University of California Los Angeles
Computer Science Department

Technical Report No. 990012

Generalized Window Advertising for TCP Congestion Control *

M. Gerla[1], R. Lo Cigno[2], S. Mascolo[3], W. Weng[1]

[1] Computer Science Department – Boelter Hall – UCLA
405 Hilgard Ave., Los Angeles, CA, 90024 – USA
e-mail: [gerla,weng]@cs.ucla.edu

[2] Dipartimento di Elettronica – Politecnico di Torino
Corso Duca Degli Abruzzi, 24 – 10129 Torino – Italy
e-mail: locigno@polito.it

[3] Dipartimento di Elettrotecnica ed Elettronica – Politecnico di Bari
Via Orabona, 4 – 70125 Bari - Italy
e-mail: mascolo@poliba.it

Abstract

Congestion in the Internet is becoming a major cause of network performance degradation. The Generalized Window Advertising (GWA) scheme proposed in this paper is a new approach for enhancing the congestion control properties of TCP. GWA requires only minor modifications to the existing protocol stack and is completely backward compatible, allowing GWA-hosts to interact with non-GWA hosts without modifications.

GWA exploits the notion of congestion feedback from the network to end hosts. It is based on solid control theory results that guarantee performance and stable network operation. GWA is able to avoid window oscillations and the related fluctuations in offered load and network performance. This makes it more robust to sustained network overload due to a large number of connections competing for the same bottleneck, a situation where traditional TCP implementations fail to provide satisfactory performance.

GWA TCP is compared with traditional TCP, TCP with RED and also ECN using *ns-2* simulation. Results show that in most cases GWA-TCP outperforms the traditional schemes. In particular, when compared with ECN, it provides smoother network operation and increased fairness. The coexistence of GWA TCP with RED and ECN is also tested. The test shows that, besides being backward compatible, GWA TCP is also friendly to older TCP versions.

*This work was done while R Lo Cigno was on leave at the Computer Science Department of the University of California – Los Angeles under grant No. 203.15.8 issued by the Italian National Research Council (CNR).

1 Introduction

The boom of the Internet is stressing the entire IP protocol implementation, as the users grow in number, along with the demand for enhanced service and performance. The majority of data services in the Internet are based on TCP (Transport Control Protocol), with application ranging from bulk data transmission (like `FTP` file transfers), to interactive, delay sensitive services (like `Telnet` remote terminal), to web browsing. TCP provides a connection oriented, reliable communication channel to upper layers. It also provides the functions of receiver flow control and network congestion control. The TCP congestion control mechanism in standard implementations is based on a simple, wired network model, with very reliable nodes and links. Packet loss is mostly caused by buffer overflow and is thus taken as indication of congestion. Several studies have shown that this control approach may fail if losses in the network are due to cases other than congestion [1, 2, 3]. Moreover, the control does not work well if the congestion is caused by an excessive number of TCP connections allowed to compete for the same bottleneck [4].

Fortunately, routers of the next generation will be more intelligent than the ones used in traditional TCP implementations. They will be able to predict impending congestion because of better environment awareness. However, a major problem consists in correctly elaborating and conveying congestion information from routers to TCP sources. A related issue is to adapt TCP to accept new feedback from the routers, ensuring more stable performance and yet retaining complete backward compatibility. The first approach in the direction of router and TCP cooperation is the Explicit Congestion Notification (ECN) scheme proposed in [5, 6]. ECN is based on a binary feedback from routers to sources that triggers a window size reduction identical to that caused by the fast retransmit – fast recovery mechanism of TCP-Reno, yet without requiring the drop of a packet. ECN is effective in reducing packet loss. However, because of its binary feedback, it cannot avoid window and network oscillations. These can negatively affect the network performance.

This paper presents an approach to TCP window control which is also based on source and network cooperation. The approach is named Generalized Window Advertising (GWA). It is backward compatible, requiring a minimum upgrade to the TCP/IP protocol suite. It ensures 100% link utilization *and* zero packet loss under the same network conditions that allow 100% link utilization in traditional or ECN TCP implementations [7]. The latter, however, do not offer bounds on packet loss ratio.

Section 2 describes the GWA implementation and associated ns-2 simulation model; Section 3 analyzes algorithm performance based on control theory and fluid flow approximation; Section 4 discuss algorithm properties and model approximations; Section 5 presents numerical results and compares GWA with traditional approaches; Section 6 show results when GWA connections are mixed together with RED or ECN connections; Section 7 concludes the paper.

2 Generalized Window Advertising TCP

TCP [8, 9, 10] performs two different, though inter-related, tasks: i) end-to-end flow control to avoid overflowing the receiver buffer; ii) connection congestion control, adapting the amount of information injected in the network to the network congestion status. This latter function is based on *congestion recovery* (rather than prevention). TCP increases the traffic offered to the network until the network is forced to drop some packets; then it backs off to give the network time to relieve from congestion, and starts again to increase the offered load to force another congestion episode. The detailed description of the TCP protocol and its dynamic in various versions (Tahoe, Reno etc) is beyond the scope of this paper and can be found in the literature. Below, we will review the features common to all TCP versions. These are the only features needed for the GWA implementation.

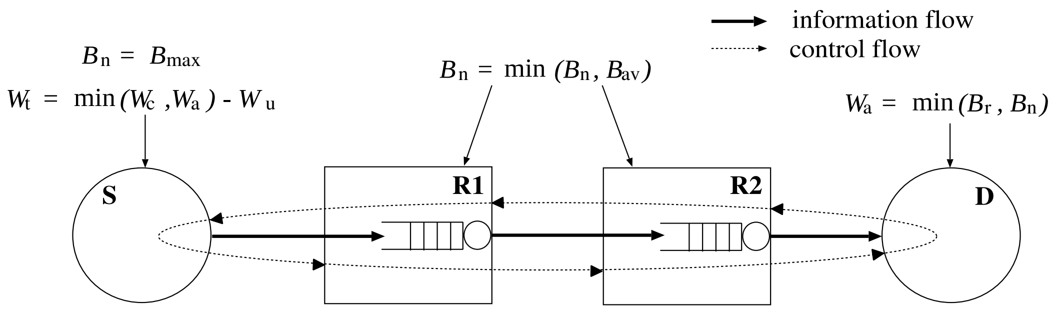

Figure 1: GWA algorithm deployment in a simple network

TCP flow control is enforced by means of the *advertised window* field W_a set by the receiver in the header of acknowledgment (ACK) segments. Congestion control is enforced by means of the *congestion window* W_c that is computed by the transmitter following a well defined algorithm (see [8] for details). Finally the amount of data W_t which TCP is allowed to transmit is computed as

$$W_t = \min\{W_c, W_a\} - W_u \tag{1}$$

W_u is the amount of outstanding data, i.e., the part of the sliding transmission window that is filled with data already sent, but not yet acknowledged.

The key idea of GWA is to "generalize" the interpretation and function of W_a, using it to convey *both* the receiver available buffer space B_r and the network congestion status B_n. Since network congestion status can be expressed as buffer space available in routers, B_n is defined as the minimum buffer space available in routers along the TCP path.

IP routers cannot access or modify the field W_a since this would be a layer violation. Moreover W_a is protected by TCP CRC. Changing W_a requires that re-computation of the CRC, which is not feasible in a high speed router. The information must be delivered to the TCP transmitter or receiver at the IP level, leaving the task of inter-layer communication to the end hosts.

The information can be sent either to the transmitter (backward notification) or to the receiver (forward notification), with minor implications on performance during transients. Conveying the information to the receiver yields a simpler solution. In fact, the receiver simply assigns

$$W_a = \min\{B_r, B_n\} \tag{2}$$

If IPv6 is used, B_n can be a 2-bytes optional field within the packet header. In IPv4, the information can be conveyed to the receiver by means of ICMP packets. For the sake of simplicity the following description assumes that IPv6 is used by all network hosts. Fig. 1 shows how the algorithm is deployed in the network. It features a TCP transmitter (or source) **S**, two routers **R1** and **R2** and a receiver (or destination) **D**.

The transmitter, before transmitting the packet, sets $B_n = B_{\max}$, where $B_{\max}$ is a predefined target value. If the window scale option is set by the TCP connections, the same scaling factor applies to B_n.

Routers along the connection modify B_n according to the buffer space B_{av} available for that connection, without ever increasing its value. When the packet arrives at the destination it contains the minimum buffer space along the followed path. The scheme works even if not all the packets follow the same route; however the analysis carried out in Section 3 no longer holds. The receiver copies B_n from the IP to the TCP level, and TCP implements Eq (2).

B_n is the buffer space that the network can provide to the connection. An implementation with per-flow queuing is an ideal match for GWA scheme. However also FCFS on the aggregate queue can be used. With aggregate FCFS queuing it is necessary to estimate the number of active flows, in order to account for the buffer space available to each. This can be a source of unfairness, but generally it does not impair the overall performance, as shown in Section 5.

2.1 Implementation in *ns-2*

GWA TCP was implemented in the *ns-2* simulator[1], developed at UC Berkeley within the VINT project[2].

