Capturing TCP Burstiness in Light-weight Simulations*

Polly Huang
Swiss Federal Institute of Technology
ETH-Zentrum, Gloriastrasse 35
CH-8092 Zurich, Switzerland
huang@tik.ee.ethz.ch

John Heidemann
USC/Information Science Institute
4676 Admiralty Way
Marina del Rey, CA 90292, USA
johnh@isi.edu

ETH TIK-Nr.92

Abstract

Burstiness in data traffic is emerging as a critical property that protocol analysis nowadays can no longer ignore. To preserve the kind of burstiness or scaling phenomena observed in aggregated TCP traffic, we develop a behavioral model that captures TCP's window-based and closed-loop control. Through a novel modeling technique – *exhaustive state exploration*, we systematically examine each TCP states over a restricted range of connection length and packet loss. This restricted range covers the TCP behavior most common to web exchanges. When connections step outside this range (becoming too long or taking more than one loss) we preserve simulation accuracy by taking a *abstraction fault* and changing to a more detailed model. By counting packets with interarrival times falling into certain critical intervals – round trip time (RTT) or retransmission timeout (RTO), we are able to create finite state automaton (FSA) with states and transitions indicating rounds of back-to-back packet transmissions. We demonstrate that an FSA approximation of TCP can produce *light-weight simulation models* of TCP suitable for background traffic, and that these models accurately reproduce multifractal scaling behavior in IP network traffic.

1 Introduction

Simulation has become a necessary tool to study networking protocols at large scale. Testbeds of 1000s of nodes are not feasible for general use. Simulation studies often consider a new protocol (transport protocol, queueing dicipline, etc.) by studing a few new flows in the context of background traffic. Simulation of background traffic is difficult for several reasons: it must accurately model burstiness inherrent in Internet traffic, it must simulate large numbers of the connections, and it must be efficient to model web traffic, the current dominate use of the Internet.

* Submitted for publication

Correctly modeling burstines in data traffic is critical to reaching appropriate conclusions. For example, studies of RED [12] have reached very different results in different simulation [12] and emulation studies [7]. A key difference in these studies is the level of burstiness in the simulated traffic.

A recent promising approach to capture burstiness is the structural approach to traffic modeling [11], closely modeling how traffic is created in a real network. Two mechanisms are crucial to capture Internet behavior over a wide range of time-scales:

1. User-level property – exponential web session arrival and heavy-tailed session duration contribute to the persistent burstiness observed throughout large time scales. (self-similarity at large scales)

2. Network mechanism – TCP's closed-loop control and window-based transmission have strong influence to the unusual burstiness in medium and small time scales. (periodicity at round-trip-time scale and multifractality at small scales)

A problem employing this approach to traffic modeling is the state cost. A simulation of a 400-user ISP network requires 1000s of random varaibles for the user-level property and 100,000s of TCP connections to capture the network mechanisms. Even with parallel simulation techniques, memory efficicency is often the critical bottleneck as simulation sizes grow.

Finally, simulation models must correctly model current Internet traffic. TCP is the dominant transport protocol on the Internet (95% of bytes, 90% of packets, 80% of flows [8, 31]), and web traffic (HTTP) typically accounts for the majority of TCP traffic (often 60-80% worth). Although the steady-state behavior of long lived TCP flows has been carefully studied and modeled analytically (beginning with Floyd [13], see Section 2 for details), these studies do not model short flows. While long-lived flows account for the majority of bytes sent on the Internet, the majority of flows are short. Several studies [8, 5, 22, 9] reported that flow or web document sizes exhibit a power-law distribution with average of 15–30 packets or 5–10 kilobytes. Furthermore, TCP is at its most aggressive when starting up (during its initial slow-start phase), so carefully modeling this period is critical to capturing the network mechanisms that contribute to the small-time-scale behavior.

The key contribution of our work is a new approach to efficiently modeling short TCP connections for background traffic. We accomplish this by taking advantage of heavy-tailed nature of Internet traffic and representing short connections very efficently while representing long connections with more detail. Since majority of flows are short, this approach greatly reduces simulation memory requirements. We efficiently represent short connections with a Finite-State Automota that models.

We generate this FSA model with a novel technique: *simulator-assisted exhaustive state anaysis*. We use a simulator to investigate all possible cases of segment loss for a given amount of data. From this exploration we construct a finite-state automaton (FSA) that approximates TCP congestion control behavior. Hand generation of this FSA from the TCP specification or an implementation would be extremely error prone. We therefore use a simulator to automatically enumerate the state space, and are developing software to fully automate FSA construction.

Our approach requires three assumptions of TCP behavior. First, we assume that most short TCP flows can be modeled as rounds of several back-to-back packets separated by a delay of about one round-trip time [15] (or longer delays in the case of error). Second, packet losses are independent and identically distributed (as opposed to burst losses). We validate these assumptions by comparing our FSA model to detailed simulations of TCP in Section 4.6. Third, our FSA models only a limited number of losses will occur in each flow. Our current models only consider at most one loss per flow, an appropriate assumption for short flows and relatively low loss rates (5%). We detect when this assumption will be violated (when a second loss will occurs) and replace our FSA TCP with an equivalent fully detailed TCP through an *abstraction fault*, thus preserving simulation accuracy. We examine this third assumption in Section 4.2, and note that we are able to correct for it when applying our FSA model to background-traffic simulation.

