TAQ: Enhancing Fairness and Performance Predictability in Small Packet Regimes

Jay Chen
New York University
jchen@cs.nyu.edu

Lakshmi Subramanian
New York University
lakshmi@cs.nyu.edu

Janardhan Iyengar
Franklin and Marshall College
jiyengar@fandm.edu

Bryan Ford
Yale University
bryan.ford@yale.edu

Abstract

TCP congestion control algorithms implicitly assume that the per-flow throughput is at least a few packets per round trip time. Environments where this assumption does not hold, which we refer to as *small packet regimes*, are common in the contexts of wired and cellular networks in developing regions. In this paper we show that in small packet regimes TCP flows experience severe unfairness, high packet loss rates, and flow silences due to repetitive timeouts. We propose an approximate Markov model to describe TCP behavior in small packet regimes to characterize the TCP breakdown region that leads to repetitive timeout behavior. To enhance TCP performance in such regimes, we propose *Timeout Aware Queuing (TAQ)*, a readily deployable in-network middlebox approach that uses a multi-level adaptive priority queuing algorithm to reduce the probability of timeouts, improve fairness and performance predictability. We demonstrate the effectiveness of TAQ across a spectrum of small packet regime network conditions using simulations, a prototype implementation, and testbed experiments.

1. Introduction

Existing congestion control schemes such as TCP-NewReno [9], TFRC [14], Cubic and many others assume the fair-share bandwidth of a flow is at least 1 packet per round trip 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 *small packet regime*, to represent environments where a TCP's per flow share to be less than k packets per RTT (for a small constant k, typically $3-4$) and we define *sub-packet regime* as a specific sub-case of small packet regimes where the TCP's fair share is less than 1 packet per RTT.

Small packet regimes are common and increasingly important in developing regions where a low-bandwidth network is often shared by many users [15, 26]. One of the primary causes for the occurrence of small packet regimes is the *content-connectivity gap*. Web pages have rapidly grown in size over the years and have out-paced the growth of connectivity in many parts of the developing world. In addition, an average web page has also increased considerably in complexity with a large number of objects assembled from different domains. In the past decade, we have observed a 30 fold increase in these metrics, thereby triggering a large number of competing TCP flows per single web session [36]. In contrast, connectivity has not grown at the same pace in developing regions with relatively non-uniform growth across regions. Global broadband penetration in developing regions still remains much lower than in North America and Europe and is primarily restricted to urban environments [2].

In this paper, we first analyze per-flow and aggregate behavior of TCP and other variants in small packet regimes and 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 large fraction of flows to observe long silence periods with no packet transmissions; (b) extreme unfairness over short time scales; and (c) unpredictable flow completion times. In addition, we explain why none of the standard TCP variants or known queu-

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

EuroSys '14, April 14–17, 2014, Amsterdam, Netherlands.

ACM 978-1-4503-2704-6/14/04. . . $15.00.
http://dx.doi.org/10.1145/2592798.2592819

ing mechanisms offer substantial performance gains in the sub-packet regime. Some of these problems have been observed in prior work by Morris [25], Qiu *et al.* [29], and Juan *et al.* [17] where they have analyzed TCP behavior in small packet regimes and in the context of many competing flows. However, in the broader context, we believe that small packet regimes have not been a traditionally important region of operation for network flows, and as a result this space has remained relatively unexplored.

This paper primarily focuses on how we can address many of the fundamental shortcomings experienced by TCP flows in small packet regimes including enhancing performance, fairness and predictability while reducing the occurrence of recursive timeouts. To better characterize the equilibrium behavior of TCP in small packet regimes, we introduce an idealized Markov model (variant of traditional TCP Markov model [11]) that can capture the repetitive timeout behavior of TCP flows. We note that since Markov models are memoryless, modeling repetitive timeouts is not straightforward since one needs memory of the prior states. Using this model, we characterize the breaking point in small packet regimes beyond which TCP flows timeout repeatedly thereby triggering the fairness problems for a majority of flows over short time scales.

Leveraging our model, we describe the design of *Timeout Aware Queuing (TAQ)*, a non-intrusive in-network middlebox solution that approximates our idealized model for improving TCP performance, fairness and predictability in the small packet regime. The design of TAQ is based on three key ideas. First, we use an approximation of the model to track the short-term behavior of every competing flow traversing a middlebox to estimate the current state of every flow. Second, we use an adaptive multi-level priority queuing algorithm that leverages the current state of every flow to perform fine-grained prioritization of individual packets. To reduce the probability of recursive timeouts, we specifically prioritize state transitions that could trigger timeouts for individual flows. Under extreme conditions beyond the TCP breaking point for a large fraction of flows, we propose an explicit admission control protocol that allows admitted flows to make progress. Our solution relies on in-network middleboxes to enable simple and transparent deployment by network operators, but preserves TCP's end-to-end semantics and requires no modifications to end hosts. We evaluate TAQ through simulation and a real-world implementation across a variety of small packet regimes and demonstrate that our system enhances fairness and performance predictability across a range of network conditions and real-world scenarios.

2. Small Packet Regimes

In this section, we first define small packet regimes 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 small packet regime to explain what makes TCP break down.

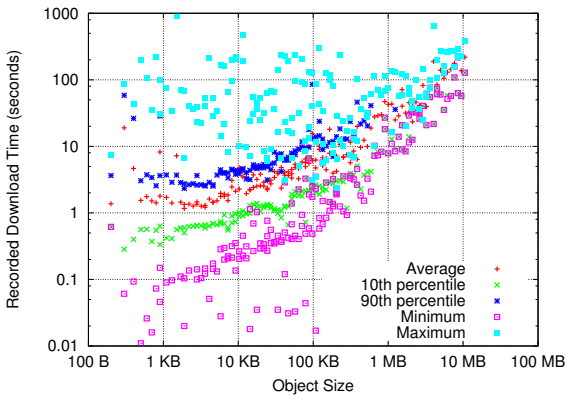


Figure 1: Scatter-plot of download times for different object sizes, taken from a 2-hour observation period at the University's Squid proxy. Each raw data point is assigned to a bucket, and the values shown here are the 10th percentile, 90th percentile, min, max, and average values per bucket.

