A Hybrid Systems Framework for TCP Congestion Control: A Theoretical Model and its Simulation-based Validation*

Technical Report

Stephan Bohacek† João P. Hespanha‡ Junsoo Lee†† Katia Obraczka§

bohacek@math.usc.edu hespanha@usc.edu junsoole@usc.edu katia@isi.edu

†*Department of Mathematics, Univ. of Southern California*
Los Angeles, CA 90089-1113

‡*Dept. Electrical Engineering–Systems, Univ. of Southern California*
Los Angeles, CA 90089-2563

††*Computer Science Department, Univ. of Southern California*
Los Angeles, CA 90089

§*Computer Engineering Department, University of California*
Santa Cruz, CA 95064

ABSTRACT

In this paper we propose the use of hybrid models to provide a description of the transient and steady-state behavior of multiple TCP flows that share a single common bottleneck link. When compared to other approaches to model TCP congestion control, our models provide a more complete description of TCP's behavior including the effect of queuing, interaction among competing flows, and finite advertised window size.

Using a hybrid model of TCP-Reno, we are able to derive well-known relationships among congestion control parameters—such as the formula $T := \frac{1.23}{\overline{RTT}\sqrt{p}}$, which relates the average throughput T, the average round-trip time $\overline{RTT}$, and the average packet drop rate p. Besides allowing us to characterize TCP's behavior more thoroughly, our hybrid models can also theoretically predict the flow synchronization phenomenon that has been previously observed experimentally. Under certain conditions, the window sizes and sending rates of all competing flows will synchronize exponentially fast at a rate of 2^{-k}, where k is the number of drops experienced by a flow.

Besides flow synchronization, our models also predict other TCP congestion control pathologies, such as unfairness, which previous models based on a single-flow analysis fail to capture. In this paper we also present mechanisms to mitigate both synchronization and unfairness. We present simulation studies that validate our theoretical predictions.

*This research was supported by the Defense Advanced Research Projects Agency and the Office of Naval Research. The views presented here are those of the authors and do not represent the views of the funding agencies.

1. INTRODUCTION

For the past decade, TCP congestion control mechanisms have been under the scrutiny of the network research community. The existence of several versions of TCP such as TCP-Tahoe, Reno, Vegas, New Reno, and Selective Acknowledgement (SACK) is evidence of the attention TCP has received over the years. More recently, motivated by the increased popularity of multimedia services, several efforts have been investigating *TCP-friendly* approaches to congestion control [1, 2, 3, 4]. One goal of TCP-friendly congestion control is to avoid the large window size variations that may be experienced by TCP flows and, at the same time, be able to coexist with TCP in a mutually fair way.

Our work on TCP congestion control was originally motivated by trying to make TCP more robust to intrusion attacks. To that effect, we set forth at trying to derive an analytical model of TCP to help us determine TCP's baseline behavior and consequently, identify potential attacks.

1.1 Yet another model of TCP?

Similarly to previous work, we model TCP when operating over a *dumbbell* topology (see Section 1.2 and Figure 1). However, unlike the single-flow analysis conducted by other TCP modeling efforts, our model considers multiple TCP flows over the dumbbell topology.

We approach the TCP congestion control problem from a control-theoretic point-of-view. More specifically, we use a hybrid systems framework which allows us to theoretically derive specific properties of TCP without the need to make oversimplifying, often unrealistic, assumptions. Hybrid systems [5, 6, 7, 8, 9, 10, 11, 12, 13, 14], which to the best of our knowledge have not yet been employed to investigate TCP, are formal models that combine both continuous time dynamics and discrete-time logic. These models permit complexity reduction through the continuous approximation of variables like queue and congestion window sizes, without compromising the expressiveness of logic-based models. For the specific case of modeling TCP congestion control, when no drops occur, all variables are treated as continuously varying variables. When a drop occurs, logic-based models dictate the discrete transitions of the state variables. A second significant reduction in complexity is achieved by normalizing time. Because TCP congestion control is window-based, its sending rate increases at a rate inversely proportional to the round-trip time. However, through a change of the time variable, the congestion window increases linearly with respect to this new variable. This novel combination of hybrid system modeling and time normalization permits us to derive well-known results (which, to some extent, validate our model), as well as new characterizations of TCP traffic.

In this paper, we develop a detailed understanding of TCP running on the dumbbell topology. For example, in the case of a drop-tail queue, we prove that, under certain conditions, the window sizes and sending rates will synchronize exponentially fast at a rate of 2^{-k} where k is the number of drops experienced by a flow (Theorem 1). To the best of our knowledge, this is the first formal proof that an optimal and fair state of TCP is exponentially stable. While synchronization of TCP flows has long been recognized [15, 16]—in the form of in-phase periodic variations of the sending rates of competing flows—, a formal proof had not been provided. Neither had the conditions for flow synchronization. For example, an apparently neglected condition for the stability of synchronization is that there must be a significant mismatch between the bandwidths of the links leading to the bottleneck link and the bandwidth of the bottleneck link. If this "bandwidth mismatch assumption" is not met, the flows will not synchronize. Also, for synchronization to occur the advertised windows must be large (sufficiently large to have no effect on the congestion window). However, as discussed in Section 5, if the advertised window assumption does not hold, then pathological situations can occur, e.g., one TCP flow ending up utilizing far more bandwidth than the competing flows. Indeed, it is possible that a sophisticated algorithm could fully adhere to the principles of TCP and yet acquire an unfairly large amount of bandwidth by manipulating its advertised window size.

Many of the recently proposed TCP-friendly algorithms are based on the well known relationship

$$T = \frac{1.23}{\overline{RTT}\sqrt{p}} \tag{1}$$

where T is the average throughput, $\overline{RTT}$ the average round-trip time, and p the average drop rate [17, 18, 19, 20]. Our hybrid systems model provides an alternative derivation of this relationship. One key difference is that our derivation did not need to make various simplifying assumptions found in previous work. For example, we do not assume that the round-trip time is constant. As it is well known, the round-trip time plays an important role in TCP: when the queue fills, the round-trip time increases and the TCP congestion window increases more slowly. In essence, the round-trip time has a stabilizing effect on the TCP flows, even *before* a drop has occurred. Interestingly enough, we observe that the relationship (1) between throughput, drop rate, and round-trip time is essentially unchanged when the variation of the round-trip time is included.

Our hybrid systems model also allows for the derivation of many other properties of competing TCP flows. For example, various relationships between the number of flows, the drop probability, the round-trip time, and the time between drops are derived (Theorem 2). With this level of detail about TCP's behavior, a source can, for example, anticipate congestion and temporarily increase the level of error correction.

1.2 Dumbbell Topology

In the topology shown in Figure 1, n persistent TCP flows are generated at a source node n_1 and are directed towards a sink node n_2. All the flows compete for the finite bandwidth B that characterizes the *bottleneck link* ℓ connecting the nodes. This configuration is known as the *dumbbell topology* and is typically used to analyze TCP congestion control. In more realistic networks, a path of several links (and intermediate nodes) would connect the source and destination. However, to analyze congestion control mechanisms, one often ignores the existence of all the intermediate links, except for the bottleneck link, i.e., the most congested link.

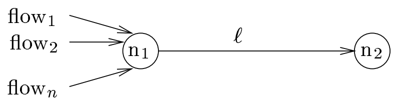


Figure 1: Dumbbell topology

This topology has been well studied in the context of TCP congestion control [21, 16, 22, 17, 20, 23, 2, 24, 25, 26] mainly for the two following reasons. First, for congestion control analysis purposes, it is believed to be (to some extent) representative of the behavior of a more general topology. In fact, while TCP behaves differently in different topologies, some characteristics are common to many of them and, hence, a complete understanding of how TCP behaves in one topology provides insight into how it might behave in other topologies. Second, analysis of more complicated topologies quickly becomes intractable. This problem of intractability has been accommodated by relying purely on simulation or, for theoretical work, on strong and often unrealistic assumptions.

In this investigation, the framework of hybrid systems is utilized to theoretically determine specific properties of TCP without the use of overly simplifying assumptions. These properties are then validated through packet-level simulations performed using the network simulator `ns-2` [27].

1.3 Outline

The remaining of this paper is organized as follows. The next section details the hybrid systems framework and defines a simplifying normalization of the temporal variable. In Section 3, the hybrid systems methodology is used to analyze TCP-Reno under a drop-tail queuing discipline. This section contains the main theoretical results: exponential stability and steady-state characterizations. Section 4 focuses on the issue of flow synchronization. As noted above, in the case of a small advertised window, synchronization can lead to drastic unfairness. The exact mechanisms for this unfairness is detailed in Section 5. Finally, Section 6 provides a summary of the results, some concluding remarks, and our future work plans.

2. HYBRID SYSTEM MODELING FOR CONGESTION CONTROL

We start by describing how TCP's congestion control mechanism can be modeled using a hybrid system, i.e., a system that combines continuous dynamics with discrete logic.