2 Related Work

Our work builds on four areas of prior work: the approach of exhaustive state analysis builds on protocol analysis through state exploration (in general) and analysis of short TCP transfers. Our general TCP behavior analysis is related to work in modeling TCP steady-state behavior and start-up behavior. Our applications of the FSA model to simulation are related to work in simulation abstraction.

Protocol analysis through state exploration: Finite-state graphs are one of several tools that have been used to understand protocol behavior, correctness, and performance. Traditionally the primary application of FSA protocol approximations has been proving protocol correctness (for example, [21, 16]). A common problem in this work is controlling the size of the state space which must be considered. Typical approaches are to make assumptions to constrain the problem or to employ techniques to merge states where possible (for example, [23]). Our work uses both of these techniques to limit state space, and it employs simulator-driven FSA construction to minimize chance of construction error and mitigate the cost of a large

state space. Furthermore, unlike most prior work with large-FSA representations, we are focused on performance analysis and approximation rather than evaluation of correctness.

Analysis of TCP steady-state performance: There has been a significant amount of work on characterizing steady-state behavior of TCP. As early as 1991, Floyd presented a simple heuristic analysis to predict TCP performance with multiple congested routers [13]. Ott, Kemperman, and Mathis completed a formal analytical study of TCP window size behavior [26], and Lakshman and Madhow [20] proposed a more elaborate model for TCP assuming high delay-bandwidth product and random losses. Mathis et al. [24] and Lakshman et al. [20] independently derived similar closed-form equations to approximate TCP bandwidth in the steady state assuming sporadic losses (without retransmission timeout),

More recently, Padhye et al. [27] solved the probability of packet losses that will incur retransmission timeouts and delayed acknowledgements. Their model was shown to predict bandwidth of long TCP connections reasonably well, and adds consideration of the maximal window size and timeout interval to the above simpler equation.

Common to these efforts is a focus on steady-state behavior of TCP with infinitely long connections. These approximations have been suggested for use as acceptable bounds on congestion control (first by Floyd [14], and in recent proposals [29, 4, 32, 28]). Studies of actual Internet traffic suggests that, although long connections account for much of the bandwidth, most connections are quite short. More precisely, the distribution of connection length is heavy-tailed with fairly small average around 15–30 packets or 5–10 kilobytes [8, 5, 22, 31, 9]. The key difference between the related work we consider next and our work is an examination of these short flows.

Analysis of TCP-slow-start performance: Two efforts have focused on analysis of short TCP connections. Heidemann et al. [15] modeled short TCP connections (slow-start phase) assuming no loss to compare several alternative request/response protocols. Cardwell et al. build upon this work by combining it with steady state models [27] to derive closed-form solutions for short and long connections in the face of loss. They model connection establishment time (including loss of the SYN), the number of segments sent in slow start and the amount of time spent in slow start as a function of initial window size slow-start window increase rate.

As in both of these efforts, we model TCP as rounds of back-to-back segments beginning when the first is sent out and ending when it is acknowledged. Unlike [15], we consider possibility of packet loss. Unlike Cardwell et al., we use the technique of exhaustive state analysis. Thus we can more accurately model non-linear slow-start rates that they approximate

with an exponential. In addition, we show how our approach can easily be apply to efficient simulation which approximate TCP traffic.

Simulation abstraction: Very large simulations require the use of abstraction: eliminating details from the simulation that do not affect the outcome. A widely used example of abstraction is treating Ethernet as a 10Mb/s "pipe" rather than modeling the details of MAC-level contention and retransmission. Ahn and Danzig proposed packet-level abstractions in flowsim [2]; rather than simulating each packet in a flow, they treat them as groups of back-to-back packets. Inspired by this work and [15] we use this approach to condensing TCP run-time state. Although we currently represent individual segments separately while in-flight, adopting Ahn and Danzig's flow representation would improve run-time in addition to memory consumption. Another promising proposal is to simulate TCP as a set of differential equations [25]. While this method takes into account some degree of traffic dynamics but does not work as accurately when simulated traffic is highly bursty.

3 Modeling TCP

Approaches to flow and congestion control are central to any network transport protocol. These algorithms control how quickly data packets are injected into networks. Open-loop protocols inject packets into the network without regard to the network or receiver; closed-loop protocols react to signals from the network or the receiver concerning congestion or buffer allocation. Closed-loop protocols employing end-to-end congestion control (such as TCP) are critical to the success of the Internet because they adapt to congestion [14].

Our goal is to capture the key aspects of TCP's closed-loop congestion control mechanism [18] well enough to accurately model TCP. Ideally our model would reproduce well enough aggregated TCP traffic that it would be indistinguishable from that of fully detailed TCP across a wide range of timescales [11, 10].

This section briefly summarizes TCP congestion control and describes how we model TCP as several rounds of back-to-back packets separated by a delay of about one round-trip time (based on [15]). We then describe how we use simulated versions of TCP to populate this model over the region we consider.

3.1 Key TCP Algorithms and Timescales

Behavior of short TCP connections is governed by TCP's congestion control algorithms [18] and delayed acknowledgements [6]. Specifically, TCP is *ACK-clocked*—new TCP segments are sent only in response to acknowledgements from the receiver, and the number of new segments introduced is limited by the congestion window, or *cwnd*. During slow start, cwnd is increased by one segment for each ACK received. Slow start exits when the amount of data specified by the *slow-start threshold*, or *ssthresh*, is sent or a segment is lost. According to the delayed acknowledgement rule, clients send ACKs whenever two full segments are received, or when a timer expires.

