Available online at www.sciencedirect.com
SCIENCE DIRECT®
Control Engineering Practice 14 (2006) 425–435
CONTROL ENGINEERING PRACTICE
www.elsevier.com/locate/conengprac

Modeling the Internet congestion control using a Smith controller with input shaping ☆

Saverio Mascolo*

Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, 70125 Bari, Italy

Received 4 May 2004; accepted 14 February 2005
Available online 8 April 2005

Abstract

The Internet has shown a great capability of endless growing without incurring congestion collapse. The key of this success lies in its TCP/IP congestion control algorithm. In this paper, we use control theoretic analysis to model the Internet flow and congestion control as a time delay system. We show that the self-clocking principle, which is known to be a key component of any stable congestion Internet control algorithm, corresponds to implement a simple proportional controller (P) plus a Smith predictor (SP), which overcomes feedback delays that are due to propagation times. Different variants of TCP congestion control algorithms, such as classic TCP Reno or the recent Westwood TCP, can be modeled in a unified framework by proper input shaping of the P + SP controller structure. Finally, we show that controllers that do not implement the Smith predictor, such as proportional (P) controllers or proportional + derivative + integral (PID) controllers, provide an unacceptable sluggish system because they do not implement dead-time compensation.

Keywords: Internet congestion control; Smith predictor; TCP/IP

1. Introduction and related works

The stability of the Internet and, in particular, the prevention of congestion requires that flows use some form of end-to-end congestion control to adapt the input rate to the available bandwidth (Jacobson, 1988; Mascolo, 1999, 2000; Allman, Paxson & Stevens 1999; Clark, 1988; Floyd & Fall, 1999; Peterson & Davie 2000). In fact, after the introduction of the transmission control protocol/Internet protocol (TCP/IP), the network suffered from congestion collapse until congestion control was introduced into the TCP stack in the late 1980s by Van Jacobson (1988).

TCP has two feedback mechanisms to tackle congestion: the *flow control* and the *congestion control*. The TCP *flow control* aims at avoiding the overflow of the receiver's buffer and is based on explicit feedback. In particular, the TCP receiver sends to the source the receiver's advertised window, which is the buffer available at the receiver. The TCP *congestion control* aims at avoiding the flooding of the network and is based on implicit feedback such as timeouts, duplicate acknowledgments (DUPACKs), round trip time measurements. In this case the source infers the network capacity using an additive–increase/multiplicative–decrease (AIMD) *probing* mechanism (Dah-Ming Chiu & Jain, 1989). The *increase* phase aims at increasing the flow input rate until the network available capacity is hit and a congestion episode happens. The sender becomes aware of congestion via the reception of DUPACKs or the expiration of a timeout. Then it reacts to light congestion (i.e. 3 DUPACKs) by halving the congestion window (*fast recovery*) and sending again the missing packet (*fast retransmit*), and to heavy congestion (i.e. timeout) by reducing the congestion window to one. Both the flow and congestion control implement

☆ Paper No: CR 2179: Enhanced version of the paper published at the IFAC '03 Workshop on Time-delay systems and recommended for inclusion in Control engineering practice.

*Tel.: +39 080 5963621; fax: +39 080 5963410.
E-mail address: mascolo@poliba.it.

0967-0661/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.conengprac.2005.02.006

the self-clocking principle, that is, when a packet exits a new one enters the network. The described mechanisms form the core of the classic Internet congestion control algorithm known as Tahoe/Reno TCP (Jacobson, 1988; Allman et al., 1999; Peterson & Davie, 2000). It is interesting to note that these mechanisms still form the main ingredients of all enhanced and successful TCP congestion control algorithms that have been proposed in the literature.

Research on TCP congestion control is still active in order to improve its efficiency and fairness, especially in new environments such as the wireless Internet (Lakshman & Madhow, 1997) or the high-speed Internet (Jacobson, Braden, & Borman, 1992; Hoe, 1996; Villamizar & Song, 1995; Fast TCP; Grieco & Mascolo, 2004; Floyd, 2003). We briefly summarize the most significant modifications that have been proposed up to now (Low, Pagnini, & Doyle, 2002).

The New Reno feature is an enhancement of Reno that has been proposed to avoid multiple window reductions in a window of data (Floyd & Henderson, 1999). TCP Vegas estimates the expected connection rate as $cwnd/\mathrm{RTT}_{\mathrm{m}}$ and the actual connection rate as $cwnd/\mathrm{RTT}$; when the *difference* between the expected and the actual rate is less than a threshold $\alpha>0$, the *cwnd* is additively increased. When the *difference* is greater than a threshold $\beta>\alpha$ then the *cwnd* is additively decreased. When the difference is between α and β, *cwnd* is maintained constant (Brakmo, O'Malley, & Peterson, 1995). Vegas TCP provides the basic ideas behind Fast TCP congestion control algorithm, which has been recently proposed by researchers at Caltech (Fast TCP). In the authors' words, "Fast TCP is a sort of high-speed version of Vegas". At the time of this paper Fast TCP is still in a trial phase and the authors have not released any kernel code or *ns*-2 implementation. Being based on RTT measurements to infer congestion, it could inherit all drawbacks of Vegas, mainly the incapacity to grab bandwidth when coexisting with Reno traffic or in the presence of reverse traffic (Grieco & Mascolo, 2004). TCP Westwood uses an end-to-end estimation of the available bandwidth to adaptively set the control windows after congestion (Mascolo, Casetti, Gerla, Sanadidi, & Wang, 2001; Grieco & Mascolo, 2002, 2004). Both Vegas and Westwood preserve the standard multiplicative decrease behavior after a timeout. TCP Santa Cruz proposes to use estimate of delay along the forward path rather than round trip delay and to reach a target operating point for the number of packets in the bottleneck of the connection (Parsa & Garcia-Luna-Aceves, 1999). The concept of *generalized advertised window* has been proposed in (Gerla, Locigno, Mascolo, & Weng, 2002) to provide an explicit indication of the network congestion status.