2.1 Defining the Small Packet Regime

We define a *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). More generally, we define a *small packet regime* of size k, which we refer to as $SPK(k)$, to denote the case where a per-flow fair share is less than k packets per RTT where k is a small constant. Given that most TCP flows use TCP CUBIC and begin with a congestion window of 10, we are typically interested in small packet regimes where k is less than 10 and typically in the range $3-4$. A TCP flow with segment size S and round-trip time of RTT, is in the small packet regime of size k 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 $k \times S/RTT$.

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. Small packet regimes are primarily triggered in environments where the bottleneck bandwidth C is limited and one observes a large number of flows (large N) creating a pathological network sharing environment. Small packet regimes occur when C/N is roughly comparable to kS/RTT for some small value of k; for values of k less than the initial TCP congestion window of 10, the congestion effect of the small packet regime is typically observed at flow initiation time due to packet losses.

2.2 Pathological Sharing: An End-User View

Next, we use a real world example to describe the pathological effects of the small packet regime triggered by high lev-

els of network sharing in a relatively well connected university campus in Kerala, India. We examine the web browsing experience of end-users in the university as observed from the perspective of a web proxy. 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.

The Y-axis spread in the graphs is striking: download times for objects vary by over two orders of magnitude! 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. We make two observations. First, a large 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 explain TCP's behavior in small packet regimes.

2.3 TCP in Small Packet Regimes

The behavior of TCP in the presence of many flows has been studied in prior work [17, 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 that result in long silence periods during which flows do not transmit a single packet. Second, flows experience 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 TCP and TFRC or existing variants of queuing mechanisms (RED, SFQ) address these problems in the small 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.

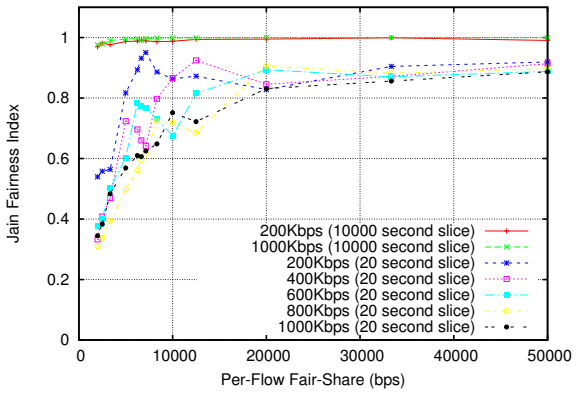


Figure 2: Long- and short-term Jain-fairness as a function of ideal per-flow fair-share, for different capacity bottleneck links (droptail queue with one RTT worth of delay).

Fairness in Small Packet Regimes: To illustrate the flow-level fairness problems observed in small packet regimes, we consider a simple simulation environment using a dumbbell topology with a single bottleneck link using a simple tail-drop queue. 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) [16]. 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). 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. Upon closer examination in the pcap traces for these simulations, we find that over 20-second time slices roughly 30% of the flows are completely shut down and roughly 40% of the flows consume more than 80% of the link bandwidth. These observations lead us to the following 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.

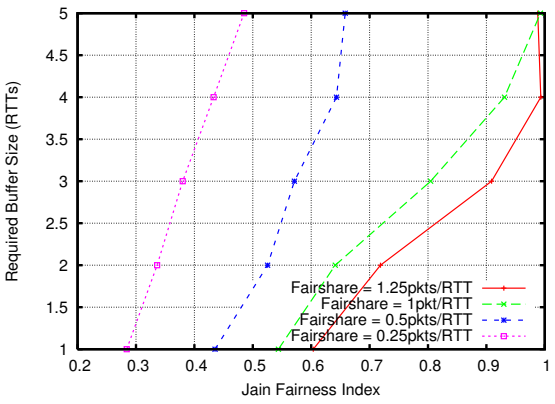


Figure 3: Droptail buffer sizes required for restoring fairness, considering (20sec time slices)

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 scenario like the one described in Section 2.2. In this simulation, we create users that spawn multiple TCP connections each (a web session pool) who all share a bottleneck link with 1Mbps capacity. The propagation RTT is 200ms, and the bottleneck link has 50 packets worth of buffer space (one RTT worth of delay). We omit the figure here due to lack of space, but with four flows per user and 200 active users we find that 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 overall congestion, we also find that that fewer connections worsen the user experience by increasing the chance that all of a user's connections stall at once.

2.4 Existing Queuing Mechanisms

Existing active queue management mechanisms offer limited help to prevent fairness or repetitive timeout problems in small packet regimes. Using detailed simulations under different small packet regimes, we observe that queueing schemes such as Random Early Detection (RED) [10] or Stochastic Fair Queueing (SFQ) [23] have similar aggregate behavior characteristics. The fundamental problem is that many active queue management mechanisms like RED and SFQ require much larger buffer sizes for the number of flows being managed to provide marginal gains in fairness without affecting the network utilization.[2] Furthermore, we observed that increasing buffer sizes simply trades in delay and delay variance for fairness. Figure 3 shows the tradeoff required for achieving fairness for a small part of the small packet regime. Increasing buffers is infeasible, particularly when the level of pathological network sharing increases since 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 applications) 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. We simulate this scenario to help illustrate that small packet regimes can occur even with larger bottlenecks.

[2] Without larger buffer sizes, the buffer is always full due to high contention, and slowing flows down (e.g. RED) results in similar behavior as droptail.

3. Timeout Aware Queuing

The fundamental problem that *Timeout Aware Queuing (TAQ)* aims to solve is minimize the probability of *timeouts* of flows, which trigger long silence periods and is the primary cause of extreme unfairness in small packet regimes. The key idea of TAQ is to track the behavior of every flow at a middlebox and perform fine-grained control of packet drops experienced by indivudal flows to minimze timeouts. TAQ is a *non-intrusive* in-network solution that requires no modifications to the end-hosts and the application layer. Characterizing timeout behavior of a flow at a middlebox is not easy due to three factors. First, the middlebox only has a limited view of a flow and is unaware of the state of the flow at the sender and the receiver. Second, there are several TCP state transitions that lead to timeout behavior and repetitive timeouts are even harder to characterize since these states are not memoryless; in essence, the length of a repetitive timeout needs memory of the length of the previous timeout value. Finally, it is very hard for a middlebox to synchronize its observations with the flow state transitions at the end-hosts since a middle-box may not be able to accurately estimate end-to-end RTT and may observe TCP traffic in only one direction.

