TCP mechanisms for Diff-Serv Architecture

Wenjia Fang
wfang@cs.princeton.edu

July 1999
Princeton University Computer Science Dept.
Technical Report TR-605-99

Abstract

Our early work [4] has shown that by using a combination of mechanisms in a Diff-Serv domain: tagging algorithms in boundary routers and a RIO algorithm in interior routers, we could create differentiations in throughput among different TCP connections during periods of network congestion. However, the effectiveness of such schemes is limited by the impreciseness and biases in the window-based congestion control algorithm of TCP. More precisely, TCP's window open-up algorithm has an intrinsic bias against long rtt connections, and TCP's window close down mechanism adapts to the perceived network congestion optimal point only, which is not sufficient to meet the underlying premise of Diff-Serv architecture.

In this paper, we propose a few mechanisms to TCP's congestion control algorithm, specifically tailored to the Diff-Serv architecture. While preserving TCP's "linear increase and multiplicative decrease" principle, these mechanisms make TCP more robust and precise in adjusting its sending rate to network congestion as well as to a predefined service profile. Simulations are used to qualitatively demonstrate the results. In addition, we discuss deployment issues.

The changes require only modifications to a TCP sender, and a TCP with these changes is inter-operable with any existing TCP implementations. The proposed mechanisms are sufficiently general to be applicable to any window-based congestion control algorithms, such as [**?**].

1 Introduction

In the general realm of providing Quality of Services (QoS) to a large variety of applications on the Internet, we have seen research efforts like Integrated Services, or Int-Serv[5] and Differentiated Services, or Diff-Serv[2]. In terms of services defined, Int-Serv aims to provide end-to-end guaranteed or controlled load service on a per flow basis, whereas Diff-Serv is an architecture to provide coarser level of service differentiation to a small number of traffic classes. In terms of implementation, Int-Serv requires each router to process *per flow* signaling messages and maintain ... rior routers. The edge routers classify packets into flows, and apply some traffic conditioning mechanisms on packets – including metering, tagging, shaping and policing. The packets are tagged with some patterns in the DS field [17], which indicate to the interior routers which PHBs should be applied to the packets. The interior routers do not need to classify packets but treat packets as an aggregate, and apply corresponding PHBs. The service provided to a particular packet stream is a combination of the traffic conditioning at edge routers (both ingress and egress routers) and a series of PHBs in interior routers. With different PHBs and different traffic conditioning mechanisms, Diff-Serv can provide differentiated services to traffic. By pushing the complexity to the edge and maintains a simple core, Diff-Serv is much more scalable than Int-Serv.

There are currently two PHBs defined in the Diff-Serv architecture: the premium service [] and the assured service []. The premium service provides the equivalent of a dedicated link of fixed bandwidth between two edge nodes. The assured service shares its root with the "best effort" service model of the current Internet and provides only certain level of assurance, or expectations to applications [3]. We limit the following discussions to mechanisms applicable to the "assured service" PHB only.

Instrumental to the Diff-Serv architecture is Service Level Agreement (SLA) – a contract between a customer and an ISP that specifies the forwarding service a customer should receive. The technical part of SLA specifies classifier rules and any corresponding traffic profiles and metering, tagging, shaping and dropping rules to be applied to the traffic streams selected by the classifier. Thus, an ISP is required to provision its network to meet the agreed service requirements in SLA.

Diff-Serv architecture changes the premise underlying the "best-effort" Internet service model. In the "best-effort" service model, the allocation of resources is based on equal and fair [1] sharing of available resources among all participating entities. For example, both RED[9] or FRED[16] queuing algorithms have the goal of local fairness in their design. In contrast, in the Diff-Serv architecture, resource allocation is based on some pre-defined policies embodied in the SLAs. This change of premise re-

among two types of packets in times of congestion, though the high level service profiles embodying policies can be of a wide range of throughputs.

In [4], we've shown that mechanisms can be deployed within a Diff-Serv domain to create differentiations among TCP connections: tagging algorithms (TSW) can be deployed at edge routers for each entity with a service profile and RIO algorithm can be deployed for each interior router to create discriminations between IN and OUT packets during congestion. Using simulations, we've shown that such mechanisms can create differentiations among a wide range of TCP connections. However, the effectiveness of such schemes is limited by the impreciseness and biases in window-based congestion control algorithms of TCP. More specifically, the rate adjustment scheme in the current Internet depends on a feedback loop completed by both TCP's congestion control algorithm and the router's congestion signals, thus, by changing mechanisms in routers alone, the rate adjustment schemes are not very effective or precise in achieving the targeted service profiles.

In this paper, we study a few mechanisms which, when applied to current TCP's congestion control algorithm, can significantly improve TCP's performance in meeting the requirement of a service profile. We believe the "linear-increase and multiplicative-decrease" principle in the current TCP is a sound one and do not hope to change it. In fact, we don't introduce any additional state variables to TCP's machinery, but merely make the observations that existing variables like *cwnd* and *ssthresh* can be used more effectively in meeting the requirements of service profiles. Thus, our proposed mechanisms can be used to any congestion control algorithms observing similar principles of that of TCP, such as "Congestion Manager" [**?**].

The rest of the paper is organized as follows. Section 2 briefly describes mechanisms to be deployed in routers within a Diff-Serv domain: RIO algorithms and tagging algorithms. Section 3 describes the proposed mechanisms to TCP congestion control algorithms and intuitions behind those mechanisms. Section 4 presents simulation results to demonstrate the effectiveness of such mechanisms qualitatively. Section 5 offers a discussion on how those mechanisms interact with Diff-Serv mechanisms deployed in routers. Section 6 offers a road-map for deployment and discusses some complications we might encounter. Section 7 discusses related work, future work and concludes.

2 Mechanisms deployed in a Diff-Serv domain

2.1 RIO algorithm to create differentiation in routers

RIO algorithm is based on RED (Random Early Drop) algorithm, and is created with two sets of parameters for two types of packets: IN packets and OUT packets. The two sets of parameters are denoted as $(min\_in, max\_in, P_{max\_in})$ and $(min\_out, max\_out, P_{max\_out})$. min_in and max_in are the low and high thresholds for IN packets, and P_{max_in} is the maximum probability with which to drop an IN packet. Similarly, min_out and max_out are the low and high thresholds for OUT packets, and P_{max_in} is the maximum probability with which to drop an OUT packet. The algorithm works as follows: when a packet arrives, RIO algorithm estimates two variables, avg_in_q, average IN packet queue and avg_q, average **total** queue, respectively. An arriving IN packet will contribute to the estimation of avg_in_q, as well as avg_q; an arriving OUT packet will only contribute to the estimation of avg_q. A dropping probability is calculated for each arriving packet depending on the current value of avg_in_q or avg_q. In the case of an IN packet, a dropping probability is calculated as $p = P_{max\_in} * (avg\_in\_q - min\_in)/(max\_in - min\_in)$. The intuition here is that an IN packet represents the traffic that is to receive priority, therefore, whether it should be enqueued is dependent on the amount of IN packets the gateway received recently, and not affected by the OUT packets or the total number of packets (both IN and OUT). In the case of an OUT packet, a dropping probability is calculated as $p = P_{max\_out} * (avg\_q - min\_out)/(max\_out - min\_out)$. Since an OUT packet represents the lower priority traffic, it should yield to IN packets in terms of queuing, therefore, its dropping probability depends not only on other OUT packets in the queue but also on the number of IN packets in the queue, therefore, we use avg_q, the average total queue, to calculate the probability for dropping an OUT packet.