Recently, nonlinear stochastic differential equations have been proposed to model the TCP dynamics (Grieco & Mascolo, 2002; Kelly, 1999; Low, 2000; Hollot, Misra, Towsley, & Wei-Bo Gong, 2002). In these models, the dynamics of the expected value of the *cwnd* is mainly expressed as a function of the packet drop probability through a nonlinear differential equation. These models, and their linearized ones, have been used to predict the long-term TCP throughput and to design control laws for throttling the packet drop probability of routers implementing active queue management (Hollot et al., 2002). In particular, the mentioned nonlinear stochastic differential model of the TCP window has been linearized around the equilibrium to derive a transfer function from the packet drop probability to the bottleneck queue length. The linearized model has been employed to design a control law for the packet drop rate aiming at stabilizing the queue average length (Hollot et al., 2002). It is not clear how effective is the model to deal with real-time dynamics of TCP and in the presence of multi-bottleneck topologies.

In this paper, we propose a general control theoretic framework to model the TCP flow and congestion control along with its variants such as Reno and Westwood. We derive the following main results: (1) different TCP control algorithms can be modeled using the same control structure, which is a proportional controller (P) plus a Smith predictor (SP) for dead-time compensation; (2) the SP plays the fundamental role of overcoming delays in the feedback loop in order to provide a stable and fast control; (3) the SP provides the control theoretic explanation of the self-clocking principle; and (4) different control algorithms such as Reno, Westwood and so on can be explained in terms of different settings of the controller reference input signal.

The work is organized as follows: Section 2 outlines the TCP flow and congestion control algorithm; Section 3 models the dynamic behavior of a generic TCP flow; Section 4 models the TCP flow and congestion control using transfer functions and a proportional (P) plus a Smith predictor (SP) controller. Section 5 shows that the SP enforces the *self-clocking* principle and provides stability; Section 6 models the TCP Reno and Westwood control algorithms by proper shaping of the set point; Section 7 shows that other simpler controllers, such as proportional-integral-derivative (PID), cannot be used in the Internet because they would provide too sluggish behavior; thus, the classic TCP remains the starting point to be considered when designing any new control algorithm; finally, Section 8 draws the conclusions.

2. The TCP/IP flow and congestion control: background

In this section we provide a background on the TCP congestion control. TCP connection is a virtual pipe between the send socket buffer and the receive socket

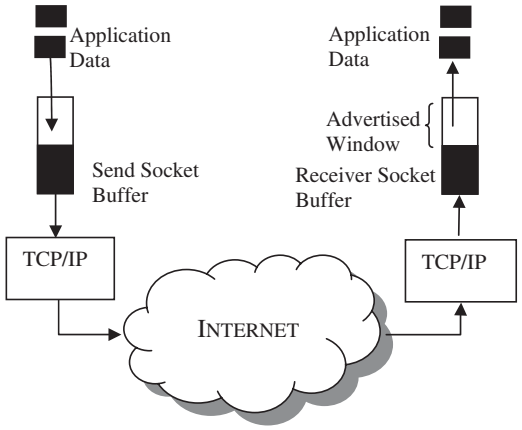

Fig. 1. Schematic of a TCP connection.

buffer (see Fig. 1). The TCP has two feedback mechanisms to tackle congestion: the *flow control mechanism* that prevents the sender from overflowing the receiver's buffer, and the *congestion control mechanism* that prevents the sender from overloading the network.

2.1. The flow control algorithm

The TCP flow control is based on *explicit feedback*. In particular, the TCP receiver sends to the source the receiver's *advertised window*, which is the buffer available at the receiver. Let *MaxRcvBuffer* be the size of the receiver buffer in bytes, *LastByteRcvd* the last byte received and *NextByteRead* the next byte to be read. On the receiver side TCP must keep

$$LastByteRcvd - NextByteRead \leqslant MaxRcvBuffer$$

to avoid overflow. Therefore, the receiver advertises a window size (*AdWnd*) of

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

which represents the amount of free space remaining in the receiver buffer. The TCP on the send side computes an *Effective Window W*

$$W = AdWnd - (LastByteSent - LastByteAcked) \quad (1)$$

which limits how many outstanding packets it can send (Peterson & Davie, 2000).

2.2. The congestion control algorithm

Considering that the network is a "black box" that does not supply any explicit feedback to the source, the issue here is to look for "some measurement" of the network capacity that must be inferred at the end nodes using implicit feedback received from the networks such as timeout and acknowledgments. Today TCP estimates the best effort capacity of the network using a variable called *congestion window* (*cwnd*). In particular, the TCP learns the appropriate value of the *cwnd* by using an additive–increase/multiplicative–decrease (AIMD) paradigm. The increasing process goes through two phases: the *slow start* and the *congestion avoidance*. During the slow start phase the *cwnd* is exponentially increased until the *slow start threshold* (*ssthresh*) value is reached. This phase is intended to quickly grab available bandwidth. After the *ssthresh* value is reached, the *cwnd* is linearly increased to gently probe for extra available bandwidth. This phase is called *congestion avoidance*. At some point the TCP connection starts to lose packets. After a timeout *cwnd* is drastically reduced to one and the slow start, congestion avoidance cycle repeats. After three DUPACKs *cwnd* is reduced by half and the congestion avoidance phase is entered (Allman et al., 1999; Peterson & Davie, 2000).

To implement both flow and congestion control, the TCP sender computes the minimum of the congestion window and the advertised window and computes the *Effective Window W* as follows

$$W = MIN(Cwin, AdvWin) - OutstandingPackets, \quad (2)$$

where

$$OutstandingPackets = LastByteSent - LastByteAcked$$

represent the in-flight packets (Peterson & Davie, 2000).

3. Modeling a generic TCP flow

The goal of this section is to derive a mathematical model of the TCP, which, in control terms, corresponds to the plant of the system. To the purpose, we start by quoting Van Jacobson (1988): "A packet network is to a very good approximation a linear system made of *gains*, *delays* and *integrators*". In this paper, we propose a detailed model of a TCP/IP connection using (a) integrators to model network and receiver buffers and (b) delays to model propagation times.

A data network is a set of store-and-forward nodes connected by communication links. A generic TCP flow goes through a communication path made up of a series of buffers and communication links.

The number of packets of the considered TCP flow that are stored at the generic *i*th buffer along the communication path is given by the following dynamic equation:

$$x_i(t) = \int_{-\infty}^{t} [u_i(\tau) - b_i(\tau) - o_i(\tau)]\, d\tau, \quad (3)$$

where $u_i(t) \geqslant 0$ models the data arrival rate, $b_i(t) \geqslant 0$ models the data depletion rate, i.e. the used bandwidth, and $o_i(t) \geqslant 0$ models the overflow data rate, i.e. the data

that are lost when the buffer is full and the input rate exceeds the output rate.

