Survey on Fairness Issues in TCP Congestion Control Mechanisms

Go Hasegawa and Masayuki Murata

Cybermedia Center, Osaka University
Toyonaka, Osaka 560-0043, Japan
E-mail: {murata,hasegawa}@cmc.osaka-u.ac.jp

Abstract

In this paper, we survey the fairness issues in the congestion control mechanisms of TCP, which is one of most important service aspects in the current and future Internet. We first summarize the problems from a perspective of the fair service among connections. Several solution methods are next surveyed. Those are modifications of TCP congestion control mechanism and router support for achieving the fair service among TCP connections. We finally investigate the fair share of resources at endhosts.

1 Introduction

In spite of a rapid growth of the Internet population and an explosive increase of the traffic demand, the Internet is still working without collapse. Of course, continuous efforts to increase the link bandwidth and router processing capacity is supporting the Internet growth behind the scenes. However, a most essential device for achieving such a success is a congestion control mechanism provided by a transport–layer protocol, i.e., TCP (Transmission Control Protocol). In TCP, each endhost controls its packet transmission rate by changing the window size in response to network congestion. A key is that the TCP congestion control is performed in a distributed fashion; each endhost determines its window size by itself according to the information obtained from the network.

In general, there are two major objectives in the congestion control mechanism. The one is to avoid an occurrence of the network congestion, and to dissolve the congestion if the congestion occurrence cannot be avoided. The other is to provide *fair service* to connections. Keeping the fairness among multiple homogeneous/heterogeneous connections in the network is an essential feature for the network to be widely accepted. The fair service also involves detecting mis–behaving flows which do not properly react against the network congestion and unfairly occupy the network resources (such as router buffer and link bandwidth).

The above–mentioned fair service could be achieved if the network explicitly allocates the network resources to each of active connections. An example can be found in the ATM ABR service where the rate of each source is determined by switches [1]. Of course, such a strategy cannot be adopted in the Internet. The basic role of the network layer is to *carry the bits* between endhosts [2], and the congestion control is performed by TCP, which is a transport protocol located at the endhosts. It is just a fundamental principle of the Internet, and only such a philosophy can realize a fully distributed network. However, it also introduces several problems. TCP at endhosts should react against the network congestion using imperfect information on the congestion level of the network. The end user can intentionally or unintentionally modify the TCP code so that it does not react against the network congestion properly. Such a *mis–behaving* connection may continue to send packets without throttling the window even if the congestion happens in the network. If other users properly perform the congestion control, that mis–behaving user may be able to receive higher throughput without window throttling. It is becoming more critical with an increase of the commercial use of the Internet, and the authors believe that fairness issues are sometimes more important than performance issues (e.g., total throughput of an aggregated flow on the link).

Actually, some recent researches impose that the network takes an active role in congestion control. It seems to violate the fundamental principle of the Internet, but we actually need a participation of the network in order to improve the fairness among connections. Perhaps, the fairness degree is much dependent on the functionality provided by the network, and functional partitioning between the network layer and the transport layer should be reconsidered in order to resolve the fairness problems for the current Internet. Keeping those facts in mind, we will focus on the fairness aspect of TCP in this paper.

This paper is organized as follows. We first summarize the congestion control mechanism of TCP and point out the fairness problems in Section 2. Router support for TCP congestion control is also described. We then review the methods for fairness improvements by TCP congestion control algorithms in Subsection 3.1, and by packet buffering and scheduling algorithms at the router in Subsection 3.2. We also describe TCP–friendly rate control for non–TCP connections (Subsection 3.3) and fair resource allocation at endhosts (Subsection 3.4). Concluding remarks are finally presented in Section 4.

2 Congestion Avoidance and Dissolution

In this section, we first summarize the existing congestion control mechanisms of TCP, which is performed at the end-hosts. The router's packet processing mechanism to support the congestion control is next introduced. Those include RED (Random Early Detection) and ECN (Explicit Congestion Notification).

2.1 TCP Congestion Control Algorithms

The TCP sender maintains *congestion window* (usually represented as *cwnd*), and it can inject new packets into the network up to the congestion window without receipt of acknowledgements. It is flow control between TCP sender and receiver. As a congestion control mechanism, the TCP sender dynamically increases/decreases the window size according to the degree of the network congestion. The congestion level is conjectured via packet losses, which can be detected by discontinuous receipts of the acknowledgements, or timeout expiration. The timeout expiration happens if more than two or three packets in the congestion window are lost [3]. The TCP sender then recognizes that the network is congested, and throttles its window. Otherwise, it determines that the network is not congested, and inflates its window.

More specifically, TCP Reno has two phases in increasing the congestion window *cwnd*; slow start phase and congestion avoidance phase. When an ACK packet is received by the TCP sender at time $t + t_A$, $cwnd(t + t_A)$ is updated from $cwnd(t)$ as follows (see, e.g., [4]);

$$cwnd(t + t_A) = \begin{cases} \text{Slow Start Phase:} \\ \quad cwnd(t) + m, \text{if } cwnd(t) < ssth(t) \\ \text{Congestion Avoidance Phase:} \\ \quad cwnd(t) + m^2/cwnd(t), \text{if } cwnd(t) \geq ssth(t) \end{cases}$$

where $ssth(t)$ is the threshold value at which TCP changes its phase from the slow start phase to the congestion avoidance phase. When packet loss is detected by timeout, $cwnd(t)$ and $ssth(t)$ are updated as follows;

$$ssth(t) = cwnd(t)/2; \quad cwnd(t) = m.$$

When packet loss is detected by a fast retransmission algorithm [4], the window size $cwnd(t)$ is halved. That is,

$$ssth(t) = cwnd(t)/2; \quad cwnd(t) = ssth(t).$$

TCP then enters the fast recovery phase [4]. In this phase, the window size is increased by one packet when a duplicate ACK packet is received, and $cwnd(t)$ is restored to $ssth(t)$ when the non–duplicate ACK packet corresponding to the retransmitted packet is received. Most of the current TCP implementations are based on the above–mentioned TCP Reno version. However, it is far from the perfect one; many problems have been pointed out in the past literature. See, e.g., [5, 3, 6].