Graphically, RIO algorithm can be demonstrated in figure 1, RIO algorithm divides up the gateway's congestion state in four phases, depending on the average queue length[2]

- Congestion free phase (phase 1)

[2]The X-axis is the number of packets of avg_q, the estimate of average queue length. We also make the max_in and min_out coincides for the sake of explanation. In actual configuration, the two doesn't have to.

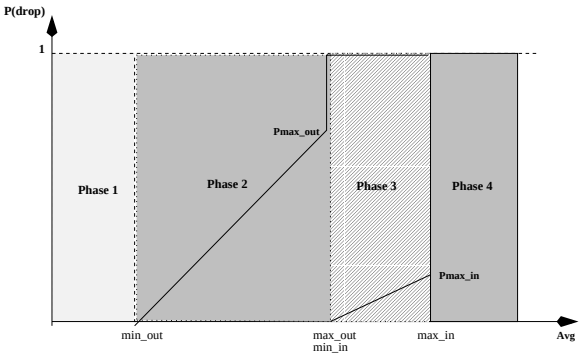

Figure 1: RIO algorithm

In this phase, the gateway is operating well: the amount of IN and OUT packets are well below its capacity. It sees very short instantaneous queue and very small average queue value. No packet is dropped.

- Congestion sensitive phase (phase 2)

  In this phase, the gateway suspects that the queue might be built up so it starts to drop packets as congestion signals, however, it drops OUT packets only. During this phase, the IN packets only see relatively long instantaneous queue and they are never dropped.

- Congestion alarm phase (phase 3)

  In this phase, all OUT packets are dropped, in addition, the gateway starts to drop IN packets as a means to keep the queue length reasonable. This is an undesirable phase for ISP because it compromises the ISP's SLAs by dropping IN packets.

- Congestion control phase (phase 4)

  In this phase, the system is congested. The gateway drops both IN and OUT packets with probability 1. In this phase, the gateway has switched its primary goal from creating differentiations among two types of packets to congestion control. The gateway degrades into a drop-tail gateway, which has other undesirable consequences, e.g., dropping multiple packets from the same TCP stream and global synchronization, etc. If the gateway constantly operating in this phase, it is a sure sign that either the system is well under-provisioned or the parameters of traffic conditioners/RIO are not set correctly.

Phase 2 is the ideal operating phase for a router because in this case, both instantaneous and average queue is short but the link is also highly utilized, the only dropped packets are OUT packets, which doesn't compromise the ISP's SLAs. When operating in phase 1, the router sees little congestion but the link capacity is not well utilized. When the input traffic is predictable, ISP should try to configure their system to avoid phases 3 and 4, and operate most in phases 1 and 2.

2.2 Tagging schemes

In Diff-Serv architecture, traffic conditioners can be modeled as logical entities sitting on the forwarding path of an edge router. In an edge router, packets are first classified, and then feed through the corresponding traffic conditioners, which can choose to 1) passively monitor packet streams and tag packets, or 2) actively buffer and shape packet streams to obtain certain traffic properties before the packet streams enter the downstream Diff-Serv domain. We consider the simpler case of tagging packets. [4] proposed a tagging algorithm – Time Sliding Window (TSW) – specifically tailored to TCP. A TSW tagger incorporates a probabilistic function which can reduce the likelihood of tagging consecutive packets within a window of packets, thus, reduce the chances of multiple packet drops within a window. This will keep TCP to operate in the "congestion avoidance" phase, thus make the rate adjustment scheme more controllable. Our simulation shows that when TCP itself incorporates mechanisms suited to Diff-Serv architecture as proposed in the following section, the rate adjustment scheme is less dependent on the intricacies of tagging algorithms. In our simulation, we could use a simple Token Bucket tagging scheme with a configured target rate for each TCP connection. See Fang99 for more discussion on tagging schemes.

3 Proposed Mechanisms to TCP's Congestion Control Algorithm

3.1 TCP's congestion control mechanism

In today's Internet, transport layer protocol TCP implements certain congestion control and avoidance mechanisms which interpret to drops as congestion signals. The mechanisms are based on [13], and have incorporated much refinements since then [6, 12].

There are two phases in TCP's window adjustment algorithm: exponential increase phase and linear increase phase, corresponding to the "slow start" and the "congestion avoidance" phases [13], respectively. TCP keeps

two variables for its congestion control algorithm: congestion control window or *cwnd* and slow-start threshold, or *ssthresh*. During the exponential increase phase, TCP sender starts with a congestion window of one packet and exponentially increase the congestion window, *cwnd*. When the congestion window hits a threshold, *ssthresh*, the sender switches into the "congestion avoidance" phase, and increases the congestion window linearly, probing the network capacity as it becomes available. TCP continues in the "congestion avoidance" phase until it receives a congestion signal – typically a packet drop or an Explicit Congestion Notification (ECN) in an acknowledgment – at which point, the sender evokes a mechanism called "Fast Retransmit and Fast Recovery" to recover the lost packet. Additionally, TCP sets its *ssthresh* to be one half of the congestion window prior to the packet loss, and resets its *cwnd* to be the same as the new *ssthresh*.

In this scheme, *cwnd* indicates the amount of packets currently outstanding, and the instantaneous sending rate of TCP can be approximated as $cwnd/rtt$, where rtt is the round trip time including queuing delays. The choice of threshold reflects an estimation of the equilibrium operating point, i.e., a packet leaves the network as a sender puts a packet into the network, is key to the performance of the algorithm. The algorithm in adjusting *cwnd* reflects the additive increase and multiplicative decrease rule which will alleviate congestion and maintain stability in a system [13, 14].

3.2 Proposed Mechanisms

3.2.1 Fair window open-up algorithms

In both "slow start" and "congestion avoidance" phases, TCP opens up its window each round trip time (rtt). In the "slow start" phase, TCP doubles its window size each rtt, and during the "congestion avoidance" phase, TCP increases its window by one packet each rtt. When the network system is operating in a stable state that TCP mostly operates in the "congestion avoidance" phase with occasional packet drops. Let r_i denotes node i's average round trip time including queuing delays. In the congestion avoidance phase of TCP, node i's window is increased by roughly one packet every r_i seconds. Thus, node i's throughput is increased by $1/r_i$ packets/sec every r_i second, or by $1/(r_i)^2$ packets/sec every second. Therefore, if a packet is dropped each for two connections with different rtts, it would take the long-rtt connection a significantly longer time to recover to its previous throughput.

