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

Stephan Bohacek bohacek@math.usc.edu
João P. Hespanha hespanha@usc.edu
Junsoo Lee junsoole@usc.edu
Katia Obraczka katia@cse.ucsc.edu

Abstract— **In this paper we make use of *hybrid systems* to model the transient and steady-state behavior of multiple TCP flows that share a single common bottleneck link. The contributions of our models include: (1) a more complete description of TCP's behavior, including the effect of queuing, interaction among competing flows, and finite advertised window size, (2) theoretical prediction of phenomena such as flow synchronization which have only been observed experimentally; our models predict that, 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, (3) theoretical prediction of other TCP congestion control pathologies, such as unfairness, which previous models based on single-flow analysis fail to capture. In this paper we also propose mechanisms that mitigate both synchronization and unfairness.**

We validate our approach by constructing a hybrid model of TCP-Reno and re-deriving 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. We also present simulation results that validate our theoretical predictions.

To our knowledge, this is the first time hybrid systems are used to model congestion control. We fully characterize TCP's behavior in the dumbbell topology, employing powerful theoretical tools available for hybrid systems. When compared with previous work, we provide a more complete characterization of TCP, demonstrating the potential of hybrid systems as a modeling tool for congestion control.

I. 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]. 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.

Yet another model of TCP?

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 [4], 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. Hybrid system modeling 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 provide a detailed understanding of TCP congestion control algorithms through the use of hybrid models. Similarly to existing models of TCP congestion control, we develop our model for the *dumbbell* topology[1]. However, our models provide new insight into the behavior of TCP congestion control. Unlike the single-flow analysis conducted by other TCP modeling efforts, our model considers multiple TCP flows, which allows us

[1]Several existing models of TCP congestion control have been developed for the dumbbell topology [5], [6], [7], [8], [2], [9]. In a dumbbell topology, TCP flows generated at source node n_1 and directed towards sink node n_2 compete for t finite bandwidth B that characterizes the bottleneck link l connecting n_1 to n_2. Figure 2 shows the dumbbell topology we use in our simulations.

to theoretically describe several phenomena that had only been observed experimentally. 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 observed empirically [10], [5]—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. 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 V, 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

$$T = \frac{1.23}{\overline{RTT}\sqrt{p}} \qquad (1)$$

where T is the average throughput, $\overline{RTT}$ the average round-trip time, and p the average drop rate [11], [12], [7], [13] or variations of (1) that consider timeouts [14], [2]). 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 between throughput, drop rate, and round-trip time given by Equation (1) 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. For intrusion detection purposes, our detailed models of TCP's congestion control algorithms allow an accurate characterization of TCP's baseline behavior and thus makes it easier to identify potential attacks.

The remainder 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 III, 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 IV 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 V. Finally, Section VI provides a summary of the results, some concluding remarks, and our future work plans.

II. 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 [15], [16], [17] but the model proposed also applies to more recent variations on Reno such as New Reno, SACK [6], and general AIMD [18]. We denote by RTT the *round-trip time*, and by w_i and w_i^{th}, $i \in \{1, 2, \ldots, n\}$ the *window size* and *slow-start threshold*, respectively, for the congestion controller associated with the ith flow. As we will see shortly, the round-trip time is a time-varying quantity.

In Reno, the algorithm to update w_i is as follows: While the window size w_i is below the slow-start threshold w_i^{th}, the congestion controller is in the *slow-start* mode and w_i is multiplied by a fixed constant m_{ss} (typically $m_{\mathrm{ss}} = 2$) every round-trip-time RTT. When the window size raises above w_i^{th}, the controller enters the *congestion avoidance* mode and w_i is incremented by a fixed constant $a \geq 1$ every round-trip-time RTT. The above takes place until a drop occurs. A drop can be detected through two mechanisms that lead to different reactions by the congestion controller. When the drop is detected because of the arrival of three consecutive duplicate acknowledgments, w_i is multiplied by a constant $m \in (0, 1)$ (typically $m = 1/2$) and the system proceeds to the congestion avoidance mode. In some cases, three consecutive duplicate acknowledgments never arrive and a drop is detected when a packet remains unacknowledged for a period of time longer than the *retransmission timeout* RTO. In this case, the slow-start threshold is set equal to mw_i and w_i is reduced to one. Unless there are many consecutive drops per flow, timeouts occur mostly when the window size becomes smaller than four and therefore no three duplicate acknowledgments can arrive after a drop.

