TCP Behavior in Sub-Packet Regimes

Jay Chen* Janardhan Iyengar† Lakshminarayanan Subramanian* Bryan Ford‡

ABSTRACT

Congestion control algorithms such as TCP-NewReno assume typical per-flow throughput is at least 1 packet per roundtrip time. Environments where this assumption does not hold, a largely unexplored space we call the *sub-packet regime*, are common in developing regions, where a 128Kbps–2Mbps access link may be shared by 50–200 users. This paper investigates the impact of pathological-sharing on TCP and other end-to-end congestion control schemes. Through experience and analysis we find that common congestion control algorithms break down in the sub-packet regime, resulting in severe unfairness, high packet loss rates, and flow silences due to repetitive timeouts. To understand TCP's behavior in this regime, we propose a model particularly tailored to high packet loss-rates and relatively small congestion window sizes. We validate the model under a variety of network conditions and discuss how the model can be potentially used in practice to enhance TCP performance in the sub-packet regime.

Keywords

TCP, congestion control, developing regions, low bandwidth, TCP model

1. INTRODUCTION

Existing congestion control schemes such as TCP-NewReno [8], TFRC [13] and many others assume the fair-share bandwidth of a flow is at least 1 packet per roundtrip time (RTT). The TCP-friendly rate of a flow [22], as defined by the approximate packet rate of $\sqrt{3/2}/(RTT\sqrt{p})$, where p is the observed loss rate, also satisfies this criterion: since the packet loss rate p must be less than 1, the TCP-friendly rate is at least $\sqrt{3/2}$ packets per RTT.[1] Conventional wisdom is that typical networks provide sufficient per-flow bandwidth to satisfy this assumption. However, there are a surprising number of environments where low-bandwidth networks are being shared by too many users, causing this assumption to be flawed.

We define such an environment, where a TCP flow's fair share is less than 1 packet per RTT, as the *sub-packet regime*. The sub-packet regime is surprisingly common and increasingly important in developing regions where a low-bandwidth network is often shared by many users [14,26]. In a developing country such as India, most academic institutions (universities, high schools) and small companies have a medium to low bandwidth network shared by many simultaneously active users. One premier university in India, for example, shares a 2Mbps connection among up to 400 active users at peak times, frequently causing users to experience stalled web sessions lasting more than 120 seconds. Even as bandwidth increases in the future, high levels of sharing will continue due to economic factors of aggregate purchasing power: a single user may not be able to afford Internet connectivity, but a group of users collectively have sufficient purchasing power to afford and share a single connection.

The sub-packet regime has not been a traditionally important region of operation for network flows, and as a result this space has remained relatively unexplored. The concept of a sub-packet regime arises in prior work in the context of understanding the behavior of TCP in the face of many competing flows [16, 25, 29]. We are specifically interested in exploring both individual and aggregate behaviors of TCP, where the per-flow share is significantly lower than 1 packet per RTT.

This paper makes several contributions towards characterizing TCP behavior in sub-packet regimes. By analyzing the per-flow and aggregate behavior of TCP and other variants in the sub-packet regime, we show that apart from the well-known problems of high loss rates and poor performance, flows experience the following: (a) repetitive timeouts which forces a significant fraction of flows to observe long silence periods with no packet transmissions; (b) extreme unfairness over the short time scale; and (c) unpredictable flow completion times. In addition, we show that none of the standard TCP variants or known queuing mechanisms offer substantial performance gains in the sub-packet regime.

To better understand this phenomenon, we introduce an analytical model to characterize the equilibrium behavior of TCP in the sub-packet regime. Our model is a simpler variant of a full Markov model for TCP operating in traditional regimes [10], but with careful attention in modeling repetitive timeouts. Since Markov models are inherently not suited to keep memory in the state transitions, modeling repetitive timeouts[2] is not straightforward (since one needs memory of the previous timeout value). We address this problem by determining aggregate transition states which both capture the memory effect while significantly reducing the number of

‡New York University, Email: {jchen, lakshmi}@cs.nyu.edu
‡Franklin and Marshall College, Email: jiyengar@fandm.edu
‡Yale University, Email: bryan.ford@yale.edu

[1] The only way to reduce the rate further is by adding timeouts.

[2] timeouts with backoff values greater than 1.

states. Our model accurately predicts the stationary distribution of a TCP flow across different states using few aggregate states. We discuss possible ways in which the model can be used in practice to enhance TCP performance and fairness in sub-packet regimes.

The paper is organized as follows. Section 2 defines the sub-packet regime and explores observed behavior of TCP aggregates. Section 3 explains what happens in this regime through a TCP model that we develop, Section 5 discusses the strengths and limitations of the model and Section 6 discusses insights gained from our model and how it can be potentially used in practice. Section 7 discusses related work, and we present our conclusions in Section 8.

2. THE SUB-PACKET REGIME

In this section, we first define the sub-packet regime and describe an end-user's view of web browsing behavior in this regime using an analysis of real-world access traces. We then use simulations to analyze TCP's behavior in the sub-packet regime to explain what makes TCP break down in this regime.

2.1 Defining the Sub-Packet Regime

We define the *sub-packet regime* as the region of TCP operation when competition between flows results in per-flow fair-shares less than 1 full-sized segment (maximum segment size or MSS) per observed round-trip time (RTT). A TCP flow with segment size S and round-trip time of RTT, is in the sub-packet regime if both of the following conditions hold at the bottleneck link, which has capacity C:

1. number of competing flows, $N \gg 1$, and
2. per-flow fair share is less than S/RTT.

The fair share of a flow on a bottleneck link is inversely proportional to its RTT and is also dependent on the RTT of competing flows on that link. If all flows have the same RTT, then the fair share is C/N. The first condition alone holds in the aggregate links that most Internet traffic goes through, but per-flow fair share on these high-bandwidth links is usually large enough to keep TCP out of the sub-packet regime. The second condition alone holds when one or a few TCP connections use a low-bandwidth link, such as a WiFi or cellular link, but the minimal sharing keeps these links out of the sub-packet regime. It is the *combination* of heavy sharing, on the order of several 10s to hundreds or thousands of competing flows, operating over low-bandwidth networks that causes the sub-packet regime. As we discussed in Section 1, such sharing is all too common; we now examine why this sharing is pathological.

We note that statistical multiplexing may cause flows to exhibit sub-packet-regime-like behavior when the average per-flow fair share *approaches* 1 packet per RTT. We thus also consider flows in the region *approaching* the sub-packet regime.

2.2 Pathological Sharing: An End-User View

We first examine the web browsing experience at a real university campus in India. The university is equipped with a 2Mbps access link to the Internet, and has about 400 computers on campus usable for Internet access. We start by examining object download times recorded by the university's web proxy, constrained to a 2-hour window to minimize the effects of time-of-day load variations. During this period, the proxy recorded 221 unique client IP addresses, and downloaded 1.5GB over the access link. Figure 1 shows a scatter plot of download times for objects of various sizes, ignoring cached objects and HTTPS connections, based on logarithmically-sized buckets of object sizes. Figure 1 shows the 10th percentile, 90th percentile, min, max, and average download times within each bucket.

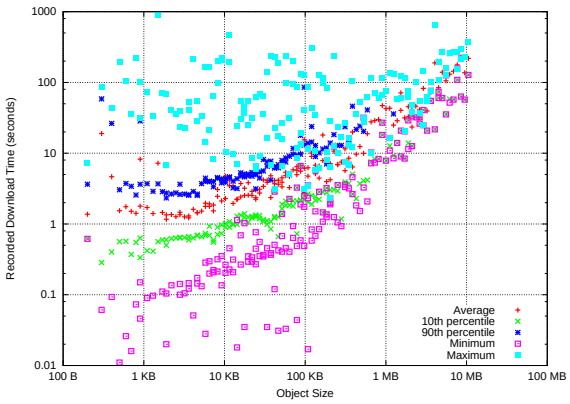

Figure 1: Scatter-plot of download times for different object sizes, taken from a 2-hour observation period at the University's Squid proxy. Each raw data point is assigned to a bucket, and the values shown here are the 10th percentile, 90th percentile, min, max, and average values per bucket.