The following extensions/modifications to the basic *ns-2* infrastructure were made.

- TCP dynamic window advertisement;
- Introduction of the B_n optional field in the IPv6 header;
- Implementation of Eq. (2) in TCP receivers;
- Implementation of IP routers which advertise B_n.

GWA IP routers have been implemented both with per-flow queuing and with FCFS queuing. Per-flow queuing is associated with a simple Round Robin (RR) serving discipline.

Routers with per-flow RR queuing discipline. Let B_t be the total buffer size and n_f be the number of active flows. Let i identify the flow and $q_i(t)$ the queue length of flow i at time t. Then the GWA scheme simply consists in assigning to the field $B_{n,i}$ of each packet the value

$$B_{n,i} = \min\left\{ B_{n,i}, \left\lceil \frac{B_t}{n_f(t)} - q_i(t) \right\rceil \right\}$$

where $B_{n,i}$ is assigned at transmission time t. $n_f(t)$ is the value of n_f at time t and is measured by the router. A simple measurement (which ensures reasonable performance if the number of flows does not vary too quickly in time) is a geometrically decaying time average:

$$n_f(t) = \alpha n_f(t-\tau) + (1-\alpha) n_f^o(t) \tag{3}$$

where τ is an appropriate measure interval, $\alpha < 1$ is the decaying factor and $n_f^o(t)$ is the number of active flows observed during the interval $[t-\tau, t]$. Flow identification can be obtained either through the optional flow label, or by assuming a different flow for each association of source address and destination address. Notice that how flows are identified does not affect the performance of the control. The optimal choices of α and τ depend on link speed, buffer size, and traffic characteristics. A conservative choice, i.e., both α and τ quite large, say $\alpha = 0.9$ and $\tau = 0.1$ s, generally leads to an overestimate the number of active flows, hence reducing the advertised buffer size. When the buffer is large enough, this conservative estimate has minor effect on performance.

Routers with FCFS queuing discipline. In FCFS queuing, the router does not keep track of the individual queues $q_i(t)$. Taking this into account and using the same notations as above, the GWA scheme uses the following estimate

$$B_{n,i} = \min\left\{ B_{n,i}, \left\lceil \frac{B_t - q_t(t)}{n_f(t)} \right\rceil \right\} \tag{4}$$

[1] *ns-2* is the last version of the simulator; all necessary information and the software can be retrieved at the URL `http://www-mash.cs.berkeley.edu/ns/ns.html`; the additional software for GWA can be obtained from the authors

[2] See the URL `http://netweb.usc.edu/vint` for additional information on VINT

where $q_t(t)$ is the overall queue length at time t; $n_f(t)$ is estimated through Eq. (3).

Though difficult to prove theoretically, this simpler scheme still provides good aggregate performance. However, there is no way to enforce fairness among flows with different round trip times. It can be expected that the throughput of individual connections is inversely proportional to their round trip time.

3 The Algorithm from a Control Theory Perspective

The theoretical analysis is restricted to the case with per-flow queuing and RR serving discipline. A fluid flow approximation is assumed together with static routing so that all packets follow the same route from the source to the destination. The algorithm is designed exploiting the *Smith predictor* [11], which is a control technique for time-delay systems. The control algorithm is first derived assuming a rate based control scheme. Then, it is modified to fit the TCP window model.

Let $r_i(t)$ be the transmission rate of i-th connection at the source and $c_{i,j}(t)$ be the service rate given to the i-th TCP connection by router j along the connection path. $c_{i,j}(t)$ is generally an unknown, bounded function. It takes into account possible higher priority traffic, as well as the other TCP flows, that interact with the considered one through the Round Robin scheduler.

If buffer capacity is large enough so that loss probability can be assumed to be zero[3], at any given router along the path we have:

$$q_{i,j}(t) = \int_0^t \left[r_i(\tau - d'_{i,j}) - c_{i,j}(\tau) \right] d\tau$$

where $d'_{i,j}$ is the delay from source i to router j. Notice that $q_{i,j}(\cdot)$, $r_i(\cdot)$ and $c_{i,j}(\cdot)$ are subject to be non negative.

Each router j stamps the available buffer $(q_i^l - q_{i,j}(t))$ in the IP header field B_n if it is smaller than the one already stored in B_n. The minimum value reaches the receiver, which, by implementing Eq. (2), reflects it back to the source.

The value $(q_i^l - q_{i,j}(t))$ arrives at the source with delay $d''_{i,j}$. Notice that $d'_{i,j} + d''_{i,j} = d_i$, where d_i is the connection round trip time, regardless of where the bottleneck j is located along the path. Since only the minimum buffer space is propagated at any time, the index j will be omitted from now on without loss of generality.

Fig. 2 depicts the block diagram of the system, where $\frac{1}{s}$ is the Laplace transform of the integrator represented by the router buffer, $G_c(s)$ is the transfer function of the controller. The block $e^{-sd'_i}$ models the delay from the source to the congested router, while the block $e^{-sd''_i}$ models the delay from the congested router to the source.

The goal is to find a controller $G_c(s)$, which stabilizes the network operation, granting no losses and full link utilization with persistent sources. Mathematically this can be expressed as (see [12] for a more detailed explanation):

$$q_i(t) \leq q_i^l \quad \forall t \geq 0 \tag{5}$$

$$q_i(t) \geq 0 \quad \forall t \geq T_t \geq d_i \tag{6}$$

Eq. (5) states that the buffer never drops packets provided that $\sum_i q_i^l \leq B$ where B is the buffer dimension; Eq. (6) guarantees the full utilization of the bandwidth assigned to flow i, after a transient time T_t, which is clearly larger than the connection round trip time d_i. Indeed Eq. (6)

[3] As we show later, the control algorithm itself guarantees that no packets are lost. So, the equation in question is satisfied for any finite buffer capacity.

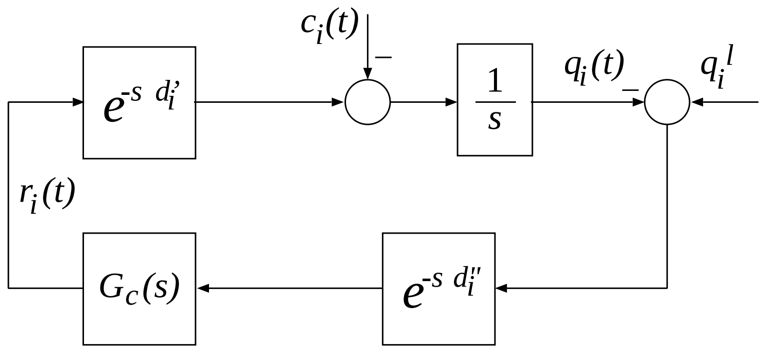

Figure 2: Block diagram of the closed loop system for TCP control

guarantees the full utilization of the link if any work conserving serving discipline, like RR or FCFS, is used.

In [11, 12] it is shown that a simple controller that satisfies Eq. (5) and (6) with $T_t = d_i$ is the following:

$$G_c(s) = \frac{k}{1 + \frac{k}{s}\left(1 - e^{-sd_i}\right)} \tag{7}$$

Eq. (7) defines the controller in the Laplace domain. In the time domain, the equation definition is:

$$r_i(t) = k\left[q_i^l - q_i(t - d_i'') - \int_{t-d_i}^{t} r_i(\tau)d\tau\right] \tag{8}$$

Eq. (8) tells us that the computed rate at the source is proportional to the available buffer space $[q_i^l - q_i(t)]$, minus the amount of information that has been sent during the last round trip time, that is $\int_{t-d_i}^{t} r_i(\tau)d\tau$. The time constant of the system is $\tau_c = 1/k$. Thus, the transient dynamics are practically exhausted after $4\tau_c$. Setting $k = \dfrac{1}{d_i}$ provides monotonic convergence in 4 round trip times. Notice that the convergence to steady state is monotonic (without overshoots) since the closed loop dynamic is first order (see [12] for the explanation). This property is important to prove loss free operation, without buffer overflow.

Control Eq. (8) can be transformed from rate to window formulation by multiplying both sides of Eq. (8) by d_i. Notice that using $k \neq \dfrac{1}{d_i}$ would make the conversion to a window form of the control less straightforward.

Recall that $B_n \triangleq q_i^l - q_i(t)$ and $W_u = \int_{t-d_i}^{t} r_i(\tau)d\tau$. B_n is directly measured by the routers and forwarded to the TCP destination. W_u, the number of outstanding packets (bytes) is known by the TCP source. Thus, the implementation of the GWA scheme through Eqs. (1) and (2) is straightforward.

4 Properties of the GWA Algorithm and Modeling Approximations

GWA TCP as outlined in Section 3 has several appealing properties. GWA *always* ensures zero loss probability, regardless of the statistical fluctuations of $c_{i,j}(t)$. This means that there are no losses even if somewhere along the path the bandwidth goes abruptly to zero. This condition holds independently from the buffer size, which only determines the link utilization.