TCP's bias towards long rtt connections has been known and studied in [7]. TCP adopts the current window open up algorithm for its simplicity in algorithm and implementation. As discussed elaborately in [7], such increase-by-one window algorithm doesn't meet either of the three fairness measures: min-max fairness [10], the fairness index proposed in [14], and the product measure, a variant of network power [15]. Two alternative window open-up algorithms, both fall in the categories of linear window open-up algorithm, will meet the criteria of fairness. In the first alternative, TCP will increases $c * rtt$ packets per round trip time. Using this scheme, a connection which goes through k bottleneck gateways will share $1/k$ of a bottleneck link bandwidth as a connection which goes through one bottleneck gateway. This will meet the criteria of fairness index proposed in [14] when the resource allocation is defined as throughput times the number of gateways. In the second alternative, TCP will increase $c * rtt^2$ packets per round trip time. Using this scheme, when n connections are sharing a single bottleneck gateway, the window open-up algorithm will allow all connections each share $1/n$ of the bottleneck bandwidth, regardless of their rtt. This will maximize the fairness index when the resource allocation is defined as throughput of individual connections.

In the Diff-Serv architecture, where each entity (potentially at the finest granularity of a single TCP connection) is associated with a "service profile" in which a target throughput is defined. Though the "service profile" definition has not been finalized by the IETF Diff-Serv working group, there are two potential definitions to choose from, which meet the criteria of fairness index. First one, a "service profile" includes both a target throughput as well as a range of rtts within which the target throughput can be met. The longer the rtt, the smaller the corresponding target throughput. This definition of "service profile" is an interpretation of the fairness index if the underlying TCP window open-up algorithm is chosen to increase $c * rtt$ each round trip time. Alternatively, a "service profile" includes simply a target throughput, which implies that the ISP is to assure the target throughput regardless of the rtts of the connection. This definition of "service profile" is an interpretation of the fairness criteria if the underlying TCP window open-up algorithm is chosen to increase window linearly $c * rtt^2$ each round trip time. The related service profile definitions and their corresponding window open-up policy, fairness criterias are tabulated in Table 1

As we described above, the current TCP window open-up algorithm fails to meet either of the above definition of fairness. This helps to explain why in our early work,

Table 1: Service Profiles and the corresponding TCP mechanisms

	Strategy 1	Strategy 2
Service profile	$[BW_{target}, (min_{rtt}, max_{rtt})]$	BW_{target}
Window Algo.	$c * rtt$	$c * rtt^2$
Fairness Critera	Throughput X # of routers	Throughput

the long rtt connections cannot meet the targeted service profile throughputs when using the current TCP implementations. In other words, the RIO gateways with preferential dropping cannot compensate for the innate bias in TCP when a long-rtt connection fails to open up quickly enough after a packet loss. In contrast, the two proposed alternatives can alleviate this bias somewhat. However, even with this proposed change, the current TCP can still be biased against long-rtt connections during the exponential window increase phase, in which, the TCP doubles its congestion window each rtt. A small-rtt connection will open up its window quicker than a long-rtt connection. This begs the question why we don't propose changes to TCP's window algorithm all the way, including exponential increase phase to be absolutely fair. There are two reasons. First, the "Slow-start" phase of TCP is relatively short because after each rtt, TCP doubles its window, so TCP can usually recover to its newly adjusted $ssthresh$ in the "Slow-start" phase quickly. From a performance perspective, the degradation in sending rate really results from the timeout period which typically proceeds the "Slow-start" phase, and not from the "Slow-start" phase itself, thus, the benefits resulted from making "Slow-start" phase fair is small. Second, as we shall discuss in section 5.1 and 6.2, the choice of c is a difficult one for policy and deployment reasons so we recommend limiting any changes to the "congestion control" phase.

It should be noted that both alternatives to the current window algorithm of TCP still fall under the "linear increase" rule. Only that the "linear increase" is by c packet per rtt (the first alternative), or by c packets per second (the second alternative).

3.2.2 Setting *ssthresh* for TCP

As discussed in section 3.1, in the current TCP window algorithm, the value of *ssthresh* reflects the perceived network available bandwidth to a TCP. *ssthresh* is initially set to a default value and is readjusted after each packet drop to be one half of the *cwnd* before the packet drop. A packet drop is recovered either through a mechanism called "fast recovery and fast retransmit" or through a timeout mechanism. When a single packet is lost, the "fast recovery and fast retransmit" mechanism can recover the lost packet successfully and both *cwnd* and *ssthresh* are reduced to one half prior to the packet drop, then TCP continues to operate in the linear window increase phase with a reduced *ssthresh*. When multiple packets are dropped within a window, current implementations of TCP usually fail to recover all lost packets because the sender won't be able to put enough packets into the network to generate sufficient duplicated acknowledgments to detect additional packet loss. Upon each detected successive packet loss, TCP reduces its *ssthresh* by one half so when eventually, TCP recovers from packet loss via a timeout mechanism, TCP operates with a much reduced *ssthresh* than that before the packet losses.

In the Diff-Serv architecture, bandwidth allocation is based on service profiles. The underlying premise is that each entity is assured of its target throughput specified in its "service profile" when congestion is experienced and can exceed such profiles when there is no congestion. With the knowledge of those target throughputs, the ISP is supposed to provision the network well so that all service profiles are satisfied. However, since Diff-Serv relies on statistical multiplexing of shared resources and not strict admission control (see section 5.2 on discussions of coarse granularity admission control at the edge of the domain.), there will be cases when either the ISP fails to provision properly or certain routes will experience incipient congestion. In the Diff-Serv domain, when a TCP connection loses a packet, how should *ssthresh* and *cwnd* be set? The underlying Diff-Serv premise implies that the ideal behavior of TCP is to reduce its sending rate when congestion is experienced – the proper mechanism to evoke in a shared environment –, but can recover to its target throughput robustly.

We propose the following change to reflect the change in underlying premise from all purely "best-effort" service model to a Diff-Serv model. We set the initial value of *ssthresh* to be the minimum of the default value and the byte equivalent of target rate as defined in its "service pro-

file". This also "gauges" the operating point of the TCP. Additionally, we propose that TCP sets its *ssthresh* to be byte equivalent of the target throughput, when congestion is detected. TCP reduces *cwnd* to be one half of the previous value before the packet drop, as it would in the current implementations. This has the effect of reducing instantaneous sending rate of TCP connections to alleviate temporary congestion, but allows each TCP connection to quickly throttle back to its target operating point.