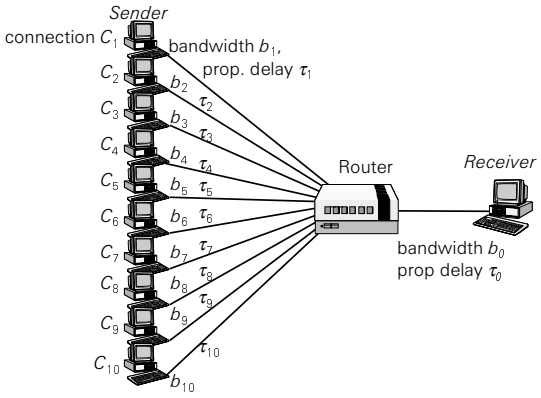


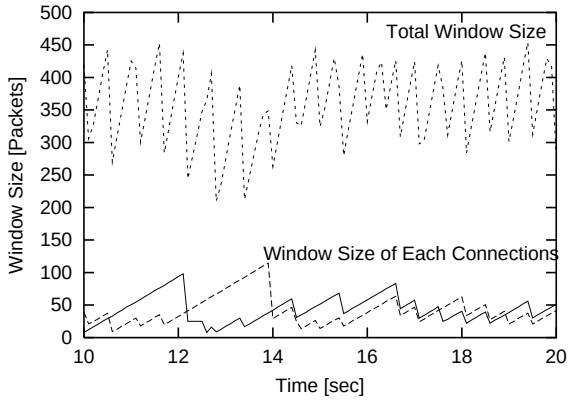
Figure 1: Simulation model

In this paper, we focus on the fairness aspect of TCP, and several simulation results are presented in the below. A simulation model is depicted in Figure 1, where the bottleneck link is shared by ten connections. The bandwidth of the bottleneck link b_0 and the propagation delay τ_0 are set to be 100 Mbps and 1 msec, respectively, throughout this paper. The router's buffer size is set to 512 packets unless otherwise specified. The access link capacity b_i ($1 \leq i \leq 10$) and the propagation delay τ_i ($1 \leq i \leq 10$) will be changed. For simulation, we used the network simulator `ns2` [7].

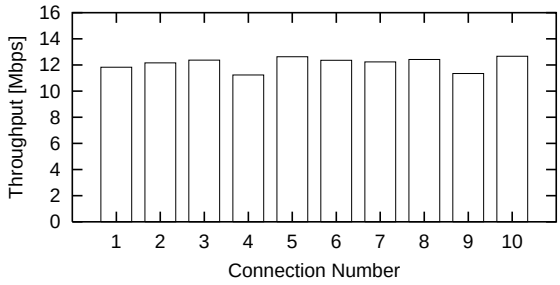
Due to an intrinsic nature of the window–based congestion control mechanism of TCP, TCP connections do not receive same instantaneous throughput even if connections share the identical end-to-end path. We set the homogeneous case using Figure 1; all b_i's and τ_i's are set to 100 Mbps and 10 msec, respectively. Figure 2(a) shows the time-dependent behavior of the window sizes of connections. Note that in the figure, only connections C_1 and C_{10} are plotted for clear presentation. Throughput values of all connections (averaged over 20 sec simulation time) are compared in Figure 2(b).

If TCP connections have different propagation delays, even the long–term throughput values become different. A simulation result is shown in Figure 2(c). In obtaining the figure, we change the propagation delays as $\tau_i = 10 \times i$ msec, while link capacities are unchanged. The figure shows that connections with smaller propagation delay clearly obtain larger throughput than other connections with longer one [8], and throughput values are almost in inverse proportion to the the propagation delays. More importantly, if the link capacities of TCP connections are different, the relative throughput of TCP connections against its access link bandwidth are much different. Figure 2(d) shows such a case. In obtaining the figure, we set the propagation delays τ_i's to be 1 msec, but the link bandwidth $b_i = 10 \times i$ Mbps for connection C_i. The result in Figure 2(d) indicates that the throughput values are far from a linear relation to the link capacity. It means that more bandwidth does not help improve the throughput.

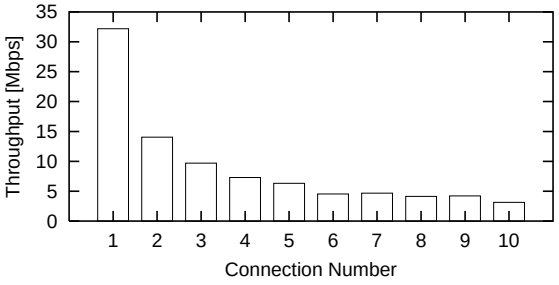
Resolving the above problems is important since the heterogeneous environment is becoming common as vari-



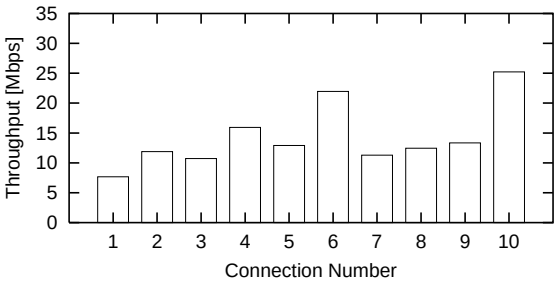
(a) Time dependent behavior of the window size



(b) Fairness comparisons



(c) The heterogeneous case with different propagation delays



(d) The heterogeneous case with different link capacities

Figure 2: The case of TCP Reno applied to the drop–tail router

ous access methods, such as DSL and Cable Modem accesses, are realized in the Internet. A main cause of the unfairness in TCP is owing to its AIMD (Additive Increase/Multiplicative Decrease) window updating algorithm. It was shown in [9] that only AIMD establishes the fairness among connections, and the window updating algorithm of TCP is just a class of AIMD. However, it is self–clocked. That is, the window size is updated only when the sender receives ACK from the receiver, and therefore the window curves of TCP connections with different propagation delays and/or different access link capacities become different. The window updating algorithm for achieving fairness can be found in, e.g., [10]. Another version of TCP to improve the fairness will be described in Subsection 3.1.

2.2 Router Support for Congestion Control

2.2.1 RED (Random Early Detection)

Most routers have employed a drop–tail (FIFO) discipline as a buffer management mechanism. The drop–tail router serves incoming packets in the arrival order, and when the buffer is full, the newly arriving packet is simply discarded. The problem is that drop–tail routers tend to discard packets in burst [11], which results in that packets from the same connection are likely to be discarded. As a result, the fast retransmit algorithm does not help avoid timeout expiration, and it leads to a global synchronization problem [12]. Accordingly, the authors in [13] proposed a RED (Random Early Detection) gateway algorithm. RED is designed to cooperate with the congestion control mechanism of TCP. It detects the beginning of congestion by monitoring the average buffer occupancy at the router, and notifies the connections by intentionally dropping packets at a certain probability. RED sets the packet dropping probability by a function of buffer occupancy (average queue size). By keeping the average queue size low, burst dropping can be avoided even when packets from the same connection continuously arrive. That is, the algorithm has no bias against bursty traffic.

More specifically, RED uses a low pass filter with an exponential weighted moving average in order to calculate the average queue size avg, which is maintained and compared with two thresholds: minimum threshold (min_{th}) and maximum threshold (max_{th}). The packet dropping probability is determined in different ways according to the queue size avg;

- If $avg < min_{th}$, all arriving packets are accepted.
- If $min_{th} < avg < max_{th}$, arriving packets are dropped with probability $p(x)$.
- If $max_{th} < avg$, all arriving packets are dropped.

The function $p(x)$ should depend on average queue length, x, and a typical (and probably ideal) form of $p(x)$ is illustrated in Figure 3. However, an adequate determination method is still not clear, and a constant dropping probability is usually used [13].