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 in the slow start mode[2]

$$\dot{w}_i = (\log m_{\text{ss}}) w_i / RTT, \tag{2}$$

and in the congestion avoidance mode

$$\dot{w}_i = a / RTT. \tag{3}$$

When a drop is detected at time t through three duplicate acknowledgements, we have

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

where $w_i^-(t)$ denotes the limit from below of $w_i(s)$ as $s \uparrow t$, and when the drop is detected through timeout, we have

$$w_i^{\text{th}}(t) = m\, w_i^-(t), \qquad w_i(t) = 1.$$

The round-trip time is given by $RTT(t) = T_p + q(t)/B$, 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

$$r := \Big(\sum_i \min\{w_i, w_i^{\text{adv}}\}\Big)/RTT$$

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,

$$\dot{q} = \begin{cases} 0 & q = 0, r < B \text{ or } q = q_{\max}, r > B \\ r - B & \text{otherwise} \end{cases} \tag{4}$$

[2]This equation leads precisely to a multiplication by m_{ss} on each round-trip time RTT.

The first branch in (4) 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. Typically, drops detected through duplicate acknowledgments will be detected roughly one round-trip time after they occur, whereas the one detected through timeout will be detected RTO seconds later. Because of this delay in detecting a drop, the rate of incoming packets will not change immediately after the queue becomes full and multiple drops are expected. To complete our model we need to know which flows will suffer a drop during this interval. To determine exactly the set of flows $\mathcal{D} \subset \{1, 2, \ldots, n\}$ 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

$$\mathcal{D}(t) = F_{\text{drop}}\big(w_1, w_2, \ldots, w_n\big). \tag{5}$$

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 essentially discrete phenomena, we can incorporate them in our hybrid model by considering distinct modes (or discrete states) for the system. Four modes should be considered to cover all possible cases: slow-start or congestion avoidance and, in each case, queue full or not. The queue-full modes are active from the moment q reaches $q_{\max}$ until the drops are detected and congestion control reacts leading to a decrease in the queue size q. The time it takes for this to happen is either RTO or RTT depending on whether w_i was smaller than 4 or not at the time of the drop. When the drop is detected, the flows in $\mathcal{D}$ will suffer the appropriate changes in their window sizes and slow-start thresholds. The transition from slow-start to congestion avoidance occurs when the window size w_i exceeds the slow-start threshold w_i^{th}. The system is initialized with all w_i equal to one and w_i^{th} equal to infinity. Figure 1 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 and slow-start thresholds w_i, w_i^{th},

$i \in \{1, 2, \cdots, n\}$, and a timing variable t_{tim} used to enforce that the system remains in the *queue-full* modes for either RTO or 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 labeled with their enabling conditions (followed by "?") and any necessary reset of the continuous state that must take place when the transition occurs (with the corresponding assignments denoted by :=). We assume here that a jump always occurs when the transition condition is enabled. The transition on the top-left entering the *slow-start/queue-not-full* represents the system's initialization. This model is consistent with most of the hybrid system frameworks proposed in the literature (cf. [4] and references therein).

Although we focused our presentation on Reno congestion control, it is also possible to construct hybrid models for other congestion control algorithms [19].

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 [20].

III. Analysis of TCP-Reno with drop-tail queuing

In this section we study the dynamics of the TCP-Reno congestion control model in Figure 1 under drop-tail queuing and infinitely large advertised window size. We assume here that the window sizes do not decrease below 4 and therefore the system only stays in slow-start for a brief initial period.

To complete the hybrid model it remains to specify the drop dynamics that determine the set of flows $\mathcal{D}$ that suffer drops while the system is in one of the *queue-full* modes. It turns out that under drop-tail queuing policy exactly one drop per flow will occur in most operating conditions [5]. 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