We consider here a simplified version of TCP-Reno congestion control [28, 29, 30][1] but the model proposed also applies to more recent variations on Reno such as New Reno, SACK [22], and general AIMD [31]. We denote by w_i, $i \in \{1, 2, \dots, n\}$, the window size of the congestion controller associated with the ith flow. In Reno, the algorithm to update w_i is as follows: While no drops occur, the window size is incremented by a fixed constant $a \geq 1$ for every w_i acknowledgments received (typically $a = 1$) and, when it is detected that a drop occurred, the window size is multiplied by a constant $m \in (0, 1)$ (typically $m = 1/2$). Let us call *round-trip time*, denoted by RTT, the time interval measured from the moment a packet is sent until an acknowledgment for that packet is received. As we will see shortly, the round-trip time is a time-varying quantity. Assuming that there are no drops, during each round-trip time the window size increases by a. Although the window size takes discrete values, it is convenient to regard it as a continuously varying variable. The following hybrid model provides a good approximation of the ith window size dynamics: While the ith flow suffers no drops we have

$$\dot{w}_i = \frac{a}{RTT}, \tag{2}$$

and, if a drop is detected on this flow at time t, we have

$$w_i(t) = m\, w_i^-(t),$$

where $w_i^-(t)$ denotes the limit from below of $w_i(s)$ as $s \uparrow t$. The round-trip time is given by

$$RTT(t) = T_p + \frac{q(t)}{B},$$

where T_p denotes the *propagation time* (together with some fixed component of the service time at nodes n_1 and n_2) and $q(t)$ is the size of the output queue of node n_1 at time t. We assume here that the bandwidth B is measured in packets per second. Denoting by w_i^{adv} the advertised window size for the ith flow, the output queue at node n_1 receives a total of

$$r := \frac{\sum_i \min\{w_i, w_i^{\mathrm{adv}}\}}{RTT}$$

packets per second and is able to send B packets to the link in the same period. The difference between these two quantities determines the evolution of $q(t)$. In particular,

$$\dot{q} = \begin{cases} 0 & q = 0, r < B \quad \text{or} \quad q = q_{\max}, r > B \\ r - B & \text{otherwise} \end{cases} \tag{3}$$

The first branch in (3) takes into account that the queue size cannot become negative nor should it exceed the *maximum queue size* $q_{\max}$. When $q(t)$ reaches $q_{\max}$ drops occur. These will be detected by the congestion controllers some time later.

As mentioned above, drops will occur whenever q reaches the maximum queue size $q_{\max}$, and the rate of incoming packets to the queue r exceeds the rate B of outgoing packets. Since a drop will only be detected after one round-trip time, the rate of incoming packets will not change for a period of length RTT and multiple drops are expected. To complete our model we need to known which flows will suffer a drop during this interval. To determine exactly the set of flows $\mathcal{D} \subset \{1, 2, \dots, n\}$ that suffer a drop, one would need to keep track of which packets are in the queue, leading to a complex packet-level model. However, for purposes of traffic flow analysis, it is sufficient to assume that $\mathcal{D}$ is a function of the window sizes of the individual flows (w_i). Denoting by $\mathcal{D}(t)$ the set of flows that suffer drops at time t, we have

$$\mathcal{D}(t) = F_{\mathrm{drop}}(w_1, w_2, \dots, w_n). \tag{4}$$

We call F_{drop} the *drop model.* As we shall see below, several drop models are possible, depending on the queuing discipline.

Although drops are an essentially discrete phenomena, we can incorporate then in our hybrid model by considering two modes for the system: one *queue-not-full* mode where the system evolves according to (2)–(3), and one *queue-full* mode where drops will occur on the flows in $\mathcal{D}$. The latter mode is active for RTT seconds. When the system leaves this mode the window sizes in all the flows in $\mathcal{D}$ are multiplied by m because of the multiplicative decrease caused by the drops. In reality, the multiplicative decrease of all flows in $\mathcal{D}$ does not occur exactly at the same time instant. However, this model provides a very good approximation for the time scales considered here.

Figure 2 contains a graphical representation of the resulting hybrid system. Each ellipse in this figure corresponds to a discrete mode and the continuous state of the hybrid system consists of the queue size q, the window sizes w_i, $i \in \{1, 2, \cdots, n\}$, and a timing variable t_{tim} used to enforce that the system remains in the *queue-full* mode for exactly RTT seconds. The differential equations for these variables in each discrete mode are shown inside the corresponding ellipse. For simplicity we assume here that the queue size q never reaches zero. The arrows in the figure represent discrete transitions between modes. These transitions are

[1] We are ignoring Reno's initial adjustment of the window size—known as *slow start*—because it has little impact on the system after a brief initial period.

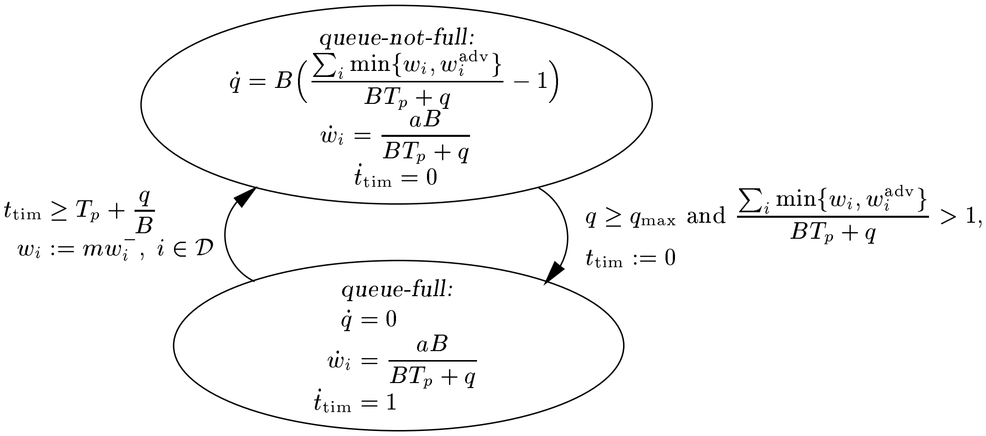


Figure 2: Hybrid model for TCP-Reno.

labeled with their enabling conditions and any necessary reset of the continuous state that must take place when the transition occurs. We assume here that a jump always occurs when the transition condition is enabled. This model is consistent with most of the hybrid system frameworks proposed in the literature [5, 6, 7, 8, 9, 10, 11, 12, 13, 14].

Although we focused our presentation on a simplified version of Reno congestion control, it is also possible to construct hybrid models for other congestion control algorithms. Figure 3 depicts the hybrid model for Tahoe congestion control [22]. In this figure, w_i^{thre} denotes the threshold that separates the slow-start phase from the congestion-avoidance phase for the ith flow, and $\bar{m}$ the multiplicative increase that occurs in one round-trip time during slow start (typically $\bar{m} = 2$).

The following are the key novel features of the hybrid models presented above:

1. They consider a continuous approximation for the queue dynamics and the window sizes. This avoids the complexity inherent to a detailed packet-level description that results from coupling congestion control with the error correction protocol.
2. They model packet drops as events that trigger transitions between discrete modes with distinct continuous dynamics. The flexibility of combining continuous dynamics with discrete logic can be exploited to model existing and novel congestion control mechanisms and queuing policies.
3. Although we utilized a deterministic hybrid system in the example above, one can incorporate in this type of model stochastic events that trigger transitions. This is needed to model random queuing disciplines such as Random Early Detection/Drop active queuing [32].

Dynamics in normalized time

The dynamics for the Reno and Tahoe models in Figures 2 and 3 are nonlinear essentially because of RTT's dependence on q. However, it is possible to make them linear by normalizing the time variable. To this effect we introduce a new time variable τ, called *normalized time*[2], defined by

$$\frac{\mathrm{d}\tau}{\mathrm{d}t} = \frac{1}{RTT} = \frac{B}{BT_p + q}, \qquad \tau(0) = 0. \tag{5}$$

This means that an interval with duration $d\tau$ in the variable τ corresponds to an interval of duration $dt = RTT d\tau$ in the variable t. We can think of τ as a time variable normalized so that one unit of τ corresponds to one round-trip time. Figures 4 and 5 show the dynamics of the hybrid systems in normalized time. In these figures, $'$ denotes the derivative $\frac{\mathrm{d}}{\mathrm{d}\tau}$ with respect to the normalized time τ. We also use the fact that in the *queue-full* mode, $q = q_{\max}$ and therefore, waiting until t_{tim} reaches $T_p + \frac{q_{\max}}{B}$ from zero with $t'_{\mathrm{tim}} = RTT = T_p + \frac{q_{\max}}{B}$, is equivalent to waiting until τ_{tim} reaches 1 from zero with $\tau'_{\mathrm{tim}} = 1$.

