Faster Network Design with Scenario Pre-filtering

[*]

Debojyoti Dutta, Ashish Goel, John Heidemann[†]
ddutta@isi.edu, agoel@usc.edu, johnh@isi.edu
Information Sciences Institute
School of Engineering
University of Southern California
4676 Admiralty Way
Marina Del Rey, CA 90292
(ISI-Technical-Report-550)

November 15, 2001

Abstract

Design and engineering of networks requires the consideration of many possible configurations (different network topologies, bandwidths, traffic and policies). Network engineers may use network simulation to evaluate changes in network configuration, but detailed, packet-level simulation of many alternatives would be extremely time consuming. This paper introduces the concept of *scenario pre-filtering*—rather than perform detailed simulation of each scenario, we propose to quickly evaluate (pre-filter) all scenarios in order to select only the *relevant* scenarios and discard those that are clearly too over- or under-provisioned. To *rapidly* evaluate scenarios, we have developed several new analytical techniques to quickly determine the steady-state behavior of the network with both bulk and short term TCP flows. These techniques apply to arbitrary topologies and routers that use both drop-tail and RED queuing policies. Since we are only interested in selecting the interesting scenarios for detailed simulation, the answers need only be approximate. However, we show that accuracy is typically within 10% of detailed simulation. More importantly, these techniques are 10–300× faster than detailed simulation, and, hence, pre-filtering is a promising technique to reduce the total simulation time when many scenarios must be considered.

[*]This work is part of the SAMAN project, supported by DARPA and the Space and Naval Warfare Systems Center San Diego (SPAWAR) under Contract No. N66001-00-C-8066. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or SPAWAR.

[†]The authors are also at the Department of Computer Science, University of Southern California

1 Introduction

The design and engineering of networks is a challenging task. Interactions between traffic load, topologies, and protocols create a huge parameter space that must be understood. Network simulation can play an important role in this understanding and in the design of better networks. For example, consider a simple link having a variable number of FTP connections going through it. A network engineer might want to find the point when the capacity of a certain link would be insufficient to meet certain goals. Or she might want to obtain the maximum number of bulk connections that can be supported by a particular network topology. To explore the network behavior, the engineer might employ a network simulator to evaluate a number of scenarios with different traffic characteristics.

Packet level simulators, such as *ns-2* [25], are discrete event driven and their granularity is a single packet. They are widely used to understand network and network protocol behavior, particularly by protocol designers. They are less commonly used to understand the behavior of operational networks in scenarios such as the ones just described for at least two reasons. First, it is not always easy to represent the current status of an operational network in a simulator because its topology or traffic may not be well known or they may be dynamic. Second, although simple simulations can be run quite quickly, simulating scenarios with many nodes and at high traffic rates can easily become quite time consuming. Understanding the behavior of the network will require many scenarios to consider alternate traffic or configuration choices. Yet often many of these scenarios are not interesting, either because the networks are significantly over-provisioned, and, hence, do not provide an understanding of the network bounds, or, they are under-provisioned. Only a few scenarios are critical i.e. provide a good balance to define the operating limits of the network. Another fact about simulations with protocols such as TCP [23] is that they often need to be run for several seconds in order to reach a steady state. Such restrictions further add to the simulation time for each scenario.

This paper addresses the second problem, i.e. to evaluate a wide range of simulation scenarios to find the relevant ones and discard the undesired ones quickly. Our work is based on the observation that there is no need to simulate the uninteresting scenarios in detail. We propose *Approx-sim*, a design tool that can very quickly evaluate the steady-state behavior of scenarios and *pre-filter* them by user-supplied criteria. It allows uninteresting scenarios to be dismissed quickly and the interesting ones to be evaluated in detail.

Approx-sim identifies the steady-state behavior of scenarios using a hybrid queuing theoretic approach for drop-tail routers, a new approach to RED modeling, and, an approximate fixed-point algorithm. It makes use of well known equations for bulk TCP behavior [22] and a new approach to approximate the short lived TCP flows. We show that *approx-sim* can evaluate scenarios an order of magnitude faster than what is possible with packet-level simulators. At the same time, the accuracy is typically within 10% with the largest observed error being 30%. In *approx-sim*, scenarios consist of a mix of long and short lived TCP traffic (“elephants” and “mice” respectively), and, have both drop-tail and RED queuing at routers over arbitrary topologies.

Packet level simulators are inherently deterministic. This is appropriate for simulation studies. However, this can lead to synchronization of traffic. Our analytical simulation engine, *approx-sim*, does not run into such problems. In fact, comparisons with *approx-sim* helped us to identify scenarios where the *ns-2* simulations were getting synchronized. In later sections, we demonstrate how we removed the synchrony in *ns-2* by adding short lived flows and by using RED gateways.