It should be noted that the argument for modifying TCP's *ssthresh* value is different from that for modifying TCP's congestion window open-up algorithm. The argument for modifying TCP's congestion window open-up algorithm is essentially a "fairness" argument but only completed in the light of Diff-Serv architecture. As discussed in [7], the current window open-up algorithm is not fair by any particular fair index. Technical merits of two proposed mechanisms have been discussed, but in absence of a policy decision on the desired fairness goal in current networks, there is no reason to deploy them. In other words, the proposed mechanisms are *necessary* but not *sufficient* conditions for deployment. With the introduction of the Diff-Serv architecture, the policies can be implicitly expressed in the service profiles. Thus, depending on how the policies are defined, the appropriate and fair TCP linear-increase window open-up algorithm can be deployed.

In contrast, the argument for setting the *ssthresh* for TCP's window both during the initial phase and the congestion control phase is entirely a policy argument, and dependent on the Diff-Serv architecture.

3.2.3 ECN-enabled TCP in Diff-Serv Domain

Some recent proposed changes to TCP include the use of Explicit Congestion Notification (ECN) mechanisms in both TCP and the RED gateway [8]. In this proposed scheme, RED routers will mark an ECN bit in a packet's header in stead of dropping the packet, and TCP will respond to the explicit congestion notifications in stead of inferring congestion from duplicated acknowledgments. This mechanism has the advantage in avoiding unnecessary packet drops and unnecessary delay for packets from low-bandwidth delay-sensitive TCP connections. A second advantage of ECN mechanisms is that TCP doesn't have to rely on coarse granularity of its clock to retransmit and recover packet losses.

Similar mechanisms could be deployed in the Diff-Serv architecture. In stead of dropping packets, the RIO gateway can also take advantage of the ECN mechanism by marking them. RIO gateways can apply its preferential algorithm in which, it marks an OUT packet as experiencing congestion with a much higher probability than an IN packet [4]. The ECN bit will be copied by the transport-layer receiver and relayed back to the sender. TCP sender has to be able to recognize the two types of packets (IN and OUT), and respond to ECN bits in them differently.

When an OUT packet arrives back to the TCP sender with the ECN bit marked, it indicates that the RIO gateway operates in the "congestion sensitive" phase (Phase 2 in section 2.1). It indicates that the gateway senses the congestion as mostly incipient with only long instantaneous queues. When a RIO gateway deploys ECN mechanism as the only mechanism for notifying the transport-layer protocols to retract its congestion window, the window reduction should be no more aggressive the recommended guidelines for ECN mechanisms [8]. We recommend that TCP reduces its *cwnd* to be one half of the current *cwnd* value, and resets *ssthresh* to be the byte equivalent of the target throughput. Depending on the value of *cwnd* and *ssthresh* prior to receiving the ECN signal, TCP can be operating in either linear increase mode or exponential increase mode again. In either case, the reduction in window size will induce a temporary reduction in TCP's sending rate to alleviate congestion but still keep TCP operating close in the targeted operating point.

When an IN packet arrives back to the TCP sender with the ECN bit marked, it indicates that the RIO gateway operates in the "congestion control" phase (phase 4 in section 2.1), in which, the gateway has seen persistent long queues and is forced to mark both IN and OUT packets with probability 1. When such packet is received, the TCP sender should react to the congestion signal more drastically. We recommend that TCP reduces its reduce its *cwnd* to be one packet, and reset *ssthresh* to be the byte equivalent of the target throughput. This is the same window reaction as in the current implementation when a packet has been dropped but TCP starts in its "Slow-Start" phase with a configured *ssthresh*. This will cause a more drastic reduction in TCP's sending rate. But since the new *ssthresh* will be greater than the new *cwnd*, TCP will recover to the target operating point using exponential increase window increase algorithm quickly. Table 7 summarizes the recommended guidelines for ECN-enabled TCP responding to IN and OUT packets.

4 Simulation Results

In this section, we will use simulation to compare the impact of different mechanisms when they are deployed in-

dependently and incrementally.

4.1 Simulation setup

We use network simulator *ns* [1] to implement simulations. We use a simple topology (Figure 2) to evaluate bulk-data transfers. We use six ftp transfers with two sets of rtts: 80ms and 30ms. Each simulation run has four different phases. The first phase is the start-up phase in which all six ftp/TCP connections reach their respective operating points. The second phase is a congested phase, in which, a constant bit rate (CBR) connection starts, running at $1/4$ of the bottleneck bandwidth. This will cause heavy congestion in the router and TCP connections will back off during this phase. The third phase is the recovery phase, in which the CBR source stops and all ftp/TCP connections will recover to their respective operating points. The fourth phase is the "over-provisioned" case, during which, one of the ftp/TCP connection (TCP1) stops sending, and the available bandwidth is shared among the rest five ftp/TCP sources. Each individual phase lasts for 25 seconds. All packet size is set to 1000 bytes. We use TCP-reno implementations and TCP's receiver's window is set to be large enough to not be a constrain on its congestion window.

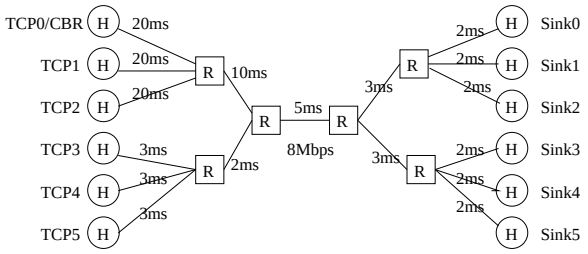

Figure 2: Simulation Topology

The parameters for RED gateways and RIO gateways are set comparably [3]. The bottleneck speed is 8Mbps. The low threshold (min_th) for RED gateway is the byte equivalent of 5ms of queue delay, and the high threshold (max_th) is the byte equivalent of 10ms of queuing delay [4] and the dropping probability P_max is 0.1. The comparable parameters for RIO are (5, 10, 0.5) for OUT packets, and (10, 20, 0.02) for IN packets. To save space, we use tables to represent the time average throughput of three representative connections during different phases. Each setup is run three times with a different random seed, and the data

[3] See Fang99 for a discussion on setting RIO parameters.

[4] In simulations, we translate this in terms of the number of packets queued.

Table 2: Configurations of TCP connections

	RTT (ms)	R_t (Mbps)
TCP0	80	2
TCP1	80	2
TCP2	80	1
TCP3	30	1
TCP4	30	0.6
TCP5	30	0.6
CBR	80	2

presented in the tables are averages of the three runs. For each scenario, we will show the throughput of 1) a long-rtt ftp/TCP (with and w/o a target throughput of 2Mbps); 2) a short-rtt ftp/TCP (with and w/o a target throughput of 0.6Mbps); 3) a CBR connection with sending rate at 2Mbps during phase 2. The constant c in TCP's window open-up algorithm is chosen to be 100, which is equivalent of increase one packet each 100ms.

The total allocated throughput is 7.2Mbps, or 90% of the bottleneck link. The details of simulation set up are listed in Table 2.