It is interesting to note that the equation that models the queue dynamics in the *queue-not-full* mode is stable. This is an important property of window-based congestion control, as opposed to other congestion control mechanisms that adapt the packets sending rates directly, instead of indirectly through the window size. This is a manifestation of the well known property that window-based mechanisms automatically adjust their sending rates to variations in the round-trip time.

The linear models derived above will be used in the next section to perform a formal analysis of TCP congestion control. It should be emphasized that these linearizations are exact and do not carry any approximation.

3. ANALYSIS OF TCP-RENO WITH DROP-TAIL QUEUING

[2]Formally, there is a bijective function f that maps normalized time τ into real time t. This function is actually defined by (5). With some abuse of notation, when we write $q(\tau)$ for some normalized time τ, we really mean $q(f(\tau))$. Similar notation is used for the remaining time-dependent variables.

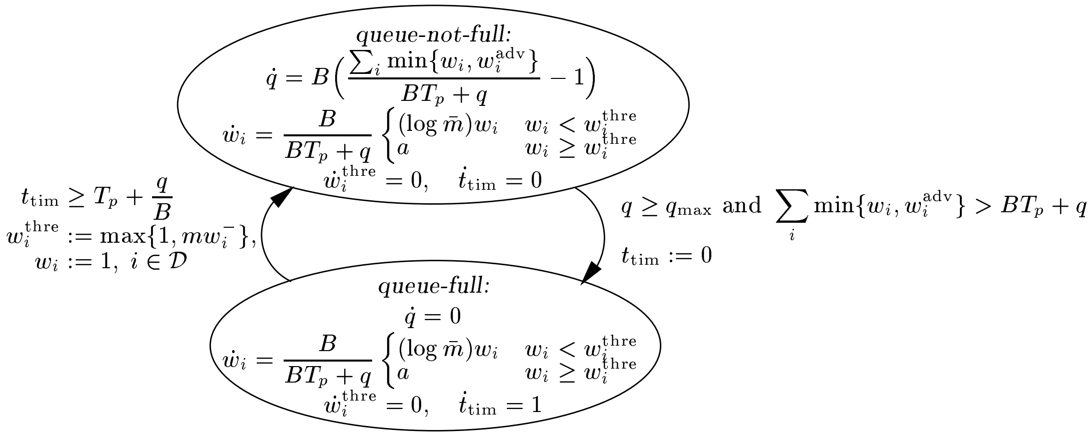


Figure 3: Hybrid model for TCP-Tahoe

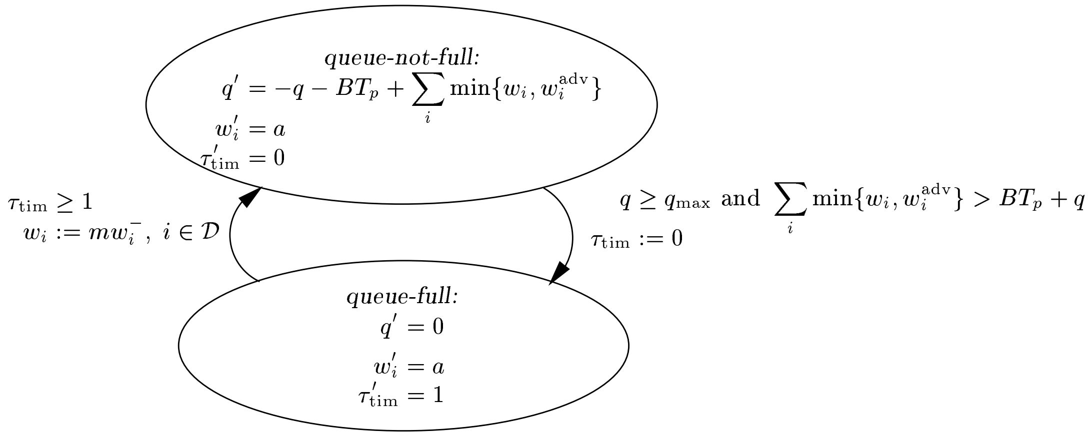


Figure 4: Hybrid model for TCP-Reno in normalized time.

In this section we study the dynamics of the TCP-Reno congestion control model in Figure 2 under drop-tail queuing and infinitely large advertised window size. For analysis purposes, we use the model in normalized time shown in Figure 4.

To complete the hybrid model it remains to specify the drop dynamics that determine the set of flows $\mathcal{D}$ that suffer drop-s while the system is in *queue-full* mode. It turns out that under drop-tail queuing policy exactly one drop per flow will occur in most operating conditions [16]. To understand why, we must recall that, while there are no drops, in every round-trip time the window size of each flow will increase because each flow will receive as many acknowledgments as its window size. When the acknowledgment that triggers the increase of the window size by $a \geq 1$ arrives, the congestion controller will attempt to send two packets *back-to-back*. The first packet is sent because the acknowledgment that just arrived decreased the number of unacknowledged packets and therefore a new packet can be sent. The second packet is sent because the window size just increased, allowing the controller to have an extra unacknowledged packet. However, at this point there is a very fragile balance between the number of packets that are getting in and out of the queue, so two packets will not fit in the queue and the second packet is dropped. Formally, this corresponds to the following *one-drop-per-flow* model

$$\mathcal{D} = F_{\text{drop}}(w_1, w_2, \ldots, w_n) := \{1, 2, \ldots, n\}. \qquad (6)$$

For a very large number of flows, a single drop per flow may not be sufficient to produce the decrease in the window size required to make the queue size drop below $q_{\max}$ after the multiplicative decrease. In this case, the one-drop-per-flow model is not valid. However, we shall see in Section 3.2 that, for most operating conditions, this model accurately matches packet-level simulations performed using the `ns-2` network simulator [27]. In fact, this hybrid model only fails when the number of flows is so large that the drop rates take unusually large values.

3.1 Transient Behavior

We proceed now to analyze the joint evolution of the window sizes of all the flows. Our analysis will show that the window sizes converge to a periodic regimen, regardless of their values at the end of the slow-start period. Because we are considering the variations of the round-trip-times caused by

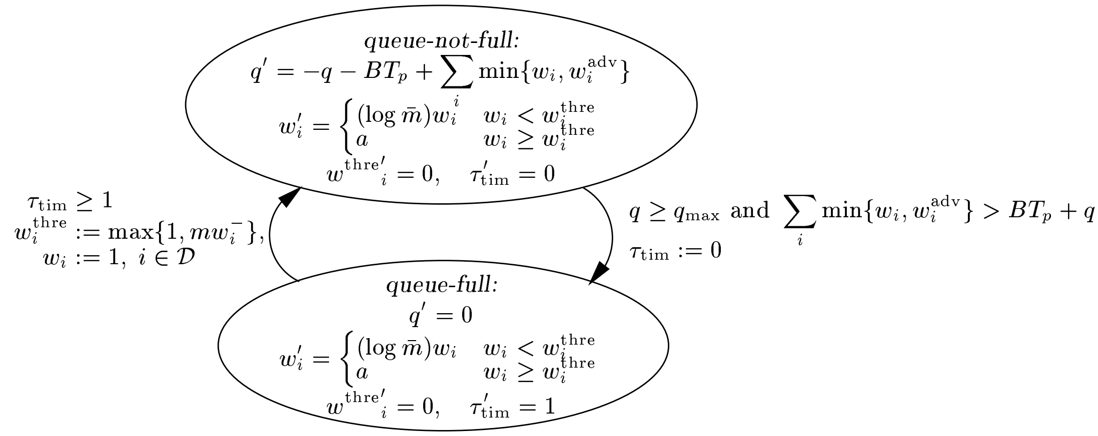


Figure 5: Hybrid model for TCP-Tahoe in normalized time

varying queuing delays, this regimen is more complex (but also closer to reality) than the simple saw-tooth wave form that is often used to characterize the steady-state behavior of this type of algorithms. We are interested here in characterizing the short-term evolution—also known as the *transient behavior*—of the window sizes until the periodic regimen is reached.

