TCP Congestion Avoidance Using Explicit Buffer Notification

by

Saverio Mascolo and Mario Gerla

CSD–TR 980008

TCP Congestion Avoidance Using Explicit Buffer Notification

Saverio Mascolo and Mario Gerla

Abstract

This paper proposes to modify the TCP congestion control mechanism by allowing network routers to supply Explicit Buffer Notification (EBN) to the TCP data sender. The TCP sender runs an algorithm that uses the EBN feedback to compute the control window. The EBN plays the role of a generalized Advertised Window so that the EBN TCP results to be a natural extension of the current TCP flow control. The key feature of the proposed congestion avoidance algorithm is that it is not based on packet loss to detect congestion. As a consequence, it results to be suitable for applications that are sensitive to the delay or loss of individual packets, such as interactive traffic, and for data transfer over wireless links where packet loss will occur more often due to unreliable links than due to congestion. The proposed control works over connection paths with any bandwidth×delay product. Its strength lies in the fact that it is designed via classical control theory and, therefore, its dynamic properties, such as no cell loss and full link utilization, can be rigorously and easily shown via mathematical analysis. Finally, Fair Queuing at the routers decouples the flows and ensures fairness.

1. Introduction

The TCP protocol was designed to provide reliable data delivery over conventional (wired) networks for a limited range of transmission rates and propagation delays. One of TCP strengths lies in its congestion control mechanism proposed in the cornerstone paper by Van Jacobson [1]. Nowadays, data transfers over communication paths with ever-larger bandwidth×delay product, quality of service (QoS) requirements for interactive traffic and communication over wireless links, are shifting the domain for which TCP was originally engineered. As a consequence, active research is in progress to extend the domain of TCP operability [2].

The congestion control algorithm used in the TCP/IP protocol is a sliding window mechanism that uses packet loss to detect congestion. In particular, the end-systems probe the network state by gradually increasing the window of packets that are outstanding in the network until the network becomes congested and drops packets. This control strategy is the best one under the assumption that the network is a "black box" that cannot supply any explicit feedback information to the data sources. This assumption is sufficient for pure best-effort data with little or no sensitivity to delay or cell loss of individual packets, but makes problematic to support interactive applications such as telnet and web-browsing.

Active queue management mechanism that detects congestion before the queue overflows is desirable because it avoids dropping cells on queue overflow and synchronization of loss across multiple flows. Random Early Detection (RED) is an active queue management mechanism that has been proposed to detect incipient congestion [3], and is currently being developed in the Internet backbone. By dropping packets based on the average queue length exceeding a threshold, rather than only when the queue overflows, RED maintains the average queue at a smaller level, and reduces the delay experienced by the flows. A further enhancement [4], [5] allows Red to set an Explicit Congestion

Notification (ECN) bit in the packet header instead of dropping the packet. This bit has to be provided in the IP header and understood by the transport protocol. In this way, the receiver would be notified by the ECN of incipient congestion. The congestion control algorithm executed at the end-system would be essentially that of a transport protocol where a dropped packet is used as an indication of congestion, that is, the source would halve its congestion window in response to an ECN indication (multiplicative decrease).

In [6] gateways maintaining separate queues per transport-level-conversation at each output-port are proposed. The idea is to control data flow by updating a conversation window that is computed as a function of the per-conservation-queue-size and the preceding window-size.

In this paper we propose to allow the routers to stamp in the packet header the per-flow available buffer room. We call this feedback Explicit Buffer Notification (EBN). The control window size is dynamically updated based on the buffer notification received from the routers. In particular, the window is computed by means of an algorithm that is the window-based "equivalent" of a rate-based algorithm for ABR traffic control in ATM networks, which has been proposed in [7]. Thanks to the explicit feedback from the network, the TCP congestion control algorithm is transformed to a congestion avoidance algorithm, where the congestion is no more intentionally provoked in order to probe the network capacity but is rather avoided guaranteeing no packet loss. The advantages of the proposed algorithm are: 1) retransmissions are reduced thus improving network utilization and delivering of traffic that is sensitive to delay or cell loss; 2) since packet loss does not indicate congestion, the proposed congestion avoidance algorithm is suitable over wireless links, where losses occur more often due to unreliable links than congestion; 3) the algorithm works well over any bandwidth×delay path, including wireless links with very large propagation delay such as satellite link.