The Y-axis spread in the graphs is striking: download times for objects vary by over two orders of magnitude![3] We can observe that this variation is pervasive: many flows witness long download times across all file sizes. Download time variation eventually decreases at larger object sizes (e.g., 1MB), but most web pages and objects reside in the size region where variation is high.[4]

We make two observations. First, a significant number of users experience poor performance with large download times even for very small object sizes where the entire object can be transmitted in a few packets. Second, the high variation in the download times for comparable object sizes indicates a high level of unfairness across flows. To understand these phenomena, we next use simulations to describe TCP's behavior in the sub-packet regime.

2.3 TCP in the Sub-Packet Regime

The behavior of TCP in the presence of many flows has been studied in prior work [16, 25, 29]. What is known from these works is that under high contention, TCP flows experience high packet loss rates leading to poor per-flow throughput and unfairness across flows. The small pipe case analysis in Qiu et al. [29] also shows that a small set of flows capture the entire bandwidth while a number of flows remain shut off; however, flows do not exhibit global synchronization problems and link utilization remains consistently high.

Building upon the analysis from prior work, we pinpoint three specific observations about TCP behavior in the sub-packet regime. First, individual flows experience repetitive timeouts frequently which results in long silence periods during which flows do not transmit a single packet. Second, flows experiences high levels of unfairness across variable time scales. While long term fairness is better than short term fairness, we observe that flows experience a random selection process where different small sets of flows progress during different short time scales. Finally, none of the existing variants of

[3]After normalizing server latencies via an emulated dumbell topology with uniform server latencies we find a reduced, but similar 1.5 to 2 orders of magnitude spread for all but the largest files.

[4]From a crawl of Alexa's top 100 websites (Jan 9, 2010), we found the mean page size (including embedded objects) to be about 350KB and the median to be about 150KB.

TCP and TFRC or existing variants of queuing mechanisms (RED, SFQ) address these problems in the sub-packet regime. We now describe these observations in greater detail using simulations. While we have performed several simulations under varied conditions, we present the simplest results that motivate these observations.

2.3.1 *Fairness in the sub-packet regime*

What happens to flow-level *fairness* under pathological sharing? We look at a simple simulation experiment, using a dumbbell topology in which flows congest a core link, which uses a simple tail-drop queue. Every edge node (user) spawns two TCP-SACK flows [21]. All traffic is one-way, reflecting download-centric web browsing. Since we wish to focus on congestion control dynamics, which are often obscured by delayed acks, our TCP receivers do not delay acks. The senders use an on-the-wire packet size of 500 bytes.

As the number of flows increases at the congested link, the overall average goodput remains consistently high (greater than 90%), for varying link bandwidths. But fairness among flows suffers, as Figure 2 demonstrates using the Jain Fairness Index (JFI) [15]:

$$fairness = \frac{(\sum x_i)^2}{n \sum (x_i)^2}$$

The JFI is a value between 1 and $1/n$, with 1 being the best fairness (exactly equal shares), and $1/n$ being the worst (one flow hogs the entire link). JFI values are most readily comparable when the number of flows is the same across the scenarios being compared; otherwise, interpreting the JFI can be problematic because worst-case fairness depends on n. We nevertheless use this index to note deviations from the ideal fairness of 1 since the focus of the experiment is not to measure the exact amount of unfairness in the system.

Long-term fairness is high, as seen for the flows that run for 10000 seconds (17 minutes). Unfairness over shorter periods of 20 seconds sets in, however, as per-flow fair-share drops below 30Kbps, or, with an RTT (including queueing delay) of about 400ms, below 3 packets per RTT. The choice of a 20 second window for our analysis is pragmatic: as expected, fairness becomes worse with shorter windows and better with longer windows.

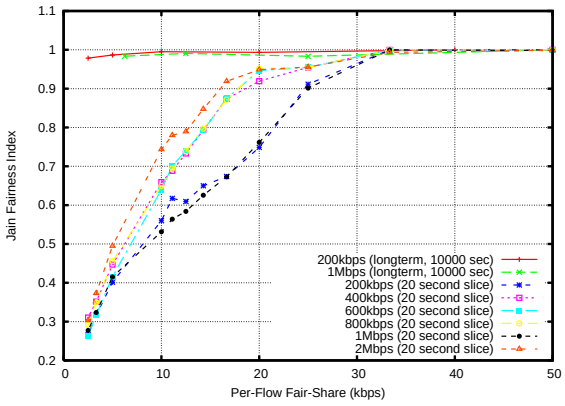

Figure 2: Long- and short-term Jain-fairness as a function of ideal per-flow fair-share, for different capacity bottleneck links.

To better understand how the network manages and how the system distributes bandwidth among sub-packet regime flows, we now take a closer look at the dynamic behavior of the flow aggregate. We simulate a 1Mbps link with a propagation RTT of 200ms and a 200ms packet buffer (1 RTT worth of buffer), leading to a maximum observed RTT of 400ms. Figure 3 shows a CDF of flow goodput when 100 flows are sharing this link (a per-flow fair-share of 10kbps or 1 packet-per-RTT). Each curve shows average goodput over a 20-second timeslice.

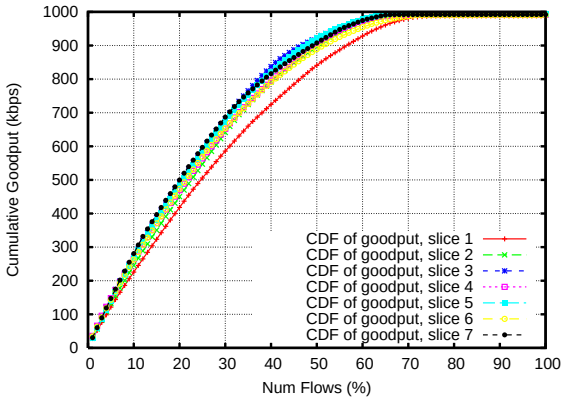

Figure 3: CDF, with 100 flows, of TCP goodput in 20-second slices

We make the following observations. First, in any given slice, at most 70 flows are active; the remaining 30 flows in every slice get zero bandwidth. Second, the distribution of goodput among flows is highly consistent across all the timeslices. While 70% of the flows are active in any given slice, there are 3 distinct groups of flows in each slice: 40% of the flows get about 80% of the link, about 30% of the flows share the remaining 20% of the link, and about 30% of the flows get "locked out" of the system in each slice. Third, no flow starves forever, since, as we've seen in Figure 2, every flow gets its fair-share of the link in the long term.

These observations lead us to the following significant result: While ensuring that all flows get admitted for equal shares in the long-term, the emergent flow management mechanism in the system is to perform *arbitrary admission control* of flows within shorter time slices. The admission control helps the few admitted flows make progress, but its arbitrariness causes the huge download time variation seen in Figure 1.

2.3.2 *Repetitive timeouts*

Consider a user who spawns a pool of TCP connections from a web browser. We define a *user-perceived hang* as an event where the user receives no data and observes no progress on the web browser for some time. The length of a user-perceived hang is the duration during which *none* of the browser's pool of simultaneous TCP connections receives any data. We look at simulation traces of such web transfers in the ns2 simulator on a pathologically-shared link, by creating users that spawn multiple TCP connections each (a web session pool), all sharing a bottleneck link of 1Mbps capacity. The propagation RTT is 200ms, and the bottleneck link has 50 packets worth of buffer space (one RTT worth of delay). The experiment is run for 10 minutes.

Figure 4 shows the longest hang experienced by the different users in the system. This figure shows only the single longest hang, not the total of hang times experienced by the user—that number would be higher.

With four flows per user and 200 active users, all users perceive at least one hang time longer than 20 seconds, and with 400 users, almost 50% of the users perceive at least one hang longer than a minute. While spawning fewer connections per user helps reduce congestion, Figure 4 also shows that fewer connections worsen the user experience by increasing the chance that all of a user's con-

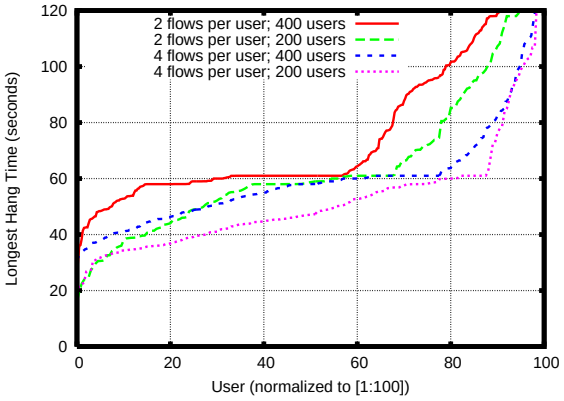

Figure 4: Longest *hang* time experienced by users, observed over a 10 minute period. Hang times are sorted in increasing order, and the X-axis shows users normalized to be in the range [1:100].