The dynamic equation of the generic communication link $(i-1)$ connecting the $(i-1)$th buffer to the next (i)th buffer is a pure delay. In particular, letting $b_{i-1}(t)$ be the link input rate at the $(i-1)$th buffer and $u_i(t)$ be the link output rate at the next (i)th buffer, it results

$$u_i(t) = b_{i-1}(t - T_{i-1}), \tag{4}$$

where T_{i-1} is the link propagation time.

Starting from the basic equations (3) and (4), we propose to model a generic TCP flow over an IP network as it is shown in Fig. 2. In particular, Fig. 2 shows a functional block diagram made of:

(1) The TCP connection receiver buffer of length $x_r(t)$, which is modeled using an integrator with Laplace transfer function $1/s$. The receiver buffer receives the inputs $u_r(t)$, $b_r(t)$, $o_r(t)$, which represent the input rate, the depletion rate and the overflow data rate, respectively.
(2) The nth buffer that the TCP connection goes through before reaching the receiver buffer, which is modeled using an integrator with output $x_n(t)$. The nth buffer receives the inputs $u_n(t)$, $b_n(t)$, $o_n(t)$, which again represent the input rate, the depletion rate and the overflow data rate, respectively. It is important to notice that the depletion rate $b_n(t)$ reaches the next buffer $(n+1)$, which is the receiver buffer, after the propagation time T_n, i.e. $u_r(t) = b_n(t - T_n)$. Moreover, it should be noted that the input rate $u_n(t)$ is equal to the depletion rate $b_{n-1}(t)$ at the previous $(n-1)$th buffer, i.e. $b_{n-1}(t - T_{n-1}) = u_n(t)$, where T_{n-1} is the propagation time from the $(n-1)$th buffer to the nth buffer. Depletion rates are unpredictable because they model the best effort bandwidth available for a TCP connection when going over a statistically multiplexed IP network.

The series of buffers shown in Fig. 2 can be recursively augmented both in the left direction, to model up to the first buffer node encountered by the TCP connection, and in the right direction, to model buffers $n+j$, with $j = 2, p$, encountered by ACK packets when going back from the receiver to the sender.

By considering a closed surface that contains the TCP path going from the first to the last buffer modeled by a set of integrators indexed from 1 to $n+p=m$, where the mth integrator models the last buffer encountered by the TCP along the connection round trip, we can invoke the flow conservation principle for the unique input rate, which is the TCP input rate $u_1(t)$, and the output rates that are (a) $b_m(t)$, which models the bandwidth used by the TCP connection, i.e. the best-effort bandwidth as viewed by the considered TCP flow through the ACK stream; and (b) the overflow rates $o_i(t)$, for $i = 1, m$, which represent packets that are lost at each buffer along the path connection.

In equations, we can write the number $x(t)$ of packets belonging to the considered TCP flow and stored into the network by adding packets stored at each buffer along the path:

$$x(t) = \sum_{i=1}^{m} x_i(t). \tag{5}$$

Substituting (3) in (5) and considering the (4) it turns out that

$$x(t) = \int_{-\infty}^{t} \left[u_1(\tau) - b_m(\tau) - \sum_{i=1}^{m} o_i(\tau) - \sum_{i=1}^{m-1} (b_i(\tau) - b_i(\tau - T_i)) \right] \mathrm{d}\tau,$$

which can be rewritten as

$$x(t) = \int_{-\infty}^{t} \left[u_1(\tau) - b_m(\tau) - \sum_{i=1}^{m} o_i(\tau) \right] \mathrm{d}\tau - \sum_{i=1}^{m-1} \int_{t-T_i}^{t} b_i(\tau)\, \mathrm{d}\tau. \tag{6}$$

Eq. (6) states that the network storage is equal to the integral of the TCP input rate $u_1(t)$ minus the output rate $b_m(t)$ leaving the last buffer of the path, minus the sum of the overflow rates $o_i(t)$, minus the sum of packets that are in flight over each link i.

Since the TCP implements an end-to-end congestion control that does not receive any explicit feedback from the network, it is not possible for the controller to know terms in (6). Thus, we consider the sum of the in *flight packets* plus the *stored packets*, which we call the total

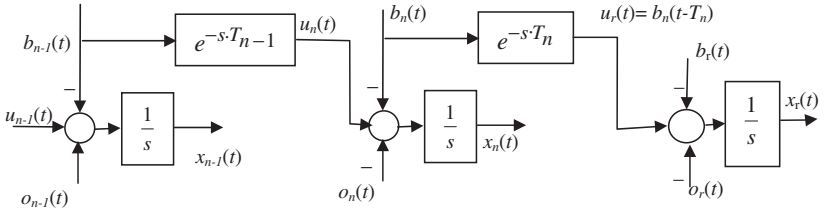

Fig. 2. Dynamic block diagram of a generic TCP/IP flow.

network storage x_t,

$$x_t(t) = x(t) + \sum_{i=1}^{m-1} \int_{t-T_i}^{t} b_i(\tau)\,\mathrm{d}\tau = \int_{-\infty}^{t} \left[u_1(\tau) - b_m(\tau) - \sum_{i=1}^{m} o_i(\tau) \right] \mathrm{d}\tau,$$

and the sum of overflow rates o_t,

$$o_t(t) = \sum_{i=1}^{m} o_i(t).$$

Thus, we can write

$$x_t(t) = \int_{-\infty}^{t} [u_1(\tau) - b_m(\tau) - o_t(\tau)]\,\mathrm{d}\tau. \tag{7}$$

By considering that the TCP establishes a "circular flow", i.e. that the data input rate comes back to the sender as an ACK rate, it can be said that $b_m(t)$ models the rate of ACK packets. Thus we can write

$$b_m(t) = u_1(t - T) - o_t(t), \tag{8}$$

which says, in mathematical words, that the ACK rate is equal to the input rate, delayed by the round trip time, minus the loss rate. By substituting (8) in (7) it turns out that

$$x_t(t) = \int_{-\infty}^{t} [u_1(\tau) - u_1(\tau - T)]\,\mathrm{d}\tau = \int_{t-T}^{t} u_1(\tau)\,\mathrm{d}\tau. \tag{9}$$

Eq. (9) states that the network total storage is equal to the integral of the input during the last round trip time T.

