traffic conditions in the Internet path had not changed dramatically. The experiment was repeated several times and although there were quantitative differences across the experiments the qualitative results were the same; MulTCP with SACK was achieving approximately twice the throughput of the standard TCP flow in the same experiment.

Although SACK improves things considerably it is still not a panacea because of the limited number of SACK blocks in each segment. According to RFC 2018 when a retransmitted packet is lost again then even TCP SACK will have to time out. The same happens when many acknowledgements are lost. The congestion control decisions concerning error recovery with Selective Acknowledgements is still an open research issue.

C. Different Receivers

MulTCP has the advantage of requiring sender-only modifications, however a connection's performance is determined to a great extent by the receiver's behaviour. Therefore we experimented with two receivers: a FreeBSD with SACK and a Windows95 one. Our results showed that only the multiplicative factor itself is not sufficient for providing consistent weighted proportional throughput differentiation between different receivers, because there are other aspects of a TCP implementation which affect the performance drastically.

In our experiments each client received two TCP flows: a standard TCP and a MulTCP with weight $N = 5$ (Figure 6). The FreeBSD receiver has a large socket buffer size which allows larger windows but it also means that it has to wait until the socket buffer is filled up to a certain point before sending an ACK to the MulTCP sender (or the delayed ACK timer expires). The Windows'95 TCP implementation on the other hand does not appear to have this problem; initially it performs better but in the long run FreeBSD outperforms it.

V. Router Mechanisms and Network issues

This Section examines the interaction of MulTCP with RED gateways and compares it with the proportional fairness obtained from router mechanisms (WFQ, CBQ) that are expected to be deployed in future IP networks. Although the scheduling and queue management disciplines are conceptually independent from the end-to-end flow control, they affect the performance of TCP flows by providing "implicit" or "explicit" feedback signals. These signals can be either an increase in the Round Trip Time (due to queueing delays that lead to poor TCP throughput) or packet drops (an implicit congestion signal). Experiments were carried out in the controlled environment using FreeBSD routers.

[4] SACK information is carried in OPTIONS field of the TCP header and the 40 bytes available there allow for a maximum of 4 SACK blocks, each block being two 32-bit sequence numbers denoting the Left and Right Edge of Block respectively.

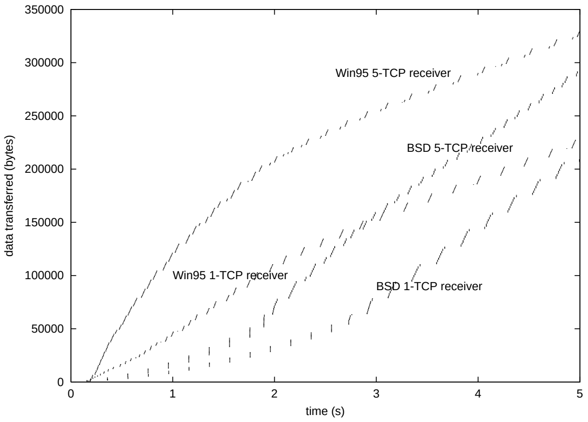

Fig. 6. MulTCP with Win95 and FreeBSD TCP receivers.

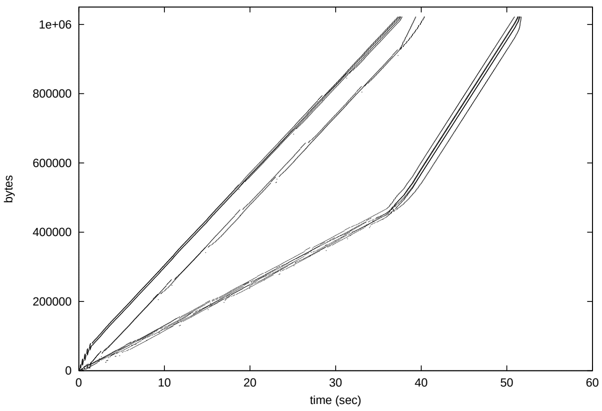

Fig. 7. WFQ: proportional fairness on the bottleneck with per flow scheduling and different weights.