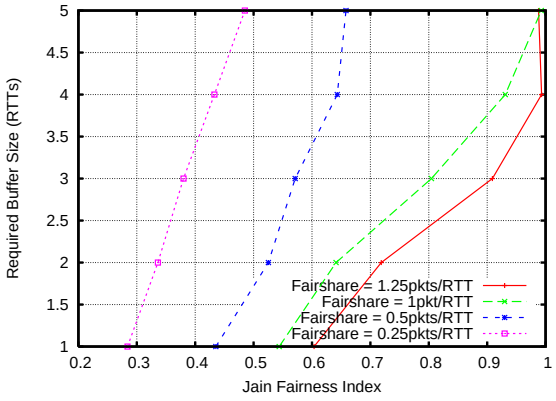

Figure 5: Buffer sizes required for restoring fairness, considering 20-second slices

nections stall at once.[5]

We briefly describe TCP's behavior under these web transfers. First, the TCP flows experience a high loss rate (30%) which is about the same as the average loss rate at the link. Second, an analysis of the timeout values used by TCP senders for setting their retransmission timers reveals huge variance. Exponentially increasing timer backoffs indicate that flows are repeatedly losing packets, without an opportunity to reset their retransmission timer with a successful new transmission. These repeated timeouts result in increasing silence periods on the link for those flows.

2.4 Understanding the Alternatives

We now consider the effect of buffer size on fairness, and briefly discuss what happens with other end-to-end schemes and queueing mechanisms.

2.4.1 Using larger buffers

In dealing with pathological sharing, Morris [25] observes that provisioning the bottleneck router to buffer up to 10 packets per flow restores fairness. Increasing buffer sizes simply trades in delay and delay variance for fairness. Figure 5 shows the tradeoff required for achieving fairness for a small part of the sub-packet regime. Increasing buffers is infeasible, particularly deep in the sub-packet regime. After increasing buffer sizes, the corresponding increases in delay can be disproportionately high. At 1000 bytes a packet, with 800 TCP connections competing at a bottleneck link of 2Mbps capacity, the maximum queueing delay necessary to achieve fairness is 32 seconds—a delay which most traditional applications are not equipped to deal with, and many others (such as real-time apps) would find unacceptable. Under low load, these large queues result also in large RTT variations that can be problematic for applications and for TCP's RTT estimator.

2.4.2 TFRC vs. TCP in the sub-packet regime

TFRC uses equation-based congestion control and pacing, which makes it less bursty than TCP and a distinctly different candidate as compared to other TCP variants. Pacing allows TFRC to perform better than TCP in terms of short-term fairness, as shown in Figure 6. but coarse-grained timers in non-realtime operating systems hinder TFRC's ability to pace well [13].

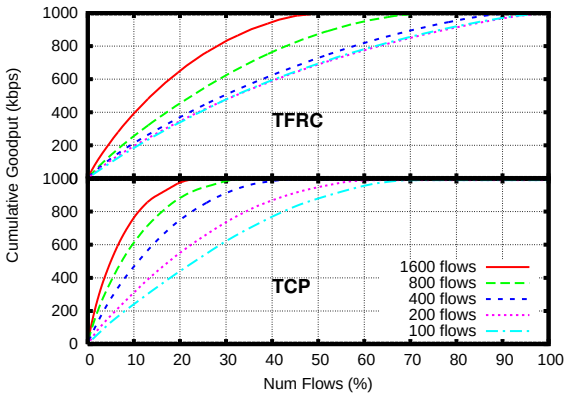

Figure 6: TFRC and TCP goodputs over 20-second slices, across different loads.

While TFRC's pacing allows for finer rate control than TCP in sub-packet regimes, TFRC too breaks down, albeit deeper in the sub-packet regime than TCP, when it encounters timeouts.

A noteworthy and unintuitive result is that while link utilization is the same for both, and while TCP is the bursty sender of the two, TFRC observes a loss rate (20.7%) much higher than TCP does (12.8%). Bursty TCP senders send more than one packet in a burst, and a loss episode often results in the entire burst being dropped; such TCP flows appear locked out and are silent afterwards. With TFRC however, the pacing of transmissions results in fewer burst losses within any flow and therefore fewer flows appearing locked out; consequently, while the flows are fairer to each other, more packet losses are observed due to more active senders than with TCP.

2.4.3 AQM and other TCP variants

Using other queueing schemes such as Random Early Detection (RED) [9] or Stochastic Fair Queueing (SFQ) [23] under the same settings does not change the behavior of the aggregate by much. RED and SFQ both need much larger buffer sizes for the number of flows being managed, and they fall short of their promise in this regime. XCP [17] offers rate controls, again assuming at least one packet per RTT. Delay-based end-to-end mechanisms such as TCP Vegas [6] break down in this domain as well, since the levels of

[5]The longest hang time experiences a plateau at 64 seconds which is an artifact of the longest TCP timeout (RTO) period of 64 seconds in the ns2 simulator [28].

flow-level contention and queue occupancy are too high, and Vegas too assumes at least a packet per RTT of bandwidth. TCP variants that are in common use—NewReno [8], SACK [21], Compound TCP [35], Cubic TCP [12]—all implicitly assume that the fair share is at least one packet in each RTT. A fair share that is less than 1 packet per RTT sends TCP into timeout periods, since that is the only way TCP can send at at rate lesser than 1 packet per RTT.

3. MODELING TCP IN SUB-PACKET REGIMES

In this section, we build a simple model particularly tailored for analyzing the behavior of TCP-NewReno in sub-packet regimes with high-packet loss-rates and with relatively small average congestion windows. The main purpose of this model is to analyze the *stationary distribution* of a set of TCP flows, which provides a detailed characterization of the state of a TCP connection. In other words, the model results in the stationary distribution probability, $P(congestion\ window\ =\ W)$, for low-values of W, and the probability of a flow being in a timeout or in a repetitive timeout phase. Since the model is ergodic, its stationary distribution corresponds to the probability that a set of flows is in a particular set of states at any point in time; the probability distribution holds irrespective of flow size.

There are several key insights we draw from this model. First, all the transition probabilities in this model are modeled using a single parameter p, the packet-loss probability at the bottleneck link. Second, under conditions where p is roughly a constant, the stationary distribution probability is only dependent on p and is independent of RTT. Third, we observe a shift in the stationary distribution beyond $p = 0.1$ where the probability of repetitive timeouts significantly increases thereby lowering the effective throughput of a TCP flow.

Our model is different from and extends previously proposed models of Padhye *et al.* [27], Fortin-Parisi *et al.* [10]. The fundamental problem with using a generic Markov model for capturing TCP behavior is state space explosion. One of the advantages in the sub-packet regime, however, is that state space is constrained and may be accurately captured using appropriate state transitions. Our model focuses on high loss rates and captures exponentially increasing silence periods due to repetitive timeouts, the dynamics of which are not captured in detail in previous work. The major challenge in the sub-packet regime model that we address is accurately capturing the timeout behavior of TCP flows, with a simple model of small windows. We distinguish our model from related work in more detail in Section 7.

The overall model is described in Figure 8. Given the complexity of the model, we will describe how we arrive at this model in stages, by first developing the *approximate model* in Figure 7. First, we describe how we start with a simple window-state based Markov model and how we arrive at transition probabilities. Then, we describe how we model simple timeouts and timeouts with memory of previous backoff value using three aggregate states. Finally, we describe how we expand the model to accurately handle timer backoffs.