2 Related Work

Our work is related to other approaches for fast simulation, either through parallelism or approximation. It also builds on analytical approaches to understanding network performance.

2.1 Rapid simulation

Parallelism has been used for many years to improve simulation performance [4, 16]. Several parallel network simulators are currently available, such as Parsec [1], SSFNET [6], and parallel versions of *ns-2* [13, 24]. Our work is complementary to these efforts—while parallelism can improve the performance of detailed simulation by up to the number CPUs devoted to the task (typically 4–8 today), pre-filtering can improve performance many-fold by never simulating the uninteresting scenarios in detail.

RPI has proposed the use of *experimental factoring*, that combines multiple sequential simulations on a network of workstations with search algorithms to choose the scenarios that should be considered [26]. This work is similar to ours in goal (rapidly exploring the design space), and largely complementary in result: their approach could gain further performance improvement by using pre-filtering techniques. In contrast to our approach, they do a random search of the parameter space while we have a deterministic algorithm.

2.2 Analytical approaches

Queuing theoretic approaches have long been used to evaluate network performance (for example, [18]). Although it is necessary to understand fundamental performance limits, these approaches must be applied to the Internet with care because of the complexity of the protocols and the networks in use there. Our approach seeks to get the best from both queuing theory and detailed packet-level simulation by using the former for a rapid approximation while using the later for detailed evaluation.

Recently, there has been extensive work in fluid-flow-based approaches to network simulation [19, 20, 21]. These approaches are promising, and, some such as Misra et al.’s [20] approach can capture the transient behavior. However, further work is needed to understand the performance of these approaches for large networks. Our work complements their strategy; we look at a different problem of finding the steady state values of the network state.

2.3 Fixed point

Fixed point approximations have been studied by Bu et al. [2]. They present formalisms to compute the fixed point for a single congested RED router with reasonable accuracy. They do not, however, have strong results for complex scenarios with both RED and drop-tail routers, and, with both long and short lived TCP flows (elephants and mice). We show that our approach allows complex mixed scenarios to be solved, approximately. Also, the formalisms used in our approach are extremely simple.

2.4 Modeling

Recent work has developed increasingly accurate analytical models for the steady state of bulk TCP [22]. Our work

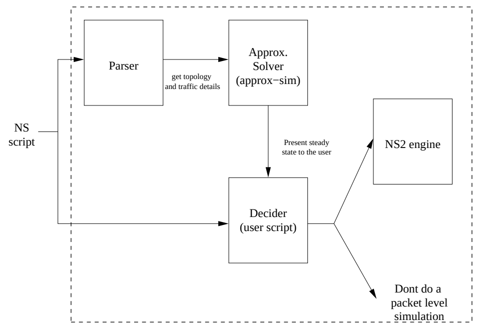
NS script
Parser
get topology and traffic details
Approx. Solver (approx−sim)
Present steady state to the user
NS2 engine
Decider (user script)
Dont do a packet level simulation

Figure 1: The structure of the pre-filtering tool

builds upon these results. Other work has modeled short TCP flows; we build upon the work there by Cardwell et al. [3] and Huang et al. [15]. More recently, Ben Fredj et al. [12] describe short flows as inelastic traffic and demonstrate that simple queuing models like $M/G/1$ are reasonably accurate for modeling drop-tail routers. We build upon their results and extend them to RED routers. Although RED has been studied in detail [10, 7, 9], and Hollot et al. presented a control theoretic model of RED [14], we believe that our work is the first to evaluate RED using a Markovian queuing model.

3 Our Approach

We designed a pre-filtering framework where we integrated a a fast approximate network simulator, a decider (to determine whether to discard a scenario) and a detailed packet-level simulator using the structure shown in Figure 1. The user feeds a regular *ns-2* script into our version of *ns-2* that has the embedded *approx-solver*, *approx-sim*. *Approx-sim* would do a fast approximate simulation of the network scenario and would present to the user the drop probabilities of the routers, the delays and the approximate aggregate throughput of the links. The user can then decide to either simulate the scenario in detail by using the packet level engine in *ns-2* or discard it.

The data structures of *approx-sim* are populated by a module within the Tcl space of *ns-2* [25]. The pre-filtering tool thus reduces to a simple Tcl script that runs the *approx-sim* module within the *ns-2* framework. The output of the *approx-sim* would be detailed steady state statistics of the network that can be accessed by the Tcl scripts. This information can be used to write simple pre-filtering tools in Tcl itself. Currently, the *approx-sim* tool is not tightly integrated into the *ns-2* framework. It runs in a stand-alone mode.