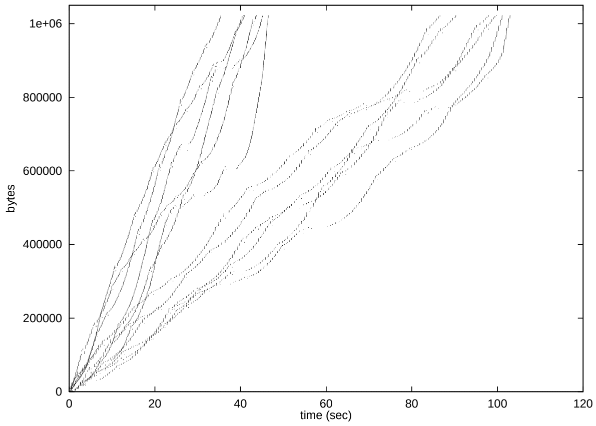

Fig. 8. TCP throughput with two CBQ classes, 66% and 33% and the same number of flows in each.

A. *WFQ and CBQ*

Network mechanisms like Weighted Fair Queueing and Class Based Queueing [12], can be used for providing local proportional fairness (i.e. on per hop basis) by appropriate allocation of weights and/or connection admission control.

Figure 7 shows an ideal situation of weighted proportional throughput differentiation, obtained using a WFQ scheduler with per flow queueing and two weights for providing two types of service to TCP flows. In the test six of the twelve TCP connections were served by WFQ queues assigned a weight of two while the weights of the other six queues were set to one. When the "weight 2" TCPs finished (at around time 37 sec) the "weight 1" TCPs saw an increase in their throughput (knee in the curve) and the output link bandwidth that became available was distributed fairly between them.

With CBQ we introduced two classes; the goal was to ensure that each TCP flow in the privileged class gets twice the throughput of a TCP flow in the other class. This could be achieved by appropriate link sharing (bandwidth assignment between the classes) and connection admission control decisions for the number of flows admitted in each class.

In Figure 8 the bottleneck link bandwidth was proportionally allocated between the two classes and the same number of TCP flows was "admitted" in each class. The proportional differentiation by a factor of two is evident. Alternatively if the link bandwidth is distributed evenly between the two classes, then connection admission control is needed to guarantee that the privileged class is proportionally populated compared to the other one. The results with CBQ look different from the WFQ case because there is no per-flow isolation and there are interactions between the flows aggregated in each queue. Assuming that per flow queueing is expensive, the CBQ with the appropriate link sharing and admission control procedures provides a viable framework for proportional differentiation.

B. *RED with flows from two classes of congestion control*

Random Early Detection (RED) [13] is a queue management mechanism that drops packets with a certain probability based on an exponentially weighted moving average (EWMA) of the queue length and some preconfigured thresholds. Since drop probability does not depend on the instantaneous queue length, small bursts can pass through unharmed and packets will be dropped only during periods of sustained overload.

RED is meaningful in environments where the flows use some sort of end-to-end congestion control based on congestion signals provided by the gateways. We attempted to evaluate the effects of RED when standard TCP and MulTCP were simultaneously active in the network. Our experiments reveal that RED helped considerably towards consistent weight proportional differentiation between the two TCP families.

Figures 9 and 10 show the results with drop tail and RED

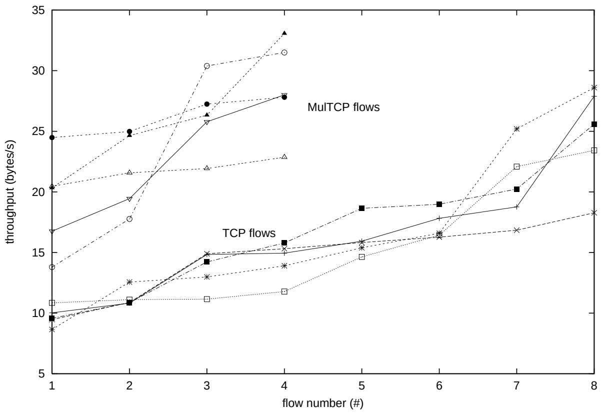

Fig. 9. Throughput for TCP and MulTCP flows with drop tail router.