2. The current TCP congestion control algorithm

In this section, we briefly describe the current TCP congestion control algorithm in order to enlighten the modifications introduced by the Explicit Buffer Notification. In particular, we will show how the proposed EBN TCP is a "natural" extension of the current TCP flow control.

TCP congestion control mechanism has two parts: the *flow control mechanism* that prevents the sender from overflowing the receiver buffer, and the *congestion control mechanism* that prevents the sender from overloading the network [8].

2.1 The flow control mechanism

Let *MaxRcvBuffer* be the size of the receiver buffer in bytes. On the receive side TCP must keep

$$LastByteRcvd - NextByteRead \leq MaxRcvBuffer$$

to avoid overflow. Therefore it advertises a window size of

$$AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - NextByteRead)$$

which represents the amount of free space remaining in the receiver buffer. The TCP on the send side computes an Effective Window W that limits how much outstanding data it can send [8]

$$W = EffectiveWindow =$$

$$AdvertisedWindow - (LastByteSent - LastByteAcked) =$$

$AdvertisedWindow–OutstandingPackets$ (1)

2.2 The congestion control mechanism

In order to keep the sender from overloading the network, TCP maintains the state variable called *Congestion Window*. The congestion window plays, in congestion control, the role of the advertised window in flow control. TCP computes the minimum of the congestion window and the advertised window. Therefore the *Effective Window* is computed as follows [8]

$$MaxWindow=MIN(CongestionWindow,AdvertisedWindow)$$
$$EffectiveWindow=MaxWindow–OutstandingPackets \quad (2)$$

Now, since TCP assumes no explicit feedback from the network, the problem is how TCP comes to learn an appropriate value for the *Congestion Window*. Van Jacobson's milestone paper [1] defines an increase/decrease mechanism to throttle the size of the congestion window. This mechanism probes the network capacity by increasing the load until packet loss is experienced; when a loss is experienced the window size is reduced.

In detail, the TCP-Tahoe and TCP-Reno [9] maintain a current congestion window size W and a threshold W_t which are updated as follows:

<u>1 Tahoe/Reno</u>

After every non-repeated ack:
if $W<W_t$,set $W=W+1$; /slow start phase
else set $W=W+1/[W]$. /congestion avoidance phase. $[x]$ denotes the integer part of x

<u>2 Tahoe</u>

After a packet loss is detected (when timer expires or the number of repeated ACKs exceeds a threshold):
set $W_t=W/2$;
set $W=1$.

<u>2 Reno</u>

When the number of repeated ACKs exceeds a threshold, retransmit "next expected" packet;
set $W_t=W/2$, then set $W=W_t$;
resume congestion avoidance phase using new window once retransmission is acknowledged.

<u>3 Reno</u>

Upon timer expiry, the algorithm goes into slow start phase as before:
set $W_t=W/2$;
set $W=1$.

The key feature of the TCP congestion control algorithm is that it assumes no explicit feedback from the network. Therefore, the congestion window size oscillates around its equilibrium value because packet losses are intentionally provoked in order to probe network capacity. This causes link under-utilization when the congestion window is small, and packet loss when the congestion window is large. Moreover, increasing the congestion window until packet are lost, fills the buffers. This increases queuing delays that are harmful for delay-sensitive applications [6]. Other inefficiencies of TCP congestion control are [2], [6]:

1) slow adaptation of the window and, therefore, low performance especially over path with large bandwidth-delay product;
2) the reaction of TCP to detection of packet loss by means of TCP retransmission timeout may take several RTTs, that is, it is no more up to date.

In the next Section we propose to modify the TCP allowing the network router to supply explicit feedback [4], [5]. This is because we believe that, under the assumption of no explicit feedback from the network, the current TCP congestion control basically already gives the best obtainable performance.