If segments are lost, TCP will detect that fact and recover in two ways. First, *fast retransmit* [19, 30] is an optimization to quickly recover from a single loss. If the receiver detects a missing segment, it begins sending ACKs for each segment received. The sender interprets these three consecutive ACKs as a loss signal for that missing packet and immediately resends it. Second, if fast retransmit is not possible, the sender eventually will *time out* (after *RTO*, the retransmit time-out delay) and resend the unacknowledged segment.

We can observe TCP's behavior at *multiple timescales*: at the coarsest granularity (seconds) users initiate new connections. Slow-start, fast-retransmit, and timeouts all occur at frequencies proportional to the round-trip time between sender and receiver. Finally, ACK-clocking operates at a very fine timescale on the order of packet-transmission times. Because of these multiple time-scales, Internet traffic exhibits a complex, multi-fractal behavior [11]. Any attempt to model TCP must capture this richness. Experience with other protocols that attempt to reproduce TCP behavior (RAP's rate-based approach [29]) demonstrate that modeling coarse-grained TCP behavior can reproduce very TCP like traffic, but lack of fine-grain adjustment is noticeable in traffic statistics.

3.2 Deriving a Simple FSA

We approximate TCP with a simple model with the goal of the reproducing medium- and coarse-granularity TCP behavior. We model the effect of TCP's slow-start, fast retransmit, and timeout mechanisms, but without directly reproducing those algorithms. We will show later that we can accurately reproduce aggregated TCP traffic across a very wide range of timescales, but we do not expect (or claim) to reproduce very fine-grained, ACK-clock-like effects.

The left portion of Figure 1 shows a typical TCP connection for a web request. After connection setup, the request is

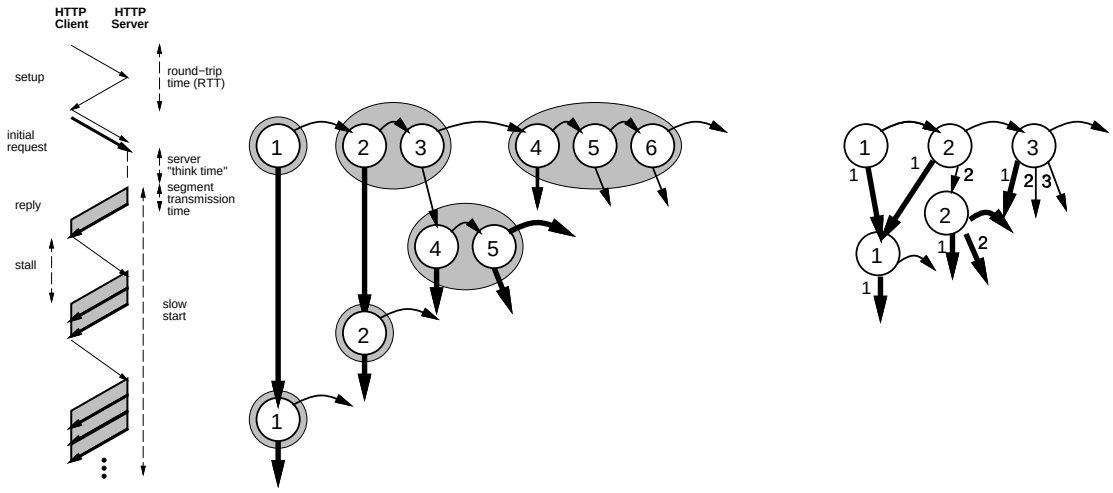

Figure 1: Left: packet exchanges in a short TCP connection. Middle: sequence-number finite-state machine. Right: round-size finite-state machine.

sent, and data returns the web client. Governed by TCP's slow-start algorithm, data is sent in a series of *rounds*. Each begins with the transmission of a data segment and ends with the receipt of an acknowledgement for that segment. When multiple segments are in flight, multiple ACKs will be returned; we assume these transmissions overlap.

We have modeled receivers both with and without delayed acknowledgements. When a receiver implements delayed ACKs we currently generate the ACK for the first data segment immediately. In real implementations this segment will be timer driven with an effectively random delay of 1–500ms, averaging 100ms for typical implementations.

We map this model to a state machine in two steps. First, consider the center part of Figure 1. The number in each state indicates the sequence number of segment sent. Short arcs (in the grey ovals) indicate delay of a packet-transmission time; longer horizontal arcs indicate a delay of about a round-trip corresponding to a wait for the return of an acknowledgement. The top row of states in the center state diagram captures the lossless behavior seen in the packet exchange at the left part of the figure. (The top-left state is transmission of one segment, an RTT delay, two segments in states 2 and 3, and so on.)

To model the effects of packet loss we expand this FSA downward. Packet loss affects both the delay until sending the next segment and how many segments are sent in the next round. A packet loss can trigger either a timeout or a fast retransmit—we show timeouts with thick, downward lines and fast retransmits with thin downward lines, both labeled with the digit of the lost segment. The new state corresponds to retransmission of the lost segment and (if possible) additional segments. The center FSA of Figure 1 shows that if segment 1 is lost, the TCP connection will wait for a retransmission

timeout and then re-send this segment. Alternatively, if segment 3 is lost, TCP sends two more segments and then times out on the missing segment 3.