Let us denote by $\{\tau_k : \tau_k \leq \tau_{k+1}, k \geq 1\}$ the set of normalized times at which the system leaves the *queue-full* mode. Using the fact that the system dynamics is essentially linear at each discrete mode, one can show (cf. Appendix) that

$$\tau_{k+1} - \tau_k = f^{-1}(s_k) + 1, \qquad k \geq 1, \tag{7}$$

where

$$s_k := \frac{q_{\max} + BT_p - \sum_{i=1}^n w_i(\tau_k)}{an}, \tag{8}$$

and $f : [0, \infty) \to [0, \infty)$ denotes the smooth bijection

$$x \mapsto \begin{cases} \frac{x}{1-e^{-x}} - 1 & x \neq 0 \\ 0 & x = 0 \end{cases}.$$

We proceed now to analyze the evolution of the $w_i(\tau_k)$. To this effect, suppose that the system left the *queue-full* mode at some normalized time τ_k, $k \geq 1$. Since it takes $f^{-1}(s_k) + 1$ units of normalized time until the system leaves the *queue-full* state again, and during this time $w_i' = a$, $i \in \{1, 2, \dots, n\}$, we conclude that

$$w_i^-(\tau_{k+1}) = w_i(\tau_k) + a f^{-1}(s_k) + a,$$

and therefore

$$w_i(\tau_{k+1}) = m\left(w_i(\tau_k) + a f^{-1}(s_k) + a\right). \tag{9}$$

From (8) and (9) we then conclude that

$$s_{k+1} = m\big(s_k - f^{-1}(s_k)\big) + \frac{1-m}{an}(q_{\max} + BT_p) - m. \tag{10}$$

It turns out that, as long as[3]

$$q_{\max} + BT_p \geq \frac{2ma}{1-m} n,$$

the map $g : [0, \infty) \to [0, \infty)$, defined by

$$s \mapsto m\big(s - f^{-1}(s)\big) + \frac{1-m}{an}(q_{\max} + BT_p) - m,$$

is a contraction[4]. In particular,

$$|g(s) - g(\bar{s})| = m|s - \bar{s} - f^{-1}(s) + f^{-1}(\bar{s})| \leq m|s - \bar{s}|, \tag{11}$$

for all $s, \bar{s} \geq 0$. Since (10) can also be written as

$$s_{k+1} = g(s_k),$$

using the Contraction Mapping Theorem [33, p. 126] we conclude that the s_k converges to the unique fixed point s_∞ of g, which is the unique solution to

$$s_\infty = m\big(s_\infty - f^{-1}(s_\infty)\big) + \frac{1-m}{an}(q_{\max} + BT_p) - m. \tag{12}$$

The convergence is as fast as m^k. From this and (9) we conclude that the following theorem holds:

Theorem 1. *Let $\{t_k : t_k \leq t_{k+1}, k \geq 1\}$ be the set of times at which the system leaves the* queue-full. *For infinitely large advertised window size w_i^{adv} and $q_{\max} + BT_p \geq \frac{2ma}{1-m} n$, all the $w_i(t_k)$, $i \in \{1, 2, \dots, n\}$ converge exponentially fast to*

$$w_\infty := \frac{ma}{1-m}\left(f^{-1}(s_\infty) + 1\right), \tag{13}$$

[3] This condition guarantees that g indeed maps nonnegative scalars to nonnegative scalars. This is because $s - f^{-1}(s) \in [-1, 0)$ for all $s \geq 0$.

[4] This is a consequence of the fact that

$$\frac{\mathrm{d}g}{\mathrm{d}s}(s) = m\Big(1 - \frac{\mathrm{d}f^{-1}}{\mathrm{d}s}(s)\Big) \in [-m, 0)$$

for all $s \geq 0$. Here, we used the implicit function theorem to conclude that

$$\frac{\mathrm{d}f^{-1}}{\mathrm{d}s}(s) = \Big(\frac{\mathrm{d}f}{\mathrm{d}x}(x)\Big)^{-1} = \frac{(1 - e^{-x})^2}{1 - e^{-1} + x^{-x}} \in (1, 2],$$

where $x := f(s)$.

as $k \to \infty$ and the convergence is as fast as m^k.

The condition $q_{\max} + BT_p \geq \frac{2ma}{1-m} n$ essentially limits the maximum number of flows under which the one-drop-per-flow model is valid. When this condition is violated, i.e., when

$$n > \frac{1-m}{2ma}(q_{\max} + BT_p),$$

a single drop per flow may not be sufficient to produce a decrease in the sending rates that would make q drop below $q_{\max}$ after the multiplicative decrease.

We defer to Sections 4 and 5 a detailed discussion of the implications of Theorem 1 and proceed with the analysis of the model.

3.2 Steady-state behavior

In the previous section we established that the window sizes converge to a periodic regimen, also known as the *steady-state* regimen. We proceed now to analyze the system when it operates under this regimen. Among other things, we will show that the relationship between average throughput, average drop rate (i.e., the percentage of dropped packets), and average round-trip time that appears in [17, 18, 19, 20] can also be derived from our model. In this section we concentrate on the case where s_∞ is much larger than one and therefore

$$f^{-1}(s_\infty) \approx s_\infty + 1. \tag{14}$$

This approximation is valid when

$$n \ll \frac{1-m}{2ma}(q_{\max} + BT_p) \tag{15}$$

and causes the system to remain in the state *queue-not-full* for, at least, a few round-trip times[5]. In practice, this is quite common and a deviation from (15) results in very large drop rates.

Suppose then that the steady-state has been reached and let us consider an interval $[t_k, t_{k+1}]$ between two consecutive time instants at which the system enters the *queue-not-full* state. Somewhere in this interval lies the time instant $\bar{t}_k$ at which the system enters the *queue-full* state and drops occur. During the interval $[t_k, t_{k+1}]$, the instantaneous rate r at which the nodes are successfully transmitting packets is given by

$$r(t) = \begin{cases} \frac{\sum_{i=1}^n w_i(t)}{RTT(t)} & t \in [t_k, \bar{t}_k) \\ B & t \in [\bar{t}_k, t_{k+1}] \end{cases} \tag{16}$$

The total number of packets N_k sent during the interval $[t_k, t_{k+1}]$ can then be computed by

$$\begin{aligned} N_k &:= \int_{t_k}^{t_{k+1}} r(t)dt = \int_{\tau_k}^{\tau_{k+1}} r(\tau)RTT(\tau)d\tau \qquad (17) \\ &\approx \frac{1-m^2}{2an}(q_{\max} + BT_p + 2an)^2. \end{aligned}$$

[5]When the system remains in the *queue-not-full* for at least 4 round-trip times, (14) already yields an error smaller than 2%.

We used here the change of integration variable defined by (5) to work with quantities in normalized time. Details on the computation of the integrals in (17) and in (19) below are given in the Appendix. Since n drops occur in the interval $[t_k, t_{k+1}]$, *the average drop rate* p is then equal to

$$p := \frac{n}{N_k} \approx \frac{2a}{1-m^2}\left(\frac{n}{q_{\max} + BT_p + 2an}\right)^2. \tag{18}$$

Another quantity of interest is the average round-trip time $\overline{RTT}$. We consider here a packet-average, rather than a time-average, because the former is the one usually measured in real networks. This distinction is important since the sending rate r is not constant. In fact, when the sending rate is higher, the queue is more likely to be full and the round-trip time is larger. This results in the packet-average being larger than the time-average. The *packet-average round-trip time* can then be computed as

$$\begin{aligned} \overline{RTT} &:= \frac{\int_{t_k}^{t_{k+1}} r(t)RTT(t)dt}{N_k} = \frac{\int_{\tau_k}^{\tau_{k+1}} r(\tau)RTT(\tau)^2 d\tau}{N_k} \\ &\approx \frac{1}{T}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\frac{q_{\max} + BT_p + 2an}{n} - a\frac{1-m}{1+m}\right), \qquad (19) \end{aligned}$$

where $T := \frac{B}{n}$ is the average throughput of each flow. We recall that, because the queue never empties, the total throughput is precisely the bandwidth B of the bottleneck link.

It is interesting to note that the average drop rate p can provide an estimate for the quantity $\frac{n}{q_{\max}+BT_p+2an}$. In particular, we conclude from (18) that

$$\frac{q_{\max} + BT_p + 2an}{n} \approx \sqrt{\frac{2a}{(1-m^2)p}}. \tag{20}$$

This, in turn, can be used together with (19) to estimate the average throughput T. In fact, from (19) and (20) we conclude that

$$T \approx \frac{1}{\overline{RTT}}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\sqrt{\frac{2a}{(1-m^2)p}} - a\frac{1-m}{1+m}\right)$$

The following theorem summarizes the results above for the hybrid model for TCP-Reno congestion control in Figure 2:

Theorem 2. *For infinitely large advertised window size w_i^{adv} and $q_{\max} + BT_p \gg \frac{2ma}{1-m} n$, the average drop rate p, the packet average round-trip time $\overline{RTT}$, and the average throughput T of each flow are approximately given by*

$$p \approx \frac{2a}{1-m^2}\left(\frac{n}{q_{\max} + BT_p + 2an}\right)^2, \tag{21}$$

$$\overline{RTT} \approx \frac{n}{B}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\frac{q_{\max} + BT_p + 2an}{n} - a\frac{1-m}{1+m}\right), \tag{22}$$

$$T \approx \frac{1}{\overline{RTT}}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\sqrt{\frac{2a}{(1-m^2)p}} - a\frac{1-m}{1+m}\right). \tag{23}$$

To verify the formulas in Theorem 2, we simulated the dumbbell topology of Figure 6, using the `ns-2` network simulator [27]. Figure 7 summarizes the results obtained for a network