3.1 Assumptions

We make the following simplifying assumptions. (1) We assume that the TCP flow is operating in sub-packet regimes, with most flows having a small *congestion window (cwnd)* [3] size. We assume a maximum window size in our model, W_{max}; the model may be extended to higher states by increasing W_{max}. (2) We assume that all TCP flows experience medium to high loss-rates in sub-packet regimes. In addition, we assume that the packet-loss probability is modeled using a single parameter p; this is a reasonable assumption since most TCP flows are operating in very low cwnd sizes ($cwnd = 1$ or $cwnd = 2$) resulting in packets of a TCP flow often being single or spaced-out. (3) We assume that traditionally "short-lived" flows do not experience the effects of losses any differently than "long-lived" flows; in sub-packet regimes, even a short flow that carries about 50 packets appears as a "long-lived" flow in the network, particularly due to extended silence periods from timer backoffs. (4) Finally, we assume that the typical *slow start threshold (ssthresh)* [3] is small enough that flows operate in congestion avoidance and not in slow start. With a maximum window size of W_{max}, the maximum ssthresh value is $W_{max}/2$. In this section, we use $W_{max} = 6$ and $ssthresh \leq 3$, but in practice, this could be extended for larger values. $ssthresh \leq 3$ also simplifies the TCP window increment function to be only additive, since window doubling does not go beyond 3.

3.2 Starting with a Simple Chain

We begin with a simple congestion window based chain model for TCP as illustrated in the top part of the approximate model in Figure 7 where S_n represents a cwnd of n. There are three possible transitions from S_n:

- $S_n \rightarrow S_{n+1}$, when all transmissions are successful, resulting in an increase of 1 in the sender's cwnd;
- $S_n \rightarrow S_{\lfloor n/2 \rfloor}$, when at least one transmission is lost and loss recovery happens through fast retransmissions;
- $S_n \rightarrow S_1$, when at least one transmission is lost and loss recovery happens through a timeout. Note that S_1 represents a timeout retransmit state, and the only way to reach S_1 is through a timeout.

Since each packet has an independent loss probability p, we have:

$$P(S_n \rightarrow S_{n+1}) = (1-p)^n \tag{1}$$

The ability to recover from a packet loss without timeout is dependent on fast retransmissions which is triggered by 3 duplicate acks (dupACKs); at least 3 packets must be successfully transmitted in the same window to recover from a packet loss. Therefore, no fast retransmission occurs in our model if a loss occurs when the sender's cwnd is smaller than 4.

While TCP-NewReno is equipped to handle multiple losses in a window [8], studies [30–32] have shown that both TCP-NewReno and TCP-SACK are unable to handle beyond a threshold of losses at low congestion windows. Also, if a flow experiences k packet losses in S_n, to recover using fast retransmit, k loss-free roundtrips are required to recover fully from all the losses [8]. Thus, in sub-packet regimes with high-loss rates, handling multiple losses for low-cwnds is fundamentally hard. In the loss recovery period following the fast retransmit (or, the fast recovery period), dropping any packet will lead the sender into a timeout period. In our model with $W_{max} = 6$, we assume that for states S_4, S_5, S_6, we are able to recover from at most one loss using a fast retransmission, and 2 or more losses result in a timeout at the sender.

The probability that a fast retransmission is triggered in state S_n is given by the probability that exactly one packet is lost, which is $np(1-p)^{n-1}$. The probability that a retransmission, once triggered, is successful is $(1-p)$. Hence, the fast retransmission transition probability from state S_n (for $n = 4, 5, 6$) is given as follows:

$$P(S_n \rightarrow S_{\lfloor n/2 \rfloor}) = np(1-p)^{n-1} \times (1-p) \tag{2}$$

In our model for $W_{max} = 6$, there are two circumstances under which a sender experiences a timeout: (i) when there are two

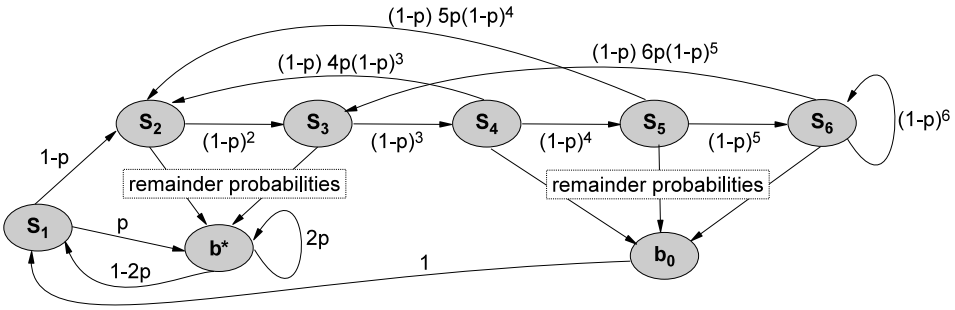

Figure 7: The Approximate Model. States S_1 and S_2 get expanded in the full model.

or more losses in a window, and (ii) when a retransmitted packet is dropped. This probability is simply computed as the residual probability:

$$P(S_n \rightarrow RTO) = 1 - P(S_n \rightarrow S_{n+1}) - P(S_n \rightarrow S_{\lfloor n/2 \rfloor}) \quad (3)$$

The upper part of Figure 7 shows these transitions put together. Note that S_2 and S_3 do not have fast retransmission transitions, and the sender never reaches a cwnd smaller than 2 through fast retransmissions [3].

3.3 Modeling Timeouts

Since timeouts are very common in the sub-packet regime, we capture two forms of timeouts in our model: (a) Simple timeouts; and (b) Timeouts with memory of the previous backoff value (repetitive timeouts). A *simple timeout* occurs when a TCP sender hits a timeout without memory of a previous timeout. In other words, when hitting a timeout, the sender's retransmission timer has a backoff value of 1. On hitting the timeout, the backoff value is doubled to 2. This increased backoff value extends the next timeout period by twice the base timer value, and "collapses" to the base value of 1 only when a new roundtrip time measurement is available, which only happens when cumulative acknowledgements arrive for newly transmitted (not retransmitted) data [28]. A *repetitive timeout* occurs when a flow hits a timeout before memory of previous timeouts is lost. In other words, when hitting a timeout, the sender's retransmission timer has a backoff value greater than 1. On hitting the timeout, the backoff value is doubled again by the TCP sender, causing increasingly extended silence periods.

3.3.1 Modeling simple timeouts

In our model with $W_{max} = 6$, we assume that we are in a simple timeout when we transition to a timeout state from a window size of $4, 5, 6$ (states S_4, S_5, S_6) since at least one new packet has been cumulatively acknowledged by the time the flow leaves state S_3 and reaches S_4, thereby resetting the timeout value. For simplicity, we consider the base timeout period $T_0 = 2 \times RTT$. Thus, in the event of a timeout, we capture the $2 \times RTT$ silence period by modeling transitions from S_4, S_5, S_6 to the first timeout state S_1 through an empty buffer state b_0; the transition to state b_0 takes one epoch (RTT), and the transition to S_1 takes another epoch.

3.3.2 Modeling repetitive timeouts

When the flow is in small window states of 2 and 3, the timeout value may contain memory of the previous timeouts, which is harder to model due to the Markov chain's memoryless property. As follows, we model these timeouts using an aggregate state to capture the expected wait time before a packet retransmission on a repetitive timeout.

RFC2988 [28] recommends that the maximum timeout period be larger than 60 seconds, and the Linux OS uses a 120 second period. Since these constants are large as compared to typical roundtrip times, we assume infinite timeout states for simplicity of modeling[6]. Let $S_{1/2}$ represent the state if the sender enters the timeout period with the base timer value of $2 \times RTT$, $S_{1/4}$ if the timer has backed-off to $4 \times RTT$, $S_{1/8}$ if the timer has backed-off to $8 \times RTT$, and so on. These states will get aggregated, and are thus not shown in Figure 7.

We model $S_{1/2}$, $S_{1/4}$, $S_{1/8}$, and other other infinite timeout states as follows. If a sender enters at $S_{1/4}$, it must wait for 3 idle periods before retransmitting, and if a sender enters $S_{1/8}$, it must wait for 7 idle periods before retransmitting. On a successful retransmission, which happens with probability $(1-p)$, the sender leaves the timeout state and enters S_2, since the new cumulative acknowledgment received in response to the retransmission increases the sender's congestion window to 2. On an unsuccessful retransmission, which happens with a probability p, the sender doubles its timer period and enters the next longer timeout state.