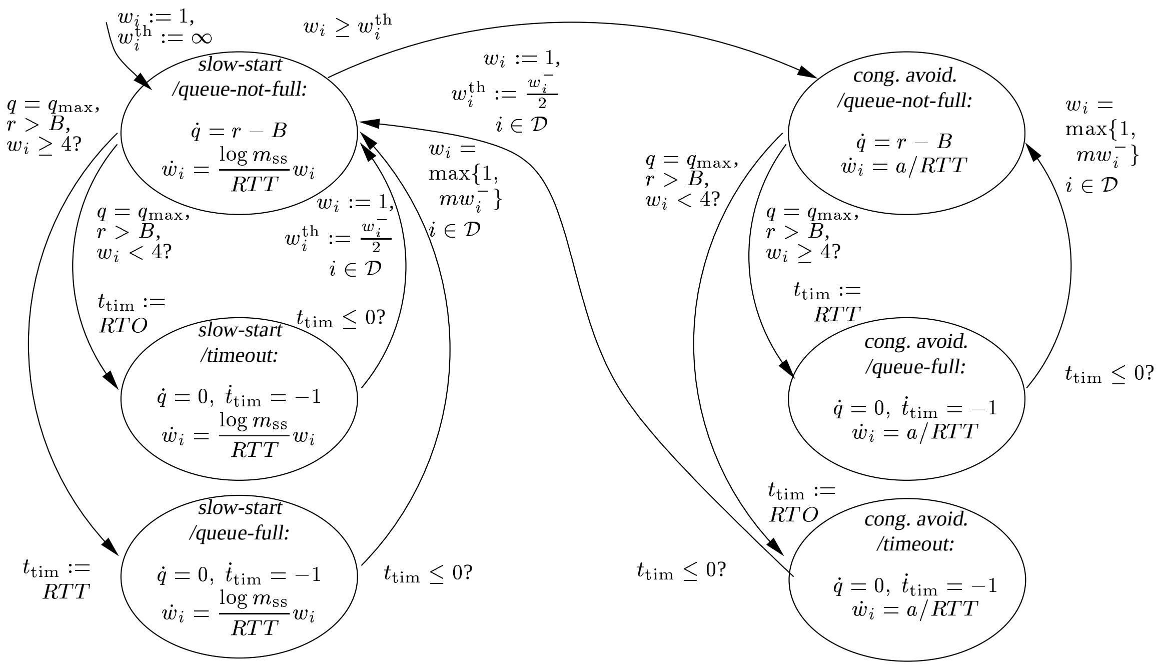
$$\mathcal{D} = F_{\text{drop}}\big(w_1, w_2, \dots, w_n\big) := \{1, 2, \dots, n\}. \qquad (6)$$

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 III-B that, for most operating conditions, this model accurately matches packet-level simulations performed using the `ns-2` network simulator [21]. In fact, this hybrid model only fails when the number of flows is so large that the drop rates take very large values.

A. Transient Behavior

We proceed now to analyze the joint evolution of the window sizes of all the flows. Our analysis shows 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 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. The following is proved in [19] (the proof was omitted here for lack of space.)

Theorem 1: Let $\{t_k : t_k \leq t_{k+1}, k \geq 1\}$ be the set of times at which the system enters the *congetion avoidance/queue-not-full* mode. For infinitely large advertised window size w_i^{adv} and $q_{\max} + BT_p \geq \frac{2ma}{1-m}n$, all the



$w_i := 1,$ $w_i^{\rm th} := \infty$
$w_i \geq w_i^{\rm th}$
slow-start /queue-not-full:
$\dot q = r - B$
$\dot w_i = \frac{\log m_{\rm ss}}{RTT} w_i$
$w_i := 1,$ $w_i^{\rm th} := \frac{w_i^-}{2}$ $i \in \mathcal{D}$
cong. avoid. /queue-not-full:
$\dot q = r - B$
$\dot w_i = a/RTT$
$w_i = \max\{1, m w_i^-\}$ $i \in \mathcal{D}$
$q = q_{\max},$ $r > B,$ $w_i \geq 4?$
$w_i = \max\{1, m w_i^-\}$ $i \in \mathcal{D}$
$q = q_{\max},$ $r > B,$ $w_i < 4?$
$q = q_{\max},$ $r > B,$ $w_i < 4?$
$w_i := 1,$ $w_i^{\rm th} := \frac{w_i^-}{2}$ $i \in \mathcal{D}$
$q = q_{\max},$ $r > B,$ $w_i \geq 4?$
$t_{\rm tim} := RTO$
$t_{\rm tim} \leq 0?$
$t_{\rm tim} := RTT$
slow-start /timeout:
$\dot q = 0,\ \dot t_{\rm tim} = -1$
$\dot w_i = \frac{\log m_{\rm ss}}{RTT} w_i$
cong. avoid. /queue-full:
$\dot q = 0,\ \dot t_{\rm tim} = -1$
$\dot w_i = a/RTT$
$t_{\rm tim} \leq 0?$
$t_{\rm tim} := RTO$
slow-start /queue-full:
$\dot q = 0,\ \dot t_{\rm tim} = -1$
$\dot w_i = \frac{\log m_{\rm ss}}{RTT} w_i$
$t_{\rm tim} := RTT$
$t_{\rm tim} \leq 0?$
$t_{\rm tim} \leq 0?$
cong. avoid. /timeout:
$\dot q = 0,\ \dot t_{\rm tim} = -1$
$\dot w_i = a/RTT$