This model produces a complete but verbose FSA. To simplify this state machine we group series of packets that are sent close together (back-to-back) and represent only timeouts and round-trip delays with arcs. The center FSA of Figure 1 groups these rounds with grey circles. On the right we represent the same information, but the number in the circle represents the number of segments sent in that round. Again bold arcs represent a timeout delay, and now all non-bold arcs represent a delay of about a round-trip time. For states that represent a round with multiple segments in flight, there are multiple downward transitions, each labeled with which segment in that round is lost. Position in the FSA now represents TCP's congestion window and slow-start threshold.

One additional optimization we employ is to merge states in the graph with identical congestion windows and slow-start threshold. The right FSA of Figure 1 shows this where the timeouts from the loss of the first packet in either the first or second rounds both transit to a state where one segment is sent. This abstraction also allows certain degree of state aggregation if different loss patterns end up in a state with same number of packets to send, congestion window size, and slow-start threshold. In the end, we get a more manageable state machine (the right FSA in Figure 1) that captures the essential dynamics of a TCP connection.

3.3 Generating Complete FSAs

Manual generation of an FSA model from the TCP specification would be very error prone. Furthermore, there are many variants of TCP that one might wish to consider. Instead we conduct a series of systematic experiments in a simulator. For each size transmission, we produce a trace of the flow (taken at the sender). We compute packet interarrival times, and knowing the RTT and RTO, back-calculate the corresponding part of the FSA. We systematically repeat this procedure for each possible segment which could be lost.

As an example of this procedure, we constructed a network of four nodes: bottleneck link of 1.5Mb/s, 100ms delay, with two edge links of 5Mb/s and 2ms delay (204ms total round trip time). A trace corresponding to the left part of Figure 1 will show *segment sequence number*/interarrival time of *1*/316.7ms, *2*/1.6, *3*/220, *4*/1.6, *5*/1.6, *6*. This sequence indicates transmission of 1 segment followed by a round-trip delay (plus an extra 100ms delayed from the delayed ACK timer), then

2 back-to-back segments followed by a round-trip delay, then 3 back-to-back segments, that translates into the top row of the FSA. When we repeat the experiment losing the third packet we see this sequence: *1*/316.7ms, *2*/1.6, *3*/316.7, *4*/1.6, *5*/898.4, *3*. Again, we see 1 segment and a round-trip and delayed-ACK delay, then two more segments followed by another round-trip and delayed-ACK delay, two more segments and a timeout until the lost segment is retranmitted. In Figure 3 this corresponds to starting in the top-left node, moving right, then following the thin line down, then the thick line down. As a rule of thumb we treat interarrivals less than 50% as back-to-back, and over 200% as retransmits, although this approach doesn't distinguish delayed-ACK timers. As can be seen, this approach differs from typical simulation-based protocol studies that explore randomly chosen or specific configurations and often consider only statistical summary of the behavior.

Using this approach we have constructed FSA models of TCP for Tahoe and Reno senders and receivers with and without delayed acknowledgements. We used ns-2's one-way TCP implementation and explored the state space with zero and one loss per flow out to transmissions of up to 31 segments. Figures 2 and 3 show the resulting FSA models for these four combinations. These figures are similar to the right part of Figure 1 (lines represent RTO or RTT delay, numbers in circles represent how many segments are sent in that round, and downward lines indicate that the ith segment in that round was lost). In addition, the two-tuples near some nodes indicate cwnd and ssthresh values after a loss.

Although generation of traces was automated, analysis of the traces was done by hand. Automation of this process would not be difficult and would allow a deeper exploration of the state space and exploration of other TCP variants.

We can reproduce the progress of a TCP connection by beginning in the upper left state (labeled 1) and keeping track of the number of segments remaining to be sent. If no segment is lost in the round, it moves to the state on the right with label 2 after delay of one round trip time. If no packets are lost for the entire connection, the FSA TCP connection will move horizontally to the right until all segments have been sent. However, if a segment is dropped, the FSA TCP connection follows the line downwards that corresponds to the dropped packet (first, second, or nth in that round). From the two-element tuples, we see that Reno and Tahoe TCP adjust their slow-start threshold in the same fashion whereas the two flavors of TCP decrease the congestion window size in slightly different manners. When a packet retransmission occurs due to duplicate acknowledgements, Reno TCP reduces its congestion window size to a half of the current window size. On the other hand, Tahoe TCP always reduces its congestion window size to one.