There were certain design choices that went into the design of the pre-filtering framework. Our initial approach was to build a pre-filtering tool along with *approx-sim* for the user. We wanted it to have features such as to determine whether a scenario has congestion above a threshold. That needed a careful designing of the query language that the user would use and features that we could possibly support. This approach could have limited the capabilities of the tool itself.

As a better design choice, we decided to build just the *approx-sim* and a good user interface to the data structures and the results of the simulation. The user would query the data structure and would take decisions based on the response. This approach has a couple of advantages. First of all, the user has complete flexibility to design the query that would suit his design. Also, the user would not have to learn yet another query-language. From the designers standpoint, it is redundant to redesign a query language that would imitate the functionality of a standard scripting language like Perl or Tcl, and, at the same time be different.

Another important design choice was to build the whole simulator in a *plug-and-play* fashion so that it is easily scalable and it is easy to incorporate newer models of network elements and traffic agents. Also, this ensures that integrating *approx-sim* into *ns-2* is easy. We hope that such a modular design will make the system more suitable for rapid validation of new protocols.

4 Solving for the Steady-state behavior

The approximate-solver is the heart of our pre-filtering tool. It evaluates characteristics of long-lived and short-lived TCP connections, including the throughput of the flows and the delays, drop probabilities and the aggregate throughputs at each router. We next describe how these are accomplished, quickly and approximately.

Figure 2 shows the flowchart we use to solve for the steady-state behavior of the network. We begin with a topology and the details of traffic agents in the topology from the user in the form of a *ns-2* script. Our module would parse the *ns-2* script and populate the internal data structures of *approx-sim*. The user can invoke the *approx-sim* module by a simple Tcl command from his *ns-2* script.

In step 1, the engine, in its *simulate* procedure, first calculates the drop probability and the queuing delays at each router from the previous iteration or the initial conditions (See Figure 2). In step 2, it uses these router statistics to calculate the end-to-end drop probability and delays encountered by each flow which are then used to obtain the per-flow throughput. Step 3 calculates the total throughput for each link by adding the per-flow throughputs of each flow that pass through it. Then the algorithm checks

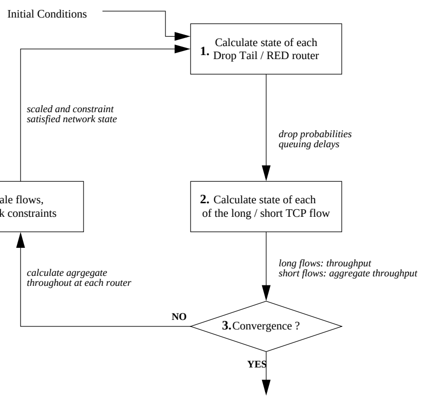
Initial Conditions
1. Calculate state of each Drop Tail / RED router
drop probabilities
queuing delays
2. Calculate state of each of the long / short TCP flow
long flows: throughput
short flows: aggregate throughput
3.Convergence ?
NO
YES
calculate agrgegate throughout at each router
4.Scale flows, check constraints
scaled and constraint satisfied network state

Figure 2: The structure of the *Approx-sim* simulator

whether convergence of a new network state is achieved. If the network state converges, we terminate. Otherwise we use a scaling algorithm in step 4 to scale the flows to meet the network constraints, and, we use the results to run another iteration of the procedure, *simulate*.

In step 1, we use simple queuing theory for drop tail routers and a new, very simple analytical model for RED [10] gateways. We describe these models below in Section 4.1. At the end of this step, we know the drop probabilities, p_i, and the average queuing delay, d_{q_i} at each router i. The state of router i is defined by $\langle \lambda_i, p_i, d_{q_i} \rangle$. The calculations in step 2 are dependent on whether the flows are short lived or not. For bulk flows, we calculate the throughput of each flow (Section 4.2.1). For short lived flows, we calculate their aggregate throughput (Section 4.2.2). An important question is how to guarantee that this process converges and leads to the steady state. We discuss this issue in Section 4.4.

4.1 Modeling network elements

Approx-sim considers models of two different packet queuing disciplines: drop-tail and RED [10]. For drop-tail routers, we use the *M/M/1/K* model [17]. Let us consider a server A processing packets from the TCP connection TCP_i. Packets arrive at the server at a particular rate (λ). The server consumes packets at the rate of μ, which is given by the bandwidth of the link. Packet losses are given by the P_K of the $M/M/1/K$ system. Hence we can plot a curve of throughput versus drop probability P_k. Call this C_{router}. Note that we could have used the more accurate $M/D/1$ or $M/D/1/K$ but that would have added to the complexity to our formalism. We have found the simple $M/M/1/K$ models to be fairly accurate as long as the network is not very heavily loaded.