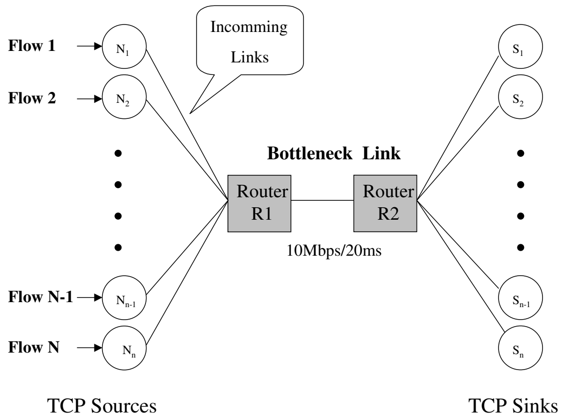


Figure 6: Dumbbell topology with n TCP flows, 10 Mbps bottleneck link, 100 Mbps incoming links, 40 millisecond round trip propagation delay, and queue size at the bottleneck link of 250 packets.

with the following parameters:

$$B = \frac{10^7 \text{ bits/sec}}{8 \text{ bits/char} \times 1000 \text{ char/packet}} = 1250 \text{ packets/sec},$$

$T_p = .04$ sec, $q_{\max} = 250$ packets, $a = 1$ packet/RTT, $m = 1/2$. As seen in the figure, the theoretical predictions given by (21)–(23) match the simulation results quite accurately. Some mismatch can be observed for large number of flows. However, this mismatch only starts to become significant when the drop rates are around 1%, which is an unusually large value. This mismatch is mainly due to two factors: the quantization of the window size and a crude modeling of the fast-recovery algorithm [29]. We are now in the process of incorporating these two features into our model to obtain formulas that are accurate also in very congested networks.

3.3 Comparison with Previous Results

For $a = 1$ and $m = 1/2$, the formula (23) becomes

$$T \approx \frac{1}{\overline{RTT}} \left(\frac{1.27}{\sqrt{p}} - \frac{1}{3} \right). \qquad (24)$$

For reasonable drop rates, the term $\frac{1.27}{\sqrt{p}}$ dominates over $1/3$ and (24) matches closely similar formulas derived in [17, 18, 19, 20]. However, the analysis presented here goes several steps further than the ones presented in these references because of the following:

1. We deduced (24) under a more general setup. In particular, our analysis considered the effect of queuing and its effect in the round-trip time.

2. Theorem 2 provides a more complete description of the steady-state behavior of TCP because it gives explicit formulas for the average round-trip time $\overline{RTT}$ and the drop rate p as a function of the number of flows n.

3. The analysis presented here takes into account the coupling between the n competing flows and we were able to show that the steady-state for which the formulas (21)–(23) are valid is actually reached asymptotically.

4. The results in Theorem 1 also differ from most of the others found in the literature in that they provide information about the transient behavior of the individual flows.

Previous derivations of (23) ignored queuing, assumed constant round-trip time, considered a single flow, and ignored transient behavior. It is important to emphasize that $\overline{RTT}$ in (22) denotes the *average* round-trip time. It turns out that the actual round-trip-time RTT varies quite significantly around this average because of fluctuations on the queue size. In fact, even after the steady-state is reached, the variation of the "instantaneous" round-trip time is often larger then 50% of the average round-trip time.

The models developed in Section 2 and the analysis above ignore the existence of timeouts. However, one can already find in the literature (e.g., in [34, 24]) formulas such as (23) that take timeouts into account, in the context of constant round-trip time, single-flow models. In future work we plan to incorporate timeouts in our hybrid models and verify if the formulas in the literature still hold when one takes into account multiple flows, queuing, etc. We should note, however, that in the `ns-2` simulations of the network in Figure 6 we only observed timeouts when the drop rates where very high.

4. FLOW SYNCHRONIZATION

One conclusion that can be drawn from Theorem 1 is that all flows converge exponentially fast to the same limit cycle. This limit cycle corresponds to a continuous increase of the window size from w_∞ to $\frac{1}{m}w_\infty$, followed by an instantaneous decrease back to w_∞ due to drops. Because all flows converge to the *same* limit cycle, this means that the flows will become synchronized. We were able to observe this synchronization effect in `ns-2` simulations using the "default" dumbbell topology described in Section 3.2. Figure 8 plots the congestion window of 8 TCP-Reno flows and the queue size at the bottleneck link. Even though each flow starts at a random time between 0 and 5 seconds, we observe that they are almost perfectly synchronized at around 30 seconds.

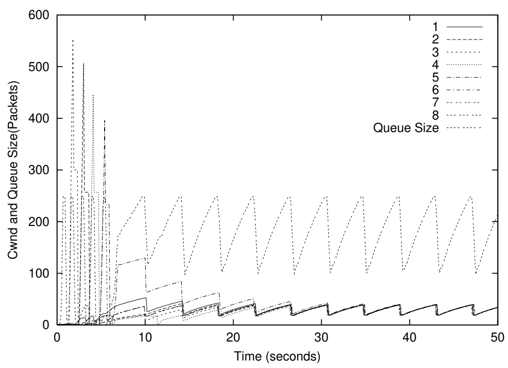


Figure 8: Congestion window and bottleneck link queue size for the default dumbbell topology with $n = 8$ flows.

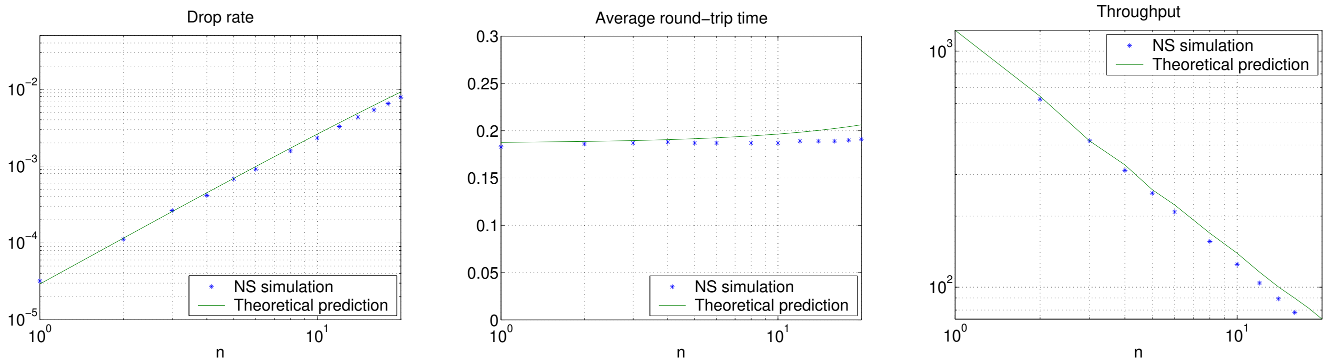


Figure 7: Comparison between the predictions obtained from the hybrid model and the results from `ns-2` simulations.

Window size synchronization had been observed in [15, 16] for TCP-Tahoe congestion control [22] and actually led to the development of Random Early Detection/Drop active queuing [35, 32]. In [16], the authors defend that synchronization is closely related to the packet loss synchronization that we also use in our model. In fact, they provide an informal explanation—supported by packet-level simulations—of how synchronization is a self-sustained phenomenon. Although [16] only deals with TCP-Tahoe congestion control, the arguments used there also apply to Reno. Theorem 1 goes much further because it demonstrates that synchronization is not just self-sustained but it is actually an exponentially attracting state. This means that synchronization will occur even if the flows start unsynchronized or lose synchronization because of some temporary disturbance. Moreover, the convergence to this state is very fast and the distance to it is reduced by at least m (typically 1/2) with each drop. In fact, initially the convergence is even faster because the upper bound in (11) is conservative for large $|s - \bar{s}|$. To the best of our knowledge this is the first time that these type of theoretical results were obtained. Note that synchronization cannot be captured by single-flow models.