Fig. 1. Hybrid model for TCP-Reno.

$w_i(t_k)$, $i \in \{1, 2, \ldots, n\}$ converge exponentially fast to

$$w_\infty := \frac{ma}{1-m}\left(f^{-1}(s_\infty) + 1\right), \tag{7}$$

as $k \to \infty$ and the convergence is as fast as m^k. In (7), s_∞ is the unique solution to

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

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

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

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 IV and V a detailed discussion of the implications of Theorem 1 and proceed with the analysis of the model.

B. 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 [11], [12], [7], [13] can also be derived from our model. In this section we concentrate on the case where s_∞ is much larger than one and therefore

$$f^{-1}(s_\infty) \approx s_\infty + 1. \tag{8}$$

This approximation is valid when

$$n \ll \frac{1-m}{2ma}(q_{\max} + BT_p) \tag{9}$$

and causes the system to remain in the state *congestion-avoidance/queue-not-full* for, at least, a few round-trip times[3]. In practice, this is quite common and a deviation from (9) 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 *congestion-avoidance/queue-not-full* state. Somewhere in this interval lies the time instant $\bar t_k$ at which the system enters the *congestion-avoidance/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

[3]When the system remains in the *queue-not-full* for at least 4 round-trip times, (8) already yields an error smaller than 2%.

is given by

$$r(t) = \begin{cases} \frac{\sum_{i=1}^{n} w_i(t)}{RTT(t)} & t \in [t_k, \bar{t}_k) \\ B & t \in [\bar{t}_k, t_{k+1}] \end{cases} \tag{10}$$

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

$$N_k := \int_{t_k}^{t_{k+1}} r(t)dt \approx \frac{1-m^2}{2an}(q_{\max} + BT_p + 2an)^2. \tag{11}$$

Details on the computation of the integrals in (11) and in (13) below are given in [19]. Since n drops occur in the interval $[t_k, t_{k+1}]$, *the average drop rate* p is then equal to

$$p := \frac{n}{N_k} \approx \frac{2a}{1-m^2}\left(\frac{n}{q_{\max} + BT_p + 2an}\right)^2. \tag{12}$$

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

$$\begin{aligned} \overline{RTT} &:= \frac{\int_{t_k}^{t_{k+1}} r(t) RTT(t) dt}{N_k} \\ &\approx \frac{1}{T}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\frac{q_{\max} + BT_p + 2an}{n} - a\frac{1-m}{1+m}\right), \end{aligned} \tag{13}$$

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 (12) that

$$\frac{q_{\max} + BT_p + 2an}{n} \approx \sqrt{\frac{2a}{(1-m^2)p}}. \tag{14}$$

This, in turn, can be used together with (13) to estimate the average throughput T. In fact, from (13) and (14) 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 1:

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

$$p \approx \frac{2a}{1-m^2}\left(\frac{n}{q_{\max} + BT_p + 2an}\right)^2, \tag{15}$$

$$\overline{RTT} \approx \frac{n}{B}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\frac{q_{\max} + BT_p + 2an}{n} - a\frac{1-m}{1+m}\right), \tag{16}$$

$$T \approx \frac{1}{\overline{RTT}}\left(\frac{2}{3}\frac{1-m^3}{1-m^2}\sqrt{\frac{2a}{(1-m^2)p}} - a\frac{1-m}{1+m}\right). \tag{17}$$

To verify the formulas in Theorem 2, we simulated the dumbbell topology of Figure 2, using the `ns-2` network simulator [21]. Figure 3 summarizes the results obtained