RED gateways represent a common AQM strategy used in today's routers. We have developed a simple model to describe the steady state behavior of a RED router. We assume that the traffic that flows into this gateway has also reached its steady state. In Bu et al.'s work [2] the authors used the RED model directly into the TCP equations of Padhye et al. [22]. Our approach is different. We determine the drop probability and the average queue length as a function of the steady-state utilization at the router.

Without loss of generality, if the service rate of a RED router is unity, the throughput at the router, λ, will be the same as the utilization of the router queuing system ρ, i.e. $\lambda = \rho$. Let the queue length at the router be l_q and the drop probability at the router be p. The RED characteristics can be expressed by

$$p = \begin{cases} 0 & \text{if } l_q < min_{th} \\ (\frac{(l_q - min_{th}) \times p_{max}}{max_{th} - min_{th}} & \text{if } min_{th} \le l_q \le max_{th} \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

As long as the queue length is below min_{th}, the drop probability is zero. If queue length is between min_{th} and max_{th}, drop probability increases linearly between 0 and p_{max}. After this limit is crossed, drop probability becomes unity.

Lemma 4.1. *The steady state average value of the queue-length with utilization ρ and drop probability p is given by*

$$q_l = \frac{\rho(1-p)}{1-\rho(1-p)} \quad (2)$$

Lemma 4.2. *In the steady state, the average queue length will never be larger than max_{th}*

Theorem 4.3. *If the drop probability seen at this router is p and the steady state queue length is between the interval min_{th} and max_{th}, the drop probability p at this router is given by*

$$p = \frac{(\frac{\rho(1-p)}{1-\rho(1-p)} - min_{th}) \times p_{max}}{max_{th} - min_{th}} \quad (3)$$

The above theorem ensures that we can obtain a quadratic equation in p. Hence, given a value of λ (or ρ), we can find a drop probability p. Then, using the value of p, we can use Lemma 4.1 to calculate the queue length and the queuing delay. Next, we define, for every RED router, two parameters ρ_{min} and ρ_{max}. ρ_{min} is the solution to the equation 2 and corresponds to the throughput that causes the buffer length of the RED router to be min_{th} and the drop probability to be 0. Similarly, we can define ρ_{max} to be the throughput that causes the buffer length of the RED router to be max_{th} and the drop probability to be p_{max}. By Theorem 4.3 and the above definitions of ρ_{max} and ρ_{min}, we can easily calculate the queue length and the drop probability in the following way: If the link throughput is less than ρ_{min}, the drop probability is 0 and the queue length is given by the $M/M/1/K$ model. Similarly, when the throughput is greater than ρ_{max}, the queue length is exactly max_{th} and the drop probability can be calculated from Theorem 4.3. Note that we cannot use Equation 1 to obtain the drop probability because we are interested in the average drop probability and not the instantaneous one. Instead, we are try to find the average state. If $\rho_{min} < \rho < \rho_{max}$, the drop probability can be computed using Theorem 4.3. We should note that we can adapt the above analysis to obtain an iterative method in order to calculate the state of RED routers that use more complex variations like Gentle RED [8].

4.2 Modeling the flows

In our work, we assume that all flows are TCP flows. This is reasonable because TCP traffic is known to account for the bulk of the traffic in the Internet. In this section, we refer to the analytical models that have been used in our *approx-sim*.

4.2.1 Modeling elephants

Padhye et al. [22] gave the throughput of bulk TCP flows ($B(p, RTT)$), or *elephants*), as a function of the probability of a loss event (p) and the round trip time RTT as

$$B(p, RTT) = k \times \frac{1}{RTT\sqrt{p} + RTO \times O(p^2)} \quad (4)$$

where k is a constant and RTO is the maximum value of the timeout.

4.2.2 Modeling mice

Short lived TCP flows, or *mice*, have been extensively studied in [3] and [15]. These efforts have focused on finding very detailed models to accurately depict the behavior of individual short term flows. In contrast, we concentrated on a much simpler model for aggregates of short term flows. We have refined Ben Fredj et al. [12] model of *mice* to incorporate the drop probabilities. They validated their work on simple topologies while we validated their model (and our refinement) on more general topologies.

We approximate aggregates of short flow between the same source-destination pair as a smooth fluid. The rationale for this kind of idea is that aggregates being *inelastic traffic* will have less correlation to the complex feedback mechanism and will be easier to model at a higher level. It is also much faster to determine the behavior of the aggregate.

Lemma 4.4. *If the rate of arrival of short lived flows between any source-destination pair is $\lambda_{arrival}$ and the data transferred by the flow is σ, then the rate of short flow traffic between the same source-destination pair is given by*

$$\lambda_{mice} = \lambda_{arrival} \times \sigma \quad (5)$$

For now, we assume that short TCP connections come according to some arrival pattern that is Markovian and each connection transfers a constant amount of data.