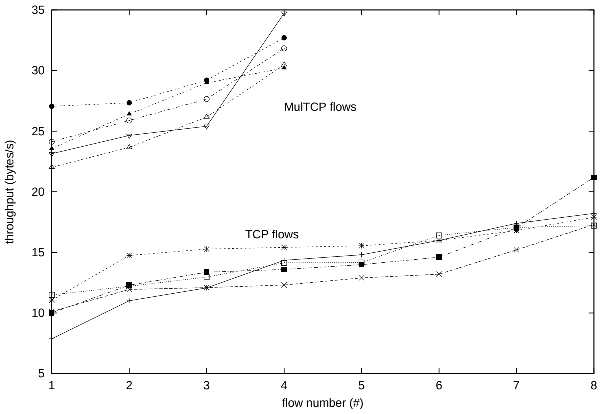

Fig. 10. Throughput for TCP and MulTCP flows with RED routers.

routers. For each one of the 5 tests we plot two lines, one for each family of flows (TCP flows 0-7, MulTCP flows 0-3) . Each line connects the average throughput achieved by the flows in each test (sorted in ascending order).

MulTCP, being by nature more bursty than standard TCP, suffers multiple consecutive packet drops in the same window with conventional drop-tail routers and this reduces its effectiveness. RED gateways on the other hand are not biased against bursty sources. Therefore MulTCP flows usually avoid timeouts because the gateway is inflicting scattered random drops which can be recovered by using either the fast retransmit/fast recovery [14] or the selective acknowledgements mechanism.

The range of values for the MulTCP throughput was better bounded in the RED case. MulTCP flows performed in statistical terms twice as well compared to the standard TCPs (avoiding the phenomena of standard TCPs outperforming MulTCPs as in Figure 9). In this work we did not address in detail quantitative issues of statistical assurance. Table I

TABLE I

THROUGHPUT AVERAGES (KBYTES/S) FOR TCP AND MULTCP FLOWS

	Drop-tail (8+4)	RED (8+4)	RED (6+6)
TCP	15.952	14.278	13.922
MulTCP	23.959	27.259	23.039
Gain	50.1%	90.9%	65.5%

summarises the average aggregate throughput for each family (class) of flows and the gain of the MulTCP class over the standard TCP class for different traffic mixes with RED or drop-tail routers.

Ensuring the appropriate traffic mix, i.e how many flows in each family, is of course very important. It is not reasonable to expect having an arbitrary number of MulTCP flows and a single TCP and still have all the MulTCPs getting twice the throughput. The experiments showed that even with a 50% allocation the gain was not satisfactory; the smaller the fraction of MulTCPs to the total number of flows the closer to the ideal the gain was. In Table I, for one third of the flows being MulTCP (8+4 allocation) the aggregate gain was 90% with RED routers and 50% for drop-tail. This both emphasises RED's importance and leaves significant margins for improvement when the ratio of the MulTCP flows to the total number of flows is lower.

VI. MULTCP PROBLEMS AND LIMITATIONS

MulTCP performance is more sensitive to TCP implementation aspects and to router mechanisms than the standard TCP because it must address the additional goal of proportional fairness. In this section we discuss the factors that mitigate MulTCP's performance.

A. MSS and window sizes

MulTCP's modified slow start is problematic when TCP's *receiver window* advertised from the client is not sufficiently bigger than the *maximum segment size* (MSS) negotiated for the connection. Assuming a connection with 8 Kbytes MSS and 16 KBytes receive window (typical maximum window size), the sender will not be able to send more than two packets without receiving an ACK rendering the modified slow start (three packets for each ACK) practically unusable. The *maximum window size* can also be a problem since there are TCP stacks that do not implement the TCP Window Scaling Option [10] limiting the maximum window size to 64 KBytes. Large MSS reduces MulTCP's effectiveness because the algorithm operates in segments and the window will not be able to grow enough to ensure throughput differentiation.

TABLE II
NUMBER OF BYTES SENT BEFORE ENTERING NORMAL SLOW START

MulTCP factor	Bytes MSS 536	Bytes MSS 1460	Bytes MSS 9140
1.5	2144	5840	36560
2	8576	23360	146240
2.5	19296	52560	329040
3	53600	146000	914000

B. Small Transfers

The nature of MulTCP's modified slow start algorithm does not make it amenable to several differentiation levels for short transfers. MulTCP connections will be exhibiting identical behaviour for a considerable length at startup so that short (web like) transfers will finish long before any differentiation based on weights is possible.