3. The TCP with Explicit Buffer Notification

In this Section we describe the TCP with Explicit Buffer Notification. This algorithm makes one further step in the direction suggested in [4] and [5], where it is assumed that the network feeds the data sources with a binary feedback information, called Explicit Congestion Notification (ECN). Here we propose a multi-bit feedback that we call Explicit Buffer Notification (EBN). Every packet, at the destination, carries stamped on its header the minimum amount of free space remaining in the buffers it traverses along the communication path. In other words, let B_i be the free space remaining in the *i-th* buffer, where the *i-th* buffer is exclusively used by the considered TCP flow (i.e. Fair Queuing is assumed at the router). The Explicit Buffer Notification is

$$EBN = \min_{\substack{i \in routers \\ of\ \ the\ \ path}} \{B_i, AdvertisedWindow\}$$

Clearly, EBN plays the role of a generalized Advertised Window.

Note that the per-flow buffering allows the flows to be decoupled. This assumption is crucial in order to effectively and easily control congestion by means of an algorithm entirely run at the source end-system.

At the destination end-system, TCP stores this EBN in the TCP header in the returning ACK packet. At the source end-system, when an ACK is received, the TCP computes the size of the control window W as follows

$$W = EBN - OutstandigPackets \tag{3}$$

Eq. 3 dynamically updates the window W of unacknowledged packets based on the EBN feedback received from the network.

Due to the explicit feedback from the network, the EBN TCP does not need to probe network capacity by inducing packet loss. Therefore, in an ideal setting, the slow start, the congestion avoidance and the multiplicative decrease phases of the current TCP could be no more necessary. In a practical setting, to ensure the TCP operability in the case of path with some routers that are not EBN capable, the current TCP window control (2) must be used where the *MaxWindow* is now

$$MaxWindow = MIN\ (CongestionWindow, EBN)$$

Remark 1*:* The window-based feedback control (3) is a "natural" extension of the current TCP flow control equation (1). In fact, it can be viewed as the generalization of the TCP flow control equation obtained by substituting the Advertised Window with the Explicit Buffer Notification. The EBN takes into account the space that is free in all the buffers along the flow path, including the Receiver Buffer.

Remark 2: FIFO queuing, differently from *Fair queuing* (FQ), does not separate packets according to the flow to which they belong. This is a problem at two different levels [8]. At one level, it makes impossible to adequately control congestion via an algorithm implemented entirely at the source. At another level, since the entire congestion algorithm is run at the source and FIFO queuing does not provide a way to police how well the sources adhere to this mechanism, it is possible for a ill-behaved flow to capture an arbitrary large fraction of the bandwidth. Fair queuing addresses both these problems maintaining a separate queue for each flow currently being handled by the router. The fact that each flow has exclusive buffers, i.e. the flows are decoupled, allows an algorithm entirely implemented at the source to effectively and easily control congestion. Moreover, the router services these queues in a round-robin manner and enforces fairness among a set of well-behaved congestion controlled flows.

4. Genesis of the TCP EBN algorithm and mathematical analysis

In general, we can model a packet switching connection as a flow of packets going through a series of buffers and communication links. This simple model captures the fundamental structure of data networks, including both Internet and ATM. We assume that every flow, for instance a TCP flow, has dedicated buffers along the communication path, that is, these buffers store only packets of the desired connection (*Fair queuing*). Under this assumption, each flow is completely decoupled from other flows so that routers can guarantee appropriate QoS requirements to each flow. Moreover, thanks to this assumption, it is possible to effectively perform congestion avoidance by means of an algorithm entirely run at the source.

Considering a flow, a link along the path will be the bottleneck for this flow. The buffer feeding this link has the maximum queue level over all the queue levels along the path. The congestion avoidance algorithm, entirely implemented at the source, must guarantee that: 1) the bottleneck queue does not overflow (i.e. no data loss); 2) the bottleneck queue level is always greater than zero (i.e. full link utilization).

Fig. 1 shows the dynamic model of the flow with the corresponding bottleneck queue. The model consists of:

1) The integrator represented in the Laplace domain by the transfer function $1/s$. It models the buffer associated to the link that is the bottleneck for the considered flow;
2) The disturbance $d(t)$, which models the link bandwidth that is available for the considered flow; $d(t)$ is an unknown and bounded function that depends on all the flows sharing the bottleneck link;
3) The transfer function $e^{-sT_{fw}}$, which models the propagation time T_{fw} from the data source to the bottleneck queue;
4) The transfer function $e^{-sT_{fb}}$, which models the propagation time T_{fb} from the bottleneck queue to the destination and then back to the source; T_{fb} includes also the bottleneck queuing time in case it is not zero;
5) The controller transfer function $G_c(s)$;
6) The data source input rate $u(t)$;
7) The reference signal $r(t)$.