The link utilization reaches 100% after one round trip time, provided that the buffer size at the congested link is larger than the amount of information that can be stored "in flight" in the network. For any router in the network the minimum buffer $B^{\min}$ is expressed as:

$$B^{\min} \geq \sum_{i=1}^{N} c_i^{\max} \cdot d_i^{\max} < c_l \cdot d^{+} \tag{9}$$

where N is the number of connection; $c_i^{\max}$ and $d_i^{\max}$ are upper bounds on the values observed on i; c_l is the link capacity and d^{+} is an upper bound on $d_i^{\max}$ $\forall i$, i.e., the maximum of the round trip delays over all connections. Note that the first inequality of Eq. 9 states the well known condition for full link utilization of traditional TCP implementations. Thus, GWA TCP does not require additional network resources to work properly. The second inequality gives an upper bound that can be used for router buffer dimensioning. It is worth noticing that this bound depends only on the local link speed and not on the speed of other links in the network. Hence, the buffers in routers supporting GWA TCP can be dimensioned with reference to local parameters only.

The above properties can be rigorously verified in our model. However, the real system does not correspond exactly to the model. There are features in the real system which require a reinterpretation of these properties, namely: i) the discrete (instead of continuous) behaviour; ii) the effect of transient TCP dynamics[4].

TCP behavior is intrinsically discrete, due to the packetized transmission of information. The difference between the fluid flow model and the real system can be expressed as sampling and quantization noise. Because of this, it can be expected that the real system needs a buffer larger than what predicted by theory in order to reach 100% link utilization. Besides, at steady state, the behavior can be affected by small ripples due to quantization: both the TCP transmission window and the network buffer occupancy may present small oscillations around the equilibrium point (while the theory predicts monotonic convergence).

Regarding slow start and congestion avoidance, both of these features have the effect of reducing the transmission window size (with respect to what is advertised by the network) during the initial phase of a connection. Since GWA guarantees zero loss, slow start and congestion avoidance will not be triggered during the data phase (except for infrequent losses due to link errors[5]) and thus will not play a major role. One effect of this "reduced offered load" is that full utilization is not reached after one round trip time; however this is a negligible performance loss.

5 Results and Comparison with Traditional TCP Implementations

The performance predicted by theoretical analysis is very promising; however both the modeling simplifying assumptions and the implementation "shortcuts" are potential sources of performance degradation. Of particular concern is the interaction between the GWA scheme and the other TCP congestion control algorithms: timeouts, slow start, congestion avoidance, etc., that must be preserved for backward compatibility and robustness.

[4] Slow Start, Congestion Avoidance, Delayed ACKs and all the other protocol characteristics are in general nonlinear, and make it difficult to mathematically analyze the closed loop dynamics. Notice that GWA properties ensure that, apart form the initial transient, Slow Start and Congestion Avoidance phases should be triggered very rarely.

[5] We assume a wired network model, with negligible loss due to link errors. If the path includes wireless segments with interference and error loss which are not locally protected, TCP GWA proper operation requires that Slow Start and Congestion Avoidance mechanisms be turned off.

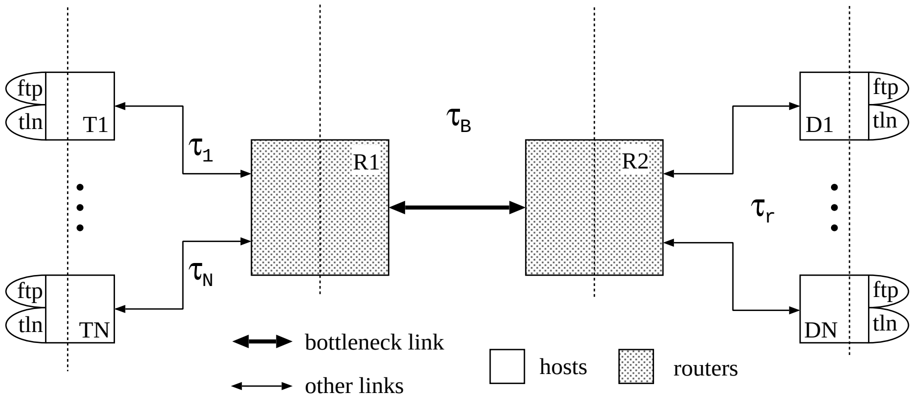

Figure 3: Single bottleneck topology used for performance comparison

This Section compares the performance of 5 different TCP implementations and network congestion control schemes, running on the simple network with a single bottleneck depicted in Fig. 3. The 5 TCP implementations are:

- TCP-Reno and standard FCFS queuing IP routers, in the sequel named *Drop-Tail*;
- TCP-Reno and RED IP routers [13], named *RED*;
- TCP-Reno with ECN (Explicit Congestion Notification) capabilities [5, 6] and tagging RED IP routers, named *ECN*;
- TCP-Reno with GWA capabilities and per-flow queuing, RR IP routers (see Section 2), named *GWA-RR*;
- TCP-Reno with GWA capabilities and FCFS queuing IP routers (see Section 2), named *GWA-FCFS*.

Simulations are run both in LAN and WAN scenarios, with Telnet-like and greedy FTP applications competing for the bottleneck resources. The performance measures taken into account are the FTP throughput at the receiver (i.e., the "goodput"), the packet loss probability and the transfer delay of Telnet packets. Telnet throughput is negligible. Thus it is not measured.

With reference to Fig. 3, simulation experiments are run with N hosts, each one running both an FTP and a Telnet application. All links are bidirectional, but the connections are all unidirectional, so that on the backward path the load is due only to ACKs and is very light. Referring again to Fig. 3, the propagation delay between source and destination i is $\tau = \tau_i + \tau_B + \tau_r$, and the round trip time experienced by the transmitter is $d_i = 2\tau + \tau_q$, where τ_q accounts for possible queuing delays both on the forward and the backward path. The TCP MSS (Maximum Segment Size) is set to 1000 bytes. The timer granularity (i.e., the TCP 'tick') is 0.1 s, hence it is somewhat "faster" than present day implementations that use values comprised between 0.2 and 0.5 s.

Telnet applications generate MSS packets with average interarrival time equal to 0.1 s. This is an unrealistic assumption for a Telnet connection, but can be representative of other time-sensitive applications. Indeed running simulations with Telnet connections offering a much lower load would result in a set of results whose statistical properties are not reliable.

The tuning of RED router parameters proved to be a non-trivial task. Parameter values must be adjusted to each specific network configuration in order to get satisfactory performance. In our experiments, the lower and upper thresholds are set to 1/12-th and 3/12-th the buffer size,

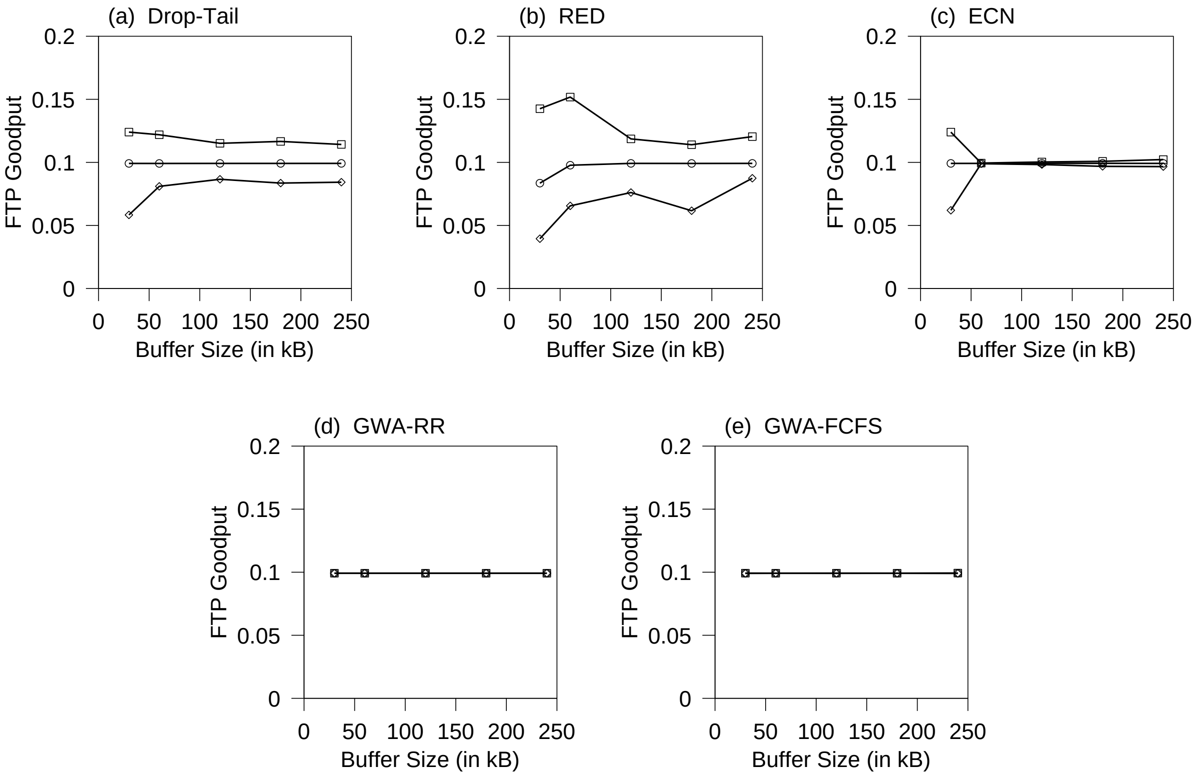

Figure 4: Receiver throughput for the 5 considered implementations versus the router buffer size in LAN environment

respectively. The drop probability of RED increases linearly (from 0 to 0.1) with average queue length between the lower and the upper thresholds. RED drops packets randomly from the queue. Above the upper threshold, RED drops an incoming packet with probability =1. For ECN, routers mark (instead of drop) the packets when average queue length is between the two thresholds. Packets arriving, when average queue length is above the upper threshold, are marked rather than dropped[6]. The other parameters are the same as in the RED experiments.