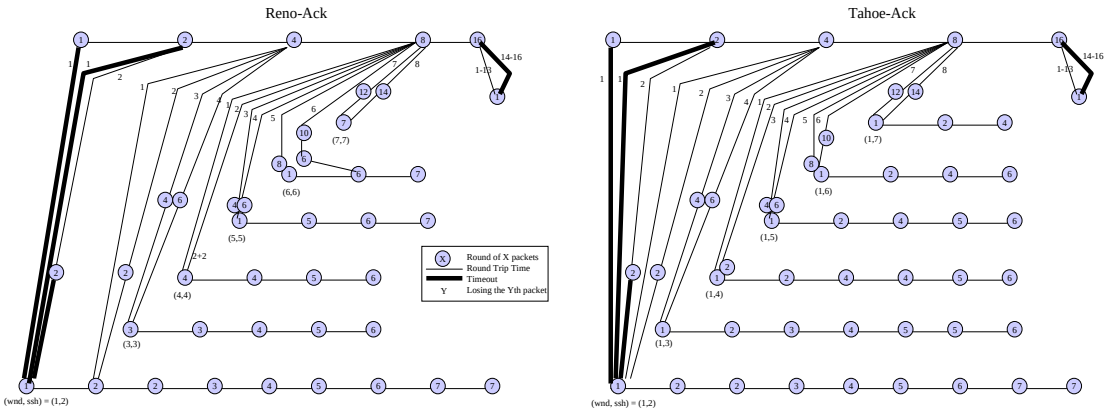

Figure 2: FSA TCP with One-ack-per-packet Sink

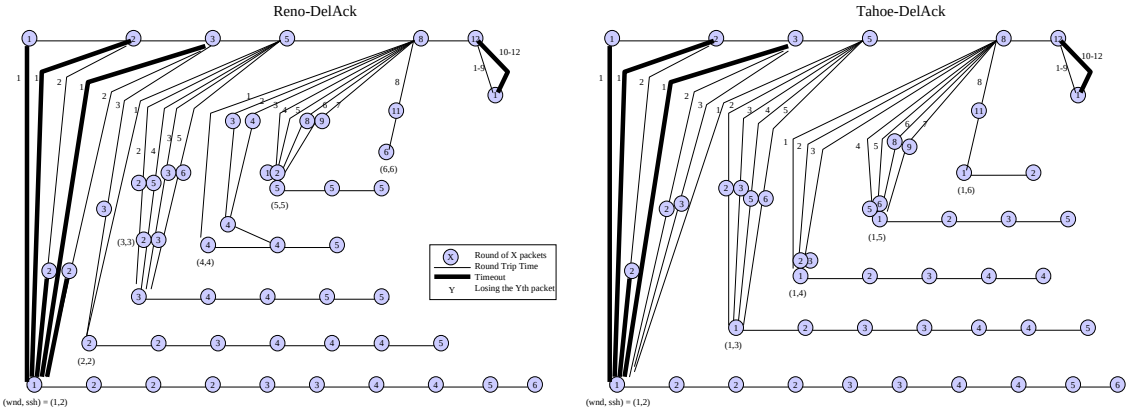

Figure 3: FSA TCP with Delayed Acknowledgement Sink

4 Generating Background Traffic for Network Simulations

One application of our FSA model of TCP is to provide a light-weight approximation of TCP that can efficiently generate large amounts of web-like background traffic for use in network simulation. Our approach is to translate the FSA model into an abstract version of TCP. We will show that this approach can substantially reduce memory consumption in simulations with many flows, while preserving simulation accuracy across a large range of timescales.

4.1 Constructing a Light-weight TCP Agent

We have implemented our FSA TCP agent in the ns-2 simulator [3]. Our simulator implementation of an FSA-driven TCP protocol consists of several parts. We directly implement the states in the FSA model of TCP as a set of C++ data structures. Each active FSA TCP connection maintains a pointer into this FSA, indicating its position in the state machine, the number of segments remaining to be sent, and an approximation of round-trip time. The FSA TCP agent sends regular packets into the simulator; we have made some optimizations to avoid memory consumption by these packets. When one of these packets is dropped (because a router queue overflows or packet corruption is simulated) the corresponding FSA TCP agent is directly informed.

One important difference between our simulator implementation of FSA and the model is how we handle violations of the model's constraints. In the simulator we can detect violation of the constraints (for example, a second packet is lost in a flow) and generate an *abstraction fault*. To recover, we substitute a regular (fully detailed) TCP agent for the abstract FSA TCP agent. We cannot regenerate all TCP state (for example, exact details of the RTT estimation), but we can preserve cwnd and ssthresh. This ability to fall back on a more detailed implementation allows us to get a very accurate simulation of TCP behavior while limiting the model size.

4.2 Single Loss Per Flow

Our current implementation of FSA TCP considers expanding the state through only one loss per flow. This limitation is because our creation of the state machine is currently only semi-automated.

Assuming i.i.d. packet loss probability we can quantify the probability of our model being not covering a given flow by summing all the paths in the FSA which result in crossing two (or more) loss paths. This simplifies to:

$$P[\text{model inapplicability}] \quad = \quad \sum_{i=1}^{n} i(1-p)^{i-1}p^2 \tag{1}$$

where p is the probability a segment is lost and n is the number of segments to send. (Intuitively, for each possible length, what is the chance that we lose two and send the rest correctly.) This equation is shown graphically in Figure 4 for several error rates. This graph quantifies how many absraction faults we expect to take in simulation. Probability of model failure

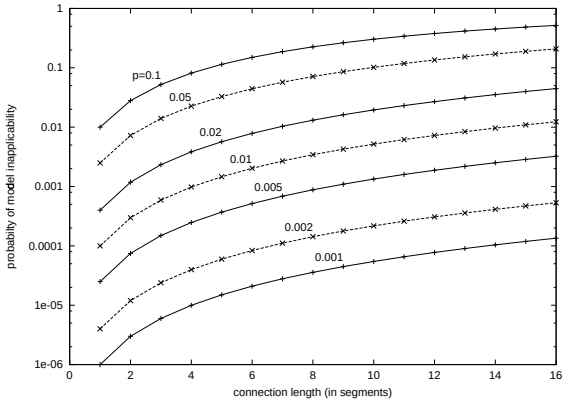

Figure 4: Failure probabilities of the single-loss model as a function of connection length for several values of q and the Reno-DelAck protocol. (Calculated from equation 1.)

is quite low for low loss rates (less than 2%) over the range we consider, but for higher loss rates and longer connections an accurate model will require handling at least two losses per flow.