The feedback control scheme works as follows: the source compares the reference signal $r(t)$ with the delayed bottleneck queue level $x(t–T_{fb})$ and then inputs the difference into the controller $G_c(s)$ whose output is the input rate $u(t)$. The change in input rate reaches the bottleneck queue after the propagation delay T_{fw}, whereas the queue level is fed back to the source after the time T_{fb}. It should be noted that the *round trip time* (*RTT*) of the connection is always $RTT= T_{fw}+T_{fb}$ anywhere the bottleneck queue may be positioned along the flow path. Note that T_{fb} includes the delay due to the bottleneck queuing time. This means that the proposed model is capable to represent the case of moving bottleneck.

Fig. 1 shows that the input rate variable u(t) fills the bottleneck queue, whereas the available bandwidth variable d(t) empties this queue. To guarantee full utilization of network links in presence of the disturbance $d(t)$, the control variable $u(t)$ has to pump enough traffic so that the queue is never empty. On the other hand, to guarantee stability, i.e. to avoid queue overflow, the control u must moderate the intensity of data pumping.

Formally, queue stability can be stated as the objective of designing a control law $u(t)$, for the input rate, such that the bottleneck queue level $x(t)$ satisfies the following *stability condition*:

$$x(t) \le r^o \quad \text{for } t > 0$$

where r^o is the buffer allocation at the network queues. Similarly, full utilization of network links is guaranteed if the bottleneck queue level of the flow satisfies the following *utilization condition*:

$$x(t) > 0 \ \text{ for } \ t > T$$

where T represents the transient time after the starting of data flow. Clearly, this condition guarantees that any link has always data to send.

Now, by letting the bottleneck buffer allocation r^o be the reference value $r(t)$, the control action tries to fill this queue at its full capacity, whereas the available bandwidth $d(t)$ seeks to empty the queue. Note that, due to the possibly large propagation delays in the control loop, queue level dynamics might exhibit oscillations, and even become unstable. Therefore the design of the linear controller $G_c(s)$, which satisfies the *stability* and *efficiency* conditions, has to be carried out carefully.

In this paper, the design of $G_c(s)$ is developed following Smith's principle, which is an important classical control technique for time-delay systems [10], [11]. In [7] this principle was exploited for designing a rate-based control algorithm for ABR traffic in ATM networks.

In wide area networks, the connection *round trip time* is large and influences the stability of the congestion control algorithm. The *round trip time* is mostly determined by the propagation delay. This quantity can be computed in advance, when the connection is established. To take into account the jitter of round trip time due to queuing time, a model containing time varying delays should be considered. However this would make mathematically difficult to deal with the model. From a practical point of view, it is possible to extend the validity of the model with constant *RTT* to the case of time varying *RTT,* where *RTT* can be estimated dynamically using TCP round trip delay measurements. Moreover, we will see in Sec. 6 that the window-based form of the control equation is intrinsically robust to the jitter in *RTT*, that is, it inherently takes into account the time varying *RTT*.

By assuming a known *RTT*, Smith's principle can be easily exploited. In particular, the system reported in Fig. 1, can be made equivalent to the delay-free system shown in Fig. 2, where K is a constant gain. Thus, equating the transfer functions of the systems reported in Fig. 1 and Fig. 2

$$\frac{\frac{K}{s}}{1+\frac{K}{s}}e^{-sT_{fw}}=\frac{G_c(s)\frac{1}{s}e^{-sT_{fw}}}{1+G_c(s)\frac{1}{s}e^{-s\cdot RTT}}$$

and, after a little algebra, the following controller is derived

$$G_c(s)=\frac{K}{1+\frac{1}{s}\left(1-e^{-s\cdot RTT}\right)} \tag{4}$$

The system in Fig. 2 is a first order system with the delay T_{fw} in cascade. From control theory, it is known that the step response of a first order system delayed by T_{fw} is $r^o[1-exp(t-T_{fw,})]\cdot 1(t-T_{fw,})$, where r^o is the amplitude of the step $r(t)=r^o\cdot 1(t)$. This response has two advantages: 1) it is bounded by r^o; 2) it has no dumped oscillations.