To address these challenges, we first present an idealized Markov model that captures TCP behavior in small packet regimes. The idealized model assumes perfect knowledge where it models the various state transitions that happen at an end-host including window increments, decrements, losses, timeouts and repetitive timeouts. In this section, we first present the idealized Markov model and describe key takeaways from these models. Through the idealized model, we learn three important properties. First, given the loss-rate in the network, what is the expected stationary distribution of the state across all flows in the network; this stationary distribution also characterizes the probability of timeouts. Second, we infer a tipping point on the network loss rate beyond which, the probability of timeouts dramatically increases. Finally, using the idealized model, we describe a

simplified and approximate version of the idealized model that can be used at a middlebox to classify the state of a flow into several known categories. This information is used by the TAQ queue management layer for packet scheduling.

3.1 An Idealized Model for Small Packet Regimes

We propose an idealized Markov model for analyzing behavior of TCP in small packet regimes especially under high loss rates and relatively small congestion windows. The model aims to capture the *stationary distribution* of a set of TCP flows across the different possible states of a flow, specifically, capturing the stationary probability of a flow across different timeout states. Our model is designed based on a single parameter p, the packet-loss probability at the bottleneck link. Our model is different from and extends previously proposed models of Padhye *et al.* [27], Fortin-Parisi *et al.* [11]. Markov models by definition are memoryless and it is difficult to capture timeout state transitions because they are dependent on the history of past window states of a flow. To address this limitation, we propose an idealized Markov model to characterize the expected behavior of flows.

We make two simplifying assumptions. (1) We assume that the TCP flow is operating in small 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 small packet regimes independent of short flows or long flows. 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.

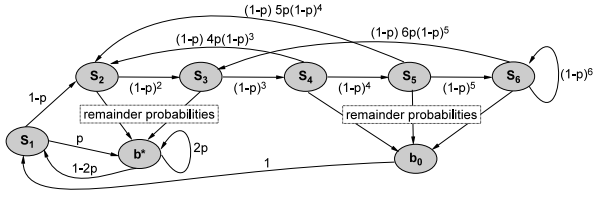


Figure 4: The Partial Model. States S_1 and S_2 get expanded in the Full model.

We begin with a simple congestion window based chain model for TCP as illustrated in the top part of the model in Figure 4 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.

Each packet has an independent loss probability p, so:

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

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 [9], 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 round trips are required to recover fully from all the losses [9]. 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 or more losses in a window, and (ii) when a retransmitted packet is dropped. This probability is simply computed as the residual probability:

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

The upper part of Figure 4 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].

3.1.1 Modeling Timeouts

Modeling timeouts is the most challenging part of our model. 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 round trip time measurement is available, which only happens when cumulative acknowledgements arrive for newly

transmitted (not retransmitted) data [28]. 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.

The challenging part of the model is to capture *repetitive timeouts* which occurs when a flow hits a timeout before memory of previous timeouts is lost. When hitting a repetitive timeout, the sender's retransmission timer has a backoff value which is doubled again by the TCP sender, causing extended silence periods. We model these timeouts using an aggregate state to capture the expected wait time before a packet retransmission on a repetitive timeout. Consider a hypothetical infinite state TCP timeout model where a flow could have infinite timeouts. 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. 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:

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

We also know that

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

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

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

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

$$\begin{aligned} Expected\ idle\ time &= 1(1-p) + 3p(1-p) + 7p^2(1-p) + \\ & 15p^3(1-p) + ... \\ &= 1/(1-2p) \end{aligned} \quad (8)$$

Thus,

$$\begin{aligned} P(b^* \rightarrow S_1) &= 1/(Expected\ idle\ time\ in\ b^*) \\ &= 1 - 2p \end{aligned} \quad (9)$$

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.

To model repetitive timeouts more precisely, we need to break up the timeout retransmit state S_1 further across multiple backoff states. As an example of an expanded model illustrated in in Figure 5, we can derive a much more accurate but complex picture of the timeout states. In this example, the original b^* state gets expanded to capture different stages of timeouts: with at least 1 backoff, at least 2 backoffs and atleast 3 backoffs. Since Markov models are memoryless, we need to different set of states to capture these transitions. The transition probabilities can be derived in a similar manner as before but the expressions become more complex. Due to space constraints we omit the details of this calculation from this paper.

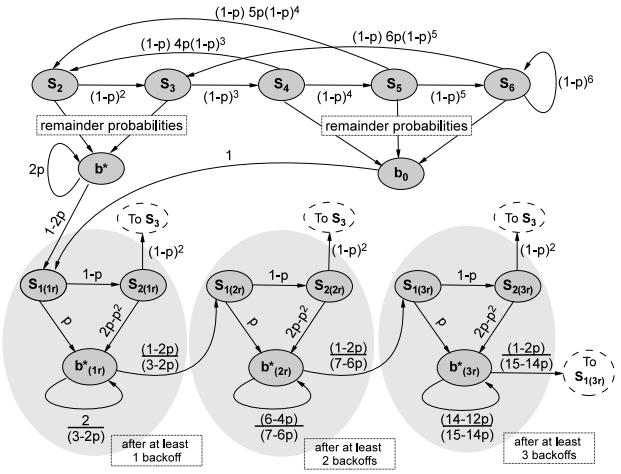


Figure 5: The Full Model. This model is limited to a max cwnd of 6, and can be easily extended to larger cwnds.