To be able to incorporate these infinite timeout states into our Markov chain, we aggregate them into two states: a buffer state (b^*) where the sender waits for an amount of time, and the retransmit state introduced earlier (S_1), where the timer goes off and the sender retransmits. With these aggregated states, the following transitions are possible:

- From a small congestion window (S_2, S_3) entering a timeout period, $S_n \rightarrow b^*$;
- staying idle, $b^* \rightarrow b^*$; and
- transition from idle period into retransmission state, $b^* \rightarrow S_1$.

The expected wait time at the aggregated buffer state, b^*, may be calculated as follows. A TCP flow waits for 1 epoch in $S_{1/2}$ before entering the retransmit state, 3 epochs in $S_{1/4}$, 7 epochs in $S_{1/8}$, and so on. Thus, once in a timeout period, the expected amount of time that a TCP flow must wait in the timeout period before retransmitting is computed as:

$$\begin{aligned} Expected\ idle\ time = & P(S_{1/2} \mid RTO) + 3P(S_{1/4} \mid RTO) + \\ & 7P(S_{1/8} \mid RTO) + ... \end{aligned} \quad (4)$$

To resolve equation (4), we note that:

$$P(S_{1/2n} \mid RTO) \div P(S_{1/n} \mid RTO) = p \quad (5)$$

[6]This assumption leads to our model being limited to $p < 0.5$, as discussed in Section 6

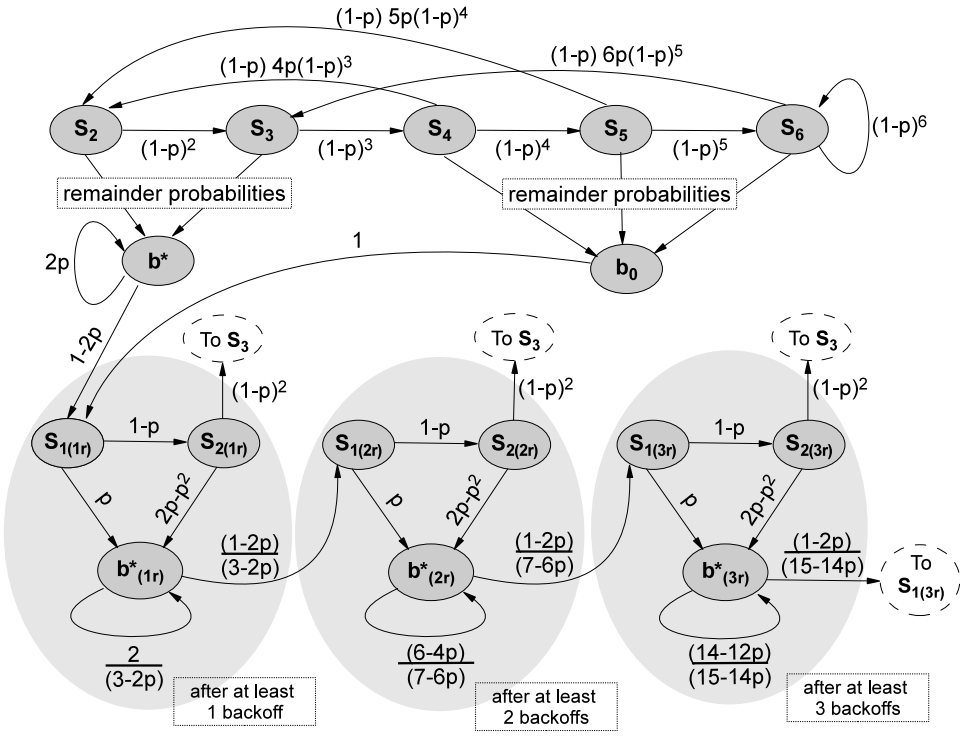

Figure 8: The Full Model. This model is limited to a max cwnd of 6, and can be easily extended to larger cwnds.

We also know that

$$P(S_{1/2} \mid RTO)+P(S_{1/4} \mid RTO)+ P(S_{1/8} \mid RTO) + ... = 1 \quad (6)$$

Combining equations (5) and (6), we get

$$P(S_{1/2} \mid RTO)(1 + p + p^2 + p^3 + ...) = 1$$
$$P(S_{1/2} \mid RTO) = (1 - p) \quad (7)$$

Equation (4) may now be resolved using equations (5) and (6) to give the expected idle time in the timeout state as follows:

$$Expected\,idle\,time = 1(1-p) + 3p(1-p) + 7p^2(1-p)+ 15p^3(1-p) + ...$$
$$= 1/(1-2p) \quad (8)$$

Thus,

$$P(b^* \rightarrow S_1) = 1/(Expected\,idle\,time\,in\,b^*)$$
$$= 1 - 2p \quad (9)$$

Consequently, the probability of staying idle,

$$P(b^* \rightarrow b^*) = 2p \quad (10)$$

Finally, on a successful retransmission in S_1, the sender enters S_2 with a probability of $(1-p)$, while an unsuccessful retransmission leads the flow back into the timeout state, b^*, with a probability of p.

3.3.3 Capturing timer backoffs accurately

To model repetitive timeouts more precisely, we need to break up the timeout retransmit state S_1 further as illustrated in the bottom part of the expanded full model in Figure 8. So far we have assumed that after one successful timeout retransmission, the flow re-enters the state S_2. What this transition misses is that while the sender's cwnd grows to 2 on successfully retransmitting after a timeout, the sender's backoff value has not yet collapsed, since TCP needs an ack for a *new* data transmission (not a retransmission) to be received to collapse the backoff value.

The approximation used so far of going back to b^* from S_1 on a timeout while simple, loses information. We know that the sender has timed-out at least once when the flow is in state S_1, so we may put a larger lower-bound on the amount of time that the sender will be in an idle state on a subsequent loss.

After one timeout, we arrive at S_1 (either with memory or without) and the minimum timeout value is $2 \times RTT$, resulting in an expected wait time of $1/(1-2p)$. However, if we experience a loss at S_1, the minimum timeout value is $4 \times RTT$ with a minimum wait time of $3 \times RTT$. This may be viewed as the same infinite state machine described earlier in equation 4 to represent infinite timeout states, but starting here with 3 wait states at $S_{1/4}$. The aggregate representation of this new infinite state may be modeled using a single state with a corresponding expected wait time calculated as:

$$Expected\,idle\,time = 3p(1-p) + 7p^2(1-p)+ 15p^3(1-p) + ...$$
$$= (3-2p)/(1-2p) \quad (11)$$

This may be represented by breaking up the S_1 and S_2 states in Figure 7 using an aggregated three state transition diagram of $S_{1(1r)}, S_{2(1r)}$ and $b^*_{(1r)}$, as shown in the bottom left part of Figure 8 (grayed oval). $S_{1(1r)}$ represents the retransmit state after the first timeout, $S_{2(1r)}$ represents the state where a new transmission is attempted after a successful retransmission in $S_{1(1r)}$, and $b^*_{(1r)}$

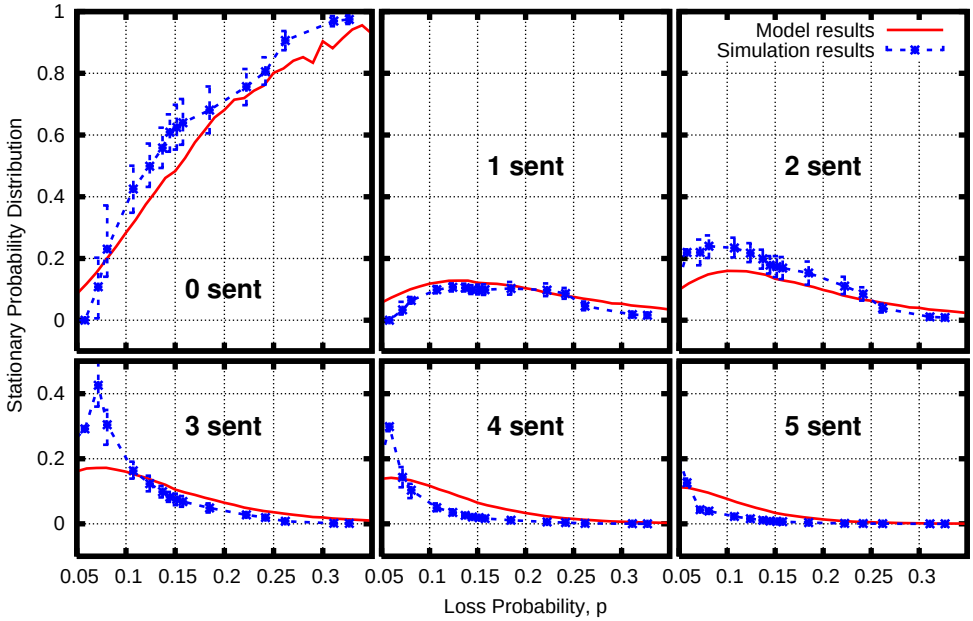

Figure 9: Stationary probabilities from the model and from simulations for 6 of the sending states that a TCP connection can be in. Error bars show 10th and 90th percentile flow values.

represents the aggregate buffer state for idle time on the first repetitive timeout (or, a timeout after at least one backoff), which occurs if the retransmission in $S_{1(1r)}$ is unsuccessful. Note that $S_{2(1r)}$ now becomes a distinct state, separate from S_2. On successful transmissions of the entire window in $S_{2(1r)}$, which happens with probability $(1-p)^2$, the flow loses backoff memory and enters state S_3, since we expect *new* transmissions to be sent in $S_{2(1r)}$.