4. Modeling the TCP flow and congestion control

This section aims at showing that the closed loop control system depicted in Fig. 3, which enlights the "plant" and the "controller", models both the TCP flow and congestion control. Moreover, the stability properties of TCP flow and congestion control will be analyzed along with an interpretation of the self-clocking principle in terms of a Smith predictor controller (Smith, 1959).

In Fig. 3, the following variables and blocks are shown:

(1) The receiver queue length x_r and the receiver capacity r_1 provide the term $r_1 - x_r$ (i.e. the *Advertised Window*), which reaches the sender after the propagation time T_{fb} that is modeled in the Laplace domain by the transfer function $\mathrm{e}^{-sT_{fb}}$.
(2) The set point $r_2(t)$ represents a threshold for the total network storage, which is modeled by the queue $x_t(t)$.
(3) The minimum block takes the minimum between the *Advertised Window* and $r_2(t)$.
(4) Delays T_{1i} and T_{ir} model the time delay from the sender to the generic node i and from the node i to the receiver, respectively; the forward delay from the sender to the receiver is $T_{fw} = T_{1i} + T_{ir}$.
(5) The controller transfer function is

$$G(s) = \frac{k}{1 + k/s(1 - \mathrm{e}^{-sT})}, \tag{10}$$

which contains the proportional gain k and the SP $(1 - \mathrm{e}^{-sT})/s$, where T is the round trip time sum of the *forward* delay T_{fw} and the *backward* delay T_{fb}. The role of the SP is to overcome the delay T, which is inside the feedback loop and is harmful for the stability of the closed-loop control system (Åström & Wittenmark, 1997; Mascolo, 1999).

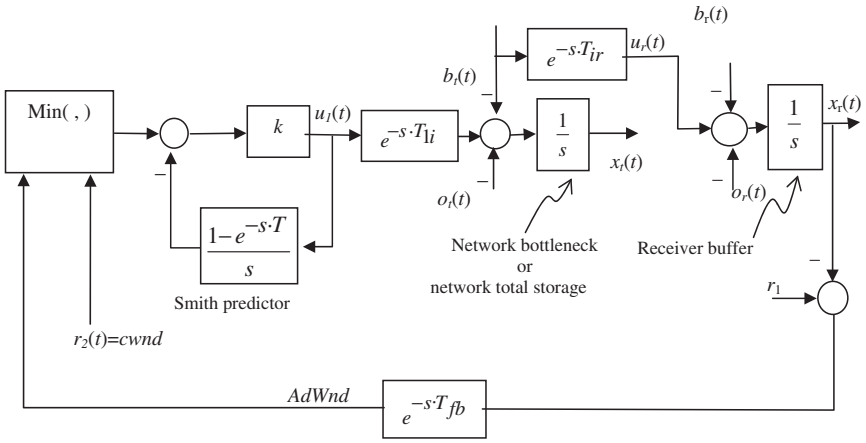

Fig. 3. Functional block diagram of the TCP flow and congestion control.

It is well-known that the SP can be successfully used when the delay and the plant dynamics are very well modeled, which is the case of TCP where buffers are perfect integrators and delays are exactly know via packet time stamping (Karn & Patridge, 1991). Moreover, it should be noted that: (a) the buffer x_t in Fig. 3 can model both the total network storage of packets but also it can model the generic buffer x_i that acts as the bottleneck of the TCP connection at time t; (b) a moving bottleneck is easily captured by the model through delays T_{1i} and T_{ir} where i is the generic moving bottleneck; and (c) the number of nodes the TCP flow goes through is taken into account by the total network storage x_t, which enlightens an important feature of TCP with respect to controllers placed at nodes such as the ones proposed in the context of ATM networks (see f.i. Mascolo, 2000; Tarbouriech, Abdallah, & Ariola, 2001).

In order to show that the block diagram in Fig. 3 models the TCP/IP flow and congestion control, first we will assume that the bottleneck is at the receiver and then that the bottleneck is inside the network.

4.1. The TCP flow control

By assuming that the bottleneck is at the receiver, it results in $\min(Adwnd, r_2(t)) = Adwnd$, $u_r(t) = u_1(t - T_{fw})$ and $o_t(t) = 0$. In other words, the connection is constrained by the receiver, and the input rate reaches the receiver after the forward delay without network queuing, i.e. $b_t(t) = u_1(t - T_{1i})$. Under these conditions, Fig. 3 can be transformed into Fig. 4 that models the TCP flow control. The following propositions can be shown.

Proposition 1. *The Smith controller* (10) *implements the TCP flow control equation* (1).

Proof. To find the input rate $u_1(t)$ computed by the TCP sender we use standard Laplace techniques, that is, we compute the Laplace transform of the input rate:

$$U_1(s) = [R_1(s) - X_r(s)]\mathrm{e}^{-sT_{fb}}\frac{k}{1 + k\left((1 - \mathrm{e}^{-sT})/s\right)},$$

which can be written as

$$U_1 = -kU_1\left(\frac{1 - \mathrm{e}^{-sT}}{s}\right) + k[R_1 - X_r]\mathrm{e}^{-sT_{fb}}.$$

By transforming back to time domain it results in

$$\frac{u_1(t)}{k} = r_1(t - T_{fb}) - x_r(t - T_{fb}) - \int_{t-T}^{t} u_1(\tau)\,\mathrm{d}\tau. \qquad (11)$$

By considering that

$$r_1(T - T_{fb}) - x_r(T - T_{fb}) = \text{advertised window}$$

and that

$$\int_{t-T}^{T} u_1(\tau)\,\mathrm{d}\tau = \text{outstanding packets}$$

Eq. (11) gives the classic window-based flow control equation (1), where $W = u_1(t)/k$. By considering that $u_1(t) = W/T$ relates the rate and the window of a window-based control, it results in $1/k = T$. □

Note that the *outstanding packets* automatically take into account the round trip time T that in general can be time varying due to queuing delays. In the case of flow control T is, to a very good approximation, constant since there is no congestion inside the network, which implies that network queuing delay is zero and round trip time is pure propagation delay.

Proposition 2. *The TCP flow control Eq.* (11) *guarantees that the receiver queue is always bounded by the receiver capacity* $\bar{r}_1$, *i.e.*

$x_r(t) < \bar{r}_1$ for any t.