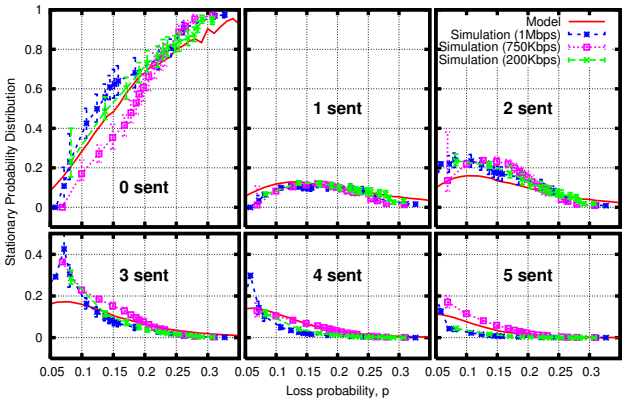


Figure 6: Validating the model with different bottleneck bandwidths

3.1.2 Validating the Model

We have extensively validated our model across a variety of small packet regimes and were able to confirm that the model provides an accurate picture of the stationary probability of flows across different states. We present a brief summary of results in this section based on ns2 simulations over a bottleneck link with flows having variable RTTs and using TCP SACK. Figure 6 shows results of the actual probabilities for simulations under a variety of link bandwidths (up to 1Mbps), The buffer size for each simulation was set to an RTT's worth. Note that "0 sent" is the sum of probabilities for all the b* states in the model where the flows do not transmit any packets, and similarly "1 sent" and "2 sent" represent the sums of the S_1 states and S_2 states, respectively. Overall, simulation results agree well with our model, especially for $p > 0.05$. These graphs were computed for $W_{max} = 6$ and for lower values of p. Many flows have higher window sizes, but for small packet regimes we are only interested in small cwnd. We also ran simulations under RED and SFQ AQM schemes, and obtained similar agreement with the model.

3.2 Applying the Idealized Model

The model leads to several important takeaways that influence the design of TAQ. First, given a loss probability p, the model provides the complete stationary distribution of a flow across all the possible states. This is a potentially powerful summary data that a middlebox can use for fine-grained queue management and decision making. Specifically, given p, the middlebox can estimate the probability of timeouts and repetitive timeouts for any given flow. Given different flows operating in different states, TAQ can use the predicted state of a flow to preferentially drop packets to enhance fairness across flows. In essence, TAQ aims to achive a Fair Queuing-like fairness model by exercising fine-grained control of packet drops across competing TCP flows.

Second, under high levels of contention, when the loss rate jumps beyond 10%, we observe from the model that the probability of timeouts in the stationary distributions rapidly increases. Under such settings, a large fraction of flows will remain in timeout states. Unfortunately, there is inherent unfairness in the flows that remain shut out for short or extended time periods resulting in an arbitrary flow admission control. To ensure fairness under such high contention regimes, TAQ introduces an explicit admission control policy (to partially limit contention over short time scales) and guarantee progress for the admitted flows.

Third, our idealized model, while accurate, is too brittle for practical use. If we were to implement the model in a middlebox, slight differences in TCP versions, TCP implementations (e.g. retransmission timeout values), and inaccuracies in real-world RTT estimation can cause wildly incorrect predictions. Rather than apply the model directly, we abstract our idealized model into a simpler approximation that captures the behaviors generally characteristic of all TCP flavors in small packet regimes. The approximate state model in TAQ is a simplification of the idealized TCP model along two dimensions. First, the approximate model uses middlebox observations of a flow behavior to learn the current state of a flow. Second, the middlebox determines the possible next state of a flow as a potential consequence of an action (successful or unsuccessful packet transmission) at a middlebox. These two inputs are critical for fine-grained packet control at a middlebox.

Finally, we note that from the model that loss of retransmissions or new transmissions immediately following retransmissions are particularly expensive as they lead to backoffs and extended silence periods. This requires TAQ to specifically handle packet retransmissions with higher priority above normal packet retransmissions. However, simple retransmission packet prioritization can be detrimental since such a policy under high flow contention can push a majority of flows into recovery state where they are constantly recovering from packet losses; in simpler terms, all origi-

nal packets get dropped and only retransmitted packets get transmitted. Next, we elaborate on how to construct a useful *approximate* model of TCP states.

3.3 An Approximate Model

A middlebox may not be completely aware of the exact congestion state of a flow as stored at the TCP sender. To apply the idealized model in practice at a middlebox, TAQ monitors the progress of every flow across *epochs* of a flow. An *epoch of a flow* in TAQ is the middlebox perceived RTT of the flow. In the conventional mode of operation, TAQ observes two-way traffic (including ACKs), thereby making it relatively easy to estimate RTT. If TAQ only observes one-way traffic, TAQ uses a simple and approximate epoch estimation approach where the initial epoch estimate is set as the time period between the SYN and the first data packet. Given that TCP flows in their normal states (not timeouts) start with short bursts in the beginning of each epoch, TAQ revises the epoch estimates (using a weighted moving average) by observing short packet bursts at the beginning of every new estimated epoch. Given the epoch size and the aggregate loss rate at the bottleneck, 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. The middlebox tracks the following parameters of a flow in each epoch: (a) number of new packets; (b) highest sequence number; (c) number of retransmitted packets and (d) packet losses in previous epoch. TAQ uses the difference between the highest sequence numbers across epochs as a measure of the progress of a flow. Using these parameters, TAQ determines the approximate state of a flow across one of the following states:

- *Slow start:* This refers to a flow that is exponentially increasing its window and is characterized by a significant growth in the number of new packets across each epoch.
- *Normal state:* This refers to the case where the flow experiences no losses at the TAQ queue and the number of new packets shows a small linear growth (or is roughly similar to the previous epoch).
- *Explicit loss Recovery state:* The middlebox explicitly drops a packet corresponding to a flow and should expect a reduction in the number of packets observed in future epochs. After a packet loss/drop, the middlebox primarily witnesses retransmission packets at the middlebox with very few packets in each epoch until the flow recovers from the loss to the normal state. Typically, even with TCP SACK, in a recovery phase, very few packets are retransmitted per RTT by a TCP flow; the typical number is one.
- *Timeout silence state:* When a flow hits a timeout, the flow hits a silence period with no packets. The typical RTO value is dependent on RTT plus a multiplicative factor of the variance in RTT. A timeout might trigger either a timeout silence state followed by recovery or a direct transition to a recovery state.
- *Timeout Recovery state:* After a timeout, a flow typically retransmits one or more packets. Upon successful retransmission of packets that triggered the timeout, the flow can recover to the slow start phase with a small window.
- *Extended silence state:* The loss of packets during a timeout recovery phase forces a flow to hit a repetitive timeout and the flow hits a silence period that lasts multiple epochs.