The RED router can avoid the occurrence of retransmit timeouts of TCP connections, and henceforth most of lost packets are retransmitted by the fast retransmit algorithm. Furthermore, the phase effect that all connections exhibit

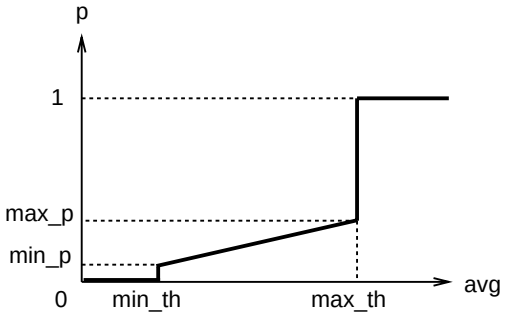


Figure 3: RED packet dropping rate $p(x)$

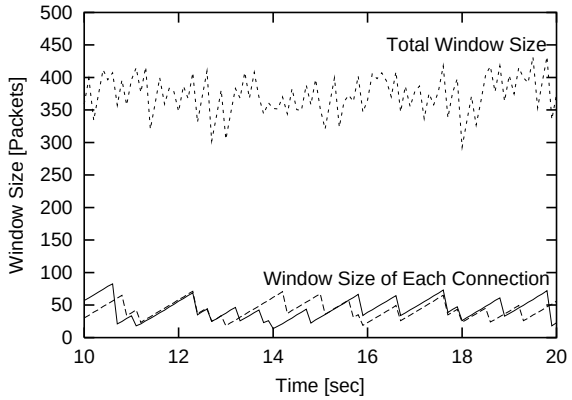
the similar window changes can also be avoided [13]. The RED router is then expected to provide higher throughput to TCP connections than the drop–tail router [13].

Figure 4 shows simulation results in order to demonstrate improvements by the RED router. We again use a simulation model shown in Figure 1. Figures 4(a) and 4(b) are for the homogeneous case and correspond to the case of the drop–tail router shown in Figures 2(a) and 2(b), respectively. Those figures show that RED can improve the total throughput. When the drop–tail discipline is used, packet losses take place continuously, and multiple TCP connections simultaneously suffer from the performance degradation by packet losses. With the RED mechanism, on the other hand, the synchronization of throughput degradation can be avoided by its probabilistic packet discarding. However, the main purpose of the RED router is not to improve the fairness among connections. Even if the access link capacity is same among all connections, throughput values of those connections are much affected by the propagation delays. See Figure 4(c), which corresponds to Figure 2(c) of the drop–tail router. Its unfair property also appears when the access link capacities of connections are different. Compare Figures 4(d) and Figure 2(d). Note again that the RED mechanism itself is not intended to establish the fairness among connections in the heterogeneous case, but it is often used as a basis of developing the new packet processing methods at the router. See Subsection 3.2.2.

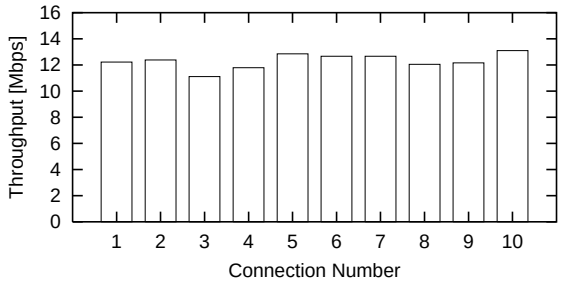
2.2.2 ECN (Explicit Congestion Notification)

An ECN (Explicit Congestion Notification) mechanism is recently proposed in [14, 15] while the basic idea is not new in the packet–switching network. The key idea of ECN is to avoid throughput degradation by packet losses caused by the RED algorithm. That is, the router *marks* packets when the network is congested and the TCP sender is notified of the network congestion by receiving marked ACK packets, and reacts in the same way as packet loss is detected, without waiting retransmit timeout or receiving duplicate ACKs. With an ECN support, the TCP sender can quickly react against network congestion.

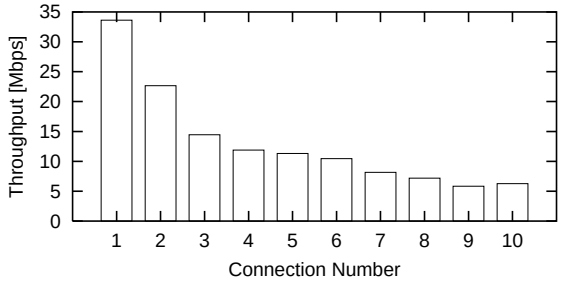
Fairness among heterogeneous TCP connections can also be improved by the ECN mechanism. A simple application



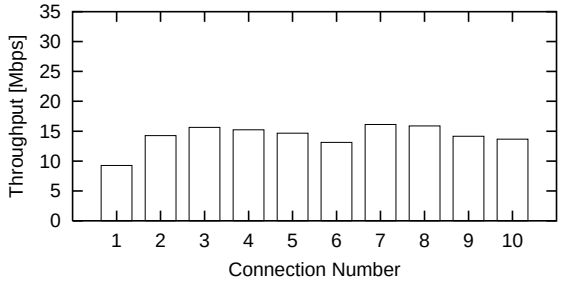
(a) Time dependent behavior of the window size



(b) Fairness comparisons for the homogeneous case



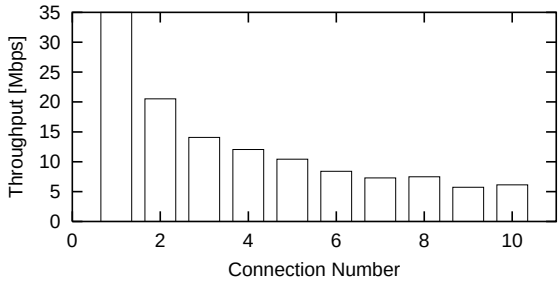
(c) Fairness comparisons for the heterogeneous case with different propagation delays



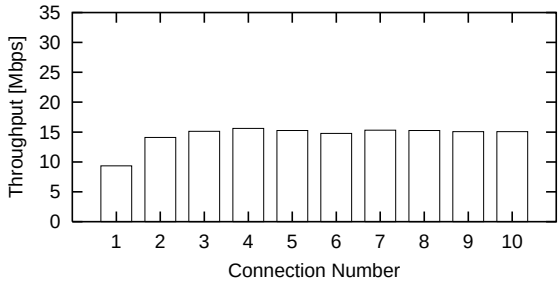
(d) Fairness comparisons for the heterogeneous case with different link capacities

Figure 4: The case of TCP Reno applied to the RED router

of the ECN mechanism is shown in Figure 5. Cases of different propagation delays and link capacities are shown in Figures 5(a) and 5(b), respectively. Those corresponds to



(a) The heterogeneous case with different propagation delays



(b) The heterogeneous case with different link capacities

Figure 5: The case of TCP Reno applied to the ECN–capable router

Figures 2(c) and 2(d) for the drop–tail router. The cases of the RED router were shown in Figures 4(c) and 4(d). The improvement is very limited in the above case. Note that in obtaining Figure 4(d), throughput values are expected to be in proportion to the link capacities since we set the different link capacities among each connections. A more careful tuning of the window updating algorithm is necessary to fully utilize the ECN capability for fairness improvement. The authors in [16] proposed the algorithm on how the router marks ECN bit to incoming packets, and how the TCP sender reacts against the ECN–marked ACK packets, so that unfair property among TCP connections with different round trip times can be avoided. However, the above–mentioned *careful tuning* limits its applicability. Perhaps, we need more router support for further fairness improvement as will be described in Subsection 3.2.