4.2 The impact of Diff-Serv mechanisms at routers and endhosts

We separate mechanisms into two groups: Diff-Serv mechanisms to be applied in the endhosts (combinations of all mechanisms proposed in section 3) and Diff-Serv mechanisms to be applied in the router (RIO algorithm and Tagging schemes in Section 2). We consider four different scenarios: 1) standard TCP-reno algorithm with RED gateways; 2) Diff-Serv enhanced TCP with RED gateways; 3) standard TCP with RIO gateways; and 4) Diff-Serv enhanced TCP with RIO gateways. Table 3 lists the results from four different scenarios.

Scenario 1 is our basis for comparison, representing the current "Best-effort" model of allocating resources. It illustrates two well-known behaviors: 1) short-RTT TCP connections have advantage over long-RTT connections when sharing the same bottleneck (first body row vs. second body row); 2) A non-congestion controlled source will create detrimental effect to TCP connections (second body column). In this case, the CBR source gets almost all its packets through a RED gateway at the expenses of other TCP connections' throughput.

Table 3: Comparison of Diff-Serv mechanisms applied to routers and endhost TCP;
Modified TCP = Standard TCP + WinAdj + Ssthresh + ECN

		Start-Up phase	Congested Phase	Recovery Phase	Over-provision Phase
Standard TCP+RED (1)	TCP0 (80ms)	0.676768	0.491638	0.723149	0.832894
	TCP3 (30ms)	1.622382	1.126404	1.585279	1.804911
	CBR		1.978168		
Modified TCP+RED (2)	TCP0 (80ms R_t=2Mbps)	1.86133	1.31369	1.81553	2.30319
	TCP3 (30ms R_t=1Mbps)	1.11268	0.84987	1.12360	1.42987
	CBR		1.92003		
Standard TCP+RIO +Tagging (3)	TCP0 (80ms R_t=2Mbps)	1.43707	1.32511	1.40382	1.49129
	TCP3 (30ms R_t=1Mbps)	1.05836	0.90249	1.11443	1.37187
	CBR		1.78891		
Modified TC + RIO +Tagging (4)	TCP0 (80ms R_t=2Mbps)	2.02678	1.89689	2.02658	2.36111
	TCP3 (30ms R_t=1Mbps)	1.04109	0.91049	1.04853	1.33992
	CBR		1.00350		

Scenario 2 illustrates the effect of Diff-Serv mechanisms incorporated into TCP. With configured knowledge of target throughputs, TCP could robustly recover to its target rate after packet losses. The proposed window open-up algorithm also corrects the bias against long-RTT connections. However, in the presence of a non-congestion controlled source, all TCP sources will suffer as a result. RED gateway is not capable in discriminating against a "out-of-profile" source.

Scenario 3 shows the results of applying Diff-Serv mechanisms in the routers only. Compared to scenario 2, RIO algorithm can discriminate against "out-of-profile" source to limit the detrimental effect OUT packets have on IN packets during congestion. In this case, the CBR source is getting 89% of its packets through vs. 96% of its packets through in scenario 2. (By configuration, the bottleneck link has enough available bandwidth to accommodate 50% of the CBR packets.) The service differentiation among TCP connections with varying RTTs is the biggest during congestion (body column 2). When the network is well provisioned, the service discrimination effect of RIO is damped by the TCP's window algorithm. Short-RTT connections will obtain most of the available bandwidth in the over provisioned situation. In other words, when free of congestion, the innate TCP biases can override the targeted bandwidth allocation created by the Diff-Serv mechanisms in routers.

Scenario 4 illustrates the effects of Diff-Serv mechanisms in both endhost TCP and routers. Compared to scenario 2, the improvement lies in the congested phase, in which, RIO algorithm is able to shield IN packets from the interference of OUT packets. In this case, the CBR source is able to get 50% of its packets through (body column 2), which is roughly what the router can accommodate besides all its pre-allocated resources. Compared to scenario 3, the improvement lies in allocation of bandwidth according to each connection's profile regardless it RTT and the network conditions. When network is congested, each TCP receives close to their targeted throughput; when network is well-provisioned, the allocation of extra available bandwidth is fair among all TCP connections.

4.3 Impact of individual host mechanisms

In this section, we isolate the effect of each of the host mechanisms proposed above. We start with scenario 3 of Table 3, which has standard TCP-reno implementations and Diff-Serv mechanisms applied to routers (tagging and RIO algorithm), and add each proposed mechanism to TCP implementations. We connotate the above proposed mechanisms with the following abbreviations: *WinOpt* for changing the window open-up algorithm with $c * rtt^2$; *Ssthresh* for configuring TCP's *ssthresh* with the target throughput; *ECN* for incorporating differential ECN mechanisms into TCP. Table 4 lists the four stages of the progressive changes. The first stage corresponds to scenario 3 in Table 3, and the last stage corresponds to scenario 4 in Table 3.

The second stage shows a slight improvement over stage 1: the long rtt connections gain more bandwidth than in stage 1 and the short rtt connections perform slight less. However, the CBR source actually gets *more* packets

Table 4: Comparison of individual endhost mechanisms applied to TCP

		Start-Up phase	Congested Phase	Recovery Phase	Over-provision Phase
Standard TCP +Tagging+RIO(3)	TCP0 (80ms R_t=2Mbps)	1.43707	1.32511	1.40382	1.49129
	TCP3 (30ms R_t=1Mbps)	1.05836	0.90249	1.11443	1.37187
	CBR		1.78891		
TCP+WinAdj +Tagging+RIO	TCP0 (80ms R_t=2Mbps)	1.45106	1.46056	1.43349	1.58436
	TCP3 (30ms R_t=1Mbps)	1.02033	0.87470	1.08831	1.34318
	CBR		1.86580		
TCP+WinAdj +Ssthresh +Tagging+RIO	TCP0 (80ms R_t=2Mbps)	1.77864	1.49092	1.8005	2.21291
	TCP3 (30ms R_t=1Mbps)	1.06224	0.94253	1.10739	1.35984
	CBR		1.43665		
TCP+WinAdj +Ssthresh+ECN +Tagging+RIO(4)	TCP0 (80ms R_t=2Mbps)	2.02678	1.89689	2.02658	2.36111
	TCP3 (30ms R_t=1Mbps)	1.04109	0.91049	1.04853	1.33992
	CBR		1.00350		

through the gateway than stage 1. This is due to the following subtle reason. We configure all TCP connections with a new window open-up algorithm using a constant c of 100, which is equivalent to the window open-up rate of 1 packet each 100ms during the "linear increase" phase. This rate is *slower* than the respective window open-up rates of TCP connections in stage 1. In other words, the new window open-up algorithm makes all TCP connections *fair* but all *less* aggressive relative to their counterparts in stage 1. As a result, the CBR connection gains more bandwidth through the gateway. This is similar to the problem discussed in deploying such fairness mechanisms in a heterogeneous environment [11]. We will come back to this point in the deployment section (Section 6.2).

The third stage shows improved throughputs for TCP connections during all phases. When a packet drop occurs, the standard TCP congestion control algorithm reduces its window by one half. If two connections have the same rtts, but one with a higher target rate, it would take the high target-rate connection a longer time to recover to its previous sending rate because the *ssthresh* has been reduced, and it would take many RTTs to open it up. The second mechanism we proposed remedies this bias. After a packet drop, the *ssthresh* is still set to be the byte equivalent of target rate, so TCP can quickly recover through "Slow-start" phase, and throttle back to the target operating point.