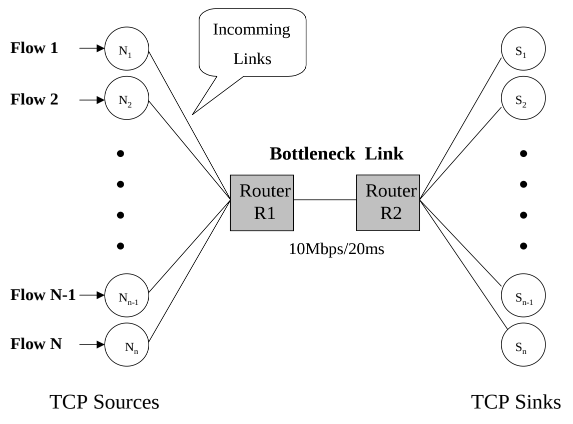


Fig. 2. Dumbbell topology with n TCP flows, 10 Mbps bottleneck link, 100 Mbps incoming links, 40 millisecond round trip propagation delay, and queue size at the bottleneck link of 250 packets.

for a network 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 (15)–(17) 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

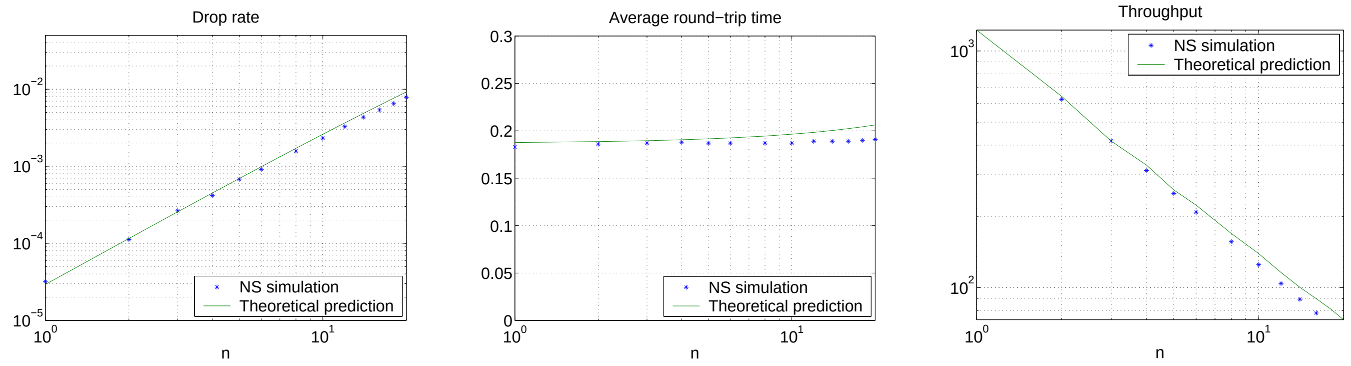


Fig. 3. Comparison between the predictions obtained from the hybrid model and the results from `ns-2` simulations.

crude modeling of the fast-recovery algorithm [16]. 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.

C. Comparison with Previous Results

For $a = 1$ and $m = 1/2$, the formula (17) becomes

$$T \approx \frac{1}{\overline{RTT}} \left(\frac{1.27}{\sqrt{p}} - \frac{1}{3} \right). \tag{18}$$

For reasonable drop rates, the term $\frac{1.27}{\sqrt{p}}$ dominates over 1/3 and (18) matches closely similar formulas derived in [11], [12], [7], [13]. However, the analysis presented here goes several steps further than the ones presented in these references because of the following: (i) previous derivations of (17) ignored queuing, assumed constant round-trip time, considered a single flow, and ignored transient behavior; (ii) the results in Theorem 1 provide information about the transient behavior of the individual flows; (iii) Theorem 2 also 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. It is important to emphasize that $\overline{RTT}$ in (16) 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.

IV. 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 dumbbell topology. Figure 4 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.

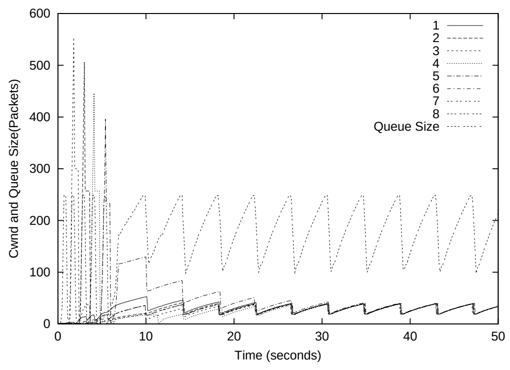


Fig. 4. Congestion window and bottleneck link queue size for the default dumbbell topology with $n = 8$ flows.