4.3 Experimental Methodology

We evaluate FSA TCP performance (memory consumption and run-time) and accuracy with a common scenario. We use an ISP-like topology (Figure 5) similar to that used for prior scaling analysis by Feldmann et al [11]. To demonstrate the scaling property of FSA TCP, we vary number of TCP connections from 10 web sessions to 100 web sessions and each session contains about 200 TCP connections. These TCP connections arrive in Poisson random distribution and the connection sizes are Pareto (heavy-tailed) with average 10KB and scaling factor (alpha) 1.2. For the set of FSA TCP simulations, we replace TCP connections that are shorter or equal to 31KB with FSA TCP and let the other longer connections run using original TCP implementation. All simulations run in detailed delivery mode and end after 4200 seconds of simulation time (slightly higher than an hour). We use a Pentium II 450MHz machine with 1GB physical memory, running FreeBSD 3.0, and our modified version of ns-2.1b5.

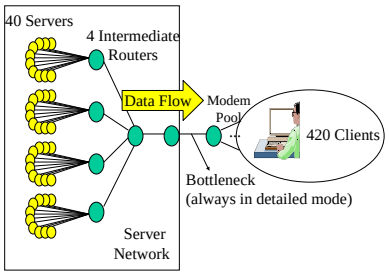

Figure 5: An ISP-like topology used to evaluate FSA TCP in simulation. 420 clients (left) connect to and 40 servers (right) through fast access media.

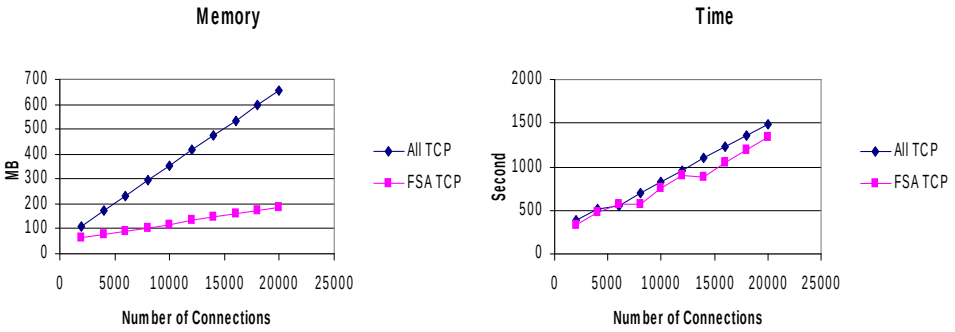

Figure 6: Memory and Time Consumption for FSA TCP Simulations

4.4 Simulation Performance

Figure 6 evaluates the memory and time performance of our FSA TCP compared to a regular (detailed) TCP in the simulator. Each point in Figure 6 is the result of one simulation. Because the simulation is deterministic, repeated runs with the same random seed generate the same results and we do not show confidence intervals. The left plot in Figure 6 shows that FSA TCP improves memory usage significantly compared to detailed TCP. The more TCP connections have to be created, the more memory FSA TCP saves. This evaluation also suggests that TCP state overhead consumes the majority of memory in simulations involving many 100s of simultaneous TCP connections[1]. FSA TCP abstracts away a great deal of details and requires very few states, essentially a pointer to the current state in the finite state automata and a floating point number for round trip time.

In the right plot in Figure 6, we do not see significant improvement in run-time. This is because simulation time is

[1]Optimizations in the 2.1b6 release of ns may change this difference; we have not yet evaluated this experiment on that platform.

proportional to the size of the event scheduler list (unless physical memory is exceeded), which is determined by the number of events or packets scheduled at times. Currently, our FSA TCP implementation in ns-2 generates the exact amount of individual packets as indicated in the finite state automata diagrams. Thus, the amounts of events or packets scheduled for detailed TCP and FSA TCP are the same, and so we do not see much improvement in simulation run-time. An optimization to FSA TCP that we are considering is to represent each round of packets with a representative packet event, as suggested by Ahn and Danzig [2]. By avoiding scheduling individual packets, this approach will substantially reduce the size of the event queue and speed up simulations.

4.5 Distortions in Individual Flows

A risk in using an abstraction is that it may introduce distortions or inaccuracies in the simulation. FSA TCP does not implement the round-trip time estimation mechanism from detailed TCP, and it does not model ACK-clock triggered spacing between individual packets and ACKs. These can result in differences in delay and delay-related metrics.

To quantify the differences, we measure the connection throughput and the queuing delay for each packet at the bottleneck queue. For each number of connections, we run two identical simulations. One uses FSA TCP and the other use detailed TCP. For each connection, we compute the difference ratio of throughput and the absolute difference in queuing delay at bottleneck. The left plot in Figure 7 shows that FSA TCP throughput differs from detailed TCP by about 3%, and that this distortion is largely insensitive to the amount of traffic. The right plot Figure 7 shows that per packet delay (at the bottleneck) FSA TCP differs from detailed TCP by about 10–20 ms, or 5–14% in a network with round-trip times ranging from 140–400 ms. These results suggest that FSA TCP might be useful in generating background traffic where coarse-grain accuracy (up to 100s ms or per round trip time) is required. However, we should avoid using FSA TCP when comparing TCP behavior in fine-grain (below 10ms or per packet) time scale.