Lemma 4.5. *Let the end-to-end drop probability between any source-destination pair be p and the arrival rate for short flows be λ_{mice}. The throughput of the mice is given by*

$$B_{mice} = \frac{\lambda_{mice}}{1-p}, \textit{ where p=drop probability} \quad (6)$$

Proof. The total short lived traffic that arrives at a router is given by Lemma 4.4. Now, if drop probability is p, the traffic generated by short flow request in one second is given by

$$B_{mice} = \lambda_{mice} \times (1 + p + p^2 + p^3 + ...) \quad (7)$$

Hence the lemma. □

Note that long lived bulk TCP traffic is said to be *elastic* since the closed loop congestion control algorithms can adjust the sending window and utilize the available bandwidth. In contrast, short lived TCP flows can be considered to be inelastic. The dynamics of the network will be heavily influenced by these mice. This is also emphasized in [12]. The intuitive idea is to assume that the long lived flows in presence of these short flows do not contribute much to increasing the *load* on the network. Hence we can calculate the short flows throughput first and use the remaining bandwidth for the long lived flows.

4.3 Does a fixed point exist ?

It is not clear whether a fixed point exists between the router characteristics and the flow characteristics in all network scenarios. Bu et al. [2] prove that it does for a single congested link. They do not have conclusive results for complex networks. We use an iterative scheme and we have found the *approximate fixed point* for all our experiments because we relaxed the conditions of convergence to include errors. Next, we present an alternate proof of the existence of the fixed point with drop-tail routers. Bu et al. [2] prove the same facts for a AQM router like RED.

Let us assume we have a single congested link. We use only long-lived TCP flows (elephants) in our subsequent proofs and arguments. Consider the throughput characteristic of each TCP flow, $B(p, RTT)$. We call this curve $C_{elephants}$. It is natural to question the existence of a fixed point as the throughput $B(p, RTT)$ is a surface. We now show that a fixed point indeed exists.

Lemma 4.6. *When the probability of loss (p) or the RTT increases, the throughput of bulk TCP flows on a single link $B(p, RTT)$ decreases.*

Proof. Follows from Equation 4. □

Lemma 4.7. *When the aggregate throughput on a link increases, both p and RTT increase.*

Proof. Increasing the throughput implies a higher rate of arrival into the queue. That results in higher queuing delays and increases the RTT and the drop probability p. □

Theorem 4.8. *The intersection of the curve C_{router} of the router with $C_{elephants}$ has a unique fixed point.*

Proof. Assume the contrary. Since our curve C_{router} assumes a fixed RTT, it is a straight line. It must intersect the curve $C_{elephants}$ at least once. Assume that it intersects at two points p_1 and p_2. Lemmas 4.6, 4.7 indicate the monotonicity of TCP performance. Assume $p_1 < p_2$. By Lemma 4.7, drop probabilities increase as we increase p from p_1 to p_2. But, by Lemma 4.6, as drop probabilities increase throughput should increase. Thus there is a contradiction. Hence the theorem. □

Note that we cannot say that the above holds for a network with many congested links.

Now we discuss the exact procedure to calculate the fixed point of each TCP flow in the network. Let there be a network with M edges/links and N TCP connections. Let each connection be C_i and let each link be denoted as l_i. Assume that connection i passes through M_i edges and $E(i)$ denotes the edge set of this connection. Let the delays of each link be d_i, the drop probability be p_i. Also if we assume that the i^{th} connection goes through the j^{th} link, it will occur a drop of P_{ij} and a delay a of D_{ij} and this link is denoted by L_{ij}. Now P_{ij} is equal to p_k, D_{ij} is equal to d_k for some link l_k.

Intuitively, the procedure is as follows: we use the appropriate model at each router to calculate the link delays and drops experienced at each router by all the flows going through that link. These link characteristics are used to estimate the end-to-end round trip time and the drop probabilities seen by each flow at the sender. We can then use either Equation 4 or Lemma 4.5 to calculate the throughput of the TCP flows. Hence we have

$$\exists k : P_{ij} = p_k \; and \; D_{ij} = d_k \tag{8}$$

For example, L_ij is the j^{th} link in the path of connection i, and, hence, it can be denoted by l_k for some k. Now we can write D_{ij} as

$$D_{ij} = prop_k + q_k \text{for some } k \tag{9}$$

where q_k denotes the queuing delay of the link l_k and $prop_k$ is the propagation delay of the link l_k. The RTT seen from the end-point, i.e. the sender, for a connection i is denoted by RTT_i and is given by

$$RTT_i = 2 \times \sum_{j=0}^{j<M_i} D_{ij} \tag{10}$$

where M_i is the number of links traversed by connection i.