3 Fairness Improvements for TCP Connections

3.1 Fairness among TCP connections

An interesting algorithm to improve the fairness among TCP connections without router support can be found in the TCP Vegas version [17, 18, 19]. TCP Vegas employs a different mechanism from the other versions of TCP in detecting the network congestion. It controls the window size by observing RTTs (Round Trip Times) of packets that the connection has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and gradually throttles the window size. If RTTs become small, on the other hand, TCP Vegas determines that the network is relieved from the congestion, and increases the window size. TCP Vegas updates its congestion window size in congestion avoidance phase as follows;

$$cwnd(t+t_A) = \begin{cases} cwnd(t)+1, & \text{if } diff < \frac{\alpha}{base\_rtt}, \\ cwnd(t), & \text{if } \frac{\alpha}{base\_rtt} < diff < \frac{\beta}{base\_rtt}, \\ cwnd(t)-1, & \text{if } \frac{\beta}{base\_rtt} < diff, \end{cases} \quad (1)$$

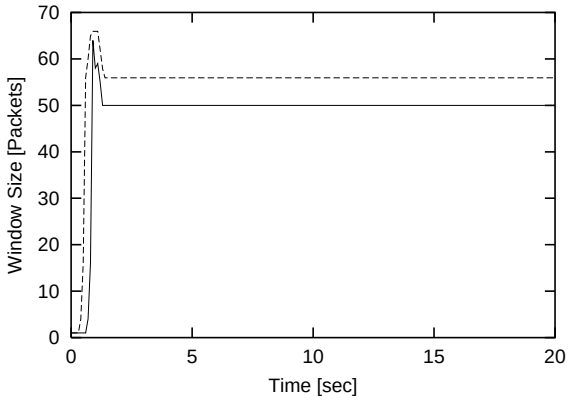
where rtt is the observed round trip time, $base\_rtt$ is the smallest value of observed RTTs, and α and β are some constant values. The quantity $diff$ is determined by the following equation:

$$diff = \frac{cwnd}{base\_rtt} - \frac{cwnd}{rtt}.$$

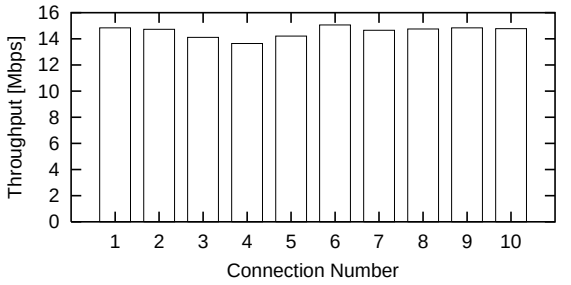
Note that Eq. (1) used in TCP Vegas indicates that if observed RTTs of the packets are identical, the window size remains unchanged.

Figure 6(a) shows the time–dependent behavior of the window size of TCP Vegas. To obtain the figure, we used the simulation model in which all connections have identical propagation delays and link capacities. In TCP Vegas, the window size is converged to the fixed value and no packet loss occurs. It is different from TCP Reno in which the packet loss is necessary to invoke the congestion control, as having been shown in Figure 2(a). TCP Vegas is then expected to achieve higher throughput than TCP Reno. Actually, in [17], the authors show that TCP Vegas achieves 40%-70% higher throughput, with one–fifth to one–half the losses, as compared to TCP Reno, through simulation and implementation experiments. The improvement using our simulation model is shown in Figure 6(b). Furthermore, TCP Vegas can slightly improve the unfairness nature found in TCP Reno, even when TCP connections have different propagation delays and link capacities. Compare Figures 6(c) and 6(d) and Figures 2(c) and 2(d). It is because TCP Vegas tries to find the available bandwidth for itself [8].

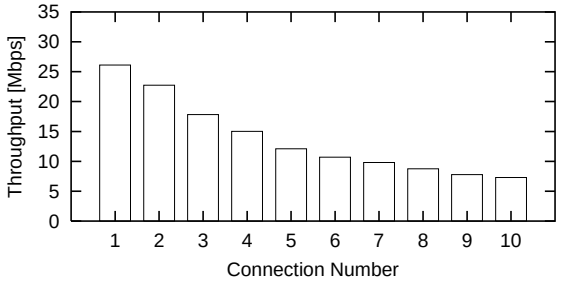
A difficulty for deploying TCP Vegas is the unfairness problem when it co-exists with TCP Reno. In [20], the authors pointed out that when TCP Reno connections and TCP Vegas connections share the bottleneck link, the TCP Reno connection occupy the link bandwidth. Then TCP Vegas connections suffer from significant performance degradation, while TCP Vegas solely works well as shown in the above. See Figure 7, in which five connections of Figure 1 are changed to utilize TCP Vegas while other five connections still use TCP Reno. Other parameters (propagation delays and link capacities) are identically set. Throughput values of TCP Vegas connections are much degraded especially when the buffer size at the router becomes large. It is due to the difference of the congestion control mechanism of TCP Reno and TCP Vegas. TCP Vegas increases its window size *conservatively*, which means that if RTT becomes



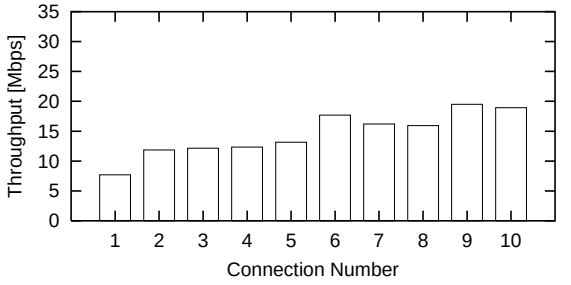
(a) Time dependent behavior of the window size



(b) Fairness comparisons for the homogeneous case



(c) Fairness comparisons for the heterogeneous case with different propagation delays



(d) Fairness comparisons for the heterogeneous case with different link capacities

Figure 6: The case of TCP Vegas applied to the drop–tail router

large, it decreases the window size. TCP Reno, on the other hand, increases the window size *aggressively* until a packet

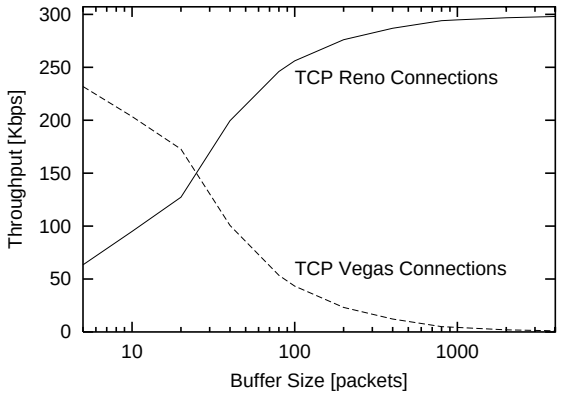


Figure 7: Comparisons of throughput where TCP Reno and Vegas share the link

is lost.