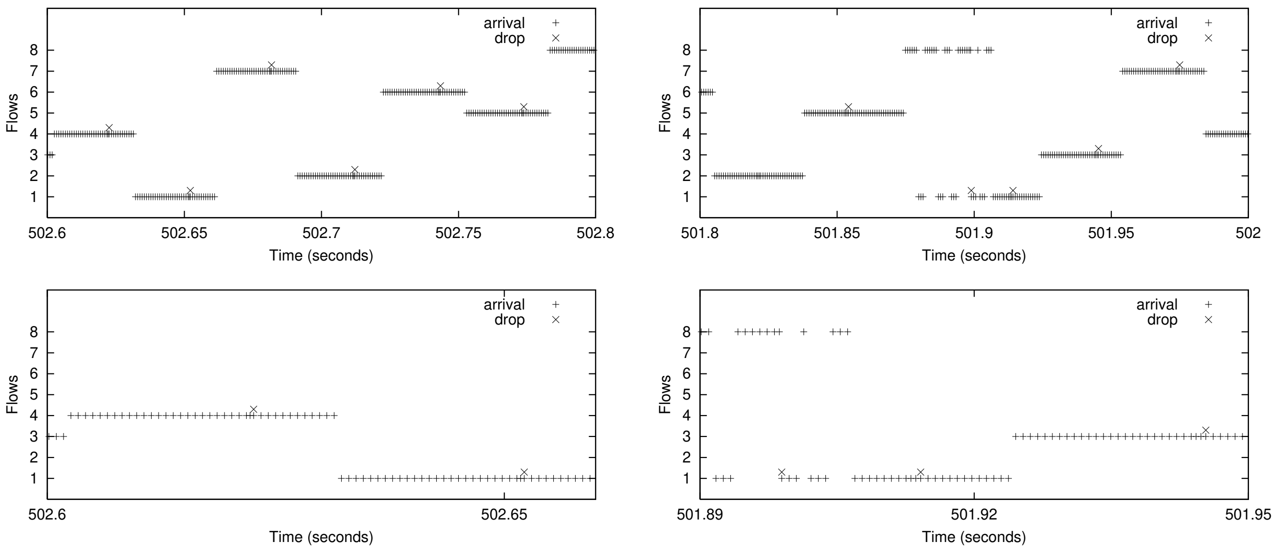
As long as the output queue of node n_1 does not get empty (which was the case in all our simulations), the bottleneck link is used at full capacity and flow synchronization does not have any effect on the average throughput. However, it does produce large variations on the size of the bottleneck queue and therefore large variations on the round-trip time. The network traffic also becomes more bursty and network resources are used unevenly. The larger variations on the round-trip time are particularly harmful to TCP because they often lead to inefficient timeout detection. They are also a problem for multimedia flows that cannot withstand large delay jitter, i.e., delay variation associated with the delivery of packets belonging to a particular flow.

Three assumptions were instrumental in the proof of Theorem 1: infinite (or very large) advertised window size, drop-tail queuing, and large bandwidth mismatch between the data sources and the bottleneck link. The last assumption is actually implicit in the dumbbell topology in Figure 1. It was under these three assumptions that we derived the one-drop-per-flow model (6). Indeed, without a limit on the advertised window size, each flow will increase its window size by one while the system is in the *queue-full* mode. When this increase takes place, the large bandwidth mismatch between incoming and bottleneck links results in back-to-back packets arriving at the bottleneck link queue. Finally, under a drop-tail policy the second packet is dropped resulting in exactly one drop per flow. The next section discusses the impact of finite advertised window sizes. In the remaining of this section, we study the link bandwidth mismatch issue and the effect of queuing policies other than drop-tail on flow synchronization.

Even though the source generates back-to-back packets when the window size is increased, these packets do not arrive back-to-back at the bottleneck link, when the incoming link has finite bandwidth. In fact, the smaller the bandwidth of the incoming link, the more spread will the packets arrive at the bottleneck link. This can be seen in Figure 9 that shows a trace of packets arriving at the bottleneck link from each of the incoming links. The two plots in figure 9(a) correspond to the topology in Figure 6 with a bandwidth of 20Mbps in the incoming link, versus 10Mbps in the bottle neck link. The bottom plot is just a zoomed version of the top one. Although the packets no longer arrive exactly back-to-back, we still have one drop per flow and synchronization occurs. In the lower plot in Figure 9(a), we can actually see the effect of a window size increase and the corresponding drop in flow 4 at time 502.623 seconds. Figure 9(b) contains similar plots but for an incoming link bandwidth of 15Mbps. We can see in this plot that a back-to-back packet in flow 8 at time 501.90 seconds actually causes a drop in flow 1.

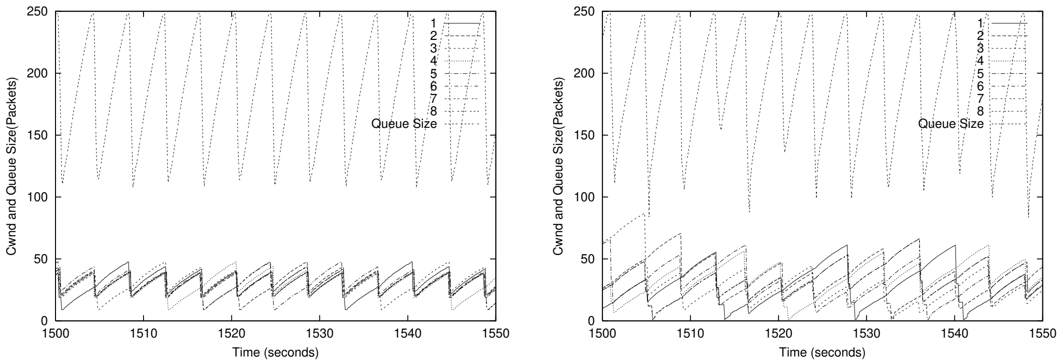
For the network parameters considered here, 15Mbps was roughly the bandwidth for the incoming link at which we start losing synchronization. Flow synchronization is completely lost, when this bandwidth is lowered to 5Mbps. We show in Figure 10 plots of the congestion window and bottleneck link queue sizes for these two cases. Note that the time scales in both graphs go up to 1550 seconds showing we are not simply observing a transient effect.

We also investigated what effect different queuing disciplines have on flow synchronization. Figure 11(a) shows that by using a simple *drop-head* queuing policy (drop the pack-



(a) (b)

Figure 9: Packet traces for (a) 20 Mbps and (b) 15 Mbps incoming link bandwidths.



(a) (b)

Figure 10: Congestion window and bottleneck link queue size for the default dumbbell topology with (a) 15 Mbps and (b) 5 Mbps incoming link bandwidths.

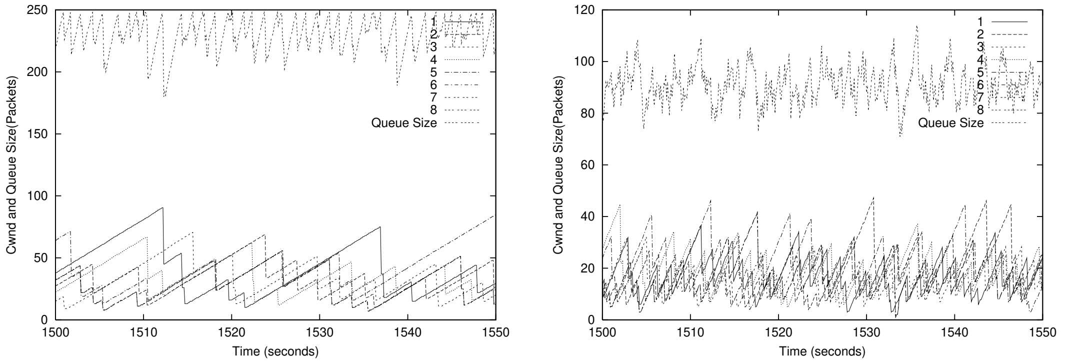
et at the head of the queue)—while keeping all the other dumbbell characteristics the same—we are able to eliminate synchronization. Figure 11(b) shows that Random Early Detection (RED) queuing also eliminates flow synchronization by adding randomization to the network, as suggested in [35].

It is interesting to note that a deterministic queuing discipline such as drop-head can be very effective in breaking synchronization. This is not completely surprising because there is not a significant difference between the distribution of the packets that are at the head of the queue and any random packet in the queue. Therefore drop-head is not significantly different from random-drop [36, 37]. However, when compared to random-drop and RED, drop-head has the significant advantage that drops will be detected sooner since queuing delay is minimized. Drop-head is therefore the most responsive of these algorithms. Incorporating different queuing disciplines (including random-drop and deterministic early drop) into our models and developing the formal analysis for them is an item for future work.

5. FAIRNESS

From Theorem 1 we can conclude that all flows converge to the same limit cycle and therefore TCP-Reno congestion control is asymptotically fair. Although this has generally been accepted as true, a formal proof of this property of TCP for an arbitrary number of flows, taking queuing into account, has not been developed before. In fact, this is one of the contributions of this work.

It turns out that fairness may be lost when the assumptions used to derive Theorem 1 do not hold. In particular, when some or all the flows have finite advertised window sizes. The graph in Figure 12(a) demonstrates this behavior. It plots the congestion window size of 8 TCP-Reno flows all of which are limited by an advertised window size of 50 pack-



(a) (b)

Figure 11: Congestion window and bottleneck link queue size for the default dumbbell topology using (a) drop-head queuing and (b) RED queuing.

ets. We observe that 4 out of the 8 flows are able to reach the maximum window size of 50 packets and keep sending at that constant rate throughout the simulation. Because they will not further increase their sending rates, these flows will never attempt to send back-to-back packets, making it very unlikely that they will suffer drops. The remaining flows were not able to reach the advertised window size (possibly because they started slightly later) and therefore they will surely suffer drops when the bottleneck queue fills up. In this specific scenario, TCP favors the first four flows to start, which end up exhibiting almost four times higher throughput than the remaining flows.