5.1 LAN Environment

LANs are characterized by negligible propagation delay (i.e., same order or less than packet transmission time). With reference to Fig. 3, LAN experiments are run with $N = 10$ hosts, $\tau_B = 40\,\mu$s, $\tau_r = 30\,\mu$s and $\tau_i = 30\,\mu$s $\forall i$. All link speeds are 100 Mbit/s. The receiver's available buffer space is 64 kbit/s. Simulations are run for 25 s. Results (except for transients) are computed after discarding the first 5 s, thus excluding the network transient behavior.

Fig. 4 reports the throughput measured at the receiver after discarding all duplicated packets. The overall throughput is normalized to 1, so that the fair share of each connection is 0.1. Each plot has three curves: the middle one is the average among all connections, while the upper and the lower ones are the maximum and the minimum, respectively. The spreading between these two latter curves is indicative of the "unfairness" of the network. The throughput is generally quite

[6] In the original *ns-2* code we downloaded, the ECN routers mark packets when the buffer is between the thresholds, but drop them when it is above the higher threshold. We were not able to obtain satisfactory results with this layout, while we obtained good results by marking instead of dropping. Good results can probably be obtained also with further adjustments of RED thresholds an drop probability P_d for ECN (which we did not attempt in our study).

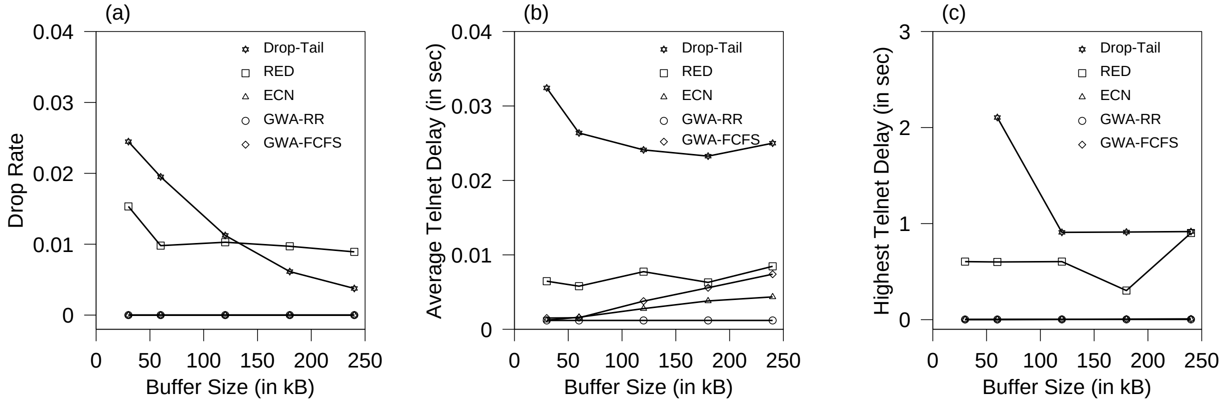

Figure 5: (a) Packet drop probability, (b) average delay of Telnet packets, and (c) maximum delay of Telnet packets versus the buffer size in LAN environment. For Drop-Tail case, the maximum delay is about 12 sec when buffer size is 30 kB.

good with any scheme. Regarding fairness, on the other hand, all traditional implementations show some degree of unfairness. The GWA algorithm, both with RR and with FCFS serving discipline, ensures perfect fairness and full utilization even with small router buffer size. Fairness is also achieved with FCFS since in LANs the difference in propagation delays is negligible.

Fig. 5 reports the packet drop rate (plot a) together with the average (plot b) and maximum (plot c) delay experienced by Telnet packets. Each curve corresponds to a different implementation. At steady state, GWA TCP and ECN yield zero losses. In fact, these three curves are overlapped in Fig. 5(a). Drop-Tail and RED show losses that decrease with increasing buffer size as expected. The behavior of Telnet packet delays can be explained by considering that the delay measured by the receiver includes retransmissions. This is the reason why TCP implementations that do not avoid loss show very high values of the maximum Telnet transfer delay. With GWA-RR, the delay experienced by Telnet packets is merely the propagation delay, since the individual queues associated with Telnet connections will always be empty. With GWA-FCFS, both the average and the maximum delay increase with buffer size, since the aggregate queue stabilizes to a higher value when the buffer is large. ECN Telnet delays are also very low. Moreover, from Fig. 5(c) we note that GWA and ECN maximum delays are in the millisecond range, while Drop Tail and RED maximum delays are in the range of seconds.

Fig. 6 plots the transient behavior of an FTP connection window (left plots) and the bottleneck buffer (right plots) during the first 3 seconds of the experiment, for router buffer size = 120 kbytes. The window size is measured and plotted every time a packet is sent and the queue length is averaged over 0.01 second intervals. Small oscillations in GWA are due to quantization effects (residual buffer space is not a multiple of TCP packet size). The inserts in the plots show a blow-up of the first 0.1 seconds, highlighting the initial transient behavior of the network, and offering an insight that is not available from steady state results. In particular, these results clearly reveal the interaction of TCP with the network congestion reaction policy.

Drop-Tail and RED exhibit oscillatory behavior as expected. Somewhat surprising, however, is the fact that RED yields a very large number of timeouts. We note 8 timeouts in the first 3 seconds of transmission for the FTP under study. Only a timeout can halt transmissions for more than 0.2 s. We note that ECN does not suffer packet drops at steady state. However, it incurs multiple drops in a single window followed by timeout at the very beginning of the session. This has no impact on steady state FTP throughput. However, it may affect the performance of applications exchanging small files.

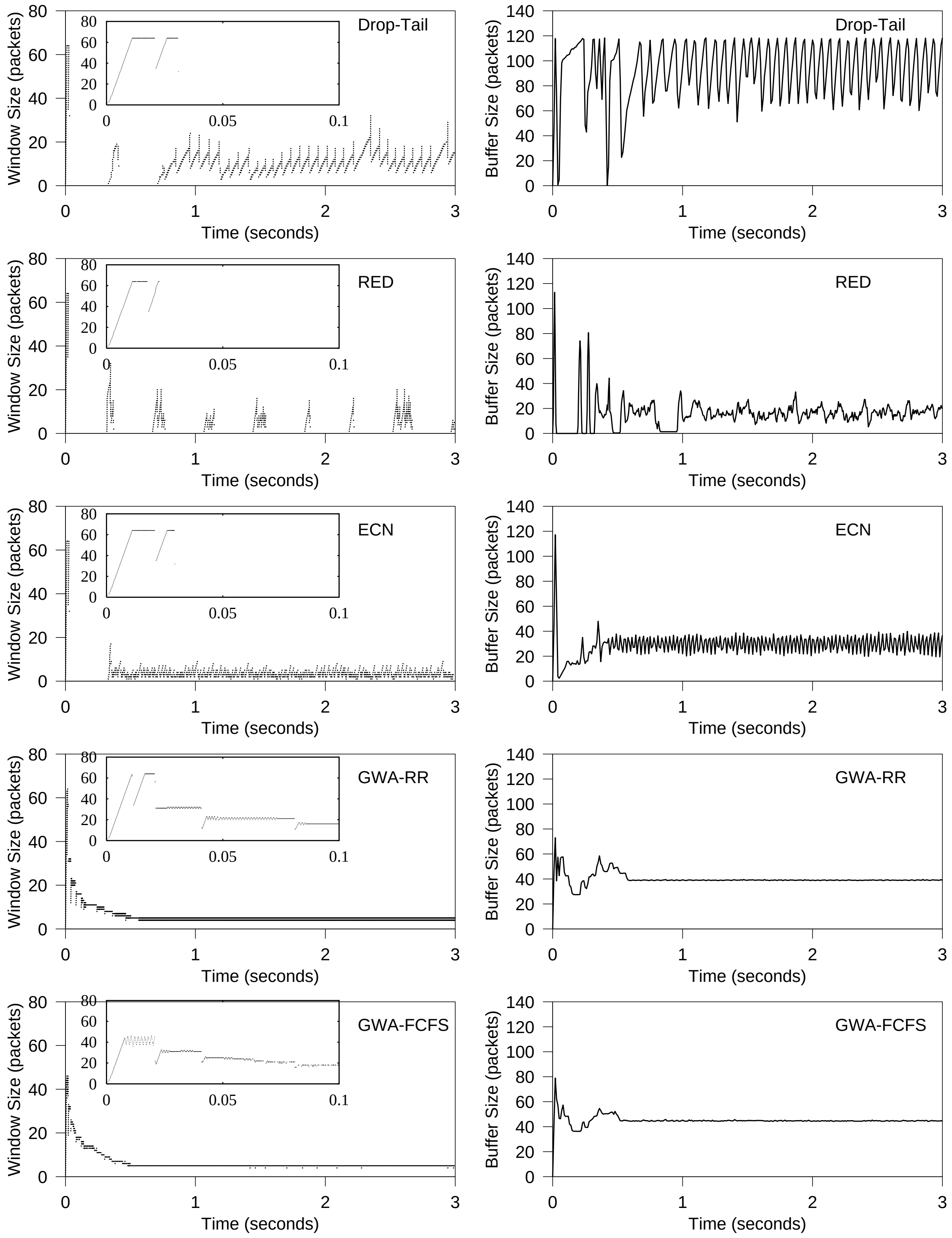

Figure 6: Window dynamic (left column) and bottleneck buffer dynamic (right column) versus time in LAN environment