Problems of TCP have been repeatedly pointed out, and in the early 90's, new light–weight transport protocols were actively discussed [21]. However, TCP is now widely deployed in the operating network. Thus, it now becomes difficult for some new protocol (including the modified version of TCP such as TCP Vegas) to take the place of TCP unless the protocol designer considers a migration path from the operating TCP. Otherwise, the new protocol would not be received. In that sense, TCP Vegas seems to be difficult to be widely accepted. However, the above–mentioned problem may be solved to large extent if the router supports some per–flow queueing mechanism, which will be introduced in the next subsection.

Fortunately, the role of the receiver in the TCP mechanism is just to return ACKs (acknowledgements) as the packet is successfully received. It indicates that we have an opportunity to incorporate some modification at the sender side of TCP to improve the performance and fairness. Such an example will be presented in Subsection 3.4.

3.2 Packet Buffering and Scheduling Algorithms for Improving Fairness

In the previous subsection, we have reviewed the congestion control mechanism at endhosts. As long as the congestion control is performed in an end-to-end fashion by TCP, it would be impossible to achieve complete fairness among connections. It is because TCP treats the network as a *black box*, and the TCP sender cannot obtain perfect information on the network resource usage and status of other connections sharing the bottleneck link. One–bit marking of ECN also has a limitation. Therefore, some packet buffering and/or scheduling mechanism should be supported at the router to further improve the fairness among connections. In what follows, we will survey several router algorithms for improving fairness among connections.

3.2.1 Per–flow Queueing

Per–flow queueing is not a new idea. The Fair Queueing (FQ) algorithm [22] and Round Robin (RR) algorithm [23] are a class of per–flow queueing methods. FQ and RR algorithms can achieve good fairness among connections, but have several problems. The computational complexity of packet scheduling in FQ is $O(\log n)$ where n is the number of active flows. RR and WRR (Weighted Round Robin; a variant of RR to support weighted fairness) also have an order of $O(\log n)$. Furthermore, those do not consider variable packet sizes.

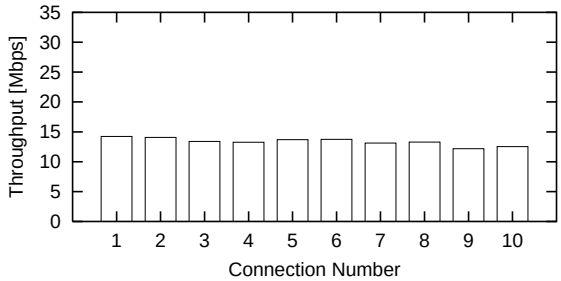
To resolve such problems, the authors in [24] proposed DRR (Deficit Round Robin). Each flow is assigned to *DRR sub-queue* in the router buffer, using the hash function. Then, DRR serves each sub-queue by *Deficit Counter*, which absorbs the difference of packet sizes. That is, each sub-queue has a deficit counter, and the counter is incremented in each round, considering the pre-defined service rate. When an incoming packet is served in the round, the counter is decremented by the amount corresponding to the packet size. If the assigned service rate is not completely utilized due to no more packets arrived at the sub-queue, the counter value is kept and will be used in the next round. Through these mechanisms, DRR provides reasonable fair service among connections and it works in $O(1)$ expected. DRR has already been implemented in a commercially available product [25]. To see the effectiveness of DRR, we show the simulation result [26]. As a simulation setting, we used the heterogeneous cases where connections have different propagation delays and link capacities, which are shown in Figures 8(a) and 8(b), respectively. By setting service rate of each DRR queue in proportion to each connection's input link bandwidth, an almost complete fairness can be achieved.

While DRR reduces the overhead of the RR algorithm, it is still much more complex than single–queue methods such as the drop–tail and RED. Therefore, it might be difficult to apply even DRR to core routers, which have to accommodate thousands of, and maybe tens of thousands of active flows. Another approach is to introduce the stateless queueing into the core routers, which will be described in the next subsection.

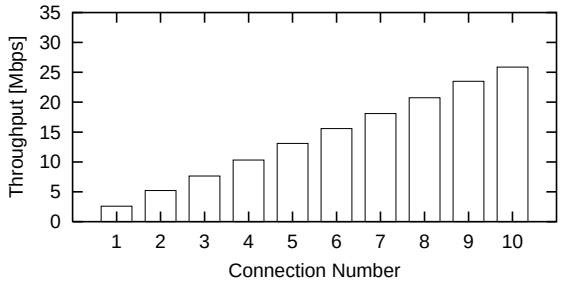
A fundamental limit of the DRR algorithm exists in its scalability. For realizing the DRR algorithm, the router should know how the bandwidth is allocated to the connections. One approach is to prepare the mapping table for the user's IP address (and possibly port address) and the user's access line bandwidth. It directly implies that the DRR algorithm can be applied to the edge routers, but difficult for the core routers.

3.2.2 RED Variants

Another approach is to use *per-flow information*, not *per–flow queueing*. One example is FRED (Fair Random Early Detection) [27]. FRED uses single–queued buffer, but it counts the number of arriving/departing packets of each active flow, and calculates its buffer occupancy. It is used to differentiate RED's packet dropping probability among connections. The authors demonstrate through some simulation experiments that FRED can improve fairness among TCP connections. To see it, the same simulation model as described in the previous subsection was used to obtain Figure 9. Those figures clearly show that FRED can provide the intermediate solution between RED and DRR mechanisms by keeping *incomplete* flow information.



(a) The heterogeneous case with different propagation delays

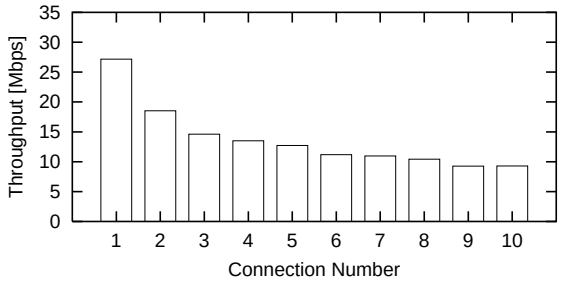


(b) The heterogeneous case with different link capacities

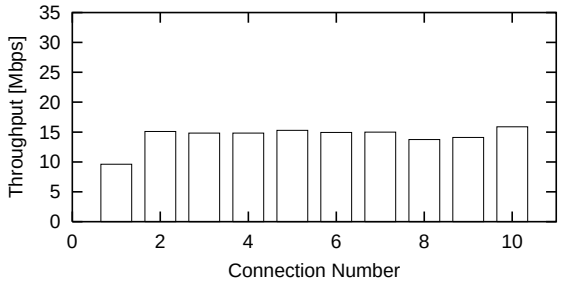
Figure 8: The case of TCP Reno applied to the DRR router

The FRED algorithm maintains information on all active flows, and henceforth, if there are many flows arriving at the router, the fairness may be much degraded due to lack of buffer space. It then fails in providing the fair service. We note that by FRED, excessive packets from UDP connections are dropped at the router for fair share of the output link. It must lead to much degradation of user's perceived QoS in real–time applications. The UDP connection should be aware that a fair share of the link is maintained in the network, and actively controls its rate according to the network status. We will describe the rate control of UDP connections in Subsection 3.3.