In the time domain, the transfer function (4) gives the following input rate control equation [7]

$$u(t)=K\left(r^o-x(t-T_{fb})-\int_{t-RTT}^{t}u(\tau)\cdot d\tau\right) \tag{5}$$

This equation can be intuitively interpreted as follows: the computed input rate is proportional, through the coefficient K, to the available room in the bottleneck queue, that is $(r^o-x(t-T_{fb}))$, decreased by the number of cells released by the source during the last *round trip time*. Equation (5) is a rate-based control equation that was presented in [7] for controlling ABR traffic. In Section 6, we will transform this equation to an equivalent window-based form.

In the next section performance evaluation of the proposed control algorithm is carried out via mathematical analysis. In particular, stability is demonstrated and buffer allocation required for full utilization is derived.

5. Performance Evaluation

Linear control theory provides an established set of tools which enables us to design algorithms whose performance can be predicted analytically rather than relying on simulations. In particular, to analyze the performance of the proposed algorithm is sufficient to use standard Laplace transform technique. The advantage of mathematical analysis is that the results are demonstrated in a general setting, whereas the validation using simulations is always restricted to the simulated scenario. It is worth noting that the mathematical analysis is here possible thanks to the simplicity of the proposed algorithm and to the assumption of *decoupled flows* (i.e. Fair Queuing).

5.1 Stability

The following proposition shows that the bottleneck queue is bounded, that is, overflow of the queues along the path is avoided.

Proposition 1*:* Considering the reference signal $r(t)= r^o \cdot 1(t)$ and the controller (4), the output of the system shown in Fig. 1 satisfies the *stability condition* $x(t) \le r^o$, for $t \ge 0$.

Proof:

The input-output dynamics of the systems in Fig. 1 and Fig. 2 are equal. Therefore, considering the system in Fig. 2, the Laplace transform of the output $x(t)$ in response to the set point $r^o \cdot 1(t)$ is

$$X(s) = \frac{r^o}{s} \frac{1}{(1 + s / K)} \cdot e^{-sT_{fw}}$$

By transforming $X(s)$ back to time domain, it follows

$$x(t) = r^o \cdot \left(1 - e^{-K\left(t - T_{fw}\right)}\right) \cdot 1\left(t - T_{fw}\right) \le r^o \qquad (6)$$

that is, the queue does not overflow.

Remark 3*:* The queue dynamics (6) is characterized by the time constant $\tau = 1/K$. Therefore, the transient dynamics can be considered exhausted after the time $T_{tr} = T_{fw} + 4\tau$. Fig. 3 shows the dynamics of $x(t)$ in response to $r(t) = r^o \cdot 1(t)$.

5.2 Modeling the available bandwidth as a disturbance

It is difficult for a source to know the bandwidth that is available at the bottleneck link. This is because the path can be a wide area connection traversing heterogeneous networks. Moreover, the bandwidth at the bottleneck link is shared by other flows so that it is impossible to predict the left over bandwidth capacity. We assume that the data source does not know the bandwidth that is available at the bottleneck link. Therefore, we model this bandwidth by means of the disturbance $d(t)$, which is a bounded and unmeasured function. The reason for which we assume that $d(t)$ is not measured is that, since it is a bursty signal, it is difficult to measure it using the sampled measurements conveyed by the acks. In fact, the Nyquist sampling theorem requires that, in order to reconstruct a signal, it is necessary a sampling frequency at least equal to two times the signal higher frequency. As a consequence, it is possible to measure only a low-pass filtered $d(t)$, i.e. a time average which cut off the higher frequencies. Now, it should be noted that the queue level is a low-pass filtered measurement of $(\textit{input rate} - d(t))$, that is, the filtered measurement of $d(t)$ is an information already contained in $x(t)$.