The fourth stage shows significant improvement over the third stage in terms of meeting each connection's service profile. This is because TCP benefits from two particular effects: 1) when ECN mechanism is used, TCP reacts to an ECN at most once per round trip time; 2) the configured *ssthresh* keeps TCP operating close to the targeted point.

4.4 Robust recovery from packet losses

In this section, we "zoom in" on the details of TCP's window behaviors before and after incorporating Diff-Serv mechanism. We illustrate the effects in Figure 3. The left graph shows TCP0's *cwnd* and *ssthresh* throughout the entire 100 seconds of simulation (stage 1 setup in Table 4, in which TCP uses standard Reno algorithm). The right graph show TCP0's *cwnd* and *ssthresh* throughout time (stage 4 setup in Table 4, in which TCP incorporates all three Diff-Serv mechanisms). The most pronounced and visible difference lies in how *ssthresh* is adjusted in the two graphs: in the left graph, *ssthresh* adjustment is according to the perceived network conditions and can be drastic and unpredictable. For example, from time 25 to 50 seconds, when there is a CBR source keeping the networks in a congested state, the TCP sources usually detects this and operates in a much reduced operating point. There are several cases *ssthresh* is adjusted multiple times, each for a packet drop within the same window. (Not visible given the granularity of the graph.) From time 80 second and onwards, the network is in a "over-provisioned" state, and the rate adjustments (packet drops) are infrequent and the *ssthresh* are high. In contrast, in the right graph, the *ssthresh* are set by the targeted throughput, so after a packet drop, TCP's *cwnd* is reduced but not the *ssthresh*. (The *ssthresh* is adjusted if the estimated RTT changes, because the *ssthresh* is set to be byte equivalent of target-rate delay product. This is shown in the graph as a few discreet values of *ssthresh*: 40, 45 and 50 packets, etc.) By keeping the *ssthresh* near its target operating point, TCP can quickly recover from its packet losses and not being affected by worsened network

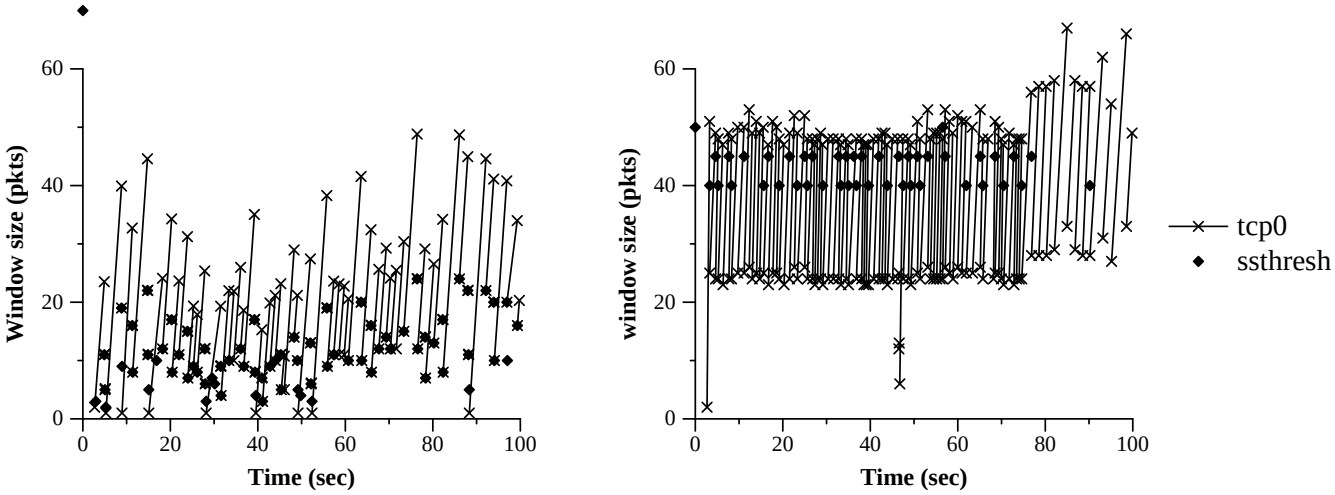

Figure 3: TCP window algorithm before and after incorporating Diff-Serv mechanism

conditions caused by non-congest control sources.

Another difference between the two graphs lies in the rate at which TCP adjusts its window, or the slope of each discrete segment of TCP window adjustments. It is not visible given the granularity at which we present the two graphs. (or Shall we present a finer granularity graph?) But during the "congestion control" phase of window increase, the enhanced TCP using a constant c of 100 is opening up its window *slower* than its counterpart before incorporating the Diff-Serv mechanisms. The right graph appears to have a fast rate of increase because most of the time, it operates in the "Slow Start" phase after a packet drop because *ssthresh* is greater than *cwnd*. (This is also a sign that TCP is not meeting its targeted throughput.) On the other hand, in the left graph, TCP operates in "congestion avoidance" phase after a packet drop because *ssthresh* and *cwnd* are both readjusted.

5 Discussions

5.1 Choice of c in fair window open-up algorithms

Another way of viewing the change in TCP's linear window open-up algorithm is the following: in stead of open up TCP's congestion window by one packet each round trip time, the proposed mechanism opens up *cwnd* by one packet for each standard unit of rtt. If all TCP implementations adopt such algorithm, then they will all increase their window at the same rate regardless of their rtt. Thus, the choice of c, which determines the value of such standard unit of rtt is a crucial one. For example, if c is chosen to be 100, then, the standard unit of rtt is implicitly set to be 100ms ($100 * (0.1)^2 = 1pkt$). In other words, all TCP implementing the above proposed mechanism will be opening up their congestion windows at the same rate as a current TCP implementation with an rtt of 100ms. Essentially, this algorithm make those TCP connections with rtt less than 100ms less aggressive than the current implementations, and those with rtt greater than 100ms more aggressive than the current implementations.

There are two potential problems arise from this. First one, how to choose a value that can be universally agreed upon. The technical merits of the proposed mechanism have been argued for, but the ultimate choice lies in the policies by which the choice of c makes sense. One problem of choosing a relatively small c (less than 100ms, for example) is that for long rtt connections, the new algorithm will result in an effective rate increase even greater than that during the "Slow-start" phase. For example, if a 1sec TCP connection uses the proposed algorithm, it means to open up its window at the rate of one packet each 100ms, which is 10 packets each rtt. Depending on the current number of packets outstanding, this rate can be greater than that during the "Slow-start" phase. The problem with

Table 5: Choice of c in TCP fair window algo.

rtt range	Constant C	Equivalent rtt
(0, 50ms)	1024	31.2ms
(50,100ms)	256	62.5ms
(100,200ms)	64	125ms
(200ms, ∞)	4	500ms

choosing a relatively large c is that this makes TCP window increase algorithm very slow and if this algorithm is universally adopted, it might result in low utilization of link bandwidth immediately after a congestion epoch.