In [28], the authors try not to use even per–flow states for further reducing processing overhead. The authors in [28] proposed a SRED (Stabilized RED) mechanism, which estimates the number of active flows in the router. In SRED, the router maintains a fixed–size table called *Zombie List*, whose entries contain the information on active flows. When the table becomes full, one of the table entries is randomly overwritten by that of the arriving packet. When a packet arrives at the router, it compares with the randomly–selected entry. If it *Hits* the entry, the packet is dropped at a



(a) The heterogeneous case with different propagation delays



(b) The heterogeneous case with different link capacities

Figure 9: The case of TCP Reno applied to the FRED router

higher probability than when it does not hit. A main purpose of the SRED algorithm is to resolve the scalability problem against the increasing number of flows, but the misbehaving flows can also be detected in SRED to some extent as noted by the authors. By the SRED mechanism, packets from mis–behaving flows are largely dropped because the Zombie List tends to contain more entries of such flows, and therefore those packets are discarded at high probability. The advantage (and disadvantage) of SRED is to keep a limited size of the list. A further investigation is necessary to identify the required size of the list to keep the fairness among active flows.

Another idea is that core routers do not maintain even per–flow information, which is described in [29]. In [29], the authors propose a CSFQ (Core Stateless Fair Queueing) mechanism, which classifies routers into the edge routers and the core routers, and assigns different functions to each kind of routers. The edge router, which does not handle many connections, maintains per–flow state to label the incoming packets according to the estimation of the packet incoming rate. If some connection sends many packets, packets are marked, which will be first dropped at the core routers. The core router maintains no per–flow state and it just employs RED–based probabilistic packer discarding, for which the packet labels (marked or not) are used. CSFQ discriminates the marked and non–marked connections, but the determination method of packet dropping probabilities is still not clear. Refer to [29] for a more detailed algorithm of CSFQ.

3.3 TCP–friendly Rate Control

It is well known that when TCP and UDP connections share the link, UDP connections tend to occupy the link [30]. It is because the UDP connection does not adequately react against the congestion while the TCP connection does [30]. Example behaviors of TCP and UDP connections are shown in Figure 10 where we again use the model shown in Figure 1 except that one connection uses UDP to transmit packets. The UDP connection starts its packet transmission at time 0 and continues it at rate 100 Mbps. Other nine connections use TCP Reno. Connection C_i starts its packet transmission at $i \times 5$ msec. We use the drop–tail, RED, and FRED routers, which are shown in Figures 10(a), 10(b) and 10(c), respectively. From the figure, it is very apparent that active queue management such as FRED is necessary to keep fair bandwidth allocation between TCP and UDP connections.

In the above example, we consider that the UDP connection keeps to transmit the packets at constant rate. It is true that most of the currently available real–time multimedia applications are equipped with its proprietary delay and/or rate adaptive control mechanisms. See, e.g., [31]. However, it is only for its own purpose of performing better–quality presentation. Accordingly, a notion of *TCP friendly* rate control is now proposed [32], where TCP friendliness is defined as *a non–TCP connection should receive the same share of bandwidth as a TCP connection if they traverse the same path.*

A key to establishing TCP–friendly congestion control is that non–TCP flow precisely estimates throughput of TCP connections, in order to regulate its rate. Fortunately, we already have a good estimation method for TCP throughput. In [33], the authors derive the following equation for estimating the TCP throughput, λ;

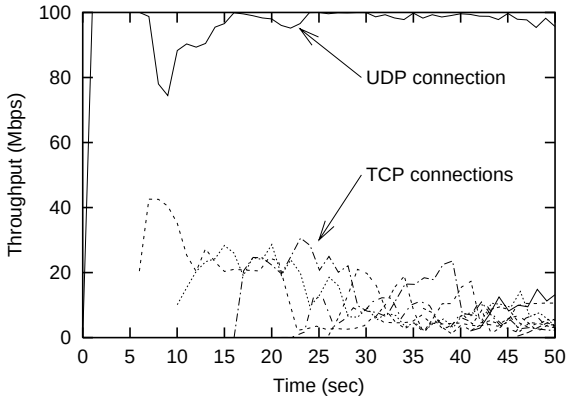
$$\lambda = C \times \frac{MTU}{RTT\sqrt{p}}$$

where RTT is the round trip time of the connection, MTU is the packet size, and p is the observed packet loss probability. C is the constant value, which is set to 0.87 when the receiver uses the delayed ACK option. Otherwise it is set to 1.22. A more detailed derivation is available in [34], where other parameters of the buffer size at the receiver (W_{max}) and TCP's timeouts (T_0) are incorporated:

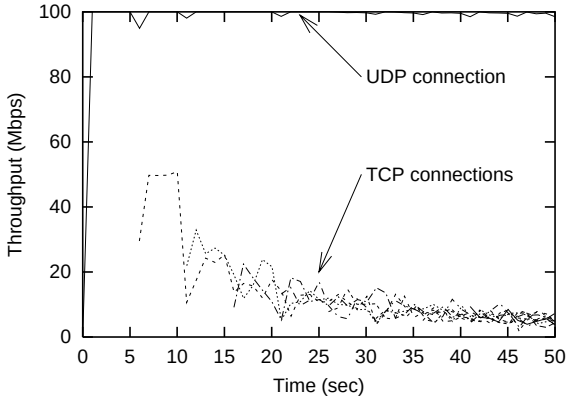
$$\lambda = \max\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0\ \min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)}\right) \quad (2)$$

where b is set to 2 if the receiver uses the delayed ACK option, and 1 if not.