Window size synchronization had been observed in [10], [5] for TCP-Tahoe congestion control [6] and actually led to the development of Random Early Detection/Drop active queuing [22], [20]. In [5], 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 [5] 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. 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 2. 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 *congestion-avoidance/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 5 that shows a trace of packets arriving at the bottleneck link from each of the incoming links. The two plots in figure 5(a) correspond to the topology in Figure 2 with a bandwidth of 20Mbps in the incoming link, versus 10Mbps in the bottleneck 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 5(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 5(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.

We also investigated what effect different queuing disciplines have on flow synchronization. Figure 6(a) shows that by using a simple *drop-head* queuing policy (drop the packet at the head of the queue)—while keeping all the other dumbbell characteristics the same—we are able to eliminate synchronization. Figure 6(b) shows that Random Early Detection (RED) queuing also eliminates flow synchronization by adding randomization to the network, as suggested in [22].

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 [23], [24]. However, when compared to random-drop and RED, drop-head has the 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.

V. 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, had not been developed before. Indeed, 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 7(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 packets. 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 simula-

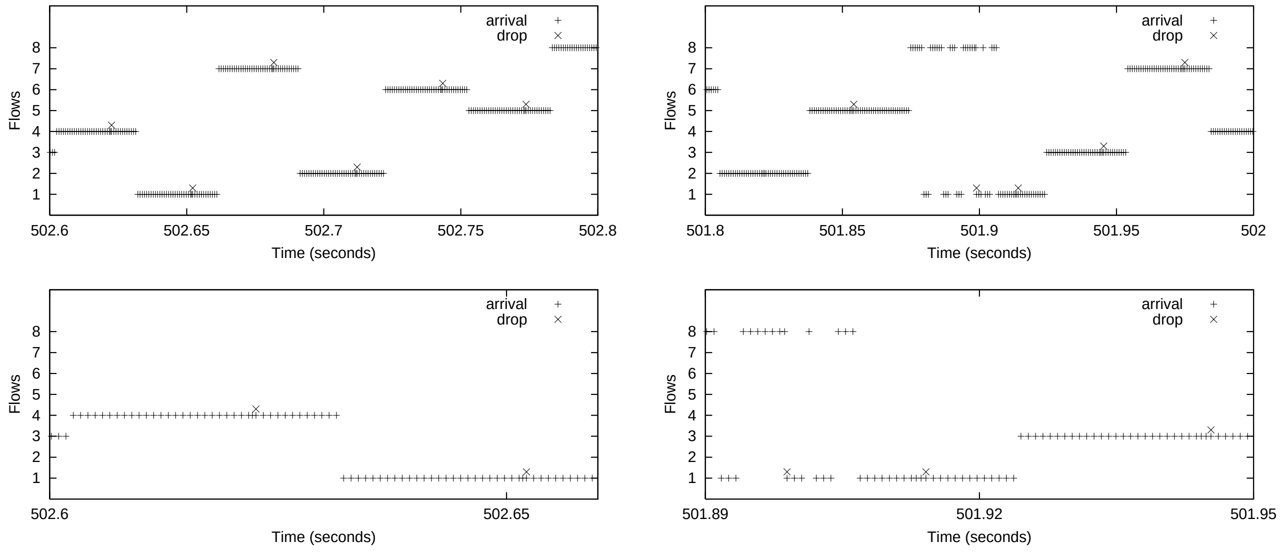

(a) (b)

Fig. 5. Packet traces for (a) 20 Mbps and (b) 15 Mbps incoming link bandwidths.