Similar behavior is observed when simulating scenarios in which only a few flows have limited advertised window size. Figure 12(b) shows the simulation results when 2 (out of 8) flows are limited by a 50-packet advertised window. It is interesting to observe that one of these flows is the second flow to start and is able to achieve the 50-packet window size limit. Since it never sends back-to-back packets, it never suffers packet drops and is thus able to keep sending packets at the maximum rate (corresponding to a 50-packet window size). The other flow, which is also limited by a 50-packet advertised window size, ends up starting last and behaves like all other flows, suffering drops periodically. This is because the bandwidth is not sufficient to allow the last flow to reach the 50-packet window size.

The next set of graphs demonstrate that, by using different queuing disciplines, unfairness can be avoided. Indeed, Figure 13 shows that both drop-head and RED queuing eliminate unfairness when all flows have finite advertised window size. Both queuing policies result in normalized average throughputs just a few percentage points away from one. When some flows have finite advertised window size and others do not, we observe that the later are at an advantage and exhibit larger normalized throughputs. This is expected and occurs both with drop-head queuing and RED. A simulation for the drop-head case is shown in Figure 14.

One item for future work is the use of formal methods to predict and explain when TCP exhibits an unfair behavior, as well as ways to avoid it. The hybrid models in Figures 2–

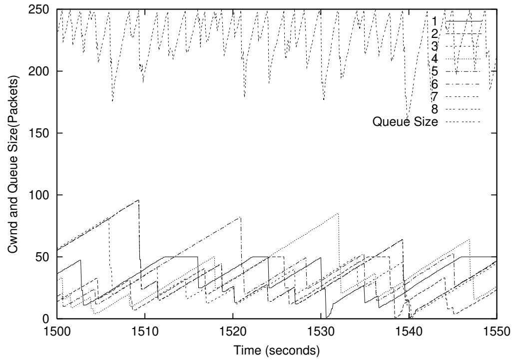


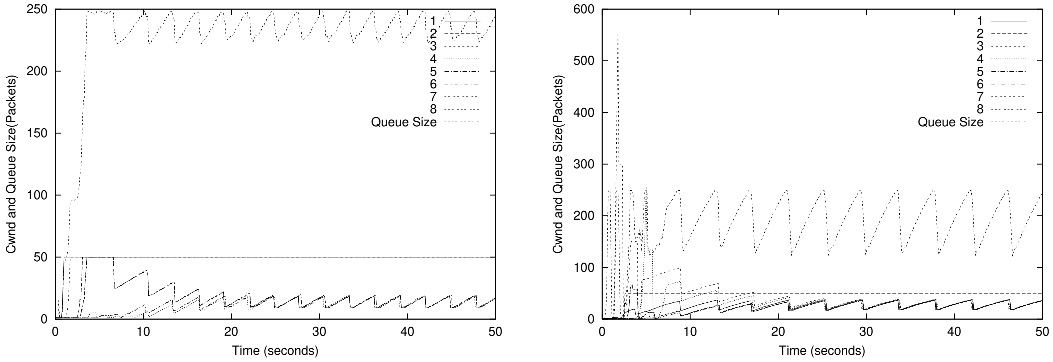
Figure 14: Congestion window and bottleneck link queue size for the default dumbbell topology where flows 1 and 2 have 50-packet advertised window size using drop-head queuing. The flows with finite advertised window size exhibit a normalized average throughput in the range $90.2\% \pm 2.1\%$ and the remaining flows in the range $103.3\% \pm 1.8\%$.

5 already take into account finite advertised windows sizes but the drop model (6) in Section 3 does not. One simple model that can capture the phenomena described above with drop-tail queuing is given by

$$\mathcal{D} = F_{\text{drop}}(w_1, w_2, \ldots, w_n) := \{i : w_i < w_i^{\text{adv}}\}, \qquad (25)$$

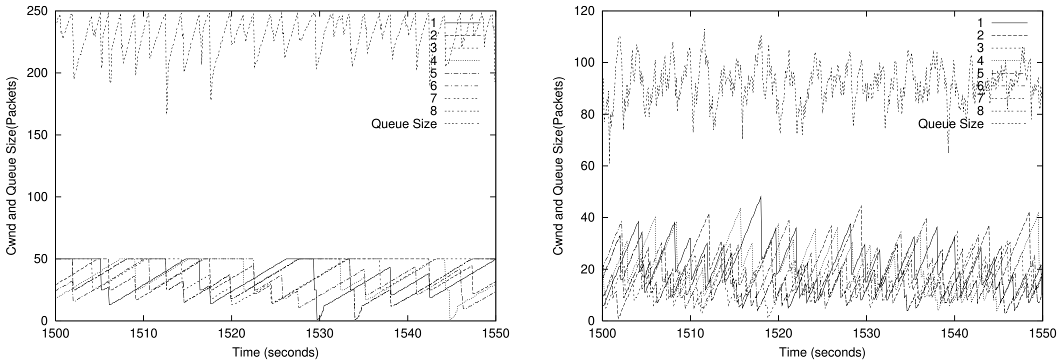
because drops will most likely only occur in those flows i that did not yet reach their advertised window sizes w_i^{adv}. Using the drop model (25), one should be able to predict when will finite advertised window sizes result in unfairness and which values for the throughputs are expected. We are now in the process of deriving drop-models analogous to (25) for drop-head queuing and RED. These should allow us to demonstrate that these queuing policies solve the fairness problem and also to quantify precisely how effective these policies are in achieving this. Note that this type of analysis is only possible with multiple-flow models like the ones considered here.

6. CONCLUSIONS



(a) (b)

Figure 12: Congestion window and bottleneck link queue size for the default dumbbell topology when (a) all flows have 50-packet advertised window size and (b) only flows 1 and 2 have finite (50-packet) advertised window size. In (a) four of the flows exhibit a normalized throughput of 138% and the remaining a normalized throughput in the range $37\%\pm1\%$, in (b) one flow exhibits a normalized throughput of 167% and the remaining a normalized throughput of $90\% \pm .5\%$



(a) (b)

Figure 13: Congestion window and bottleneck link queue size for the default dumbbell topology where all flows have 50-packet advertised window size using (a) drop-head and (b) RED queuing. In (a) all flows exhibit a normalized average throughput in the range $1 \pm 3.6\%$ and in (b) in the range $1 \pm 2.5\%$.

In this paper we proposed a hybrid model for TCP congestion control. Using this model, we analyzed both the transient and steady-state behavior of n competing TCP flows on a dumbbell network topology. Besides using our model to confirm well-known formulas, we also used it to derive new relationships and thus provide a more complete description of TCP's steady-state behavior. Our model enabled us to explain the flow synchronization phenomena that have been observed in simulations and in real networks but, to the best of our knowledge, have not been theoretically justified. We were also able to demonstrate that the limit cycle that corresponds to flow synchronization is global exponentially stable. This means that synchronization will occur even if the flows start unsynchronized or loose synchronization because of temporary disturbances.

Another contribution of our hybrid model was that it allowed us to identify conditions under which TCP fairness is lost. We observed TCP's unfair behavior when some or all flows have finite advertised window size.

In the paper, we also showed methods to avoid synchronization and unfairness. Throughout the paper, experimental results obtained through `ns-2` simulations were used to support our theoretical analysis.

We are now in the process of generalizing the analysis presented here to more complex network topologies, other congestion control mechanisms (such as TCP-Tahoe, Vegas, and Equation-Based), and different queuing policies (such as drop-head, random-drop, RED, and SRED). We are also investigating alternative mechanisms to avoid synchronization and unfairness. Another direction we are exploring is the application of the hybrid models derived here to detect abnormalities in TCP traffic flows. This has important applications in network security.

APPENDIX

Derivation of equation (7)

Suppose that drops occurred at some normalized time τ_k at which the system entered the *queue-not-full* state and therefore that $q(\tau_k) = q_{\max}$. Denoting by $\bar{\tau}_k$ the normalized time at which the next drop occurs, for $\tau \in [\tau_k, \bar{\tau}_k)$, we have[6]

$$w_i(\tau) = w_i(\tau_k) + a(\tau - \tau_k), \qquad i \in \{1, 2, \dots, n\}, \tag{26}$$

$$q(\tau) = e^{-(\tau-\tau_k)}\Big(q_{\max} + BT_p + an - \sum_{i=1}^{n} w_i(\tau_k)\Big) + + an(\tau - \tau_k) + \sum_{i=1}^{n} w_i(\tau_k) - BT_p - an. \tag{27}$$

We assumed here that q remains positive during the whole interval. Since a new drop occurs at the normalized time $\bar{\tau}_k$, we must have $q(\bar{\tau}_k) = q_{\max}$. Because of q's continuity, we must then have

$$q_{\max} = e^{-(\bar{\tau}_k-\tau_k)}\Big(q_{\max} + BT_p + an - \sum_{i=1}^{n} w_i(\tau_k)\Big) + + an(\bar{\tau}_k - \tau_k) + \sum_{i=1}^{n} w_i(\tau_k) - BT_p - an.$$