Table II shows how many bytes MulTCP connections with various factors can send before entering the normal slow start phase. The actual number depends on the Maximum Segment Size because *cwnd*, although measured in bytes, is based on MSS for its opening strategy. It can be seen, for example, that there is no differentiation between MulTCPs of weights 2 and above when there are less than 23 Kbytes to be transfered with the most commonly used MSS of 1460 bytes.

Moreover the fixed overhead required by TCP to establish a connection (3-way handshake) cannot be avoided by MulTCP. This is a considerable fraction of the overall transfer time for short connections and diminishes the effectiveness of MulTCP in short transfers.

C. Burstiness

MulTCP is admittedly more bursty than standard TCP because for the same number of returning ACKs it sends more back-to-back packets. Bursts are harmful because they impose excessively high demands for buffer space in the routers' queues that moreover have to be satisfied within very tight time limits and this can lead to packet drops and overall service degradation. The burstier the traffic becomes the higher the packet loss rate on a link for the same level of link utilisation. Another ill effect of burstiness is the increased *jitter*, variance in end-to-end delay times. Jitter affects all the connections that share the same queue in the router and it is important for real-time flows but usually has smaller effect on *elastic* TCP-like traffic. Burstiness is a fact of life and can be dealt with at the sender by pacing out packets with a certain rate.

In the Internet today all traffic flows coexist in the same FIFO queues in the routers, therefore packet drops caused by bursty connections that overflowed the buffer are interpreted as an indication to slow down indiscriminately by *all* sources that happened to lose packets. This assumes that these sources have some notion of flow control but does not necessarily mean that they were the culprits of the congestion incident in the first place.

When standard TCPs share the same bottleneck with MulTCP connections, the first achieve less throughput than in the case where all connections were standard TCPs, but this after all is the idea behind MulTCP with the additional feature that differentiation has to be proportional to a given weight.

Moreover, MulTCP cannot cause congestion collapse; it *does* have congestion control, admittedly more relaxed but still in place, and is able to sufficiently reduce its rate in presence of congestion.

D. MulTCP weight

The simulation results reported in [1] showed that for N between one and two the MulTCP flows get about N times the throughput of standard TCP but this does not hold for values above 2.5. In the Internet experiments however MulTCP/Reno was problematic in general and could not achieve consistent throughput differentiation.

With MulTCP/SACK the simulations showed that throughput can increase proportionally with weights ranging up to 10; again in practice the maximum weight proved to be much lower. However the highest weight allowed for MulTCP/SACK depends on the bandwidth delay product of the end-to-end path; for paths with higher bandwidth delay products the maximum usable weight should be higher. We simply make a conjecture which we intend to investigate but we have no data from different Internet paths to prove this at present.

In all experiments we observed cases of MulTCP connections performing overall better than TCP but the *effective gain* did not correspond to the assigned *weight*. The behaviour was becoming increasingly unpredictable with large number of flows and higher weight values, especially in the Internet case. The inconsistency is obvious in Figure 11, where MulTCP with weight four performs better than MulTCP with weight six.

Clearly differences in `multcp_cwthresh` (in segments) cannot guarantee throughput differentiation because the resulting `cwnd` values (in bytes) are unrealistic even for paths with large bandwidth-delay products. Losses occur usually long before this threshold is reached and this renders it practically unusable, making proportional differentiation impossible for high N values. As the weight N grows MulTCP connections become in fact *equally aggressive* in their congestion window opening strategy (`multcp_cwthresh`) and almost *equally unresponsive to congestion* (`multcp_cwdecr`) regarding the ratio by

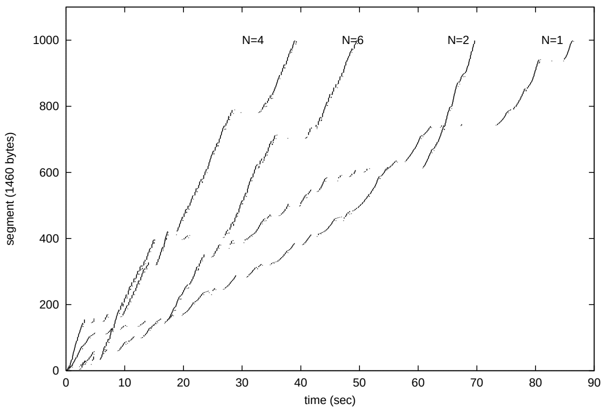

Fig. 11. SACK MulTCP for various Ns.

which the congestion window is decreased as result of loss. Figure 2 shows the values of `multcp_cwthresh` and `multcp_cwdecr` ratio for different weights N.