If we make the assumption that packet losses are independent on each link, the following theorem is obvious

Lemma 4.9. *The drop probability seen by the connection i, $DROP_i$ is given by*

$$DROP_i = 1 - \prod_{j=0}^{j<M_i} (1 - P_{ij}) \tag{11}$$

Hence, the throughput of a TCP flow C_i, B_i can now be calculated by

$$T_i = B(DROP_i, RTT_i) \tag{12}$$

4.4 Initial conditions and convergence

It is trivial to design an algorithm to calculate the approximate steady state throughput from the discussion in the preceding sub-section. Thus the algorithm is shown in Figure 3.

The above algorithm requires us to start with the correct initial values of λ_i for each link L_i. But, we do not want to make any assumptions apriori on the state of the network. Without such a restriction we can always solve the network in the following fashion: Run *ns-2* for a few seconds in virtual time. The throughput of each link will give us the initial λs for *approx-sim*. A more elegant solution is not to use any prior knowledge of the intermediate *ns-2* results. This is our approach.

4.4.1 Our approach to convergence

Initially we assume that the links are not loaded when there are only bulk flows or elephants present. We argue that the load due to the *elastic* flows is such that they will share all the available bandwidth. In this first iteration of our fixed point algorithm, the bulk TCP flows get what Equation 4 with low drop probabilities. That may result in window limited or large throughput. Now we run the algorithm and

```
set initial conditions
while(convergence not reached) do
        if (not initial) then scale connections
        for i = 1 to nLinks do
                calculate the queueing delays and drop probabilities
        endfor
        for j = i to nConnections do
                sum the drops and delays from the link list
                       of edge for connection i
                calculate throughput from the TCP equations
        endfor
        for i = 1 to nLinks do
                calculate the total throughput of each link
        endfor
endwhile
```

Figure 3: Aggregation algorithm after every round of fixed point

compute the new throughput of each connection and sum them up to find the new λs of each of the links. This gives us the initial throughput.

The throughput of a bulk connection (elephant) is very sensitive to small changes in probability, which makes it hard to achieve convergence using the iterative process described earlier. Specifically, if the drop probability is very low, then the computed throughput of the bulk connections on a link can be much higher than the capacity of the link. To speed up convergence, we scale down the computed throughput of bulk connections so that link capacities are not exceeded. A brief description of the scaling algorithm is given below.

The scaling algorithm: Let $\lambda_1, \ldots, \lambda_n$ represent the computed throughput of the n bulk flows after each iteration of the fixed point algorithm. Initially, we mark each bulk flow as being *unscaled*. For each link l define C_l, the *unscaled capacity*, as the capacity of the link minus the throughput of all the short flows (mice) on this link. Also, for each link l, define X_l to be the combined throughput of all the *unscaled* bulk flows on the link. Define the congestion γ_l as X_l/C_l. Now, we repeat the following process. while there exists a link l with $\gamma_l > 1$: Let l denote the link with the largest value of γ_l. Scale down the throughput of all the unscaled bulk flows using this link by a factor γ_l, and mark all these flows as being *scaled*. Now the total throughput of this link exactly matches the capacity of the link and hence $\gamma_l = 1$. For each newly scaled flow i, and each link $k \neq l$ such that flow i uses link k, we reduce the unscaled capacity C_k of link k by the new throughput of flow i and the combined throughput X_k of link k by the old unscaled throughput of flow i. .

When the above algorithm terminates, the throughput on any link does not exceed its capacity. In practice, we found the scaling step to be critical for fast convergence. We call this step *Link capping*. This step ensures that a particular link is put back into a stable state before the averaging process in the convergence algorithm discussed in the previous subsection.

The performance of the scaling algorithm is given by the following theorem:

Theorem 4.10. *The worst case running time, T, of the scaling algorithm on a network of size N connections, M links is given*

$$T(M, N) \; = \; \theta(N \log M + N.H) \tag{13}$$

where H is the average number of links traversed by each connection

Proof. Finding the most congested link takes $\theta(\log M)$ time with suitable data structures. When we scale each connection i, we need to change the unscaled capacity of M_i links. This takes time $M_i \log M$ with suitable data structures. $N \log M + \sum_{i=1}^{N} M_i$ where M_i is the number of links the connection i traverses. Hence the theorem. □

5 Evaluation and Results

We next evaluate how well *approx-sim* meets its three goals: speed, accuracy, and generality. First, we consider its performance relative to packet-level simulation. Second, we show that it is reasonably accurate, typically within 10–15% of packet-level simulation for the scenarios we consider. Only some scenarios were 20% accurate but they were under very heavy load. A very high level of accuracy is not required for *approx-sim* because we expect final simulation results to be done with packet-level simulation; *approx-sim* merely selects those scenarios. Finally,

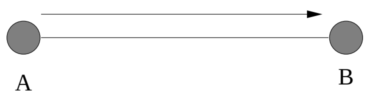

Figure 4: The line topology

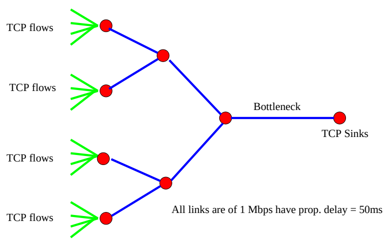

Figure 5: The symmetric tree topology: a sample binary tree of height 2 with four clients at each leaf.