One possible solution is to define a set of inclusive rtt ranges, and within which, modified TCP connections will open up their window at the same rate, but each range has a different window open rate. A reasonable heuristic is that the longer rtt, the slower the window increase rate is because the longer the connection, the more resources (buffer space, or packets in the pipe) it would take. Such ranges of rtts can be easily specified in the service profiles as the ISP will set a lower expected throughput for longer connections. The range of rtts can be chosen to reflect actual market concerns. For example, we could define four ranges of RTTs, inclusive. (0, 50ms) for LANs and WAN range of connections; (50ms, 100ms) for intra-continental connections; (100ms, 200ms) for inter-continental connections; and (200ms, ∞) for non-tether connections. Of course, such policies have to be universally agreed upon and standardized. Those ranges define the particular algorithm and the corresponding values for c.

Another problem with the choice of c lies in incremental deployment of such algorithm. When TCPs with different implementations operate in a heterogeneous environment, TCPs observing the fair algorithms might be at a disadvantage. Fortunately, Diff-Serv mechanisms offer a solution for migrating TCPs to the fair algorithms. See section 6.2 for a detailed discussion on this.

5.2 Limitations of RIO in shielding IN packets from OUT packets

Despite the fact that RIO algorithm can be configured to create strong discrimination against OUT packets, the power of RIO gateway is reduced when it operates in phases 3 and 4, during which, arriving IN packets see long instantaneous queues and are subjected to a probability of dropping. In the above simulations, when the CBR source is sending, the RIO gateway is operating in the "congestion sensitive" phase (phase 2), in which only OUT packets are dropped and no IN packets are dropped. One could easily conceive a situation in which a RIO gateway is kept congested with arriving OUT packets and will have to affect the IN packets as well. This suggests that an admission control at an aggregated level is needed. The aggregate of all services profiles determine the amount of IN packets a gateway should expect, in addition, the traffic conditioners should control the amount of OUT packets admitted into the network.

5.3 Interactions with Tagging Algorithms

Our early work proposed specific tagging algorithms for entities using TCP as transport layer protocol, and used a probabilistic function to reduce the likelihood of multiple packets being tagged and dropped within a TCP window. We find through our simulations that when TCP itself is incorporated Diff-Serv mechanisms, the end-to-end performance relies less on the intricacies and accuracies of tagging algorithms. TCP would perform well without using a probabilistic tagging function. This switches the role of tagging schemes in edge routers from tagging a TCP connection accurately to managing a number of connections sharing a common service profile. This is the topic of our future research.

6 Deployment issues

6.1 Backward compatibility

Among the above three proposed mechanisms, the first and second mechanisms require only TCP sender to change its window adjustment algorithm, and does not require TCP receiver's cooperation.

The second mechanism requires a signaling protocol for communications between transport-layer at the end host and the edge router, or policy servers, which keeps the information about service profiles. This information is used to configure TCP with its initial *ssthresh* value and the *ssthresh* value after each packet drop.

The third mechanism requires TCP to be aware of the IN/OUT bit (or TOS field) of the IP header. This mechanism can be deployed the same time as ECN fields. The mechanism works as follows: a TCP sender always sends out packets with IN/OUT bit as "OFF". A packet goes through a traffic conditioner, which in turn will tag the packet's TOS field as either "ON" or "OFF". A RIO and ECN capable gateway will mark packets differentially, and

turn on ECN field for those packets if necessary. The transport layer at the receiver side has to copy both the ECN field and the TOS field of the IP header in the due acknowledgment packet. The sending TCP will react to a packet with both ECN and TOS bits (an IN packet) set differently from that with only ECN bit set (an OUT packet). The behaviors of TCP sender, receiver and RIO gateways are summarized in Table 7.

6.2 Deployment in a heterogeneous environment

Among the mechanisms we proposed, the first mechanism has been studied in a context of improving fairness for TCP connections with varying rtts [11]. One important problem pointed out by [11] lies not in the algorithm itself, but its interaction with the standard TCP algorithm when both exist simultaneously in a heterogeneous network environment. As discussed before, the fair algorithm make all TCP connections open up their windows at the same rate. With a chosen constant c corresponding to some standard unit of rtt, this algorithms makes any TCP connections with rtt shorter than the standard rtt *less aggressive* than their current implementation, and any TCP connections with rtt longer than the standard rtt *more aggressive* than their current implementations. As a result, if two TCP implementations co-exist in a heterogeneous network environment and their rtts are both shorter than the standard unit of rtt, the connection with current implementation will be more aggressive than the connection with the fair algorithm implementation. Thus, this takes away any incentives for people to deploy the fair algorithm. (Of course, connections with rtt longer than the standard unit rtt will be more aggressive than their current implementation, and there would be incentives for people to deploy such algorithm).

The first half of the Table 6 illustrates this case. We includes another 30ms TCP connection (TCP5) in presentation. Scenario 1 is the case when all TCPs use the standard algorithm and RED is used by routers as the queuing discipline. The two 30ms TCP connections have clear advantage over the 80ms TCP connection, as expected from the current TCP window algorithm. Scenario 2 illustrates the case when TCP0 and TCP3 have upgraded to use the new and fair window algorithm whereas TCP5 has remained the same. The constant c is chosen to be 100, which makes both TCP0 and TCP3 less aggressive than their counterparts in scenario 1. We see TCP0 and TCP3 achieve comparable results whereas TCP5 has gained advantage over both. (TCP0 performs slightly better than its counterpart in scenario 1 but TCP3 performs much worse.)

This confirms that the fairness argument for changing TCP window open-up algorithm is a *necessary* but not *sufficient* condition. Fortunately, we find that Diff-Serv mechanisms in routers can be used to assist in such migration. We find that when Diff-Serv router mechanisms are deployed first and TCPs incorporate all three proposed mechanisms, the allocation of bandwidth is according their respective service profiles (for those TCP which has a service profile), and there is no clear advantage for standard TCP over enhanced TCP. Scenarios 3 and 4 in Table 6 illustrate this. In scenario 3, all TCPs have upgraded to incorporate the Diff-Serv mechanisms, and the allocation of resources is according to their respective service profiles regardless the state of the network. When the network is over provisioned, the available bandwidth is equally distributed among all connections. (When TCP1 stops sending during the last phase, there is 2Mbps "extra" bandwidth shared among five remaining connections.) In scenario 4, TCP4 (not shown) and TCP5 both use the standard TCP window open-up algorithm. The results show that there is no clear advantage of current TCP algorithm over the fair TCP algorithm in the Diff-Serv environment. This preserves the incentives for customers to update their TCP algorithms to incorporate the fair algorithm.