4.6 Distortions in Aggregate Traffic

Our main goal in using an FSA approximation to TCP in simulation was to simulate large amounts of background traffic in limited resources. To evaluate its effectiveness in this role we must understand how the distortions observed in Section 4.5 will appear in aggregate traffic. We will show that these effects do not interact (potentially magnifying each other), but instead

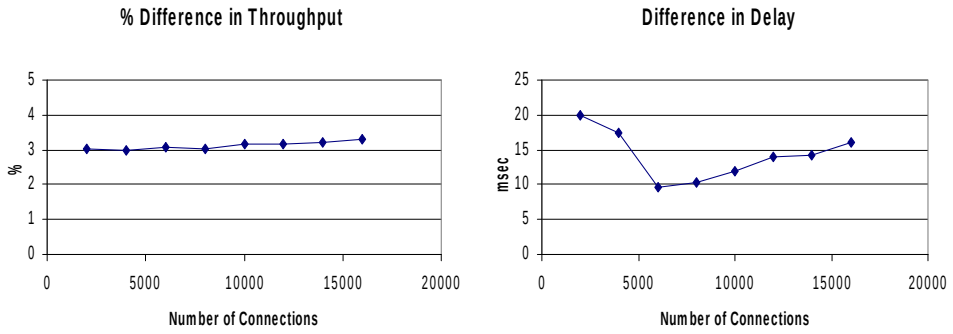

Figure 7: Distortion by FSA TCP: % Difference in Throughput and Delay

are unnoticable across medium and large timescales (longer than 10ms).

To evaluate the scaling behavior of aggregate FSA TCP traffic we employ wavelet-based analysis [10, 11]. In this section, we begin with a succinct description of the wavelet analysis and ways to interpret the scaling plots generated by the wavelet analysis. We then compare aggregate FSA and detailed TCP traffic with this tool.

4.6.1 Wavelet analysis

We use the wavelet transform of a time series to study global scaling properties in traffic. In particular, we examine the average energy contained in each scale of the trace and examine how that quantity changes as we move from coarser to finer scales. The average energy at scale j is the average of the sum of the squared wavelet coefficients $|d_{j,k}|^2$; i.e.,

$$E_j = \frac{1}{N_j} \sum_k |d_{j,k}|^2,$$

where N_j is the number of coefficients at scale j. To determine the global scaling property of the data, we plot $\log(E_j)$ as a function of scale j, from coarsest to finest scales, and determine qualitatively over what range of scales there exists a linear relationship between $\log(E_j)$ and scale j; that is, over what range of time scales there exists self-similar scaling (see [1] for more details). When periodicities at a particular time scale are added into an exact self-similar trace, a dip emerges; that is, there exists a higher frequency packet interarrival in that time scale. It is often we observe a pattern of linear relationship in large time scales and a dip in round trip time scale when applying this global scaling analysis with live network traces [10, 11]. In Figure 8, the scale j is on the bottom axis and the corresponding time (in seconds) is plotted on the top axis for reference.

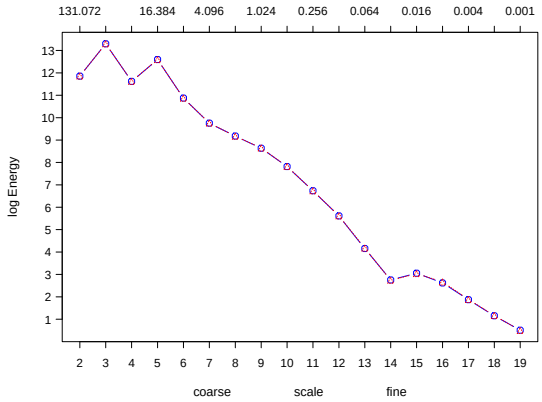

Figure 8: Comparison of global scaling of detailed and FSA TCP. (Two lines are shown, plotted nearly on each other.)

4.6.2 Applying wavelet analysis to FSA TCP

We run our ISP-like scenario employing FSA TCP where possible (when connections are short). We obtain 10 ms time series by parsing packet traces collected at the bottleneck link. After applying the wavelet-based analysis on the time series, we get the global scaling plot in Figure 8. The two lines, one for detailed TCP traffic and the other for FSA TCP traffic, overlap throughout all time scales. This shows our FSA TCP works well in preserving self-similarity and irregular small time scale behavior in aggregated traffic. This suggests that the FSA TCP abstraction will accurately simulate background traffic at timescales larger than 10 ms.

Packet-level data network traffic is characterized as self-similar or fractal with non-trivial scaling behavior at the fine timescales. Feldmann et al. suggest that the cause non-trivial scaling behavior at small time scales is TCP closed-loop control [11]. The intuition behind the effectiveness of FSA TCP at accurately reproducing detailed behavior is that it captures the medium- and coarse-grain effects of TCP's closed-loop control, and that we are able to fault out and replace connections that are too long or experience multiple losses with fully detailed representations.

5 Conclusion and Outlook

We have described how to generate a finite-state autotomaton representation of TCP based on an abstraction of TCP packet-exchange behavior. We have shown how to build a abstract FSA representation of TCP state for generation of background traffic in simulation, and how to select between the abstract and a detailed model at run-time as necessary to optimize both memory and accuracy. We demonstrated that FSA TCP simulations can accurately reproduce the same results as a fully detailed simulation of background traffic. These results suggest that the FSA model can capture the key TCP characteristics at medium and large timescales and can be employed where it is applicable.