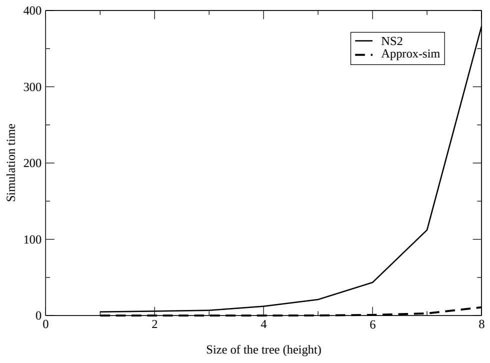

Figure 6: Running time comparisons between *approx-sim* and *ns-2* for the symmetric tree topology with elephant traffic only

we evaluate the generality of *approx-sim* by showing that it is applicable to increasingly complex scenarios in terms of traffic mix, topology and network elements.

In this entire section, we use a particular terminology. Long lived flows and *elephants* are used interchangeably. Similarly we refer to short lived TCP flows as *mice*. For throughput, units of packets/s and kB/s are used interchangeably since all our simulations use a packet size of 1kB. We start with simple topologies topologies (lines and symmetric trees) and move to more complex topologies (asymmetric trees and circular topologies) to validate *approx-sim* progressively.

5.1 Elephant traffic alone

First we consider results that we obtained for the experiments with elephant-only traffic. We evaluated *approx-sim* on the line topology (Figure 4) as well as symmetric (Figure 5) and asymmetric trees (Figure 8). This gradual increase in the complexity of the topologies will help us to evaluate *approx-sim* with just bulk flows.

The line topology shows good accuracy between *ns-2* and *approx-sim* so we jump directly to symmetric trees. Symmetric trees were initially chosen because it allows us to study the effect of many similar flows passing through a bottleneck link. Figure 5 shows the symmetric tree topology. We place the TCP sinks at the bottleneck link i.e. at the root of the tree, and four bulk TCP sources at each of the leaves of the tree. All the links are assumed to have a capacity of 1Mb/s.

Figure 6 shows the run-time performance of *approx-sim* compared to packet-level simulation with *ns-2* for symmetric trees as a function of tree height. *Approx-sim* is 10-300$\times$ faster than packet-level simulation. Although the performance of both *approx-sim* and *ns-2* is linear with network size (and increases exponentially as a function of tree height), the very large difference in constant factor makes *approx-sim* one to two orders of magnitude faster than packet-level simulation.

Speed is not useful if the simulation is completely inaccurate. Figure 7 compares *approx-sim* and *ns-2* accuracy by evaluating mean flow bandwidth for the bottleneck link. (No error bars are shown in this case because *approx-sim* is deterministic and the standard deviation between the *ns-2* flows is less than 5%.) This graph shows that *approx-sim* is quite accurate compared to *ns-2*. The simulators are typically with 10–20%; the worst case is with a hight of 8 when the network is very heavily loaded where they are 40% apart. *approx-sim* is more accurate when we look at aggregates of many flows. The accuracy is much higher for the links close to the root. At the root bottleneck link, the accuracy was 7.6%. We have also conducted experiments for high link capacities and the results have been better with less utilization.

Next we consider asymmetric trees (as shown in Figure 8) to avoid biases in evaluation due to symmetry. We examine asymmetric trees of varying heights. Figure 8 shows a tree with height two. In general, we construct an asymmetric tree of height h by expanding the leftmost node of a tree of height $h-1$ to have two children. All traffic terminates at the lower-left-most node of the tree; traffic begins at all the other leaves of the tree with 16 elephants.