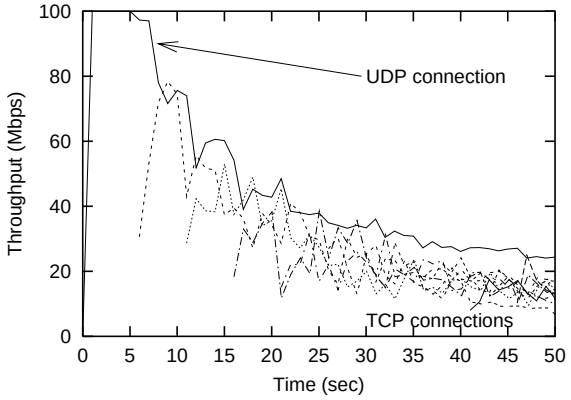
According to the above estimation methods, a non–TCP flow can performs TCP–friendly rate control by observing parameters (packet loss rate, RTT and RTO), and by setting its packet sending rate using the above equation. Applications of TCP–friendly rate control to real–time multimedia



(a) The case of the drop–tail router



(b) The case of the RED router



(c) The case of the FRED router

Figure 10: Fairness comparisons in the case of link sharing by TCP and UDP connections

can be found in [35, 36, 37]. An important issue of the TCP–friendly rate control is how to define a time scale of the fair share, which is still not clear. It is especially important when TCP friendliness is applied to real–time multimedia; a most TCP–friendly rate control is probably to use the same AIMD used in TCP. However, the rate variation of TCP–friendliness may be too frequent for video and audio presentation.

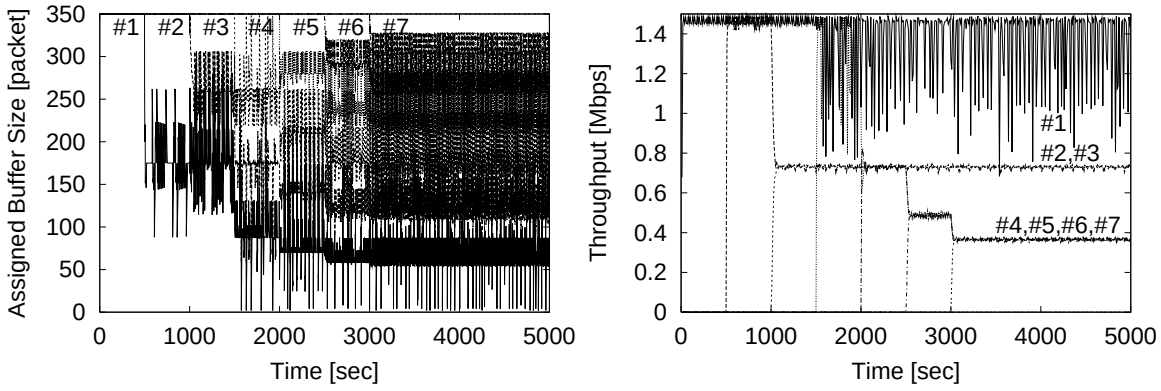
3.4 Fair Resource Allocation at Endhosts

The link bandwidth has been drastically improved by new technologies such as WDM (Wavelength Division Multiplexing), and the bottleneck point is now shifting from network to endhosts when we expect very high–performance communication. It is therefore important to consider fair allocation of the resources at endhosts. Those include socket buffer and processing power. Of course, the performance improvement at the endhosts is not new. See, e.g., [38, 39, 40]. This subsection is devoted to the fairness aspect of protocol processing at the endhosts.

Suppose that a server host is sending TCP data to two clients of 64Kbps dial–up (say, user A) and 100Mbps LAN (user B). If the server host assigns an equal size of the socket buffer to both users, it is likely that the amount of the assigned buffer is too large for user A, and too small for user B, because of the difference of capacities (more strictly, bandwidth–delay products) of two connections. For an effective buffer usage, fair allocation of the socket buffer should be taken into account. In [41], the authors proposed a buffer tuning algorithm called ABT (Automatic Buffer Tuning), which dynamically adjusts the size of socket buffer at the sender host according to the change of the TCP sending window size of connections. However, since the average window size of the connection must reflect the expected throughput of that connection, buffer allocation based on the window size becomes too frequent.

Accordingly, SABT (Scalable ABT) is proposed in [42] to assign the buffer according to the *expected* throughput calculated from RTT, RTO and packet loss probability using Eq. (2). If the expected throughput is determined from those parameters, an adequate buffer size to be allocated to each connection can be obtained. If the total socket buffer becomes short, the fair allocation of the socket buffer can also be realized by taking account of the expected throughput values of connections.

We last show simulation results of ABT and SABT algorithms [42]. Figure 11 depicts the network model used for simulation experiments. The model consists of a sender host, seven receiver hosts, six routers, and links interconnecting routers and sender/receiver hosts. Seven TCP connections are established at the sender host (connections C_1 through C_7). As shown in the figure, connection C_1 occupies router R_1, connection C_2 and C_3 share router R_2, and the rest of connections C_4 through C_7 share router R_3. The capacities of links between routers are identically set to be 1.5 [Mbps], and those of links between routers and the sender host, and between routers and the receiver hosts are set to be 155 [Mbps] and 10 [Mbps], respectively. Further, propagation delays of links between routers, and that between routers and sender/receiver hosts are 100 [msec] and 1 [msec], respectively. In this simulation experiment, connection C_i starts sending packets at time $t = (i-1) \times 500$ [sec]. The simulation ends at 4,000 [sec]. Therefore, when seven connections join the network, the ideal through-



(a) ABT Case: Changes of Assigned Buffer Size

(b) ABT Case: Changes of Throughput

(c) SABT Case: Changes of Assigned Buffer Size

(d) SABT Case: Changes of Throughput

Figure 12: Comparative results of ABT and SABT

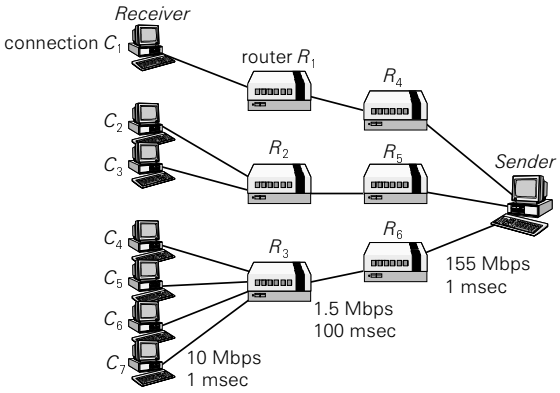


Figure 11: Network model for ABT and SABT

put becomes 1.5 [Mbps] for connection C_1, 0.75 [Mbps] for connections C_2 and C_3, and 0.375 [Mbps] for connections C_4 through C_7.

Figure 12 shows the results for the throughput and assigned buffer size as a function of time. We set the send socket buffer size of a sender host B to 350 [packets]. Figures 12(a) and 12(b) clearly show that ABT tries to control the assigned buffer size according to each TCP connection's window size, but it cannot provide a stable buffer assignment. It is because ABT assigns the buffer to each TCP connection according to the current window size of the connection, which oscillates dynamically due to an inherent nature of the TCP window mechanism. Therefore, the assigned buffer sizes of connections which do not need large buffer (connections C_2 through C_7 in the current case) are often inflated, leading to the temporary decrease of the buffer assigned to connections which need large buffer (C_1). As a result, the throughput of C_1 is often degraded. On the other hand, SABT keeps stable and fair buffer assignment as shown in Figure 12(c), leading to fair treatment in terms of throughput (Figure 12(d)).