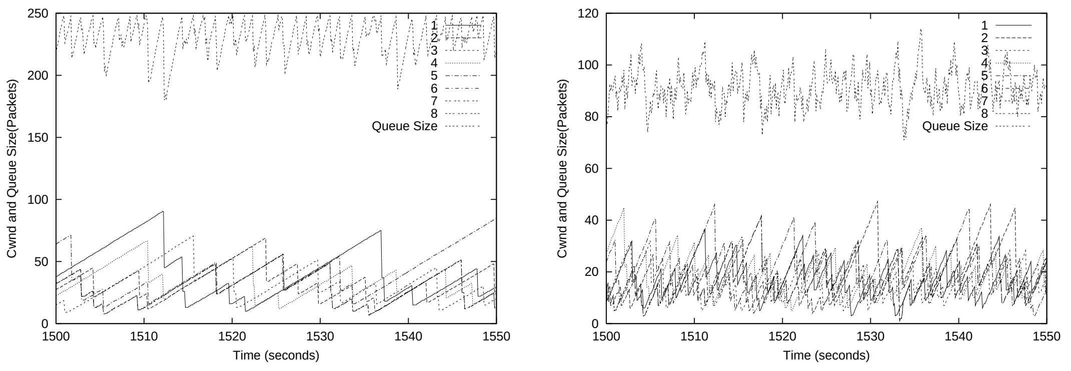

(a) (b)

Fig. 6. Congestion window and bottleneck link queue size for the default dumbbell topology using (a) drop-head queuing and (b) RED queuing.

tion. 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 7(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 8 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 latter 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

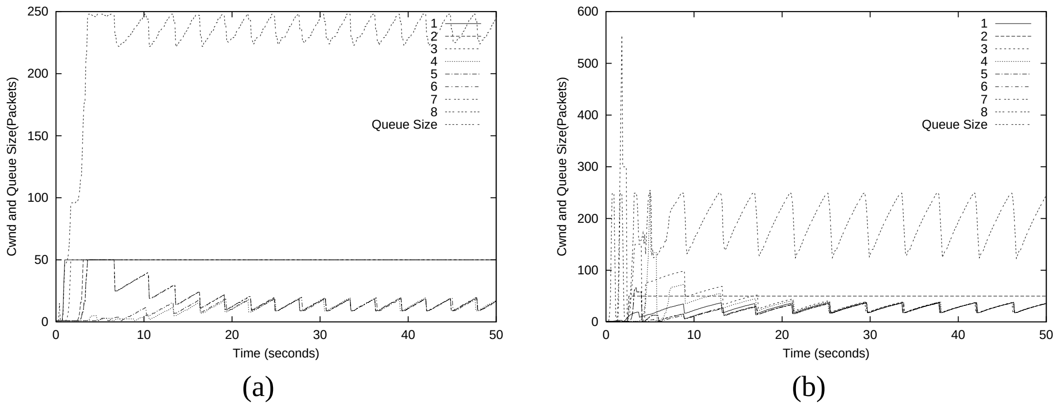


Fig. 7. Congestion window and bottleneck link queue size for the default dumbbell topology when (a) all flows have 50-packet advertised window size and (b) only flows 1 and 2 have finite (50-packet) advertised window size. In (a) four of the flows exhibit a normalized throughput of 138% and the remaining a normalized throughput in the range $37\% \pm 1\%$, in (b) one flow exhibits a normalized throughput of 167% and the remaining a normalized throughput of $90\% \pm .5\%$

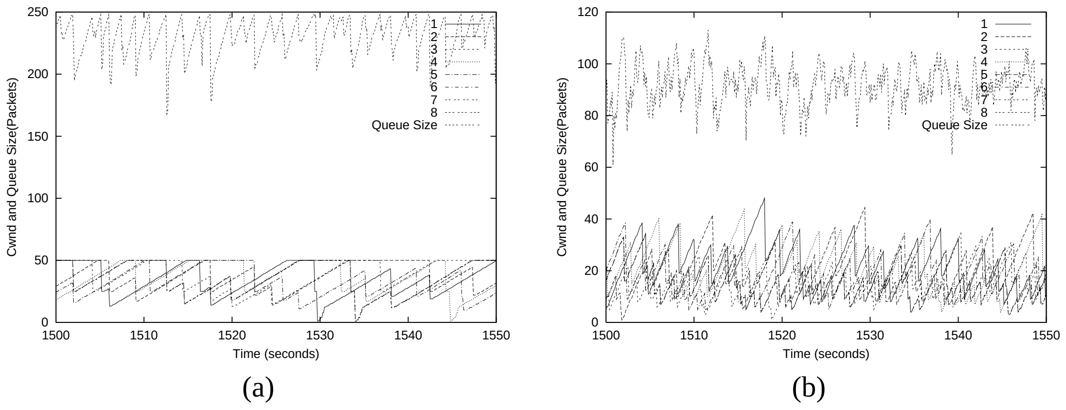


Fig. 8. Congestion window and bottleneck link queue size for the default dumbbell topology where all flows have 50-packet advertised window size using (a) drop-head and (b) RED queuing. In (a) all flows exhibit a normalized average throughput in the range $1 \pm 3.6\%$ and in (b) in the range $1 \pm 2.5\%$.

shown in Figure 9.