Proof. The queue length can be computed by exploiting the superposition property of linear systems. In particular, it is easy to compute the input–output transfer function from $R_1(s)$ to the receiver queue length $X_r(s)$ that is

$$\frac{X_r}{R_1} = \frac{k}{k + s}\,\mathrm{e}^{-sT}$$

and the transfer function from $B_r(s)$ and $O_r(s)$ to $X_r(s)$ that is

$$\frac{X_r}{O_r + B_r} = -\frac{1}{s} + \frac{k}{s(s + k)}\,\mathrm{e}^{-sT} = -\frac{1}{s} + \frac{\mathrm{e}^{-sT}}{s} - \frac{\mathrm{e}^{-sT}}{s + k}.$$

By assuming $r_1(t) = \bar{r}_1 \cdot 1(t)$, where $\bar{r}_1$ is the receiver buffer capacity and $1(t)$ is the step function that models a connection starting at $t = 0$, it results $R_1(s) = \bar{r}_1/s$ results. By exploiting the superposition property of linear systems and by transforming back to time domain

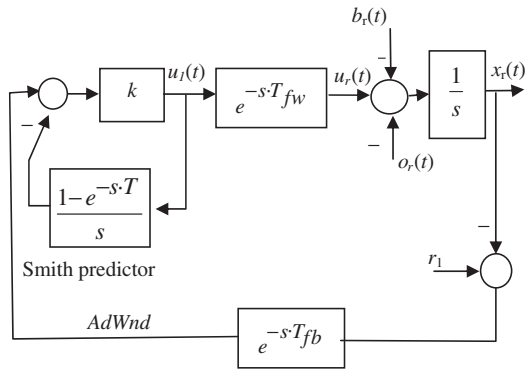

Fig. 4. Functional block diagram of the TCP flow control.

it results:

$$x_r(t) = L^{-1}\left\{\frac{\bar{r}_1}{s}\frac{k}{s+k}\,\mathrm{e}^{-sT}\right\} - L^{-1}\left\{\frac{B_r + O_r}{s+k}\,\mathrm{e}^{-sT}\right\} - \int_{t-T}^{t}[b_r(\tau) + o_r(\tau)]\,\mathrm{d}\tau,$$

which satisfies the condition

$$x_r(t) \leqslant L^{-1}\left\{\frac{\bar{r}_1}{s}\frac{k}{s+k}\,\mathrm{e}^{-sT}\right\} = \bar{r}_1 \cdot (1 - \mathrm{e}^{-k(t-T)}) \cdot 1(t-T) < \bar{r}_1$$

since $o_r(t)$, $b_r(t)$ are always nonnegative. This concludes the proof. □

Lemma 1. *Proposition* 2 *guarantees* $o_r(t) = 0$ *for any* t.

Proof. Proposition 2 proves that the receiver queue length is always upper bounded by the receiver queue capacity, which implies that receiver overflow is always avoided, i.e. $o_r(t) = 0$ for any t.

4.2. The TCP congestion control

By assuming that the bottleneck is localized inside the network, there results $\min(\mathrm{Adwnd}, r_2(t)) = r_2(t)$ and we can ignore the outer feedback loop. Therefore, Fig. 3 can be transformed into the equivalent one shown in Fig. 5, which models the TCP congestion control.

Proposition 3. *The Smith controller* (10) *implements the TCP congestion control Eq.* (2).

Proof. By assuming that the bottleneck is inside the network, there results $\min(\mathrm{Adwnd}, r_2(t)) = r_2(t)$. From Fig. 5, the output of the Smith predictor in the Laplace domain is

$$Q(s) = U_1(s)\frac{1 - \mathrm{e}^{-sT}}{s}.$$

By transforming back to time domain it results

$$q(t) = \int_{t-T}^{T} u_1(\tau)\,\mathrm{d}\tau = \mathit{outstandingpackets}.$$

Therefore the output of the controller is

$$u_1(t) = k(r_2(t) - \mathit{outstandingpackets}), \tag{12}$$

which can be rewritten as

$$\frac{u_1(t)}{k} = r_2(t) - \mathit{outstandingpackets}. \tag{13}$$

Eq. (13) gives the classic window-based congestion control Eq. (2), where $W = u_1(t)/k$, and $r_2(t) = \mathit{cwnd}$. This concludes the proof. □

Remark 1. It should be noted that (12) and (13) are the rate-based and window-based versions of the same control equation.

Proposition 4. *The TCP congestion control eq.* (13) *guarantees a total network storage* x_t *that is always bounded by the threshold* $r_2(t) > 0$, *i.e.*

$x_t(t) \leqslant r_2(t)$ for any t.

Proof. From (9), the total network storage is

$$x_t(t) = \int_{t-T}^{t} u_1(\tau)\,\mathrm{d}\tau = q(t).$$

Since $u_1(t)$ and $q(t)$ are always nonnegative, and $r_2(t)$ is strictly positive, from the control law

$$u_1(t) = k\left(r_2(t) - \int_{t-T}^{T} u_1(\tau)\,\mathrm{d}\tau\right),$$

it turns out that $r_2(t) \geqslant q(t) = x_t(t)$, which concludes the proof. □

Lemma 2. *If a TCP flow finds in each buffer it goes through a space of* c_i *packets, where* $c_i > r_2(t)$ *for any* t *and* i, *then Proposition* 4 *guarantees* $o_t(t) = 0$ *for any* t.

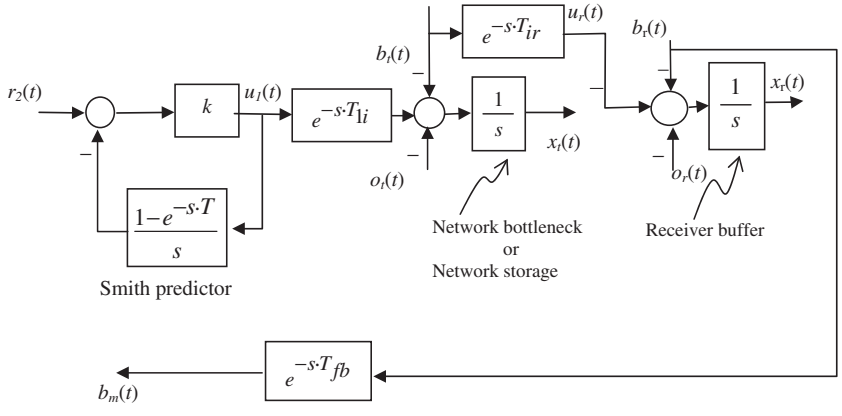

Fig. 5. Functional block diagram of the TCP congestion control.