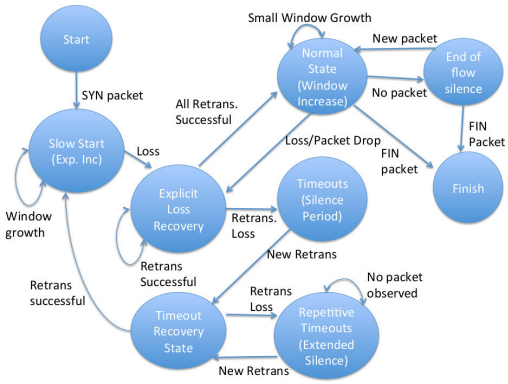


Figure 7: The Approximate Model of a flow at a Middlebox

Figure 7 illustrates the state diagram that our middlebox uses to track the state of a flow across the aforementioned states. For completeness, we introduce one more dummy silence period state to consider the case where flows in normal state do not transmit any packets, since they may have no more packets to send (for example, HTTP pipelining resulting in several objects transmitted on a single flow).

This model represents an approximation of the idealized states of a flow as represented in the idealized model. The different states corresponding to the window sizes are represented by a single normal state and the number of packets observed within an epoch is reflective of the approximate window size of the flow. This information is stored separately and is not associated with the state transition diagram. The three timeout states can be directly mapped to the three timeout states in each stage of a repetitive timeout in the full version of the idealized model. Similarly the explicit loss recovery state can be mapped to the intermediate states before the timeout states. In addition, we do not maintain any probability transitions in this approximate model; instead, in the approximate model state transitions are driven by observation of the four state parameters.

4. TAQ Queue Management

The central design goal of TAQ is to leverage the model in Figure 7 to control the behavior of a flow by altering its packet drop policies on a per-flow basis. The idea in TAQ is to use the number and nature of packet losses at the middlebox queue to predict the next state of a flow and

determine if the middlebox packet drop action could trigger the flow to a timeout or a repetitive timeout.

Each of the states corresponding to a flow are associated with memory for queue management purposes. The slow start and normal states are associated with the number of new packets observed in an epoch as a rough measure of the window size. The explicit loss recovery state and the timeout recovery states maintain information of how many packets were dropped and correspondingly, the number of dropped packets at the TAQ queue that need to be recovered. The silence periods and extended silence periods maintain memory of the previous state of a flow at the last transmission.

4.1 Predicting the Next State

A central goal of TAQ is to predict the effect of a packet loss on the next state of a flow. Here, we assume that TAQ has control over the losses that occur on the bottleneck link but may not have any control that occur on other links. When a flow experiences losses beyond the losses at a TAQ queue, TAQ automatically adjusts the state of the flow in future epochs to adjust for this behavior. We outline the effect of different types of packet drops and no losses on a flow.

No losses: In the event a flow is in a normal or slow start state and does not experience any losses during an epoch, the expectation is for the flow to remain in those states unless there are external packet losses (outside of the bottleneck link). A flow in recovery state experiencing no losses may return back to a normal state depending on the number of observed retransmitted packets in the specific recovery stage and the number of known dropped packets in this recovery stage. A flow in timeout recovery state that has successfully retransmitted lost packets will recover to a slow start state.

Normal packet loss: A packet loss in TAQ is defined as a normal packet loss if the packet dropped is a new packet. A packet loss can trigger a flow into a fast retransmit or in a recovery state. While flows can typically quickly recover from a single loss, multiple losses are much harder to deal with. In TAQ, we maintain knowledge about the number of packet drops experienced by a flow and track the recovery progress of a flow by tracking the number of retransmitted packets that immediately follow in future epochs. In TAQ, flows that have experienced packet losses in the current or previous epochs are given higher priority in future epochs for retransmitted packets and existing packets within the sliding window to prevent timeouts and bursty losses.

Retransmitted packet loss: When a retransmitted packet is dropped, a flow hits a timeout state. TAQ tracks retransmitted packets to specifically prioritize these packets to prevent timeouts for flows. In the inevitable case where a retransmitted packet needs to be dropped due to lack of network resources, TAQ needs to keep a track of the previous state of the flow for the last normal transmission, prior state of the flow (length of silence period) and number of packets in the recovery stage. In essence, any retransmission from a flow in an extended silence period should be prioritized over a retransmission from a flow in a silence period which should be further prioritized over a first retransmission. TAQ uses this length of the silence period and the last normal transmission to determine the priority of a retransmitted packet.

4.2 TAQ Multi-class Priority Queuing

TAQ uses a multi-class priority queue scheduling policy which is primarily centered around maintaining the balance between flows hitting timeouts and ensuring fairness across individual flows. TAQ can adopt either the standard fair-queuing based fairness model or can support the proportional fairness model using the RTT estimates of flows.[3] TAQ maintains several queues corresponding to different classes of packets:

- *Recovery Queue:* The recovery queue corresponds to only retransmitted packets of flows. The recovery queue operates as a priority queue where the priority of a packet is inversely proportional to the length of the silence period of the flow; the longer the silence period, the higher the priority of the packet in the recovery queue.
- *NewFlow Queue:* This queue corresponds to new flows that have just begin transmissions and are in slow start.
- *OverPenalized Queue:* This queue corresponds to normal packets of flows that have been experienced multiple packet drops in the past and current epochs cumulatively; the goal of treating this flows separately is to not over-penalize these flows, but to minimize timeouts and repetitive timeouts.
- *BelowFairShare Queue:* This queue corresponds to normal packets of flows whose current transmit rate is below the fair share; here, fair share can be calculated based on either the fair queuing model or the proportional fairness model.
- *AboveFairShare Queue:* This queue corresponds to normal packets of flows that are sending above their fair share.

TAQ uses a multi-level hierarchical queuing algorithm to manage these 5 different queues which we elaborate next.

Level 1: At the top level with the highest priority is the recovery queue that operates as a strict priority queue where recovery packets are prioritized based on the previous silence periods of the flows. To prevent the scenario where a large fraction of flows operate in recovery state alone, this queue is capacity limited so recovery packets cannot occupy more than a certain amount of network resources.