To model just one repetitive timeout, we could put a loop from $b^*_{(1r)}$ to $S_{1(1r)}$. However, to model further repetitive timeouts for better accuracy, we replicate the $3-state$ diagram to $S_{1(2r)}, S_{2(2r)}$ and $b^*_{(2r)}$—states representing a second repetitive timeout (or, a timeout after at least two backoffs)—where the minimum backoff is $8\times RTT$. The expected wait time is calculated as $(7-6p)/(1-2p)$. To model a third repetitive timeout, we add three more states with a minimum timeout of $16 \times RTT$. This is modeled in Figure 8 with the states: $S_{1(3r)}, S_{2(3r)}$ and $b^*_{(3r)}$.

While more repetitive timeout states may be similarly modeled, we end our model in Figure 8 at three repetitive timeouts, with a loop back from $b^*_{(3r)}$ to $S_{1(3r)}$.

4. VALIDATING THE MODEL

We validate the model using ns2 simulations of TCP flows operating in sub-packet regimes. We use a simple network topology with a single bottleneck link. TCP-SACK is used at the endpoints and the simulations are run for 5000 seconds each. We measure loss rate as the fraction of packets that are dropped at the bottleneck queue and consider epochs of average-RTT size to estimate the senders' probabilities of being in one of the model's states. We get the different loss rates shown in the figure by modifying the number of competing flows at the bottleneck link.

Figure 9 shows the model's predicted probability distribution for varying loss rate p, overlaid with results from simulations where we measure and plot probability against observed loss rate. For this simulation set, the flows all have a propagation RTT of 200ms, the bottleneck capacity is 1Mbps, and the bottleneck link is equipped with an RTT's worth of buffer (50 packets, at 500 bytes per packet).

Note that "0 sent" is the sum of probabilities for all the b* states in the model, and similarly "1 sent" and "2 sent" represent the sums of the S_1 states and S_2 states, respectively. The other graphs are self-explanatory[7]. Simulation results agree well with our model at loss rates greater than or equal to $p = 0.1$. We note that the simulations slightly differ from our model for $p < 0.1$ for the following reason: under lower loss rates flows grow to window-sizes larger than 6, whereas state S_6 in our model (Figure 8), representing a flow window-size of 6, is a "limiting" state. Flow window-sizes that are larger than 6 are considered anomalies which the model does not capture. There will thus be some probability that flows back from S_6 return to the lower states in our model that will not occur in reality when loss rates are low.

Figure 10 shows results for simulations under a variety of link bandwidths (up to 1Mbps), The buffer size for each simulation was set to an RTT's worth. The simulations all agree equally well with the model. Figure 11 shows results from a simulation set with mixed-RTT flows. We simply divide the flows evenly into eight groups, and each group is assigned one of eight RTTs chosen randomly between 200ms and 400ms. While measuring the average probabilites, note that since the flows have different RTTs, epoch-sizes used for the different flows are different, and depend on the flow's RTT. We find that simulation results agree well with our model.

We also ran simulations under a variety of propagation RTTs, and under RED and SFQ AQM schemes, and obtained similar agreement with the model. Note that RED and SFQ do not significantly influence the behavior of TCP aggregates in the sub-packet regime, and thus do not impact the agreement of simulation results with the model.

[7]We do not show S_6 in this graph, but the agreement is similar.

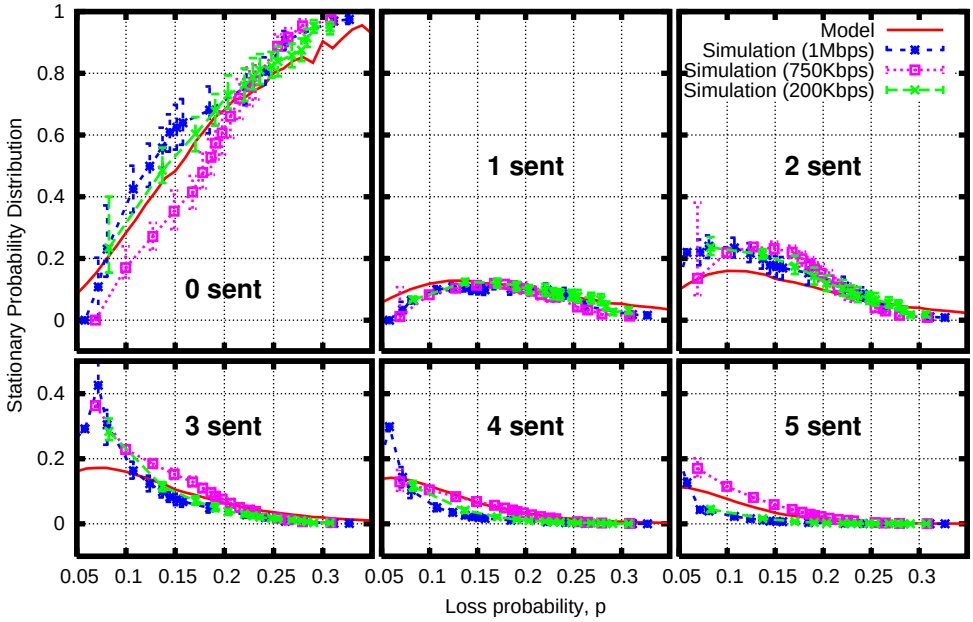

Figure 10: Validating the model with different bottleneck bandwidths

5. MODEL STRENGTHS AND LIMITATIONS

The most important strength of the model is that it provides detailed information about the states of a TCP flow in the sub-packet regime. This information is more fine-grained than the expected throughput (as predicted by the Padhye model [27]) and can be used to control flow dynamics in the network. Our model predicts the stationary probability distribution of a flow across its different states, which is more detailed information and therefore harder to estimate than an expected value of a single flow parameter such as throughput. The stationary distribution of a flow across states is representative of the long-term behavior of a flow; hence, a single trial run of a flow may not exactly follow the exact distribution unless the flow is run over a very long time period. This explains the variance in the actual observed values in the validation experiments.

Our model is specifically designed to accurately capture the details of the TCP's timeout behavior including repetitive timeouts, idle time, and backoff behavior. While the example version of the model presented in this Section is for $P_{max} = 6$, we can certainly extend the model by adding states higher than S_6, to more accurately capture TCP behavior with larger windows. However, our model is not designed for analyzing TCP behavior under low-loss conditions ($p < 0.05$) where timeouts are a relatively rare occurrence. The model can also be extended to handle other variants and extensions of TCP such as Early Retransmit [2] (that modify the loss detection and recovery behavior of TCP) by simply adjusting the probabilities of the fast-retransmit transitions in the model.

Our model trades-off broader applicability for simplicity. Many of the states in our model, such as the wait states, are aggregate states which model the expected behavior across sets of infinite states. This aggregation achieves two important simplicity goals: it avoids state space explosion, and it allows for a compact representation of the full state diagram. The model still achieves a good approximation of the stationary probability distribution of a flow at a given loss rate albeit only within the sub-packet regime. Our model is also simple in that it depends on only parameter p, the loss rate. A TCP flow that sends several packets in a short time period may experience bursty loss patterns; however, a TCP flow in a sub-packet regime transmits very few packets every RTT and packet losses for such flows are more spread out. Hence, using a single loss parameter p is a reasonable approximation for flows within this regime. The state transitions in the model are at the granularity of RTT epochs, which was done to make the model independent of flow RTT. As shown in the validation, the model is fairly accurate even when flows have different RTTs and when the bottleneck capacity varies.