Several areas of future work are apparent: We would like to fully automate FSA creation. We have automated trace generation of each loss scenario, but analysis of these traces can also be automated. This would allow us to easily explore multiple-losses per connection, burst losses, and the details of how delayed ACK timers affect performance. We would also like to explore other TCP variants. Comparing the selective acknowledgement extension to TCP [17] with these tools would be interesting. Finally, in Section 4.4 we suggested that run-time performance of FSA TCP simulations can be improved by representing back-to-back packets with a single object. We would like to explore this as well.

References

[1] P. Abry and D. Veitch. Wavelet analysis of long-range dependent traffic. *IEEE Transactions on Information Theory*, 44:2–15, 1998.

[2] J. Ahn and P. B. Danzig. Speedup vs. simulation granularity. *IEEE/ACM Transactions on Networking*, 4(5):743–757, October 1996.

[3] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, S. Shenker, K. Varadhan, H. Yu, Y. Xu, and D. Zappala. Virtual InterNetwork Testbed: Status and research agenda. Technical Report 98-678, University of Southern California, July 1998.

[4] H. Balakrishnan, H. S. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proceedings of the ACM SIGCOMM*, pages 175–188, Cambridge, MA, USA, September 1999. ACM.

[5] P. Barford and M. E. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance'98/ACM Sigmetrics'98*, pages 151–160, 1998.

[6] R. Braden. Requirements for Internet hosts—communication layers. RFC 1122, Internet Request For Comments, October 1989.

[7] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith. Tuning red for web traffic. In *Proceedings of the ACM SIGCOMM*, pages 139–150, Stockholm, Sweden, August 2000. ACM.

[8] K. Claffy, G. Miller, and K. Thompson. The nature of the beast: Recent traffic measurements from an Internet backbone. In *Proc. of the INET '98*, July 1998.

[9] C. R. Cunha, A. Bestavros, and M. E. Crovella. Characteristics of WWW Client-based Traces. Technical Report BU-CS-95-010, Computer Science Department, Boston University, July 1995.

[10] Anja Feldmann, Anna Gilbert, and Walter Willinger. Data networks as cascades: Explaining the multifractal nature of internet wan traffic. In *Proceedings of the ACM SIGCOMM*, pages 42–55, Vancouver, B.C., Canada, September 1998. ACM.

[11] Anja Feldmann, Anna C. Gilbert, Polly Huang, and Walter Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of the ACM SIGCOMM*, pages 301–313, Cambridge, MA, USA, August 1999. ACM.

[12] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, September 1993.

[13] Sally Floyd. Connections with multiple congested gateways in packet-switched networks part 1: One-way traffic. *ACM Computer Communication Review*, 21(5):30–47, October 1991.

[14] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the internet. *ACM/IEEE Transactions on Networking*, 7(4):458–473, August 1999.

[15] John Heidemann, Katia Obraczka, and Joe Touch. Modeling the performance of HTTP over several transport protocols. *ACM/IEEE Transactions on Networking*, 5(5):616–630, October 1997.

[16] Ahmed Helmy, Deborah Estrin, and Sandep Gupta. Fault-oriented test generation for multicast routing protocol design. In *Proceedings of the Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE/PSTV)*, pages 93–109, Paris, France, November 1998. IFIP.

[17] V. Jacobson and R.T. Braden. TCP extensions for long-delay paths. RFC 1072, Internet Request For Comments, October 1988.

[18] Van Jacobson. Congestion avoidance and control. In *Proceedings of the SIGCOMM '88*, pages 314–329, Stanford, California, August 1988. ACM.

[19] Van Jacobson. Berkeley TCP evolution from 4.3-Tahoe to 4.3-Reno. In *Proceedings of the 18th Internet Engineering Task Force*, pages 363–376, Vancouver, Canada, July 1990. Talk slides and notes.

[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*, 3, 1997.

[21] F. Lin, P. Chu, and M. Liu. Protocol Verification using Reachability Analysis. *Computer Communication Review*, 17(5):126–135, 1987.

[22] B. Mah. An empirical model of HTTP network traffic. In *Proceedings of the IEEE Infocom*, pages 592–600, Kobe, Japan, April 1997. IEEE.

[23] G. H. Masapati and George M. White. Algorithms for the reduction of timed finite state graphs. In *Proceedings of the ACM SIGCOMM*, pages 198–216, Stowe, Vermont, USA, August 1988. ACM.

[24] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM Computer Communication Review*, 27(3):67–82, July 1997.

[25] V. Misra, W.-B. Gong, and D. Towsley. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *Proceedings of the ACM SIGCOMM*, pages 151–160, Stockholm, Sweden, August 2000. ACM.

[26] T. Ott, J. H. B. Kemperman, and M. Mathis. The Stationary Behavior of Ideal TCP Congestion Avoidance. `ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps`, August 1996.

[27] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. of the ACM/SIGCOMM'98*, pages 303–314, Vancouver, B.C., September 1998.

[28] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A Model Based TCP-Friendly Rate Control Protocol. In *Proc. of the NOSSDAV*, 1999.

[29] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *Proceedings of the IEEE Infocom*, page to appear. IEEE, 1999.

[30] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, Internet Request For Comments, January 1997.

[31] K. Tompson, G. J. Miller, and R. Wilder. Wide-area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):10–23, November 1997.

[32] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like Congestion Control for Layered Multicast Data Transfer. In *Proceedings of the IEEE Infocom*, 1998.