A link transmission capacity normalized to unity is assumed so that, if all link bandwidth is available for the considered flow*,* $d(t)$ is equal to the step $1(t)$. The coexistence with other flows reduces the bandwidth which is available for the considered flow to $d(t) = 1(t) - b(t) \ge 0$*,* where $b(t)$ is the bandwidth used by other flows. By defining $b_m = \min_t \{b(t)\}$ it results: $d(t) \le 1(t) - b_m \cdot 1(t) = a \cdot 1(t)$ where $a = (1 - b_m) < 1$. In other words, a is the maximum available bandwidth for the considered flow.

The objective of the control algorithm is to keep the bottleneck queue level greater than zero and less than the queue capacity in presence of a time varying, bounded and unknown depletion rate $d(t)$.

5.3 Full utilization of networks links

To guarantee full utilization of network links in presence of time varying available bandwidth, the *utilization condition* must be satisfied. Intuitively, a buffer allocation equal to (available bandwidth)×(RTT) is necessary to ensure full link utilization. In fact, assuming that the available bandwidth goes instantaneously from zero to a, then, in the time *RTT,* the amount of data $a \cdot RTT$ can be drained, whereas no feedback can reach the buffer before the round trip time [12]. The following proposition states the minimum buffer allocation that is necessary to fully utilize the link bandwidth.

Proposition 2*:* The control law (5) guarantees full utilization of network links if the condition

$$r^o > a(\tau + RTT) \tag{7}$$

is satisfied.

Proof:

By considering that the available bandwidth $d(t)$ is an unknown and bounded function such that $d(t) \le a \cdot 1(t)$, with $a<1$, the worst case disturbance $a \cdot 1(t)$ is assumed. Using the controller (4), the transfer function from the available bandwidth $d(t)$ to the queue level $x(t)$ is given by

$$\frac{X_d(s)}{D(s)} = -\frac{1}{s} + \frac{K}{s(s+K)} e^{-s \cdot RTT}$$

which, for $D(s) = \frac{1}{s}$ gives

$$x_d(t) = a\left(-t \cdot 1(t) + \left(t - RTT\right) \cdot 1\left(t - RTT\right) - \frac{1}{K}\left(1 - e^{-K\left(t - RTT\right)}\right) \cdot 1\left(t - RTT\right)\right)$$

The dynamics of $x_d(t)$ is show in Fig. 4.

Letting $x_r(t)$ be the response to the step input $r^0 \cdot 1(t)$, the global queue dynamics is

$$x(t) = x_r(t) + x_d(t) = r^o\left(1 - e^{-K\left(t - T_{fw}\right)}\right) \cdot 1\left(t - T_{fw}\right) +$$
$$a\left(-t \cdot 1(t) + \left(t - RTT\right) \cdot 1\left(t - RTT\right) - \frac{1}{k}\left(1 - e^{-K\left(t - RTT\right)}\right) \cdot 1\left(t - RTT\right)\right)$$

which, for $t \rightarrow \infty$, gives:

$$x_s = r^o - a \cdot RTT - \frac{a}{k}$$

In order to fully utilize the bandwidth, x_s must be greater than zero so that the output link of the queue has always data to send. Therefore, condition (7) is derived.

Fig. 5 shows the global bottleneck queue dynamics in response to $r(t)=r^o \cdot 1(t)$ and $d(t)=a \cdot 1(t-t_1)$.

5.4 Max-min fairness

Proposition 3*:* The *Fair Queuing* service discipline guarantees fair allocation of available bandwidth to each flow. This function is performed independently from the congestion control algorithm.

Proof:

The queuing service discipline distributes the available link bandwidth to each flow according to a given definition of fairness, such as, for instance, fair queuing. In other words, the queuing service discipline gives a fraction a_j of the available bandwidth $d(t)$ to each *j-th* flow. This bandwidth $a_j d(t)$ is the disturbance for the *j-th* flow. Clearly, the queuing service discipline is not "seen" by the control algorithm, that is, congestion control and queuing service discipline are completely decoupled.

6. Window–based control

So far, we have described a control scheme that computes the input rate. Eq. 5 has been derived for controlling best effort traffic in a general packet switching network modeled as a graph of nodes (switches) and links.