7 Conclusions

The rate adjustment scheme in the current Internet relies on congestion control mechanisms in both transport-layer TCP and congestion signals in gateways. When the premise of resource allocation has changed from "best-effort" model in the current Internet to a "defined-service" model in Diff-Serv architecture, the underlying mechanisms have to be changed to support it as well.

Our early work focused on the mechanisms to be deployed in gateways to provide differentiations among TCP connections. A logical extension of that is to devise mechanisms to be deployed in transport-layer TCP to support the change in premise. This is the focus of our paper. Without introducing any new state variables to the existing TCP machinery, we propose three simple mechanisms to make TCP operate significantly better in a Diff-Serv domain. Those mechanisms preserve the "multiplicative decrease and linear increase" principle of TCP congestion control algorithm and can be applied to similar congestion control algorithms preserving the same principle.

We use simulations to qualitatively verifying the ideas, and discuss incremental deployment of those mechanisms.

Table 6: Heterogeneous Deployment of TCP mechanisms

		Start-Up phase	Congested Phase	Recovery Phase	Over-provision Phase
Standard TCP+RED	TCP0 (80ms)	0.676768	0.491638	0.723149	0.832894
	TCP3 (30ms)	1.622382	1.126404	1.585279	1.804911
	TCP5 (30ms)	1.541346	1.122553	1.610749	1.850088
(1)	CBR		1.978168		
Mixed TCP algorithms +RED	TCP0 (80ms, new)	0.851694	0.499243	0.898498	0.90222
	TCP3 (30ms, new)	0.950140	0.584215	0.792326	1.3719
	TCP5 (30ms, old)	1.893473	1.462454	1.845942	2.018788
(2)	CBR		1.986283		
Uniform TCP algorithms +Tagging +RIO(3)	TCP0 (80ms R_t=2Mbps)	2.02678	1.89689	2.02658	2.36111
	TCP3 (30ms R_t=1Mbps)	1.04109	0.91049	1.04853	1.33992
	TCP5 (30ms R_t=0.6Mbps)	0.659625	0.533941	0.629245	0.969653
	CBR		1.00350		
Mixed TCP algorithms + Tagging +RIO (4)	TCP0 (80ms R_t=2Mbps)	1.984425	1.917876	1.991106	2.18545
	TCP3 (30ms R_t=1Mbps)	0.993548	0.924187	0.991756	1.182006
	TCP5 (30ms,R_t=0.6Mbps,old)	0.602984	0.424578	0.591179	0.940206
	CBR		1.151985		

A relevant and important issue is how an ISP can configure and provision its network, given this set of mechanisms. Future work also includes implementations of those schemes in a testbed environment.

Appendix I: Summary of the proposed mechanisms to TCP

- 1. Change TCP's linear window increase algorithm to be $c * rtt^2$ per round trip time, to correct bias against long-rtt connections during the "congestion control" phase. The value of c is determined by the range of estimated rtt.

- 2. Initialize *ssthresh* to be byte equivalent of target throughput of TCP. Reset *ssthresh* to be the same value when a congestion signal is received. Set congestion window *cwnd* use the standard "Fast Retransmit and Fast Recovery" algorithm.

- 3. If TCP is ECN-capable, TCP's window reduction algorithm will differ depending whether the packet marked with ECN bit is an IN packet or an OUT packet. An IN packet with ECN bit marked indicates a more severe congestion and TCP should retract its *cwnd* to one; an OUT packet with ECN bit marked indicates a less severe congestion and TCP needs to retract its *cwnd* using standard "Fast Retransmit and Fast Reccovery" algorithm.

References

[1] *Ns network simulator*. Available via http://www-nrg.ee.lbl.gov/ns/.

[2] BLACK, D., BLAKE, S., CARLSON, M., DAVIES, E., WANG, Z., AND WEISS, W. An architecture for differentiated services. In *IETF RFC 2475*. IETF, December 1998.

[3] CLARK, D. D. Internet cost allocation and pricing. In *Internet Economics* (1997), J. B. L. McKnight, Ed., MIT Press, pp. 215–253.

[4] CLARK, D. D., AND FANG, W. Explicit allocation of best effort packet delivery service. *IEEE/ACM Transactions on Networking 6*, 4 (1998).

[5] CLARK, D. D., SHENKER, S., AND ZHANG, L. Supporting real-time applications in an integrated services packet network: Architecture and mechanisms. In *Proceedings of ACM SIGCOMM* (Baltimore, 1992), pp. pp 14–26.

[6] FALL, K., AND FLOYD, S. Simulation-based comparisons of tahoe, reno and sack tcp. In *Computer*

Table 7: Summary of Mechanisms in routers and endhost TCP

	TCP sender	*Tagger*	*RIO*	*TCP receiver*

ECN-capable	Turn ECN bit off
	
	if (ECN) {
	  if (IN)
	  cwnd = 1;
	  else
	  cwnd = cwnd/2;
	  ssthresh = byte equi. of BW_{target};
	}
	else {
	  increase *cwnd*;
	}
		Mark IN/OUT bit according to profile	if (ECN bit off) {
			  mark packets differentially;
			}
			else {
			  drop packets differentially;
			}
				Copy ECN and TOS bits to ack pkts

ECN-incapable	Turn ECN bit ON	Mark IN/OUT bit according to profile	Drop packets differentially	Copy ECN and TOS bits to ack pkts

Communications Review (July 1996), vol. 26, pp. 5–21.

[7] FLOYD, S. Connections with multiple congested gateways in packet-switched netwokrs part 1: One-way traffic. *Computer Communication Review 21*, 5 (October 1991), 30–47.

[8] FLOYD, S. Tcp and explicit congestion notification. In *Computer Communication Review* (October 1995), vol. 24.

[9] FLOYD, S., AND JACOBSON, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* (August 1993), 297–413.

[10] HAHNE, E., AND GALLAGER, R. Round robin scheduling for fair flow control in data communications networks. *IEEE International Conference on Communications* (June 1986).

[11] HENDERSON, T. R., SAHOURIA, E., MCCANNE, S., AND KATZ, R. On improving the fairness of tcp congestion avoidance. In *Proceedings of IEEE Globecom '98* (Sydney, 1998).

[12] HOE, J. Improving the start-up behaviors of a congestion control scheme for tcp. In *Proceedings of ACM SIGCOMM* (Stanford, CA, 1996).

[13] JACOBSON, V. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM* (Stanford, CA, 1988).

[14] JAIN, R., CHIU, D., AND HAWE, W. A quantitative measure of fairness and discrimination for resource allocation in shared systems. Tech. rep., Digital Equipment Corporation, 1984.

[15] K, B.-K., AND JEFFERY, J. A new approach to performance-oriented flow control. *IEEE Transactions on Communications 29*, 4 (1981).

[16] LIN, D., AND MORRIS, R. Dynamics of random early detection. In *Proceedings of SIGCOMM'97* (1997).

[17] NICHOLS, K., BLAKE, S., BAKER, F., AND BLACK, D. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. In *IETF RFC 2474*. IETF, December 1998.