Level 2: At Level 2, TAQ considers the BelowFairShare, NewFlow and OverPenalized queues and allocates all these equal priority levels. Within each queue, we use a simple FIFO policy. We explicitly limit the NewQueue capacity to limit the number of new connections in the system to specifically prevent congestion collapse of admitted flows. This is useful for implementing admission control policies of new flows. Between the OverPenalized and BelowFairShare

[3] We focus on the standard fair queuing based fairness model in this paper.

queues, we proportionally allocate resources based on the queue demands.

Level 3: At Level 3, the AboveFairShare queue operates as a FIFO queue with strictly lower priority compared to Level 2 flows. Once a flow is penalized with more than 2 packet drops in an epoch, the flow is automatically assigned to the OverPenalized queue.

There are several reasons we adopt this 3-level multi-level hierarchical queuing algorithm in TAQ. First, the first level is strictly designed based on reducing the probability of timeouts. This Level is given strictly higher priority compared to Level 2 and 3, though the net output rate is restricted to limit network overload with packet retransmissions. Second, Level 2 consists of 3 different flows which are hard to differentiate. Hence, these queues are associated with similar priority levels though the NewQueue is restricted to curtail rate of admission of new flows. Finally, flows above their fair share are given the lowest priority to maintain fairness.

4.3 TAQ with Admission Control

One of the key takeaways from the model is the dramatic increase in the probability of timeouts of flows beyond a loss rate of $p_{thresh} = 0.1$. TAQ monitors the packet drop rates at the queues and if the drop rate exceeds this threshold, TAQ adopts an admission control policy to limit the flow contention to reduce the loss rate below the critical threshold. Without admission control, existing flows could spiral deeply into the small packet regime where a majority of a flows may remain in repetitive timeouts and not make progress regardless of the underlying queuing algorithm.

When applying admission control, we need to understand the application level dependencies between flows given that several real-world applications generate multiple TCP flows for each session. Middleboxes perform admission control at the granularity of *flow pools*, which is a set of inter-related flows corresponding to the same application; the canonical example being HTTP web traffic. In the case of $HTTP/1.0$ a separate TCP connection is setup for each request, and in $HTTP/1.1$ requests may be pipelined. A typical web session may simultaneously retrieve content from several web sites in parallel. We define a "flow pool" as a collection of inter-related flows from the same source to different destinations that are all initiated within a short time-period (e.g. a few seconds). For simplicity, we assume that users do not switch sessions across applications within a few seconds.

Admission control can be performed as a separate process on NewQueue by limiting the number of new connections that are queued in this queue. A flow is admitted if: (a) it belongs to a flow pool that has already been admitted, or (b) it belongs to a new flow pool and the current loss rate is less than p_{thresh} (in practice, we use a threshold slightly smaller than p_{thresh} as a congestion avoidance strategy). After a flow pool is admitted, the middlebox tracks the status of each individual flow within a flow pool. However, TAQ can implement fair sharing across flow pools instead of across individual flows to maintain fairness across applications. Once a flow pool is identified, TAQ's queuing policy does not change except the fair share calculation.

If a flow is not admitted, in certain applications such as web browsing, one can make simple modifications to provide feedback to the user on the expected wait time or a spoofed HTTP 503 "Service Unavailable". If the middlebox acts as a proxy (instead of a transparent proxy), it can provide useful feedback as an appropriate response as maintaining a visible queue of requests with expected wait times and finish times for each browsing request. Such an approach has been integrated in prior work on delay-tolerant web browsing solutions [7]. In a feedback oriented solution, after a specific wait time, T_{wait}, the user is guaranteed admission for one flow pool. In practice, we set the value of T_{wait} to be small (few seconds) and less than the TCP SYN connection timeout; hence, the TCP connection request is still alive in timeout phase and a future SYN retry can be admitted.

4.4 TAQ in Practice

TAQ can also be implemented using only a single middlebox which sits prior to the bottleneck link. TAQ can be implemented at a router level or can be realized using a combination of software routers or transparent proxies (that tunnel traffic between them) at either end of a constrained link.

For simplicity of discussion, we assume that we have control over the low-bandwidth network resources and the underlying channel has very low loss rates and all losses are caused due to congestion at the middleboxes. In reality, if the middleboxes are overlay nodes where the traffic between them experience unpredictable losses due to cross traffic, then we would build our entire solution on top a system such as OverQoS [35], which provides a controlled-loss virtual link abstraction with very low loss rates between the two overlay nodes. Unless we have control over which packets are dropped at the middleboxes, it becomes fundamentally hard to provide any form of quality of service in these highly constrained small packet regimes. In both the router-level model or the OverQoS-like overlay network model, the TAQ nodes are constantly aware of the available bandwidth on the underlying network. This information is in turn used in the multi-level queuing algorithm of TAQ.

5. TAQ Evaluation

We implemented TAQ both in ns3 and also as a collection of elements in Click [18] (with admission control). In our experience, because most machines found in developing regions run Windows we also have a separate implementation of TAQ in 1300 lines of C# code using the SharpPcap library (without admission control).[4]

[4] We did currently implement admission control here because it would "break" existing HTTP/browsing semantics and we would have to also deploy a separate intermittency aware browser (e.g. RuralCafe [7]) to accommodate this change.

To evaluate the effectiveness of TAQ, we use both ns3 simulations and a physical testbed. In simulation we continue to use the same basic dumbell topology as the simulations results presented previously. In our physical testbed we setup a simple 100Mbps Ethernet/switched testbed with four 64-bit 2.8Ghz Intel Core Duo machines with 4GB memory running Linux Mint 15 Olivia to ensure that our testbed generally reflects the relatively underprovisioned machines found in our target environment. For our experiments we emulate a bottleneck link with a specific bandwidth and delay. We conduct a similar set of experiments in both our simulations and testbed across a variety of synthetic and real workloads. For our real workloads, we use two types of traces: (a) web access logs of users in low bandwidth environments in India and Ghana; (b) tcpdump network traces in India and Ghana to understand network bandwidth conditions of real world networks. Where possible, we emulate these conditions in our emulated and simulated network environments.

We primarily compare TAQ with Droptail (DT) queuing since as discussed earlier in Section 2, most queuing mechanisms are not suited for these environments and Stochastic Fair Queuing (SFQ) and Random Early Detection (RED) offer similar results to DT. In small packet regimes, almost every flow operates over very small TCP windows and has very few packets under contention within every epoch. At any time, we observe that the number of outstanding packets corresponding to a single flow in the buffer is very small, if not often just zero or one packets. At that granularity when there is very limited choice across flows, queuing mechanisms like SFQ have limited scheduling choices and offer similar results as Droptail.