We can then solve this equation to compute the normalized time interval $\bar{\tau}_k - \tau_k$ and obtain

$$\frac{q_{\max} + BT_p - \sum_{i=1}^{n} w_i(\tau_k)}{an} = \frac{\bar{\tau}_k - \tau_k}{1 - e^{-(\bar{\tau}_k-\tau_k)}} - 1, \tag{28}$$

which is equivalent to

$$\bar{\tau}_k - \tau_k = f^{-1}(s_k). \tag{29}$$

Equation (7) is a consequence of this and the fact that the system enters the *queue-not-full* state again at time $\tau_{k+1} := \bar{\tau}_k + 1$. ∎

Derivation of equations (17) *and* (19)

The integrals in (17) and (19) were computed symbolically using Wolfram Research's Mathematica software. For $r(\tau)$ in (17) and (19) we used

$$r(\tau) = \begin{cases} \frac{\sum w_{i=1}^{n}(\tau)}{RTT(\tau)} & \tau \in [\tau_k, \bar{\tau}_k) \\ B & \tau \in [\bar{\tau}_k, \tau_{k+1}] \end{cases} \tag{30}$$

where τ_k, $\bar{\tau}_k$, and τ_{k+1} are the normalized time instants corresponding to t_k, $\bar{t}_k$, and t_{k+1}, respectively, which are given by (7) and $\bar{\tau}_k = \tau_{k+1} - 1$. Since we are assuming that steady-state has been reached we have that

$$s_k = s_\infty \approx \frac{1-m}{an}(q_{\max} + BT_p + 2an) - 2.$$

The second equality above is obtained from (12) and (14). The window size $w_i(\tau)$ in (30) is given by (26) with $w_i(\tau_k) = w_\infty$ taken from (13). The round-trip time RTT needed in (30) and also in the integral (19) is given by

$$RTT(\tau) = T_p + \frac{q(\tau)}{B},$$

with $q(\tau)$ taken from (27) for $\tau \in [\tau_k, \bar{\tau}_k)$ and equal to $q_{\max}$ for $\tau \in [\bar{\tau}_k, \tau_{k+1}]$. The precise equation for N_k derived from the formulas above is given by

$$N_k = \frac{1-m^2}{2an}(q_{\max} + BT_p + 2an)^2 - \frac{3}{2}an.$$

Fro simplicity, in (17) we ignored the term $\frac{3}{2}an$ because it negligible for any reasonable maximum queue size $q_{\max}$. The precise equation for $\overline{RTT}$ is

$$\overline{RTT} = \frac{1}{T}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\,\frac{q_{\max} + BT_p + 2an}{n} - a\frac{1-m}{1+m}\right) - \frac{2a^2 n}{3B(1+m)}\,\frac{3m(2+m)\frac{q_{\max}+BT_p+2an}{n} - (2+11m)a}{(1-m^2)\big(\frac{q_{\max}+BT_p+2an}{n}\big)^2 - 3a^2}$$

In (19), we discarded the second term because it is dominated by the first one.

[6] Let us verify this: From these formulas we get $w_i' = a$ and

$$\begin{aligned} q' &= -e^{-(\tau-\tau_k)}\Big(q_{\max} + BT_p + an - \sum_{i=1}^{n} w_i(\tau_k)\Big) + an \\ &= -\Big(e^{-(\tau-\tau_k)}\Big(q_{\max} + BT_p + an - \sum_{i=1}^{n} w_i(\tau_k)\Big) + \\ &\quad + an(\tau-\tau_k) + \sum_{i=1}^{n} w_i(\tau_k) - BT_p - an\Big) + \\ &\quad + \sum_{i=1}^{n}\big(w_i(\tau_k) + a(\tau-\tau_k)\big) - BT_p \end{aligned}$$

A. REFERENCES

[1] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, "A TCP-friendly rate adjustment protocol for continuous media flows over best effort networks," Tech. Rep. TR 89-04, UMASS-CMPSCI, 1998.

[2] S. Floyd, M. Handley, and J. Padhye, "A comparison of equation-based and AIMD congestion control." Available at `http://www.aciri.org/tfrc/`, May 2000.

[3] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications: The extended version," Tech. Rep. TR-00-003, ICSI, 2000.

[4] D. Bansal and H. Balakrishnan, "TCP-friendly congestion control for real-time streaming applications." To appear in INFOCOM, Apr. 2001.

[5] L. Tavernini, "Differential automata and their discrete simulators," *Nonlinear Anal. Theory, Methods, and Applications*, vol. 11, no. 6, pp. 665–683, 1987.

[6] A. S. Morse, D. Q. Mayne, and G. C. Goodwin, "Applications of hysteresis switching in parameter adaptive control," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1343–1354, Sept. 1992.

[7] A. Back, J. Guckenheimer, and M. Myers, "A dynamical simulation facility for hybrid systems," in Grossman *et al.* [38].

[8] A. Nerode and W. Kohn, "Models for hybrid systems: Automata, topologies, stability," in Grossman *et al.* [38], pp. 317–356.

[9] P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon, "Hybrid system modeling and autonomous control systems," in Grossman *et al.* [38], pp. 366–392.

[10] R. W. Brockett, "Hybrid models for motion control systems," in *Essays in Control: Perspectives in the Theory and its Applications* (H. L. Trentelman and J. C. Willems, eds.), pp. 29–53, Boston: Birkhäuser, 1993.

[11] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: Background, model and theory," in *Proc. of the 33rd Conf. on Decision and Contr.*, vol. 4, pp. 4228–4234, Dec. 1994.

[12] M. S. Branicky, *Studies in Hybrid Systems: Modeling, Analysis, and Control*. PhD thesis, MIT, Cambridge, MA, June 1995.

[13] J. Lygeros, *Hierarchical hybrid control of large scale systems*. PhD thesis, Univ. California, Berkeley, CA, 1996.

[14] J. Lygeros, C. Tomlin, and S. Sastry, "Controllers for reachability specifications for hybrid systems," *Automatica*, vol. 35, pp. 349–370, Mar. 1999.

[15] E. Hashem, "Analysis of random drop for gateway congestion control," tech. rep., MIT, 1990.

[16] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," in *Proc. of SIGCOMM*, Sept. 1991.

[17] T. Ott, J. H. B. Kemperman, and M. Mathis, "Window size behavior in TCP/IP with constant loss probability," in *Proc. of the DIMACS Workshop on Performance of Realtime Applications on the Internet*, Nov. 1996.

[18] J. Mahdavi and S. Floyd, "TCP-friendly unicast rate-based flow control." Technical note sent to the end2end-interest mailing list, Jan. 1997.

[19] T. V. Lakshman, U. Madhow, and B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: A study of TCP/IP performance," in *Proc. of INFOCOMM*, Apr. 1997.

[20] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *Computer Communication Review*, vol. 27, July 1997.

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

[22] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe Reno and SACK TCP," *Computer Communication Review*, vol. 27, pp. 5–21, July 1996.

[23] R. Morris, *Scalable TCP Congestion Control*. PhD thesis, Harvard University, Cambridge, MA, Jan. 1999.

[24] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications." To appear in SIGCOMM, May 2000.

[25] E. Altman, C. Barakat, and E. Laborde, "Fairness analysis of TCP/TP," in *Proc. of the 39th Conf. on Decision and Contr.*, pp. 61–66, Dec. 2000.

[26] J. P. Hespanha, S. Bohacek, K. Obraczka, and J. Lee, "Hybrid modeling of TCP congestion control." To appear in the 4th Int. Workshop on Hybrid Systems: Computation and Control (HSCC 2001), Mar. 2001.

[27] The VINT Project, a collaboratoin between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC, *The ns Manual (formerly ns Notes and Documentation)*, Oct. 2000. Available at http://www.isi.edu/nsnam/ns/ns-documentation.html.

[28] V. Jacobson, "Congestion avoidance and control," in *Proc. of SIGCOMM*, vol. 18.4, pp. 314–329, Aug. 1988.

[29] V. Jacobson, "Modified TCP congestion avoidance algorithm." Posted on end2end-interest mailing list, Apr. 1990. Available at ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[30] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," *RFC 2581*, p. 13, Apr. 1999.

[31] Y. Yang and S. Lam, "General AIMD congestion control. technical report," Tech. Rep. TR-200009, Department of Computer Science, University of Texas at Austin, May 2000.

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

[33] A. W. Naylor and G. R. Sell, *Linear Operator Theory in Engineering and Science*. No. 40 in Applied Mathematical Sciences, New York: Springer-Verlag, 1982.

[34] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proc. of SIGCOMM*, Sept. 1998.

[35] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, pp. 115–116, Sept. 1992.

[36] V. Jacobson, "Minutes of the performance working group." Proc. of the Cocoa Beach Internet Engineering Task Force, Reston, VA, Apr. 1989.

[37] A. Mankin, "Random drop congestion control," in *Proc. of the ACM Symposium on Communications Architectures and Protocols*, pp. 1–7, Sept. 1990.

[38] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rishel, eds., *Hybrid Systems*, vol. 736 of *Lecture Notes in Computer Science*. New York: Springer-Verlag, 1993.