Proof. The proof follows from Proposition 4, which proves that $x_t(t) \leqslant r_2(t)$, and assumptions of Lemma 2. □

5. The self-clocking principle

Based on the theoretical control framework outlined in the previous section, we are now ready to show that the *self-clocking* principle can be theoretically explained using the SP. It is largely known that the *self-clocking* principle is a key feature of the TCP congestion and flow control (Jacobson, 1988). This has been recently recognized also in the context of Transport Friendly Rate Control (TFRC) algorithms (Bansal, Balakrishnan, Floyd, & Shenker, 2001), where it has been shown that algorithms that do not employ the self-clocking principle may exhibit a huge settling time, that is, they may require many RTTs to adapt the input rate to the bandwidth available in the network. As a consequence, to overcome the disastrous effects due to the violation of the self-clocking principle, the original TFRC has been enhanced with the self-clocking mechanism. In this section, we show that the *self-clocking* can be mathematically interpreted as the effect of the SP branch.

Proposition 5. *The SP branch* $(1-\mathrm{e}^{-sT})/s$ *enforces the self-clocking principle.*

Proof. By transforming back to time domain the quantity $q(s) = U_s(s)(1-\mathrm{e}^{-sT})/s$, it results in

$$q(t) = \int_0^t u_s(\tau)\,\mathrm{d}\tau - \int_0^{t-T} u_s(\tau)\,\mathrm{d}\tau = \int_{t-T}^{t} u_s(\tau)\,\mathrm{d}\tau,$$

where $q(t)$ represents the data that have been sent since the last round trip time T up to now, i.e. the outstanding packets. When the time t advances of Δ, i.e. at time $t+\Delta$, the amount of data

$$q_{acked} = \int_{t-T}^{t-T+\Delta} u_s(\tau)\,\mathrm{d}\tau$$

are acknowledged so that the control equation (3) or (5) can send this amount of data in the time interval $[t, t+\Delta]$, that is, the self-clocking principle is enforced.

6. Modeling Reno or Westwood TCP by input shaping

In this section we show that the dynamic model depicted in Fig. 5 is able to model successful variants of TCP congestion control, such as for example Tahoe/Reno (Jacobson, 1988) or the recent Westwood TCP (Mascolo et al., 2001). Other TCP variants, such as Vegas or Santa Cruz, could also be modeled in the same unified framework.

We have seen that the congestion control algorithm aims at estimating the available bandwidth using a probing mechanism. The classic TCP probing mechanism, which is currently used in all successful variants of the TCP such as Tahoe/Reno, New Reno or Westwood, comprises two mechanisms: the *slow-start* phase, which exponentially increases the congestion window up to the *ssthresh*, and the *congestion avoidance* phase which linearly increases the *cwnd* when $cwnd \geqslant ssthresh$. Now, we show that both these mechanisms can be modeled in the control theoretical framework reported in Fig. 5 by properly shaping the controller input $r_2(t) = cwnd$.

6.1. The Reno algorithm

The TCP Reno *slow-start* phase can be modeled by setting the reference input $r_2(t)$ as follows:

$$r_2(t) = r_0 \cdot 2^{t/T} \quad \text{while } r_2(t) < ssthresh,$$

where the initial window r_0 is generally equal to 1 or 2 (Allman, Floyd, & Partridge, 1998). TCP Reno enters the *congestion avoidance* phase when $r_2(t) = ssthresh$ at $t_1 = T\log_2(ssthreh - r_0)$. This phase can be modeled by setting the reference input $r_2(t)$ as follows:

$$r_2(t) = ssthresh + \frac{t-t_1}{T} \quad \text{when } r_2(t) \geqslant ssthresh.$$

The TCP probing phase ends when 3 DUPACKSs are received or a timeout happens, which indicates that the network capacity has been hit. In these cases the *cwnd* behavior can be modeled using the following settings for $r_2(t)$:

After a timeout at t_k

$$ssthresh = \max\left(\frac{r_2(t)}{2}, r_0\right),$$

$$r_2(t) = r_0,$$

$$r_2(t) = r_0 \cdot 2^{t-t_k/T} \quad \text{if } r_2(t) < ssthresh,$$

$$r_2(t) = ssthresh + \frac{t-t_k}{T} \quad \text{if } r_2(t) \geqslant ssthresh.$$

After 3 DUPACKs at t_k

$$ssthresh = \max\left(\frac{r_2(t)}{2}, r_0\right),$$

$$r_2(t) = ssthresh + \frac{t-t_k}{T}.$$

6.2. The Westwood algorithm

TCP Westwood employs the same probing mechanism of Reno. It differs from Reno because of the behavior after congestion. In fact, Westwood sets the

cwnd and *ssthresh* using an end-to-end estimate of the network bandwidth $b_m(t_k)$ available at the time of congestion. In particular, the Westwood TCP window behavior after congestion can be modeled as follows:

After a timeout at t_k

$$ssthresh = b(t_k) \cdot RTT_{\min},$$

$$r_2(t) = r_0,$$

$$r_2(t) = r_0 \cdot 2^{t-t_k/T} \quad \text{if } r_2(t) < ssthresh,$$

$$r_2(t) = ssthresh + \frac{t-t_k}{T} \quad \text{if } r_2(t) \geqslant ssthresh.$$

After 3 DUPACKs at t_k

$$ssthresh = b(t_k) \cdot RTT_{\min},$$

$$r_2(t) = ssthresh + \frac{t-t_k}{T}.$$

7. Why a PID controller is not efficient to control the Internet

We have seen that the Internet flow and congestion control problem reduces to the issue of controlling an integral mode with a time delay in cascade. It can be said that, in general, congestion control in data networks consists of controlling a time-delay system.

The proportional-integral-derivative (PID) controller is by far the most common control algorithm and performs satisfactorily well in many practical cases (Ästrom & Hägglund, 1995).

In this section, we show that a standard PID cannot satisfactorily control a data network such as the Internet since in order to provide stability for the closed-loop system it is necessary to use a low gain that turns out an unacceptable sluggish system.

To start the discussion we consider the simple proportional controller k shown in Fig. 6. In order to study the stability of this system, we invoke the Nyquist stability criterion. To the purpose, the polar diagram of the open loop-transfer function $(k/s)e^{-sT}$ is depicted in Fig. 7, which also shows the vertical asymptote of abscise $-kT$ and the circle with unity magnitude.