Now, to apply the scheme to TCP source, we transform the rate-based control equation (5) to the window-based control equation (3) described in Section 3. To do this, the control equation (5) can be rewritten as

$$\frac{u(t)}{k} = u(t) \cdot \tau = \left(r^o - x(t - T_{fb}) - \int_{t-RTT}^{t} u(\tau) \cdot d\tau \right). \qquad (8)$$

We can interpret $u(t) \cdot \tau$ as the *Window* of unacknowledged data that can be sent. By considering that

$$OutstandingPackets = \int_{t-RTT}^{t} u(\tau) \cdot d\tau$$

and letting *EBN* be

$$EBN = r^o - x(t - T_{fb})$$

Eq. 8 gives the Eq. 3

$W=EBN - OutstandingPackets$

already introduced in Sec. 3.

Every time the source gets an ack, the window W is updated in according to Eq. (3). Note that the window control equation is robust to the jitter in *RTT*. In fact, the number of outstanding packets inherently takes into account the time varying *RTT*. Therefore, we can state that the window-based form (3) of the rate-based equation (5) does not suffer the problem of the jitter of *RTT*. On the other hand, it is worth noting that the traffic generated by a window controlled source is bursty. Therefore, a TCP using the rate-based control equation (5) could be exploited to avoid bursty traffic.

Finally we remark that, in case some routers along the path are not EBN capable, then the current TCP congestion control equation (2) should be used where *MaxWindow=MIN* (*CongestionWindow*,*EBN*).

Conclusions

In this paper, a window-based TCP congestion avoidance algorithm based on feeding back Explicit Buffer Notification has been presented. A major advantage is that window control need not to be driven by packet loss. The window is dynamically updated taking into account the explicit feedback from the network. The proposed algorithm is simple and it results to be a natural extension of the current TCP flow control. It has a common root with a rate-based control algorithm proposed for ABR traffic in ATM networks [7]. In particular, it is the equivalent window-based form of that algorithm. Therefore, it is not only a natural enhancement of the current TCP flow control, but it offers a uniform mechanism for integrating the control of Internet traffic over ATM. Since the control algorithm is simple and designed using linear control theory, its dynamic properties, such as full link utilization and no data loss, are rigorously shown via mathematical analysis. As a consequence, the validation of the algorithm is general and it is not restricted to simulated network scenarios.

References

[1] V. Jacobson, "Congestion Avoidance and Control," *Proc. of the SIGCOMM'88 Symposium*, Stanford, CA, pp. 314-329, 1998.

[2] V. Jacobson, "TCP Extensions for High Performance," *Internet Draft Paper*, February 1997.

[3] S. Floyd, and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, August 1993, pp.397-413.

[4] S. Floyd, "TCP and Explicit Congestion Notification", ACM Computer Communication Review, vol. 24, no. 5, October 1994, pp. 10-23.

[5] K. K. Ramakrishan, S. Floyd, "A proposal to add Explicit Congestion Notification (ECN) to Ipv6 and to TCP", *Internet Draft Paper*, November 1997.

[6] T. Ziegler, H. D. Clausen, "Congestion Avoidance with BUC (buffer utilization control) gateways and RFCN (reverse feedback congestion notification)", *IEEE Int. Performance, Computing and Communications Conference*, Phoenix, February 1997, pp. 410-418.

[7] S. Mascolo, D. Cavendish, M. Gerla, "ATM rate based congestion control using a Smith Predictor: an EPRCA implementation", *IEEE Proc. of Infocom 96*, San Francisco, 1996.

[8] L. L. Peterson, B. S. Davie, *Computer Networks,* San Francisco, CA, 1996.

[9] W. Stevens, "TCP/IP illustrated", vol. 1, Addison Wesley, Reading, MA, 1994.

[10] O. Smith, "A Controller to Overcome Dead Time", *ISA J.*, vol.6, no.2, pp.28-33, 1959.

[11] G. F. Franklin, J. D. Powell, A. Emami-Naeini, *Feedback Control of Dynamic systems*, Addison Wesley, 1994.

[12] C. Villamizar and C. Song, "High Performance TCP in ANSNET", ACM Computer Communication Review, v. 4, no. 5, October 1995.