As a summary of our evaluations, we show that across a variety of small packet regimes, TAQ consistently achieves the following results: (a) improves short-term fairness; (b) provides smooth and predictable evolution of flows; (c) gracefully handles the introduction of short-flows; (d) provides more predictable download times; and (e) admission control can provide predictaiblity gains under heavy contention in small packet regimes.

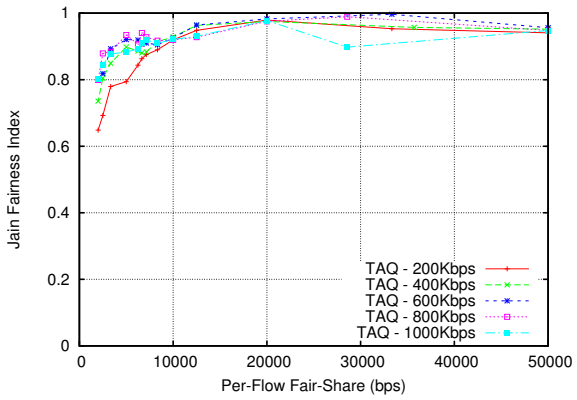
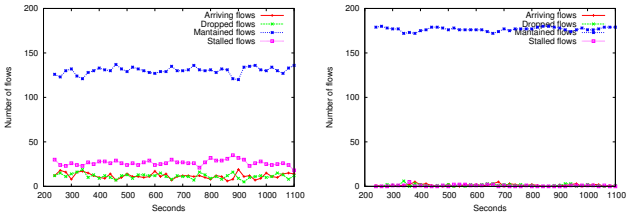


Figure 8: Short-term Jain-fairness for TAQ (20sec slices)

5.1 Short-term Fairness

Figure 8 shows the simulation results from typical short-term fairness characteristics of TAQ over 20 second time slices across different bottleneck link configurations. Comparing these results with Figure 2 for DT, we observe that TAQ improves fairness characteristics across the entire spectrum of bandwidth configurations in comparison to DT. Across even very low bandwidth conditions, TAQ achieves a better Jain's fairness index. In many cases, the fairness achieved by TAQ is higher than 0.8 which clearly shows that TAQ maintains the goodput of the individual flows within a restricted range without affecting the link utilization; in most cases, the link utilization is close to 1. An interesting observation is that when a flow experiences a drop at a TAQ queue, the overall link utilization is not affected by this drop since the loss occurs before the link at the TAQ queue.



(a) Flow Evolution for DT (b) Flow Evolution for TAQ

Figure 9: Flow Evolution for DT and TAQ

5.2 Flow Evolution

To further elaborate on the fairness, we contrast the evolution of flows across different states in TAQ with DT. Figure 9a and Figure 9b show the simulation results for flow evolution of DT and TAQ, respectively, for 180 flows across a 600Kbps bottleneck link configuration. Here, we classify flows into four categories based on the evolution from previous state to the current state: (a) Maintained flows reflect the flows that continue to make progress remaining in a normal state or a slow start phase across continuous epochs; (b) Dropped flows signifying a transition from a normal state to a silent state due to packet drops (just experienced timeout); (c) Arriving flows signifying a transition from a silent state to an active state of recovery or normal; (d) Stalled Flows that continue to remain in repetitive timeout state. We observe that in TAQ, the number of flows in a stalled state is nearly zero. indicating that TAQ nearly eliminates the number of flows that experience even 2 continuous silent epochs. We also find that TAQ increases the number of flows in the maintained state compared to DT signifying a larger number of flows that make continuous progress in TAQ as opposed to a highly bursty behavior in DT. This result shows that TAQ enhances the performance predictability of flows. Finally, TAQ has a nearly no states transitioning from Arriving and Dropped flows across each epoch and most flows are active within even 2 epoch windows. Overall, we observe

that flows experience a much smoother evolution under TAQ than in DT. We repeated this analysis and found similar results under a wide variety of workloads and network environments (including SFQ and RED).

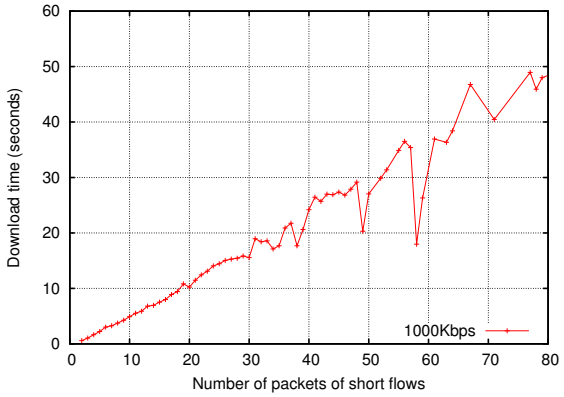


Figure 10: Behavior of TAQ with short flows

5.3 Short Flows

The use of NewFlow Queue in TAQ enables TAQ to smoothly and gracefully handle short flows. Specifically, if a flow transmits a very small number of packets, TAQ can automatically lower the flow completion time. To demonstrate this, we introduce a mixture of short flows of variable length (number of packets) with a background of long-running flows to measure their effect. Figure 10 shows the download times of different flows as a function of number of packets (in this experiment, we introduced 32 short flows over a 1Mbps bottleneck with 50 long flows - 20Kbps fair share per flow). We observe that short flows with small window sizes have more predictable download times that vary roughly linearly with the number of packets of the flow; the download time has larger variations once the flow blurs the boundary of a short flow.

5.4 Predictability of Web Flows

To show that TAQ improves fairness and is able to achieve good rates in the real world and with an underprovisioned server, we use our C# implementation and artificially constrained the network bandwidth, latency, and queue size at the middlebox to be generally consistent with our trace parameters. On the server machine we ran a standard Ruby web server and routed packets to our TAQ middlebox implementation connected to the LAN with two 100Mbps Ethernet cards. On the two designated client machines we run a Ruby script that opens long lived requests to the webserver. Figure 11 shows the Jain fairness as a function of the per-flow fair-share for 600Kbps and 1000Kbps links. As expected, we observe that even on realistically basic hardware TAQ is able to easily handle these flow rates.