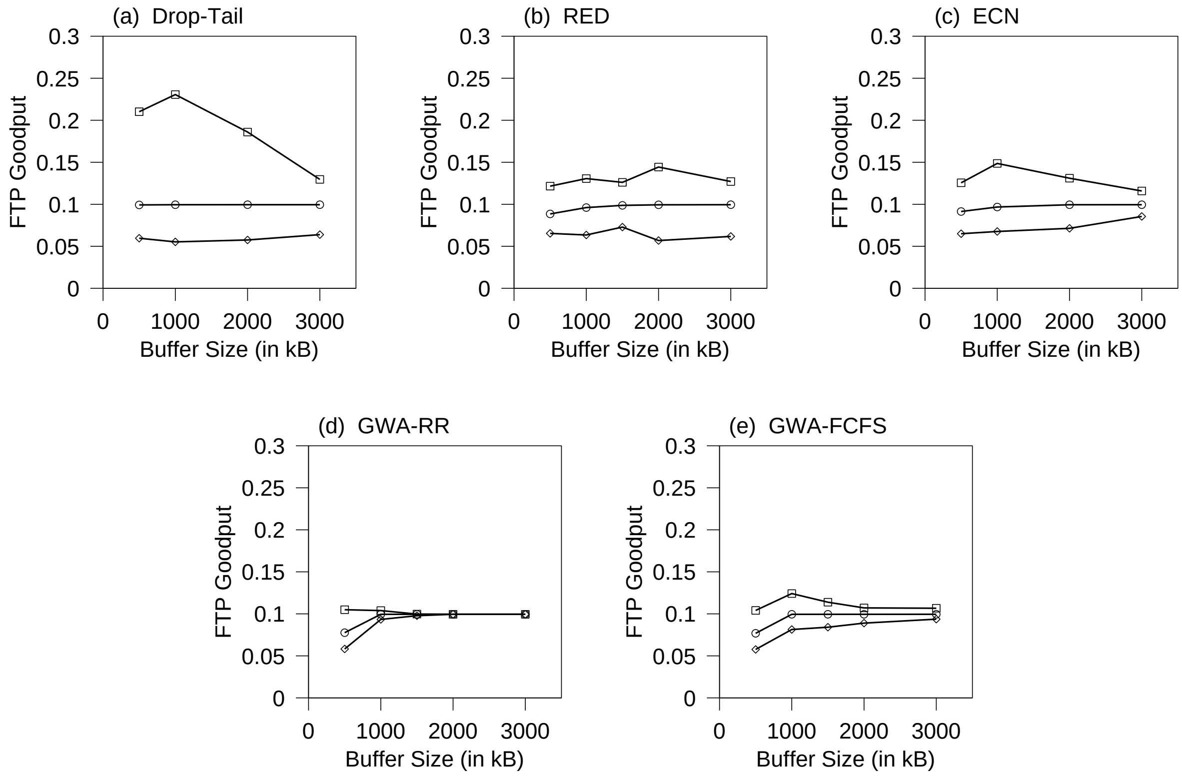

Figure 7: Receiver throughput for the 5 considered implementations versus the router buffer size in WAN environment

GWA schemes exhibit very stable behavior, without packet drops even during the initial transient. Steady state is achieved within 0.5 s. This value is much larger than the value indicated by fluid flow analysis. The reason lies essentially in the difficulty of correctly estimating the exact number of active flows during the early transmission phase. Simulation experiments performed assigning the exact number of active flows 'a priori' show a shorter, smoother transient. However, they are unrealistic and are not included here.

Considering now the right column of Fig. 6 (buffer size behavior), we notice that buffer occupancy at steady state is quite similar in both GWA-RR and GWA-FCFS. If only FTP connections were present, the steady state level would be about 1/2 of the buffer size. This is explained by the fact that the stable operating point for each TCP connection is when the advertised window is equal to the data transmitted but not acknowledged. So, every time a packet is ACKed, a new one is transmitted. Since the propagation delay is negligible, all the non ACKed packets are in the bottleneck queue. The equilibrium is reached when the buffer is half full. The plots in Fig. 6 show a smaller buffer occupancy. The reason is that Telnet connections offer less traffic than the maximum allowed. Telnet connections which are detected as being active and are counted in n_f reduce the buffer occupancy.

5.2 WAN Environment

Wide Area Network (WAN) behavior is influenced, if not dominated, by the propagation delay. Difference in propagation delay between different connections may lead to unfairness: it is well known that TCP throughput is inversely proportional to the round trip time. When competing for shared resources, longer connections are penalized. With reference to Fig. 3, WAN experiments are

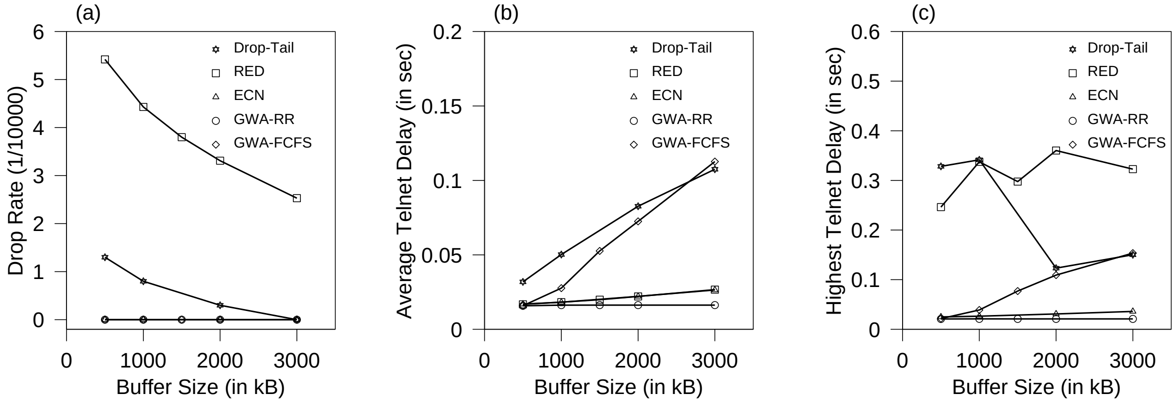

Figure 8: (a) Packet drop probability, (b) average delay of Telnet packets, and (c) maximum delay of Telnet packets versus the buffer size in WAN environment

run by setting the bottleneck link speed to 155 Mbit/s (the speed of the other links is 100 Mbit/s). $\tau_B = \tau_r = 5$ ms. The other propagation delays (τ_i in Fig. 3) are uniformly distributed between 1 an 10 ms. The window scale option for TCP window is activated, setting the maximum window size to 512 kbytes to allow full link utilization.

The average number of MSS packets (or ACKs) that are outstanding in the network at any time is about 450. Both GWA theory and heuristic considerations (see [7] for example) tell that the minimum buffer space for satisfactory network operation is 500 kbytes.

Fig. 7 shows the average, maximum and minimum throughput achieved by FTP connections for the 5 TCP implementations. Experiments last 25 s, and results exclude the first 5 s. The average throughput performance over all connections is generally satisfactory for all implementations. Perfect fairness is achieved only with GWA-RR when the buffer is large enough to support full link utilization. Notice that the required buffer size is consistent with theory. The fairness improvement as the buffer size increases for GWA-FCFS is explained by the increased buffer occupancy and therefore increased round trip time (RTT) seen by TCP sources. Recall that the throughput difference is proportional to the *relative* RTT difference among connections. ECN fairness behavior is due to the same reason, although less evident due to the buffer oscillations (see Fig.9).

The throughput performance is consistent with the packet loss and delay results reported in Fig. 8. The loss-free properties of GWA and ECN implementations lead to good delay performance of real time applications like Telnet. This is a definite advantage of both GWA and ECN algorithms. Note, however, that while GWA *guarantees* loss-free operation, the same can not be said for ECN. Further experimentation with a richer set of topologies and traffic scenarios, is needed to assess the results. Due to the large buffer size and propagation delays, the drop rate for Drop-Tail and RED is quite low in this case, never exceeding 10^{-3}.