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 Figure 1 already take into account finite advertised windows sizes but the drop model (6) in Section III does not. One simple model that can capture the phenomena described above with drop-tail queuing is given by

$$\mathcal{D} = F_{\text{drop}}\big(w_1, w_2, \dots, w_n\big) := \{i : w_i < w_i^{\text{adv}}\}, \quad (19)$$

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 (19), 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 (19) 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.

VI. Conclusions

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

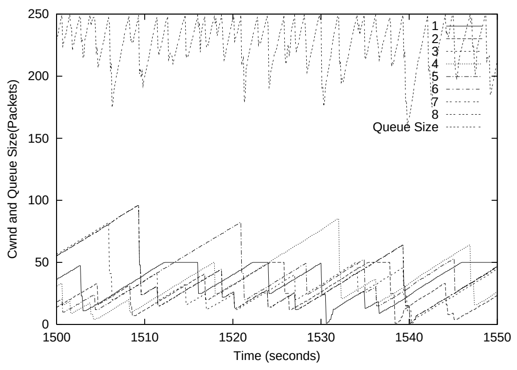


Fig. 9. Congestion window and bottleneck link queue size for the default dumbbell topology where flows 1 and 2 have 50-packet advertised window size using drop-head queuing. The flows with finite advertised window size exhibit a normalized average throughput in the range $90.2\% \pm 2.1\%$ and the remaining flows in the range $103.3\% \pm 1.8\%$.

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.

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.

To our knowledge, this was the first time hybrid systems were used to model congestion control. We fully characterized TCP's behavior in the dumbbell topology, employing powerful theoretical tools available for hybrid systems. When compared with previous work, we provided a more complete characterization of TCP, demonstrating the potential of hybrid systems as a modeling tool for congestion control algorithms. We are now in the process of generalizing the analysis presented here to more complex network topologies and traffic loads (e.g., flows with different propagation delays and different duration), other congestion control mechanisms (such as delayed acknowledgments, fast recovery), other TCP variations (e.g., TCP-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.

REFERENCES

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

[2] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer, "Equation-based congestion control for unicast applications," in *Proc. of SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 43–56.

[3] D. Bansal and H. Balakrishnan, "Binomial congestion control algorithms," Proc. of INFOCOMM, Apr. 2001.

[4] Arjan van der Schaft, *An Introduction to Hybrid Dynamical Systems*, Number 251 in Lecture Notes in Control and Information Sciences. Springer-Verlag, London, 2000.

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

[6] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe Reno and SACK TCP," *Computer Comm. Review*, vol. 27, no. 3, pp. 5–21, July 1996.

[7] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *Computer Comm. Review*, vol. 27, no. 3, July 1997.

[8] Robert Morris, *Scalable TCP Congestion Control*, Ph.D. thesis, Harvard University, Cambridge, MA, Jan. 1999.

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

[10] E. Hashem, "Analysis of random drop for gateway congestion control," Tech. Rep., MIT, 1990.

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

[12] 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.

[13] V. Misra, W. Gong, and D. Towsley, "Stochastic differential equation modeling and analysis of TCP-windowsize behavior," in *In Proceedings of PERFORMANCE99*, Istanbul, Turkey, 1999.

[14] 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.

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

[16] Van 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.

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

[18] Y. Yang and S. Lam, "General AIMD congestion control," in *Proc. of ICNP*, Osaka, Japan, Nov. 2000.

[19] Stephan Bohacek, João Pedro Hespanha, Junsoo Lee, and Katia Obraczka, "A hybrid systems framework for TCP congestion control: A theoretical model and its simulation-based validation," Submitted to the 39th Annual Allerton Conference on Communication, Control, and Computing., July 2001.

[20] Sally Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[21] 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.

[22] Sally Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, pp. 115–116, Sept. 1992.

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

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