Our model is not usable for $p > 0.5$ due to the self-loop at state $b*$ in Figure 8. While this may appear restrictive, it actually identifies a potential instability in TCP. Our model uses the assumption of infinite backoffs with TCP's retransmission timer; without a limit on the backoffs, TCP becomes unstable for $p > 0.5$, with expected timeout idle periods of $\inf$. It is possible to replace the infinite timeout assumption with a finite timeout to extend the model for $p \geq 0.5$. While this may be of theoretical interest, experience indicates that a loss rate of 50% is not actually sustainable even in the context of pathological-sharing.

6. DISCUSSION: APPLICATIONS OF THE MODEL

Our model can be applied in a variety of ways at a middlebox to both predict the status and behavior of a flow as well as to potentially design non-intrusive middle-box solutions to enhance performance in sub-packet regimes. We outline some potential applications of the model in this section.

6.1 Predicting network and flow state

The most straightforward application of the model is to predict the state of the flows across a bottleneck link using a middlebox. Given the aggregate loss rate at the bottleneck, the model currently gives us the probability of finding a flow in one of several states. Similarly, the distribution can also be used to estimate the fraction of flows that are currently in timeout states on a pathologically-

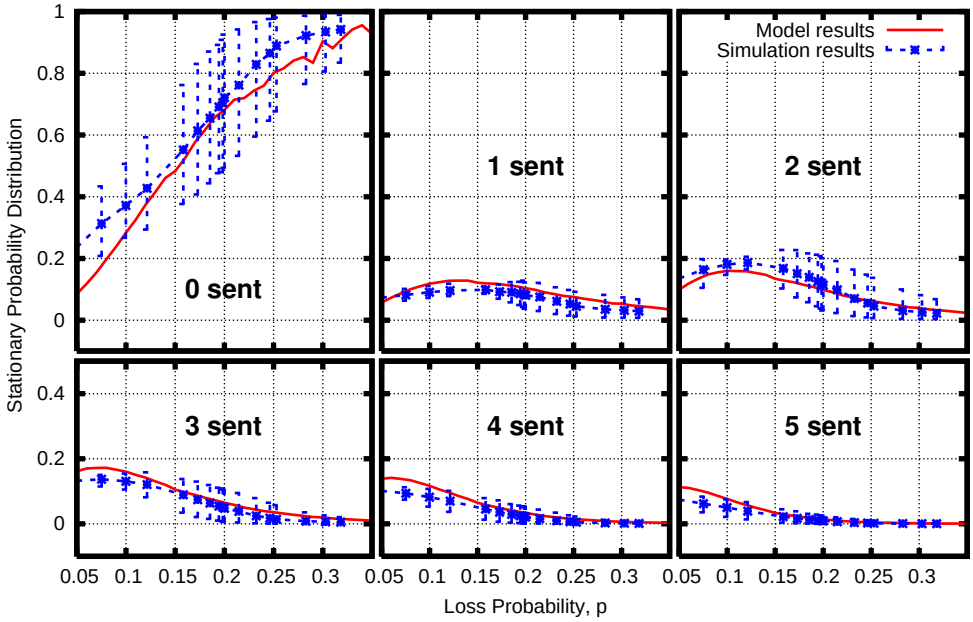

Figure 11: Validating the model with mixed-RTT flows

shared link. While these aggregate measures are useful for monitoring purposes, a key value of the model is in its ability to predict the possibility of a timeout or a repeated timeout. For a single flow, the model can be used as a mechanism for estimating the probability of hitting a timeout state. Specifically, given (i) the aggregate loss rate at the bottleneck, (ii) the current state of a flow, and (iii) the size of an epoch (flow RTT), the middlebox can track the number of packets within each epoch and predict the probability that a flow could potentially hit a timeout state in the following epoch. All of this information is available to a middlebox at the pathologically-shared bottleneck link. Passive RTT estimation techniques can be used to estimate epoch size for a flow, and aggregate loss probability can be observed. For a middlebox managing the pathologically-shared link, this information is particularly useful in advance to implement a range of different flow-state aware queue management policies. We outline some of the possible queue management policies next.

6.2 Middlebox Queue Management

A middlebox could leverage the model to control the behavior of a flow by altering its packet drop policies on a per-flow basis. For example, a middlebox can use the probability of a packet drop triggering a flow timeout to dictate the priority of a packet drop (or transmission) in several ways; we discuss two ways below.

A simple scheduling policy at a middlebox can use different service classes for flows based on the probability of a flow hitting a timeout or a repetitive timeout. More vulnerable flows are placed in higher service classes and less vulnerable flows are placed in lower service classes; the scheduling queue management policy at the middlebox can be careful to drop packets from lower service class flows before going up to higher service classes.

Our model uses number of losses in an epoch for a flow to predict its next state; a middlebox could similarly control the state changes of a flow by controlling losses. In this case, the middlebox performs per-flow state monitoring and keeps track of how many packet drops a flow can sustain without a timeout, and uses this information to make a decision about an impending packet drop at the bottleneck queue.

While we can thus achieve finer control over packet and drop priorities using our model at a middlebox, we now show, to a first degree of approximation, how such control can be effective by outlining a simple optimization that prioritizes retransmissions.

Repetitive timeouts are a more serious problem in sub-packet regimes and they are triggered due to the loss of retransmitted packets. While packet losses trigger a reduction in window sizes, loss of retransmitted packets incur the high cost of increasing timeout periods, resulting in some flows appearing to be "shut-off" for a while, until the retransmission(s) followed by at least one new transmission all get through. It is difficult for an endpoint in the flow to control whether a retransmission gets dropped in the network, but it is possible for an intelligent router/middlebox at the congested link to distinguish and prioritize scheduling of retransmissions. In sub-packet regimes, the ISP providing a pathologically-shared access link could implement and deploy such a router/middlebox.

Prioritizing retransmissions can help in enhancing fairness across flows. We illustrate this using a simple example simulation scenario where several long term flows compete over the 1 Mbps bottleneck link with a buffer of 25 packets and an RTT of 200ms. We consider two well known queuing methodologies for this test: (a) Drop-tail; (b) Stochastic Fair Queuing (SFQ). We compare these queuing mechanisms with and without retransmission prioritization. Figure 12 shows the results for the fairness across flows for various configurations over 20 second timeslices. In all configurations the link is 1Mbps, 200ms RTT, with a total queue size of 25 packets (bandwidth * delay * 1000byte packets). The SFQ implementation is composed of five separate buffers of each of length 5 packets. The gateway + SFQ consists of five separate buffers each of length 4 packets along with another buffer of size 5 packets for our retransmitted packet queue. The gateway + droptail is identical to gateway + SFQ except the 20 packets of buffer space are allocated to a single droptail queue.

The main result in this experiment is that our retransmission prioritization (RP) improves upon both SFQ and droptail. A middlebox that uses our model can thus further improve fairness in the system through similar packet and drop prioritization.

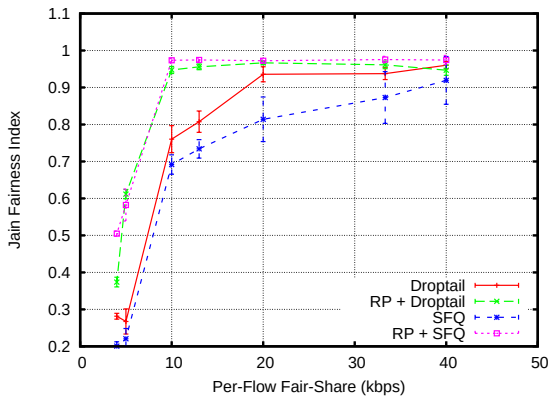

Figure 12: Fairness across individual flows over a 1 Mbps link for various configurations

7. RELATED WORK

The past three decades have seen a tremendous amount of work on analysis of TCP congestion control. We outline only work closely related to ours.