For the Nyquist stability criterion, the polar plot must not encircle the point -1. The crossover frequency ω_0

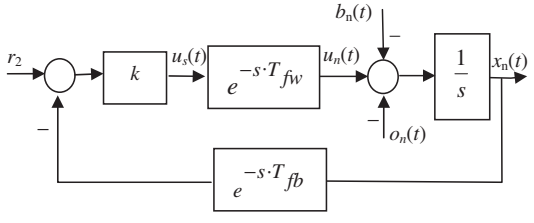

Fig. 6. A data flow controlled by a proportional controller.

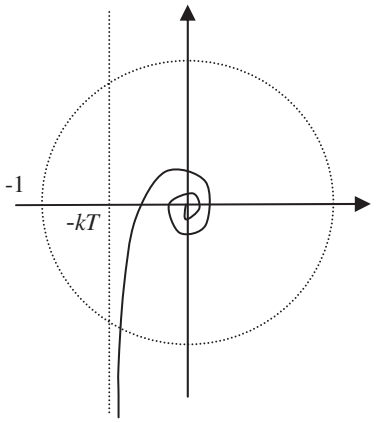

Fig. 7. Nyquist plot of the open loop-transfer function.

(frequency for which the magnitude of the loop frequency response is unity) is obtained from the equality

$$\left| \frac{k}{j\omega_0} e^{-j\omega_0 T} \right| = 1,$$

which results in $\omega_0 = k$.

The closed-loop system is stable if the phase margin

$$M_F = \pi - \pi/2 - \omega_0 T = \pi/2 - kT$$

is positive, which results in the stability condition

$$k < k_u = \pi/2T, \tag{14}$$

where k_u is the ultimate gain.

The stability condition (14) states that the gain k must be lower and lower with increasing round trip time T. As a consequence, the proportional controller provides a too sluggish closed-loop behavior in the case of data networks which are characterized by large propagation delays, such as in the case of high-speed networks, wide area networks or satellite connections.

The use of a PD controller does not help much to improve the promptness of the controller (Ästrom & Hägglund, 1995). In fact, when using a PD, the open-loop transfer function becomes $(k(1+s\tau_D)/s)e^{-sT}$. The crossover frequency is now $\omega_1 = k/\sqrt{1-(k\tau_D)^2} > \omega_0$. Thus, even though the derivative action adds the positive contribute $\text{arctg}(\omega_1\tau_D)$ to the phase margin, it must be considered that at the new cross-over frequency ω_1 the negative contribute to the phase margin due to the time delay is now increased of $(\omega_1 - \omega_0)T$. Thus, in the presence of large delay T, the lead action of a PD controller may not be useful or may be even pejorative of the stability margin because it may happen that $(\omega_1 - \omega_0)T > \text{arctg}\ \omega_1\tau_D$.

Finally, the integral action of the standard PID controller is surely not recommended for the system we are considering because it would reduce the phase

margin of $\pi/2$ at any frequency, thus making the system stability even more critical.

To get a further insight we compare the proportional controller with a proportional controller plus a SP using computer simulations. We consider a connection with $T = 1000$ units of time. From (14), the ultimate gain is $k_u = 0.001571$. Following the Ziegler–Nichols rules, we choose the proportional gain $k = k_u/2 = 0.000785$. We have also tried the Ziegler–Nichols rules for tuning a PI and a PID controller but in this case we have found that they do not provide a stable system, which confirms that it is not easy or efficient to control a system with large delays using a PI or a PID controller.

We set the reference signal $r(t) = 10000 \cdot 1(t)$, which corresponds to setting a queue threshold of 10000 packets at the initial time $t = 0$. We assume an available bandwidth of 4 pkts/unit of time. This pattern is very appropriate to test the promptness of the congestion control algorithm in matching the time-varying available bandwidth. Fig. 8 shows the input rate dynamics obtained using a proportional controller (P) and a proportional controller plus a Smith predictor (P + SP) both with $k = 0.000785$. It shows that the input rate obtained using the SP is much faster in reaching the steady state value of 4 pkts/unit of time. Moreover the Smith predictor provides a much smaller band of oscillation for the input rate. Fig. 9 shows that the SP provides a much more smaller queue length that is a very important feature for networks since it means much smaller queuing delays.

It is worth noting that another important advantage of the SP controller is that system dynamics can be made faster by increasing the proportional gain k without risking instability.

To conclude, we look at results of this section from the perspective provided by Proposition 5. In particular, we observe that a controller without an SP, such as a PID, does not implement the self-clocking principle. Therefore, as it has been also noted in (Bansal et al., 2001), it may exhibit a huge settling time, that is, it may require many RTTs to adapt the input rate to the bandwidth available in the network.

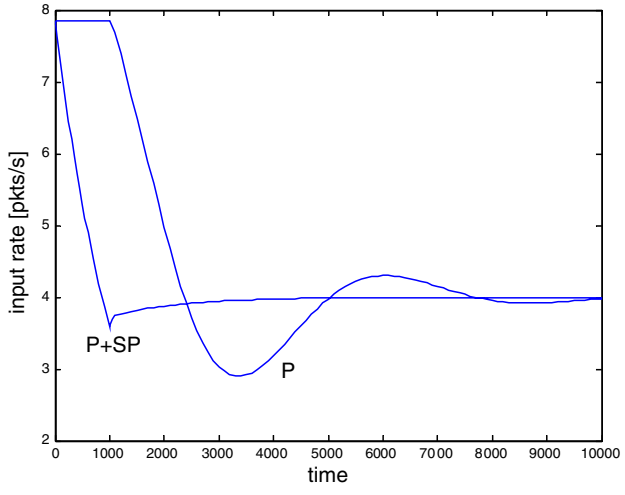

Fig. 8. Input rate using a P or P + SP controller.

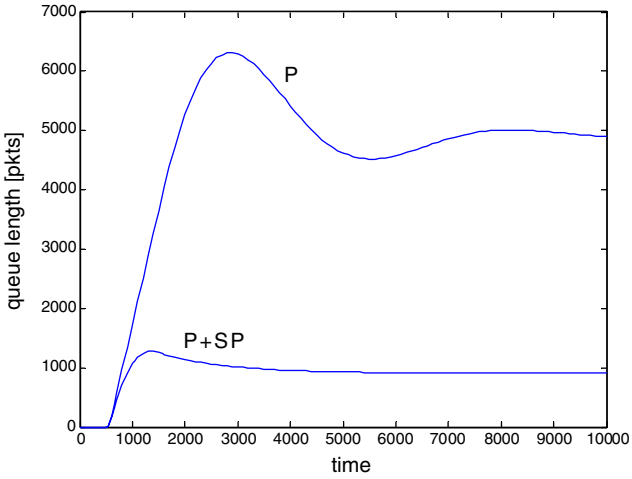

Fig. 9. Queue length using a P or P + SP controller.