Fig. 9 show the window and buffer dynamic behavior in a WAN environment for 2 Mbyte router buffer. GWA transient behavior is quite good, although a high frequency ripple is noted, especially with the RR discipline. This is due to measurement errors and jitters, both in n_f and in the available buffer space. For this reason the RR implementation is more "noisy", since the buffer measure is not averaged over the connections. In addition, as discussed in Section 4, the discrete nature of the system can cause small oscillations that add up to those due to measurement errors.

Drop-Tail, RED and ECN behave as expected. It is noticed that there are fewer timeouts here than in the LAN scenario. This is due to the fact that here packet drop rate for RED and Drop Tail is less than 10^{-3}, while it was 10^{-2} in the LAN experiment.

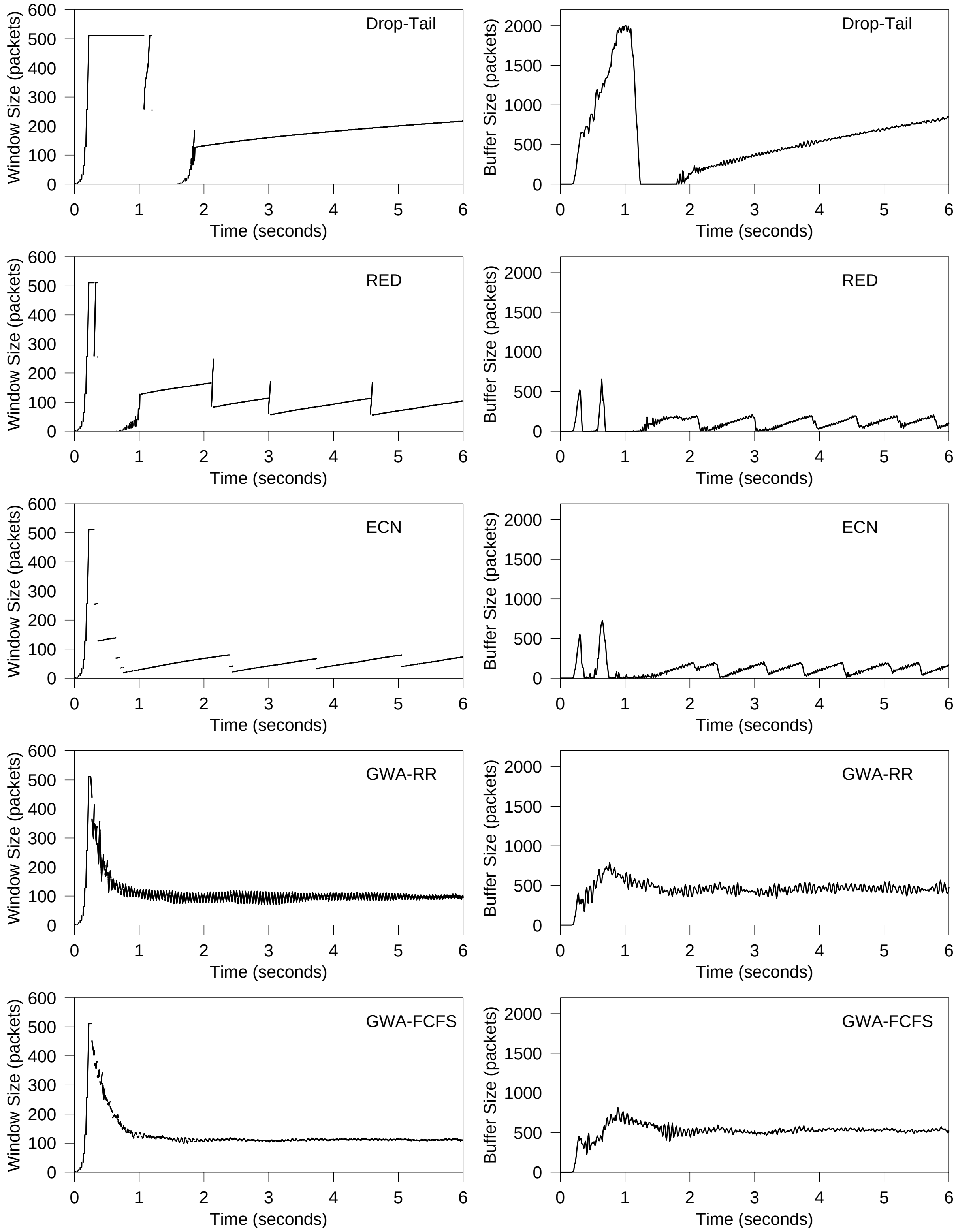

Figure 9: Window dynamic (left column) and bottleneck buffer dynamic (right column) versus time in WAN environment

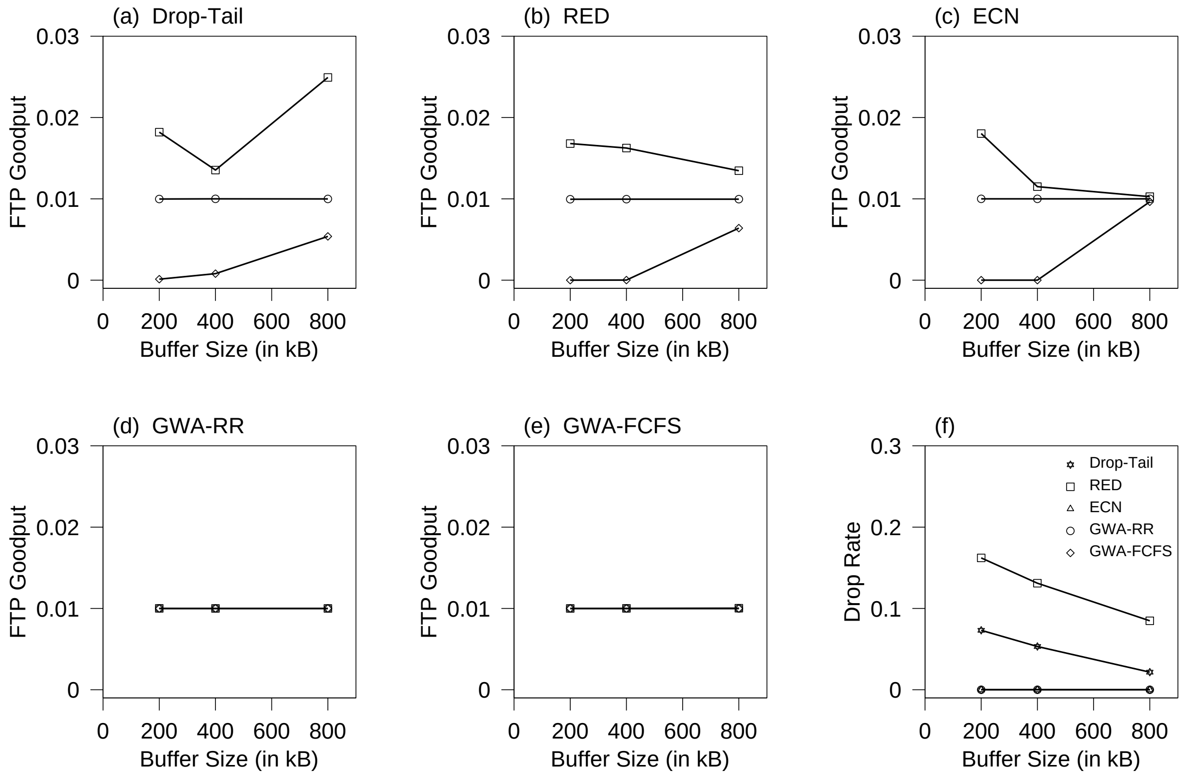

Figure 10: Throughput and loss probability with Many Flows

5.3 Behavior with Many Flows

One of the concerns in traditional TCP implementations is the degradation of performance when the number of connections competing for the same bottleneck becomes large. Two recent papers [4, 14] highlight the problem and give good explanation of this phenomenon. Essentially, TCP connections oscillate between timeout state and active state. Besides, if a large number of flows are simultaneously active, their windows are small. This implies that the load offered to the network increases very fast even during congestion avoidance, since the window is increased by 1 packet for every ACK returned, up to the congestion threshold.

The definition of what is a “large number of connections” (or flows) is somewhat vague. However, both [4] and [14] agree in defining the critical number roughly equal to the total number of packets that can be stored in the bottleneck buffer plus the number of packet and ACKs that are “in flight” between source and destination.

Unfortunately, due to the limited scalability of *ns-2* and the insufficient computer resources, it was impossible to run true “many flow” experiments in a WAN scenario with 155 Mbit/s bottleneck link. To overcome in part this problem, and using again the example in Fig. 3, all link speeds are reduced to 10 Mbit/s. Propagation delays are equal to those given in Section 5.2. The number of “in flight” packets and ACKs is around 50; the number of FTP connections is $N = 100$. Telnet connections are not simulated in this case since the main focus is on throughput behavior.