The earliest work on pathological-sharing among TCP flows by Morris appears about a decade ago [25]. Morris recognizes the limitations of TCP operating in regimes where the fair share is under a single packet per roundtrip time and provides some insight into observed flow behavior and aggregate behavior at the bottleneck. Our paper is an extension of this previous line of analysis. As discussed in Section 2.4 we find Morris' solution to be impractical; our solution is more practical and deployable.

TCP's loss of fairness under pathological sharing is not a surprise, and has been noted in [16, 20, 29]. We build on these previous observations, and contribute a more systematic understanding of the dynamics that dominate in these regimes. Even in simply observing fairness results from simulations, we find that the emergent admission control behavior in the short- to medium- term, and long-term fairness are novel observations that inform our understanding of the problem.

Prior work in stochastic models for TCP, such as [1, 4, 7, 18, 19, 22, 24, 27, 33], derive analytical expressions for the steady-state throughput of a TCP flow based on its roundtrip time and loss probability. Of these models, ours is closer to [7, 27], which consider a discrete-time model and a discrete evolution of the window size. The Padhye model is a much better fit when the packet loss rates, p, are relatively small; at high values of p, however, we observe extended and repetitive timeouts, the dynamics of which are not captured in detail in the Padhye model. While the Padhye model provides the expected average throughput, our stationary distribution is a more complete characterization of the state of a TCP connection. Finally, one subtle difference is that, the value of p in Padhye model represents the probability of loss indication (or loss episodes) while we explicitly model the (in)ability of TCP to recover in the face of a bursty loss of several packets.

Our TCP model supplements Markov models for TCP that have been proposed in [10, 11]. These models focus on TCP behavior when the packet loss rates are relatively small. Specifically, Fortin-Parisi-Sericola (F-PS) [10] build an extensive model that is built to yield expected goodput. Our model yields a detailed characterization of the states of a TCP connection, which is fundamentally harder than finding the expected goodput. Yet, and inspite of the complexity due to modeling repetitive timeouts, our model is simpler and more intuitive than the F-PS model because we assume the sub-packet regime, high loss-rates, and small windows.

Work in low bandwidth access links typically assumes a low degree of sharing at the link. For instance, Spring *et al.* [34] and Andrews *et al.* [5] both propose solutions for improving performance at low bandwidth access links, but both operate under low degrees of sharing and assume per-flow fair share of at least 1 packet per RTT.

8. CONCLUSION

The growth in the popularity of the Internet has contributed to growth in demand of Internet users especially in the developing world. However, growth in network connectivity has not kept up with growth in network demand, resulting in pathologically-shared connectivity becoming common in many parts of the developing world. The problem of pathological sharing has been further exacerbated by an increase in the complexity of websites, and a corresponding increase in the number of TCP connections created by web browsers. All these factors are contributing to the increased prevalence and increased importance of sub-packet regimes.

TCP breaks down in the sub-packet regime. As TCP probes for bandwidth, it cannot send fewer than 1 packet per RTT in a controlled manner. Consequently, in the sub-packet regime, TCP uses the only coarse-grained mechanism it has available to send at a lower rate than 1 packet per RTT—timeouts—yielding the individual and aggregate effects that we studied in this paper.

To the best of our knowledge, our work is the first to formally characterize the sub-packet regime and highlight its importance. We characterize TCP behavior in this regime and show that TCP and its many variants under different queuing conditions result in severe unfairness, high packet loss rates, and flow silences due to repetitive timeouts. We have proposed a detailed model to characterize the TCP dynamics that dominate in this regime. We briefly discussed how this model could be used in practice.

Much work to be done in examining the solution spaces for addressing transport performance in sub-packet regimes. While we briefly discuss a potential part of a solution with middleboxes, purely end-to-end mechanisms, including graceful TCP degradation into the sub-packet regime, and other full systems solutions, all form a problem-rich and immediately-relevant area that is yet to be explored.

9. REFERENCES

[1] A. A. Abouzeid, S. Roy, and M. Azizoglu. Stochastic Modeling of TCP over Lossy Links. In *INFOCOM 2000*, pages 1724–1733, 2000.

[2] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, and P. Hurtig. Early Retransmit for TCP and SCTP, Jan. 2010. draft-ietf-tcpm-early-rexmt-04 (IETF work in progress).

[3] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control, Sept. 2009. RFC 5681.

[4] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of tcp/ip with stationary random losses. *IEEE/ACM Trans. Netw.*, 13(2):356–369, 2005.

[5] L. L. H. Andrew, S. V. Hanly, and R. G. Mukhtar. Clamp: Active queue management at wireless access points. In *European Wireless*, 2005.

[6] L. Brakmo and L. Peterson. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, Oct. 1995.

[7] N. Cardwell, S. Savage, and T. Anderson. Modeling tcp latency. In *IEEE INFOCOM*, Mar. 2000.

[8] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm, Apr. 1999. RFC 2582.

[9] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Transactions on Networking*, 1(4):1063–6692, Aug. 1993.

[10] S. Fortin-Parisi and B. Sericola. A Markov model of TCP throughput, goodput and slow start. *Performance Evaluation*, 58(2-3):89 – 108, 2004. Distributed Systems Performance.

[11] J. Gil. Modelling TCP with a Discrete Time Markov Chain. In *HET-NETs: Performance Modelling and Evaluation of Heterogenous Networks*, July 2005.

[12] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[13] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification, Jan. 2003. RFC 3448.

[14] E. Huerta and R. Sandoval-Almazán. Digital literacy: Problems faced by telecenter users in Mexico. *Information Technology for Development*, 13(3):217–232, 2007.

[15] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, Sept. 1984. DEC Research Report TR-301.

[16] C. P. Juan, J. C. S. Agrelo, and M. Gerla. Using back-pressure to improve tcp performance with many flows. In *IEEE INFOCOM*, 1999.

[17] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM*, Aug. 2002.

[18] A. Kumar. Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link. *IEEE/ACM Transactions on Networking*, 6:485–498, 1998.

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

[20] L. Massoulié and J. Roberts. Arguments in favour of admission control for tcp flows. In *ITC 16: 16th International Teletraffic Congress*, pages 33–44. Elsevier, 1999.

[21] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options, Oct. 1996. RFC 2018.

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3), 1997.

[23] P. E. McKenney. Stochastic fairness queueing. In *INFOCOM*, 1990.

[24] V. Misra, W.-B. Gong, and D. Towsley. Stochastic differential equation modeling and analysis of tcp-windowsize behavior. In *Performance '99*, 1999.

[25] R. Morris. Tcp behavior with many flows. In *ICNP '97: Proceedings of the 1997 International Conference on Network Protocols (ICNP '97)*, page 205, Washington, DC, USA, 1997. IEEE Computer Society.

[26] B. Oyelaran-Oyeyinka and C. N. Adeya. Internet access in africa: empirical evidence from kenya and nigeria. *Telemat. Inf.*, 21(1):67–81, 2004.

[27] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: a simple model and its empirical validation. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, 1998.

[28] V. Paxson and M. Allman. Computing TCP's Retransmission Timer, Nov. 2000. RFC 2988.

[29] L. Qiu, Y. Zhang, and S. Keshav. Understanding the performance of many TCP flows* 1. *Computer Networks*, 37(3-4):277–306, 2001.

[30] T.-L. Sheu and L.-W. Wu. An analytical model of fast retransmission and recovery in tcp-reno. In *IEEE International Conference on Networks (ICON)*, Nov. 2004.

[31] T.-L. Sheu and L.-W. Wu. An analytical model of fast retransmission and recovery in tcp-sack. *Perform. Eval.*, 64(6):524–546, 2007.

[32] T.-L. Sheu and L.-W. Wu. Modeling of multiple tcp segment losses in slow start and congestion avoidance. *Journal of Information Science and Engineering*, 2007.

[33] B. Sikdar, S. Kalyanaraman, and K. S. Vastola. Analytic models for the latency and steady-state throughput of tcp tahoe, reno, and sack. *IEEE/ACM Trans. Netw.*, 11(6):959–971, 2003.

[34] N. T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad. Receiver based management of low bandwidth access links. In *IEEE INFOCOM*, Mar. 2000.

[35] K. Tan, J. Song, Q. Zhang, and M. Sridharan. Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. In *IEEE INFOCOM*, Apr. 2006.