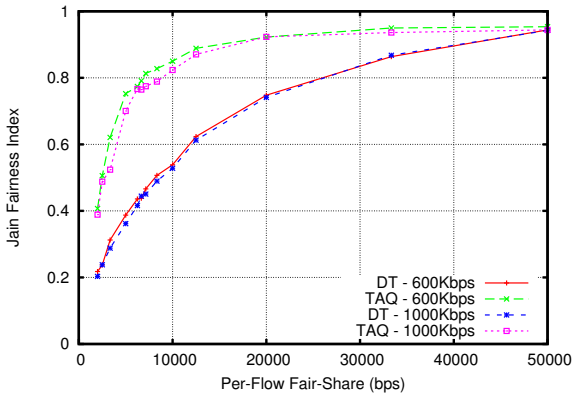


Figure 11: Short-term Jain-fairness for TAQ on our testbed (20sec time slices)

5.5 Admission Control and Web Predictability

To demonstrate the effectiveness of admission control and how it positively impacts web flows in a real world setting. We consider an environment similar to the network setting in Figure 1 where we have clients spawn web requests based on a real-world access log. We specifically replay a 2 hour peak load log. The physical setup is identical to the previous scenario. Here, in our client scripts our two clients are each configured to open up to four connections at a time, and request objects as soon as possible rather than the logged request time to simulate the dependence of a request on previous requests. During transient periods, there may be unfairness due to overbooking due to the fact that the admission controller may not admit new flows since it must honor commitments to allow flows belonging to an admitted flow pool; however the same flows are admitted within a short time period (since we emulate flows which are not admitted to constantly retry till admission). We specifically include the additional waiting time for flows not initially admitted as part of the overall download times of flows.

Figure 12 is a CDF of time taken to download objects of different sizes (10KB buckets) for the same 1Mbps network configuration as before. We observe that TAQ significantly reduces the overall object download time in the median and the worst case (inclusive of the additional waiting time for non-admitted flows). The gap is particularly pronounced for short flows where TAQ reduces flow completion time by a factor 5 for both the median and the worst case. For large objects, TAQ reduces the overall download times by a factor 2 for the median and by a factor 1.6 in the worst case. We also observe that the overall variance in the download times are significantly reduced across the board.

6. 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.

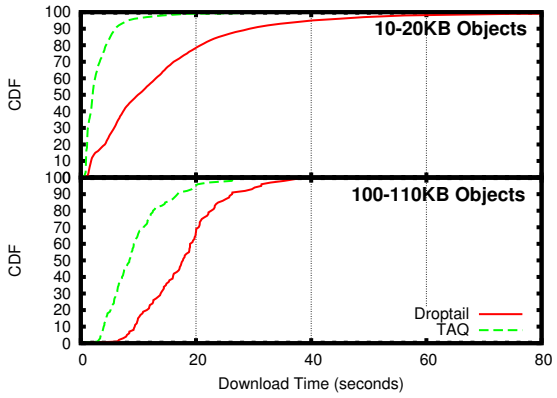


Figure 12: Object download times CDF for TAQ with admission control in comparison to a droptail queue for small and medium sized objects

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 round trip 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.

TCP's loss of fairness under pathological sharing is not a surprise, and has been noted in [17,21,29]. 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, 6, 19, 20, 22, 24, 27, 33], derive analytical expressions for the steady-state throughput of a TCP flow based on its round trip time and loss probability. Of these models, ours is closer to [6, 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.

Our TCP model supplements Markov models for TCP that have been proposed in [11, 12]. These models focus on TCP behavior when the packet loss rates are relatively small. Specifically, Fortin-Parisi-Sericola (F-PS) [11] 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 harder than finding the expected goodput. Yet, despite the complexity of 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. In mainstream networking literature, user-perceived web latency has been a concern [8], but these works focus on link losses and latency rather than small packet regimes caused by bottleneck bandwidth. Han et. al. consider more flexible transports to accommodate diverse application and network requirements that emphasizes dynamic adaptation based on pricing [13].

7. Conclusion

Three factors have led to the emergence of small packet regimes: rapid growth in web content complexity, limited growth in connectivity, growth in network sharing. The limited growth in connectivity is particularly true in developing regions and mobile web contexts where TCP flows break down in small packet regimes resulting in unfairness, long download times, repetitive timeouts and long hang periods.

To partially alleviate this problem without any modifications to the end-hosts, we propose Timeout Aware Queuing (TAQ) as a middlebox solution that can address many of the aforementioned problems of TCP flows in these regimes. Since TCP is inherently not designed to operate in these regimes, designing a middlebox-based solution to alleviate the problem is a challenging proposition. TAQ leverages a detailed TCP flow model to predict state transitions of flows at a middlebox by performing fine-grained packet drops coupled with flow tracking to reduce the impact of timeouts. To the best of our knowledge, our work is the first to formally characterize the small packet regime and propose measures to address TCP problems in these regimes. In the future we plan to investigate end-host congestion control mechanisms for small packet regimes.

8. Acknowledgements

We would like to thank Cesar Lopez Ramirez for his help implementing the latest version of TAQ. We are grateful to our shepherd Alan Mislove and anonymous reviewers for their feedback. This research was supported in part by the Center for Technology and Economic Development (CTED).

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] Akamai: State of the Internet. http://www.akamai.com/.

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

[7] J. Chen, L. Subramanian, and J. Li. RuralCafe: web search in the rural developing world. In *Proceedings of the 18th international conference on World wide web*, pages 411–420. ACM New York, NY, USA, 2009.

[8] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013.

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

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

[11] 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.

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

[13] D. Han, R. Grandl, A. Akella, and S. Seshan. Fcp: a flexible transport framework for accommodating diversity. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013.

[14] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control TFRC: Protocol Specification, Jan. 2003. RFC 3448.

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

[16] 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.

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

[18] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[19] 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.

[20] 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.

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

[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] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation-Volume 1*, page 6. USENIX Association, 2004.

[36] Websiteoptimization.com. http://www.websiteoptimization.com.