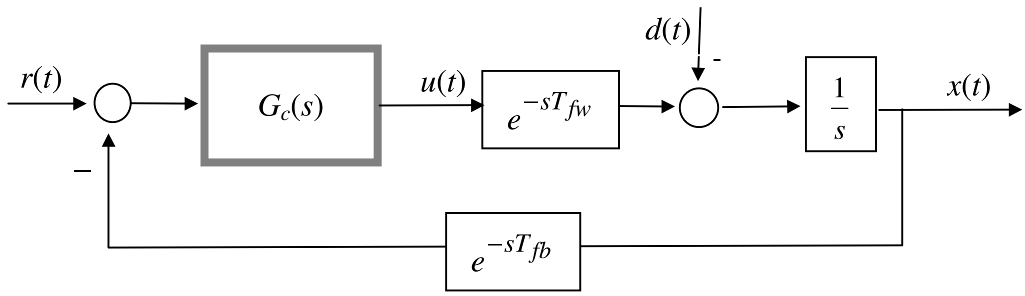

Fig.1 Dynamic model of the controlled data flow.

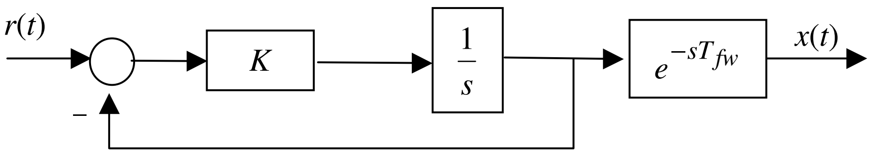

Fig. 2 Desired input-output dynamic

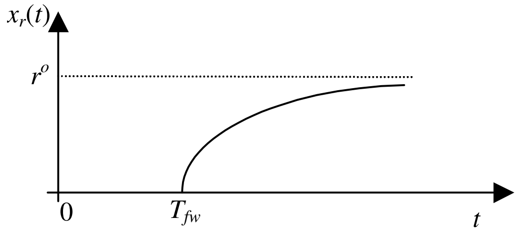

Fig. 3 Bottleneck queue level dynamics in response to the set point $r(t)=r^o\cdot 1(t)$.

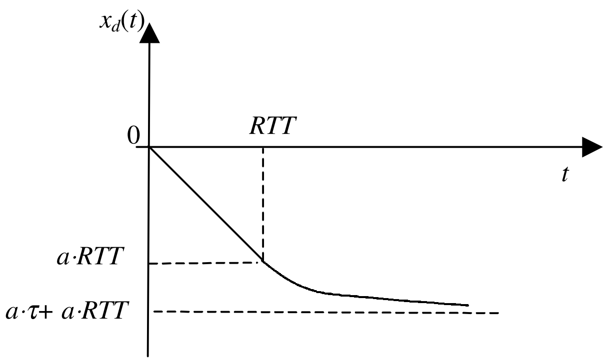

Fig. 4 Bottleneck queue level dynamics in response to the disturbance $d(t)$.

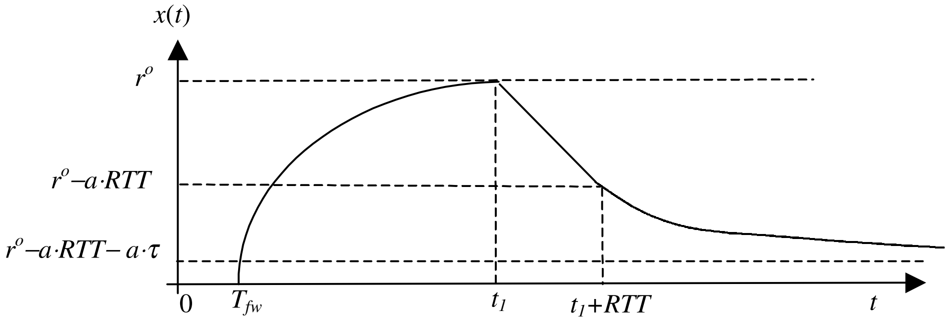

Fig. 5 Global queue level dynamics in response to $r(t)=r^o\cdot 1(t)$ and $d(t)=a\cdot 1(t-t_1)$.