Of course, the resource at the endhosts is not limited to the socket buffer. One of future topics is then to appropriately assign CPU processing power to TCP connections as well as the socket buffer of the sender hosts, in order to provide more effective and fair service to the connections.

4 Concluding Remarks

In this paper, we have discussed the fairness issues in congestion control mechanism of TCP. The router support is also presented to establish the fair service among TCP connections. The fairness issue arising at endhosts is also described. One important point that we should note here is that our simulation results shown in this paper only exhibit one aspect of the mechanisms. The results would be changed according to the simulation setting. An adequate method for performance evaluation suitable to the Internet environment is also an important research issue in the field [43].

We now summarize the fairness issues in TCP.

- Because TCP has already been used widely in the current Internet, it is difficult for the new protocol to be accepted if it is not downward–compatibleto the existing TCP.
- Even if a novel congestion control mechanism is upward–compatible, a migration path from the existing mechanism should also be considered. Otherwise, it is difficult for the new mechanism to be widely accepted. TCP Vegas is such an example as described in this paper.
- Fair share of network resources among multiple heterogeneous connections is one of key issues especially for the commercial use of the Internet.
- To improve the fairness among connections, the routers should be equipped with some mechanism to achieve the fairness. In that case, the scalability issue should be carefully taken into account.
- A compromise between the fairness degree and scheduling complexity exists. For the edge routers, DRR or some other mechanisms to support per–flow queueing seems to be a good choice. Stateless scheduling would be necessary for the core routers.
- Fair allocation of resources of the endhost is also inevitable especially for the busy server, where endhost resources is likely to be short.

Note that above statements are the authors' opinions and some of them may not be widely accepted.

References

[1] M. W. Garrett, "A service architecture for ATM: From applications to scheduling," *IEEE Network Magazine*, vol. 10, pp. 6–14, May/June 1996.

[2] A. S. Tanenbaum, *Computer Networks*. Prentice Hall, 1996.

[3] K. Fall and S. Floyd, "Simulation–based comparisons of Tahoe, Reno, and SACK TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 5–21, July 1996.

[4] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, Massachusetts: Addison-Wesley, 1994.

[5] M. Perloff and K. Reiss, "Improvements to TCP performance," *Communications of ACM*, vol. 38, pp. 90–100, February 1995.

[6] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme of TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 270–280, October 1996.

[7] N. `ns` (version 2) available from `http://www-mash.cs.berkeley.edu/ns/`.

[8] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of the congestion control mechanism of TCP," in *Proceedings of IEEE INFOCOM'99*, pp. 1329–1336, March 1999.

[9] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Journal of Computer Networks and ISDN Systems*, pp. 1–14, June 1989.

[10] P. Hurley, J.-Y. L. Boudec, and P. Thiran, "A note on the fairness of additive increase and multiplicative decrease," in *Proceedings of 16th International Teletraffic Congress*, pp. 336–350, June 1999.

[11] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, pp. 397–413, August 1992.

[12] L. Zhang, D. D. Clark, and Scott, "Oscillating behavior of network traffic: A case study simulation," *Intetnetworking: Research and Experience*, vol. 1, pp. 101–112, 1990.

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

[14] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, pp. 8–23, October 1994.

[15] S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," *RFC 2481*, January 1999.

[16] T. Hamann and J. Walrand, "A new fair window algorithm for ECN capable TCP (new-ECN)," in *Proceedings of IEEE INFOCOM 2000*, March 2000.

[17] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM'94*, pp. 24–35, October 1994.

[18] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Jounal on Selected Areas in Communications*, vol. 13, pp. 1465–1480, October 1995.

[19] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas revisited," in *Proceedings of IEEE INFOCOM 2000*, March 2000.

[20] K. Kurata, G. Hasegawa, and M. Murata, "Fairness comparisons between TCP Reno and TCP Vegas for future deployment of TCP Vegas," to be presented at *INET 2000*, July 2000.

[21] W. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meister, H. Rudin, and R. Williamson, "A survey of light-weight transport protocols for high-speed networks," *IEEE Communications Magazine*, vol. 38, November 1990.

[22] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proceedings of ACM SIGCOMM'89*, vol. 19, pp. 1–12, October 1989.

[23] J. Nagle, "On packet switches with infinite storage," in *IEEE Transaction on Communications*, vol. 35, April 1987.

[24] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 375–385, June 1996.

[25] "Cisco 12016 Gigabit Switch Router," available at `http://www.cisco.com/warp/public/cc/cisco/mkt/core/12000/12016/`

[26] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara, "Comparisons of packet scheduling algorithms for fair service among connections on the Internet," in *Proceedings of IEEE INFOCOM 2000*, March 2000.

[27] D. Lin and R. Morris, "Dynamics of random early detection," in *Proceedings of ACM SIGCOMM'97*, pp. 127–137, October 1997.

[28] T. J. Ott, T. V. Lakshman, and L. Wong, "SRED: Stabilized RED," in *Proceedings of IEEE INFOCOM'99*, March 1999.

[29] I. Stoica, S. Schenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately bandwidth allocations in high speed networks," in *Proceedings of ACM SIGCOMM'98*, pp. 118–130, September 1998.

[30] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 6, August 1999.

[31] "Real.com," available at `http://www.real.com/`.

[32] "The TCP-friendly Website," available at `http://www.psc.edu/networking/tcp_friendly.html`

[33] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, pp. 67–82, July 1997.

[34] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proceedings of ACM SIGCOMM'98*, pp. 303–314, August 1998.

[35] W. T. Tan and A. Zakhor, "Real-time Internet video using error resilient scalable compression ans TCP-friendly transport protocol," *IEEE Transactions on Multimedia*, vol. 1, pp. 172–186, June 1999.

[36] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "A model based TCP-friendly rate control protocol," in *Proceedings of ACM SIGCOMM'98*, September 1998.

[37] N. Wakamiya, M. Murata, and H. Miyahara, "On TCP-friendly video transfer with consideration on application-level QoS," to be presented at *IEEE International Conference of Multimedia & EXPO*, July 2000.

[38] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications Magazine*, pp. 23–29, June 1989.

[39] C. Partridge, "How slow is one gigabit per second?," *ACM Computer Communications Review*, vol. 20, pp. 44–53, January 1990.

[40] P. Druschel and L. L. Peterson, "Fbufs: a high-bandwidth cross-domain transfer facility," in *Proceedings of the Fourteenth ACM symposium on Operating Systems Principles*, pp. 189–202, December 1993.

[41] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," in *Proceedings of ACM SIGCOMM'98*, pp. 315–323, August 1998.

[42] T. Matsuo, G. Hasegawa, and M. Murata, "Scalable automatic buffer tuning to provide high performance and fair service for TCP connections," to be presented at *INET 2000*, July 2000.

[43] V. Paxson and S. Floyd, "Why we don't know how to simulate the Internet," in *Proc. 1997 Winter Simulation Conference*, December 1997.