The router buffer space ranges from 200 to 800 kbytes. Thus, the sum of packet buffers and packets in flight is always larger than the number of connections. Strictly speaking, we are below the “many flows” threshold. However, the experiments exhibit the very pronounced unfairness trends typical of the “many flows” behaviour. Simulation experiments are run for 100 s and the first 10 s are discarded as transient. Simulation runs are longer in this case to allow collecting statistically meaningful results for all the connections.

Fig. 10 reports the steady state results for the 5 considered implementations. Plots (a)–(e) reports the average, maximum, and minimum throughput; plot (f) reports the drop rate for all cases. This latter plot shows that GWA and ECN still ensure loss-free operation, while the drop rate for Drop-Tail and RED are nearly three orders of magnitude higher than when only 10 FTP connections were present.

Observing the throughput performance in plots (a)–(e) of Fig. 10, the limitations of binary feedback schemes based on statistical principles become evident. On one hand, the average throughput performance is excellent for any TCP implementation. In fact, full bottleneck utilization is achieved in all schemes. On the other hand, when fairness is considered, neither Drop-Tail, nor RED, nor ECN provide satisfactory performance. Indeed plot (c) shows that ECN works properly when the buffer can hold several packets per connection. It becomes grossly unfair when the buffer size is not large enough. Basically, some of the connections never get a chance to transmit. We did not attempt a careful optimization of the ECN and RED parameters. Thus, we are not claiming that these schemes do not work properly. However, we wish to point out that the results suggest at the very least the need of rather sophisticated, configuration dependent tuning of the RED router parameters.

Both GWA implementations perform remarkably well in a "many flows" situation, with full link utilization and perfect fairness. Moreover, in comparing Fig. 10(e) with Fig. 7(e), one is surprised to find that GWA-FCFS fairness actually improves when increasing the number of active connections and reducing the available bandwidth. The reason is the following. When few connections share a high bandwidth link, like in Fig. 7(e), several packets per connection are "in flight" within the network. If the buffer is not very large, the number of queued packets is smaller or equal to the number of packets "in flight" (see for instance Fig. 9). In this case the throughput obtained by each connection is inversely proportional to RTT, leading to unfairness. When there are many connections sharing a smaller amount of bandwidth, each connection has at most 1 packet in flight in the network: in the case considered here there are about 50 in flight packets and 100 connections. The number of queued packets is larger than the number of in flight packets. The FCFS queuing delay has an equalizing effect of RTT and therefore on throughput.

6 Mixing GWA-FCFS with RED or ECN

One of the main concerns in developing and deploying new TCP versions is their ability to coexist with older versions and to be "friendly" to them; at the same time, it is required that they do not suffer too much from the interaction.

If routers do provide per flow queueing, different connections are isolated one another. Hence TCP-RR can coexist with any other TCP implementation and the performance of the single connections will be driven by the respective TCP version.

Of major concern is the possibility to mix *within the same buffer* GWA-TCP with other TCP connections. We consider here the case of mixing GWA-FCFS with either RED of ECN connections.

The routers are upgraded to perform different functions depending on the type of packet they are forwarding, just like ECN enabled routers mark ECN packets instead of dropping them. If the TCP packet belong to a connection that is neither ECN nor GWA capable, then the router queues it or drops it following the standard RED algorithms. If the packet belong to an ECN capable connection then it is randomly marked when the queue level is between the two RED threshods, while it is deterministically dropped if the global queue size is above the high threshold. Packets belonging to GWA connections are marked with the available buffer size $B_{n,i}$ according to a modification of Eq. (4) to take into account the presence of TCP connections that are not GWA enabled. Let n_G be the estimated number of GWA connections, while n_f is the estimated number

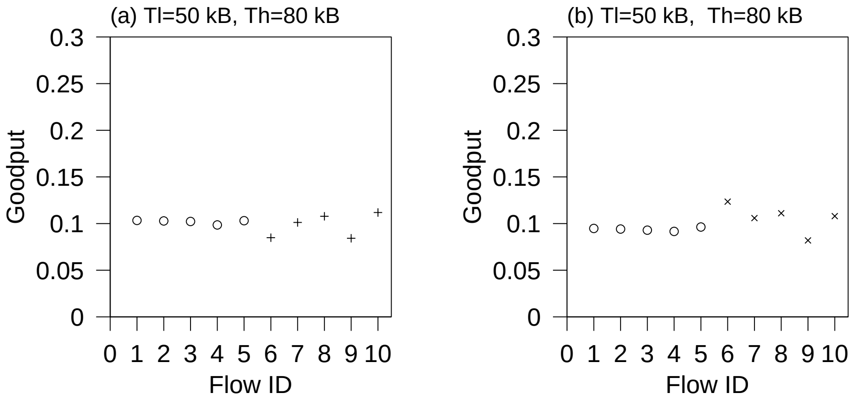

Figure 11: Throughput for each individual connection in LAN environment; GWA mixed with RED on the left and with ECN on the right

of all the TCP connections competing for the buffer resources. Substituting $B_t = \frac{n_G}{n_t} \cdot B_t$ in Eq. (4) yelds

$$B_{n,i} = \min\left\{ B_{n,i}, \left\lceil \frac{B_t}{n_f(t)} - \frac{q_G(t)}{n_G(t)} \right\rceil \right\} \tag{10}$$

where $q_G(t)$ is the buffer occupation due to GWA packets. GWA packets are dropped only when the buffer fills up.

In these experiments we use the topology reported in Fig. 3 with 10 hosts, but without telnet connections. In each experiment 5 connections are GWA and the other 5 are either RED or ECN connections. Results are obtained both for LAN and WAN environment; the bottleneck buffer size is 120 kbytes in LAN experiments and 2 Mbytes in WAN experiments. In the figures different TCP implementations are sorted out using a different markers: the first five are GWA and are identified with circles, while number from 6 to 10 are either RED, identified with "plus" signs, or ECN, identified with "times" sign.

Fig. 11 reports the throughput in a LAN environment, averaged over the last 25 s of 30 s simulation experiments. Every conections obtains roughly the same throughput, indicating that neither GWA prevails on RED/ECN, nor the other way round. In this case individual GWA connections receive roughly their fair share of bandwidth. The same is instead not true for either RED or ECN connections: While the average throughput is the same, RED and ECN connections show some degree of internal unfairness. RED low and high thresholds, Tl and Th are set to 50 kbytes and 80 kbytes respectively. As explained in the following, presenting WAN results, the threshold values do influence the results and care must be taken in setting the threshold values.

Fig. 12 shows the results for three different values of the thresholds for simulation experiments lasting 50 seconds (the first five are not counted as usual). It is quite clear that different threshold values makes either GWA prevail on RED/ECN or the other way round. This result is predictable, since assigning different thresholds is equivalent to differently balance the resources between GWA and non GWA connections. Notice that the most favourable case, where neither GWA nor RED/ECN prevails, it the case when $Tl = 1$ Mbytes, i.e., half of the buffer, while GWA prevails if the thresholds are smaller and RED/ECN prevails when they are larger. This hints to the fact that a careful *dynamic* setting of the thresholds can ensure good and fair performance with traffic where the ratio of GWA and non GWA connections varies in time. Once more GWA proves to be more fair than either RED or ECN, even if the connection propagation delays are not equal.