Exploring the weight range it has been found that N must be less than 4, even with MulTCP/SACK, for consistent differentiation. Fractional values of N in this range can also be used; a 1.5 MulTCP does not have a natural equivalent in terms of number of virtual TCPs but it implies a TCP with 50% increase in throughput compared with standard TCP.

E. *RTT bias*

When several TCP connections with various RTTs share the same bottleneck link then severe unfairness can result from the fact that the additive increase factor is the same for connections of the same weight N (linear increase phase in Figure 1) regardless of their RTT. This happens because connections with the same weight increase their window by the same number of segments (i.e one segment for standard TCP) per RTT providing a faster increase of the sending rate when the RTT is shorter. This is a well known problem inherent to TCP window management [4]. Moreover the RTT bias can influence the effective gain of a MulTCP over another with smaller weight and shorter RTT.

VII. CONTEXT OF DEPLOYMENT

MulTCP can be effective in a number of emerging popular environments. MulTCP is easy to deploy because it requires sender-only modifications (although the receiver TCP stack, with delayed ACKs for example, can influence performance) and all the clients need is a contract with the network provider. SACK proved to be essential and this needs both sender and receiver modifications. However SACK is already an Internet Proposed Standard and Microsoft Windows98, probably the most widely TCP in end hosts today, has the SACK option turned on by default.

A. *Web caches*

An environment where MulTCP can have an economic importance is that of the web caches. When a cache receives an HTTP request from a client, it sends the page back immediately if the page is available locally, otherwise the cache requests the page from another cache higher up in the hierarchy or directly from the server itself.

The cache server will be able to detect from the client's IP address a certain *user profile* i.e. whether the user is eligible for MulTCP or standard TCP service and which weight to be used (if several available). In this way the clients only need a contract with their ISP, and the ISP guarantees that statistically they get "faster downloads". In case of a cache miss, the proxy opens a new connection to a parent cache and the new connection must be able to convey the user profile semantics of the original client. This can be done either by introducing a new TCP option that negotiates the MulTCP weight to be used or by having the parent cache listening on known "special" ports that correspond to certain service levels.

Web caches could generally benefit from MulTCP and leverage their service. Moreover usually they are not far from their users which makes MulTCP behaviour more predictable and ISPs able to guarantee more accurately the TCP service levels they provide to their clients. A potential drawback is that Web traffic involves short-lived flows although this is expected to change with the deployment of HTTP 1.1 with persistent connections.

Having MulTCP-style capabilities on every Internet host raises concerns about global Internet stability due to the increased number of more aggressive and less congestion responsive flows. The counter-argument comes from UDP which is available to all end hosts and can be much more harmful than MulTCP in the hands of a malicious user, moreover it is only limited by the bandwidth of the access line. Nevertheless it makes more sense to keep mechanisms like MulTCP in the servers where they can be centrally administered; this restricts scalable transfer quality to the download path only, but usually that is what most users care about.

B. *Satellite links and other proxies*

Other possible environments include satellite links that usually have various types of proxies operating at their ends. These links with the large *bandwidth delay product* are particularly suitable for the differentiation based on larger windows.

Generally *split connection proxies* that are used as security firewalls, encryption servers, mobile proxies for addressing network heterogeneity, are places that are expected to gain in importance in future IP networks and are also key points for dealing with quality of service and deploying transport level differentiation. From the economic viewpoint, providers could be offering different levels of TCP service in exactly the same fashion they provide different access speeds.