8. Conclusions

In this paper, the TCP congestion and flow control have been modeled as a time-delay system controlled using dead-time compensation. We have shown that a proportional controller plus a Smith predictor provides an exact model of the Internet flow and congestion control. In particular, we have shown in the following: (1) enforcing the self-clocking principle corresponds to implementing the Smith predictor; (2) the Smith predictor controller guarantees stability and provides efficient congestion control; (3) different TCP congestion control algorithms, such as the classic TCP Reno or the recent Westwood TCP, can be modeled by shaping the reference input. Finally, we have shown that controllers that do not implement the Smith predictor, such as PID controllers, provide an unacceptable sluggish system because they do not implement the self-clocking principle.

Acknowledgments

I would like to thank Prof. Marco Ajmone Marsan and the MIUR-FIRB Project no. RBNE01BNLS "Traffic models and Algorithms for Next Generation IP networks Optimization (TANGO)" for supporting this work.

References

Allman, M., Floyd, S., & Partridge, C. (1998). *Increasing initial TCP's initial window*. RFC 2414.

Allman, M., Paxson, V., & Stevens, W. R. (1999). *TCP congestion control*. RFC 2581.

Åstrom, K., & Hägglund, T. (1995). *PID controllers: Theory, design, and tuning*. ISA.

Åström, K. J., & Wittenmark, B. (1997). *Computer controlled systems*. Englewood Cliffs, N.J.: Prentice-Hall.

Bansal, D., Balakrishnan, H., Floyd, S., & Shenker, S. (2001). Dynamic behavior of slowly-responsive congestion control algorithms. *Proceedings of sigcomm*, 2001.

Brakmo, L. S., O'Malley, S. W., & Peterson, L. L. (1995). TCP vegas: End-to-end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications (JSAC)*, *13*(8), 1465–1480.

Clark, D. (1988). The design philosophy of the DARPA Internet protocols. In *Proceedings of sigcomm' 88*. In: *ACM Computer Communication Review*, *18*(4), 106–114.

Dah-Ming Chiu, & Jain, R. (1989). Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, *17*(1), 1–14.

Fast TCP at http://netlab.caltech.edu/FAST/

Floyd, S. (2003). *Highspeed TCP for large congestion windows*. IETF Internet draft draft-ietf-tsvwg-highspeed-00.txt, work in progress.

Floyd, S., & Fall, K. (1999). Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, *7*(4), 458–472.

Floyd, S., & Henderson, T. (1999). *Newreno Modification to TCP's fast recovery*. RFC 2582.

Gerla, M., Locigno, R., Mascolo, S., & Weng, R. (2002). *Generalized window advertising for TCP congestion control, European Transactions on Telecommunications*, *6*.

Grieco, L. A., & Mascolo, S. (2002). TCP Westwood and easy RED to improve fairness in high-speed networks. *Proceedings of the VII international workshop on protocols for high-speed networks (PfHSN'2002)*, April 2002, Berlin, Germany. *Lecture notes on computer science* (*Lcns*). Berlin: Springer.

Grieco, L. A., & Mascolo, S. (2004). Performance evaluation and comparison of Westwood+, new Reno, and Vegas TCP congestion control. *ACM Computer Communication Review*, *34*(2).

Hoe, J. C. (1996). *Improving the start-up behavior of a congestion control scheme for TCP, Proceedings of ACM sigcomm'96*. (pp. 270–280).

Hollot, C. V., Misra, V., Towsley, D. F., & Wei-Bo Gong (2002). Analysis and design of controllers for AQM Routers supporting TCP flows. *IEEE Transactions on Automatic Control*, *47*(6), 945–959.

Jacobson, V. (1988). Congestion avoidance and control. *ACM Computer Communications Review*, *18*(4), 314–329.

Jacobson, V., Braden, R., & Borman, D., (1992). *TCP extensions for high performance*. RFC 1323.

Karn, P., & Partridge, C. (1991). Improving round-trip time estimates in reliable transport protocols. *ACM Transaction on Computer Systems*, *9*(4), 364–373.

Kelly, F. P. (1999). Mathematical modeling of the internet. *Proceedings of fourth international congress on industrial and applied mathematics*, July 1999.

Lakshman, T. V., & Madhow, U. (1997). The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, *5*(3).

Low, S. H. (2000). A duality model of TCP flow control. *Proceedings of ITC specialist seminar on IP traffic measurements, modeling and management*, September 2000.

Low, S. H., Paganini, F., & Doyle, J. C. (2002). Internet congestion control. *IEEE Control Systems Magazine*, *22*, 28–43.

Mascolo, S. (1999). Congestion control in high-speed communication networks using the Smith principle. *Automatica*, *35*(12).

Mascolo, S. (2000). Smith's principle for congestion control in high speed data networks. *IEEE Transactions on Automatic Control*, *45*(2), 358–364.

Mascolo, S., Casetti, C., Gerla, M., Sanadidi, M., & Wang, R. (2001). TCP Westwood: End-to-End bandwidth estimation for efficient transport over wired and wireless networks. Proceedings of the ACM Mobicom 2001 Conference, July, Rome, Italy.

Parsa, C., & Garcia-Luna-Aceves, J. J. (1999). Improving TCP congestion control over internets with heterogeneous transmission media. *Proceedings of IEEE international conference on network protocols*, Toronto, October 31–November 3, 1999.

Peterson, L. L., & Davie, B. S. (2000). *Computer networks*. San Francisco, CA: Morgan Kaufmann.

Smith, O. (1959). A Controller to overcome dead time. *ISAJ*, *6*(2), 28–33.

Tarbouriech, S., Abdallah, C. T., Ariola, M. (2001). Bounded control of multiple-delay systems with applications to ATM networks. *40th IEEE conference on decision and control*, Orlando, USA (pp. 2315–2320).

Villamizar, C., & Song, C. (1995). High performance TCP in ANSNET. *ACM Computer Communication Review*, *24*(5), 45–60.