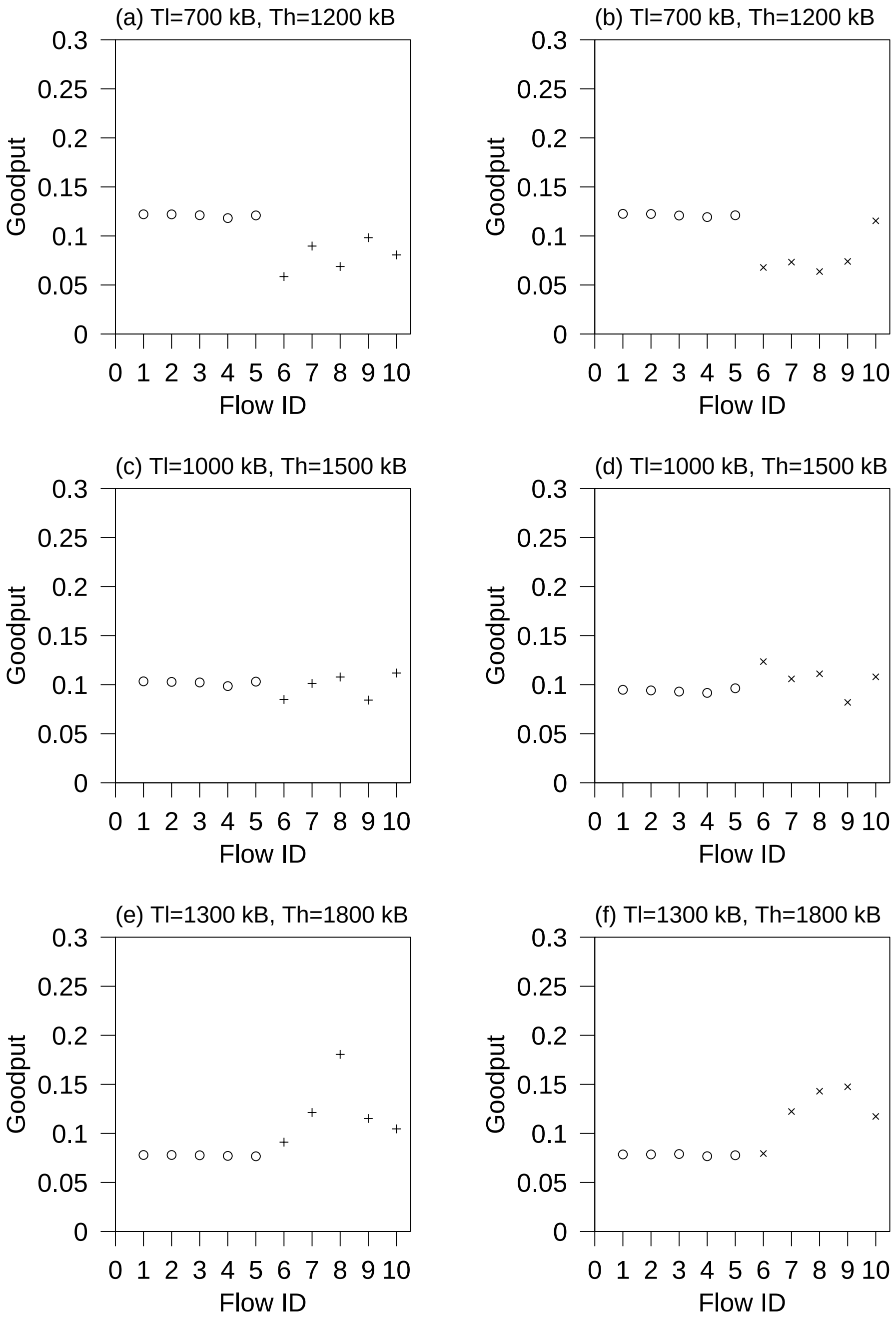

Figure 12: Throughput for each individual connection in WAN environment; GWA mixed with RED on the left and with ECN on the right; 50 seconds simulations

Fig. 13, finally, refer to the WAN case with $Tl = 1$ Mbytes and $Th = 1.5$ Mbytes, but considering only the first 10 s of simulation, i.e., results are averaged only on 5 seconds, since the first five are always discarded to avoid taking into account the different starting times of connections. These results show that the transient behavior of GWA is much smoother and reaches steady state faster than both RED and ECN. In fact in the short period ECN proves to be unfair, while plain RED, besides being unfair, underutilizes the resources. In this latter case GWA is able to exploit the resources non utilized by RED connections, unexpectedly resulting in a Max-Min fair utilization of

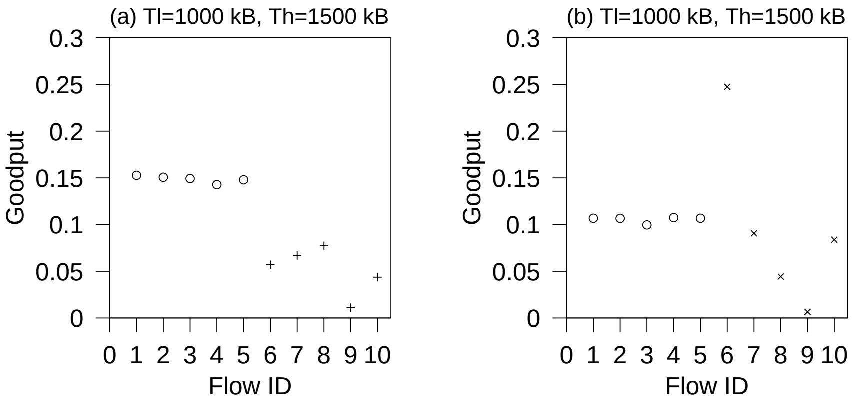

Figure 13: Throughput for each individual connection in WAN environment; GWA mixed with RED on the left and with ECN on the right; 10 seconds simulations

resources at the bottleneck. This is much more remarkable considering that, when RED connections reach steady state, the average utilization between GWA and RED is the same.

This Section proves that the coexistence of GWA and non GWA connections sharing the same buffer is feasible, though it might be difficult to tune the thresholds of RED routers. However it is worthwhile to notice that the router implementation presented here is only a possible one, and it is not claimed that it is the best one for the purpose.

Although the behavior of mixed GWA and non-GAW connections still need more investigation, the smoother behavior of GWA and its faster convergence to steady state transmnission are a great advantage especially in the transfer of small to medium amount of data, that are in fact the vast majority of data transactions on the Internet.

7 Discussion and Concluding Remarks

The Generalized Window Advertising (GWA) algorithm presented in this paper is a new approach to TCP congestion control. The scheme relies on the cooperation between IP network entities (routers) and host based TCP entities. The window control algorithm is based on a classical control theory approach known as Smith Predictor. While the original Smith Predictor scheme requires the knowledge of the round trip time, the paper shows that such explicit knowledge is not needed. It is sufficient to monitor the amount of data that has been transmitted, but not yet acknowledged. This value is normally monitored by TCP. Moreover, the original scheme requires per flow queuing. The paper shows that even FCFS can achieve full link utilization. This greatly reduces complexity (and cost) of the GWA implementation.

The GWA scheme was implemented in the *ns-2* simulator, and it was compared with other TCP implementations and buffer management schemes. Results show that GWA in general attains a more stable network operation as well as a higher degree of fairness. It guarantees a loss free operation as predicted by theory. Throughput performance depends on propagation delays and router buffer size. In typical operating conditions, GWA ensures full link utilization.

GWA is backward compatible with all TCP versions. Namely, GWA and non-GWA TCP hosts can always communicate with each other. In addition it does not require extensive modifications to the existing protocols.

Research is in progress in several directions. First, the performance in presence of high priority and variable background traffic is investigated. This study will give insight on the behavior in an IntServ or DiffServ scenario. Given the properties of Smith Predictor controllers, it can be anticipated that GWA TCP will perform well in this scenario. Second, the buffer sizing and management in IP routers needs further improvement and optimization. The controller designed in this paper ensures zero packet loss rate under any circumstance. However, it requires a fairly large amount of buffering along the path. Besides, if strict fairness is an issue, it is necessary to resort to per-flow queuing. Relaxing the requirements on packet loss probability, and devising intelligent buffer management techniques, may lead to more efficient and easier to deploy implementations. Finally, the issue of coexistence of GWA with traditional TCP can be studied more deeply. While backward compatibility is guaranteed, and it has been shown that GWA and traditional TCP implementations can share the same buffer, the optimization of network resources use still need more investigation. On the one hand it can be concluded that steady state behavior is fair and neither GWA prevails on standard or ECN enabled TCP, nor the other way round. On the other hand the transient behavior of the network whith mixed TCP implementations is still not fully understood and need more investigation.

References

[1] T. V. Lakshman, U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Product and Random Loss," *IEEE/ACM Transactions on Networking*, Vol.,5, No. 3, June 1997.

[2] M. Gerla, R. Bagrodia, L. Zhang, K. Tang, L. Wang "TCP over Wireless Multihop Protocols: Simulation and Experiments", To appear in Proceedings of *IEEE ICC'99*, Vancouver, Canada, Jun. 1999.

[3] M. Gerla, K Tang, R. Bagrodia, "TCP Performance in Wireless Multihop Networks", To appear in Proceedings of *IEEE WMCSA'99*, New Orleans, LA, Feb. 1999.

[4] R. Morris, "TCP Behavior with Many Flows", *Proc. of IEEE ICNP'97*, Atlanta, GE, USA, Oct. 28 – 31, 1997.

[5] S. Floyd, "TCP and Explicit Congestion Notification", *ACM Computer Communication Review*, V. 24 N. 5, pp. 10-23, Oct. 1994.

[6] K. K. Ramakrishnan, S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, IETF, Jan. 1999.

[7] C. Villazamir, C. Song, "High Performance TCP in ANSNET", ACM Computer Communication Review, Vol. 4, No. 5, Oct. 1995

[8] R. Stevens, "TCP/IP Illustrated", Vols. 1 & 2 Addison Wesley, 1994.

[9] J. Postel, "Transmission Control Protocol – DARPA Internet Program Protocol Specification", RFC 793, DARPA, Sept. 1981.

[10] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, IETF, Jan. 1997.

[11] O. Smith, "A Controller to Overcome Dead Time", ISA J., Vol. 6, No. 2, pp. 28–33, 1959.

[12] S. Mascolo, "Smith's Predictor for Congestion Control in TCP Internet Protocol", To be presented at *American Control Conference 1999*.

[13] S. Floyd, V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance”, *IEEE/ACM Transaction on Networking*, Vol. 1, No. 4, pp. 397–413, Aug. 1993

[14] C. M. Pazos, J. C. Sanchez Agrelo, M. Gerla, “Using Back-Pressure to Improve TCP Performance with Many Flows”, Accepted for publication at *IEEE INFOCOM'99*, 21st - 25th March 1999, Hotel Sheraton, New York, NY, USA