VIII. Discussion and Conclusions

Internet traffic is dominated by web transfers that originate from large servers that are few compared to the Internet host population. This makes transport level approaches to differentiated services a rather appealing concept; they are easier to deploy, manage and they make minimal assumptions about the underlying network infrastructure. However in order to be successful, like any other differentiated services scheme, they must sufficiently address issues of spatial granularity (i.e to which parts of the network) and levels of service assurance.

Providing deterministic QoS guarantees (intserv) has proved quite complex. The most recent approach (diffserv) is to push complexity to the edges and focus on statistical service assurance to flow aggregates that have previously been policed at the edges. A transport level approach to service differentiation applies the differentiation mechanism on key nodes (hot spots) inside the network. This is enabled by recent TCP enhancements that significantly improve performance like SACK. This approach may also prove strategic taking in mind the growing trend in using "agents", proxies and other active elements inside the network for various services.

The motivation behind differentiated services is no doubt differential pricing; its deployment is still an unresolved issue basically because it is unclear who is going to make profit out of the Internet services in order to be charged accordingly. A simple solution is to provide relative quality so that regardless of the amount of Internet traffic the user who pays more should get more, without absolute guarantees for a specified throughput. However the issue of "better service" with respect to what, does not have a straightforward answer and there are no indications that there is sufficient client demand for this kind of service (i.e. just better).

Mechanisms like WFQ and CBQ can provide proportional fairness on a link basis (per hop behaviours, PHBs) or inside a domain but service level agreements (SLAs) across multiple domains is yet another thorny issue. However these mechanisms are fairly static in nature and the *weight* of each queue in the router will have to be dynamically adjusted depending the number of flows currently active in each class. Alternatively dynamic Connection Admission Control (CAC) could be used with fixed class allocations to control the number of flows in each class.

Our work focused on an end-to-end approach. We modified the congestion control algorithm of TCP and tested it both in a controlled environment and over real Internet path, with different TCP flavours. Under real network tests MulTCP proved to be less resilient than in the simulations [1]. Our experiments showed that MulTCP is a *soft* differentiation mechanism in the sense that it provides statistical assurance of proportional throughput differentiation across an ensemble of TCP connections that share the same bottleneck and have similar round trip times. It fits nicely in certain popular environments and provides statistically assured service without requiring support from network level mechanisms. Thus it should be a useful mechanism to deploy when coupled with differential pricing.

Acknowledgements

This work has been motivated and largely benefited from discussions with Jon Crowcroft. It has been partially supported by Centro Studi E Laboratori di Telecomunicazione (CSELT), Torino (Italy). We would also like to thank George Uhl from the NASA GSFC and Luigi Rizzo from the University of Pisa for their help and cooperation.

References

[1] Jon Crowcroft and Philippe Oechslin, "Differentiated End to End Internet Services using a Weighted Proportional Fair Sharing TCP," *ACM Computer Communication Review*, vol. 28, no. 3, July 1998.

[2] Van Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[3] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, 1989.

[4] Sally Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 1–One-way Traffic," *ACM Computer Communication Review*, vol. 21, no. 5, Oct. 1991.

[5] Frank Kelly, Aman Maulloo, and David Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," 1997, http://www.statslab.cam.ac.uk/frank/rate.html.

[6] "CAIRN: Collaborative Advanced Interagency Research Network," http://www.cairn.net/.

[7] Luigi Rizzo, "Issues in the implementation of selective acknowledgements for TCP," available at http://www.iet.unipi.it/ luigi/sack.diffs, 1996.

[8] "ALTQ: Alternate Queueing for FreeBSD," http://www.csl.sony.co.jp/person/kjc/software.html.

[9] Panos Gevros and Jon Crowcroft, "Experimental Results on Weighted Proportional TCP Throughput Differentiation," in *Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH '98)*, University College London, June 1998.

[10] D. Borman, B. Braden, and V. Jacobson, "TCP extensions for high performance," Request for Comments (Proposed Standard) 1323, Internet Engineering Task Force, May 1992.

[11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," Request for Comments (Proposed Standard) 2018, Internet Engineering Task Force, Oct. 1996.

[12] Sally Floyd and Van Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, Aug. 1995.

[13] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[14] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and fast recovery algorithms," Request for Comments (Proposed Standard) 2001, Internet Engineering Task Force, Jan. 1997.