Early comparisons of results for asymmetric trees show large differences between *approx-sim* and *ns-2*. In *ns-2* all long RTT flows (eg. between nodes 2, 3, Figure 8) had very low throughput while short RTT flows (eg. between

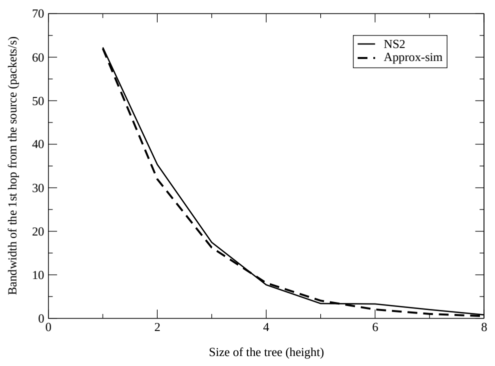

Figure 7: Accuracy of *approx-sim* throughput compared to *ns-2* for the symmetric tree topology with elephant traffic only

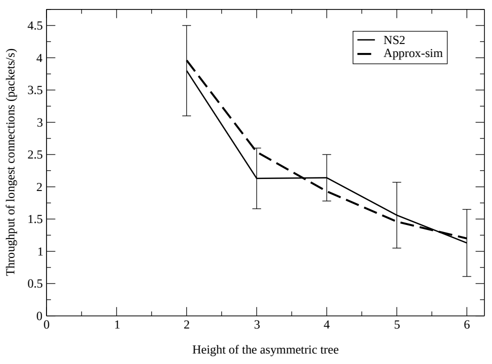

Figure 9: Comparison between *approx-sim* and *ns-2*: throughput of the longest TCP connections in several asymmetric trees - all links have a capacity of 1MB/s

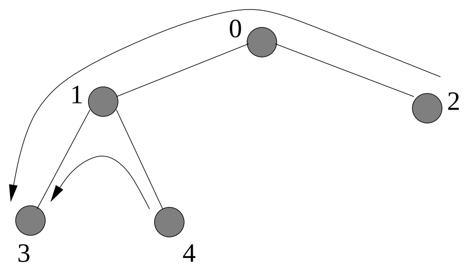

Figure 8: The asymmetric tree topology

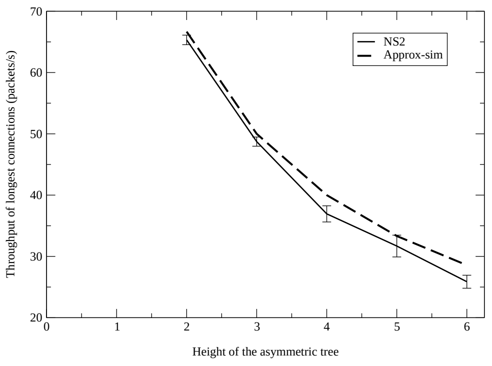

Figure 10: Comparison between *approx-sim* and *ns-2*: throughput of the longest TCP connections in several asymmetric trees - all links have a capacity of 45MB/s

4, 3 in Figure 8) had high throughput. Although it is well known that TCP is unfair to flows with different RTT and Equation 4 predicts a throughput ratio of $2 : 3$ between the short RTT and long RTT flows respectively (assuming no queuing delay), we observed a ratio of more than $40 : 1$.

We believe that this disparity occurs due to synchronization in *ns-2*. Because packet-level simulators are purposefully deterministic, packets from different senders can arrive at queues at exactly the same virtual time, and the flows can remain synchronized because there is no variation in the simulated environment. In real-world experiments, inevitable timing variations prevent consistent, fine-grained synchronization.[1] This problem with packet-level simulation has been recognized, both at small scale where the *ns-2* TCP model includes optional "jitter", and at larger scales where researchers add a small amount of additional background traffic to the simulation to de-synchronize flows [12].

Since *approx-sim* predicts the steady-state behavior, it is immune to such artificial synchronization. To avoid synchronization in our *ns-2* simulations, we introduced a small amount of background traffic. For the asymmetric tree, we filled 10% of the bottleneck link bandwidth with randomly generated web-like traffic. For our elephant-only experiments *approx-sim* does not have this traffic, therefore we expect it to slightly overestimate performance. An interesting fact is that flows in *approx-sim* can never get synchronized unlike in *ns-2*. Hence, engines like our *approx-sim* could be useful to to get an alternate opinion of a large class of scenarios.

Figure 9 compares *approx-sim* to *ns-2* with this background traffic as the height of the tree varies. We see that the results of *approx-sim* are accurate within 20% of the *ns-2* results. This is good accuracy given that the network is very heavily loaded and *approx-sim*'s approximations are least accurate under heavy load. We expect *approx-sim*

[1] Although at coarse scales, some protocol synchronization has been observed [11].

to be more accurate when the network is less loaded. We therefore also considered the same scenario with 45Mb/s-bandwidth links (See Figure 10). This graph shows that in less loaded networks *approx-sim* is even closer to *ns-2*, within 2-10%. Again, *approx-sim* underestimates bandwidths compared to *ns-2* because it does not consider background traffic.

5.2 Mixed mice and elephants

In this section, we evaluate *approx-sim* with a mix of traffic sources i.e. with both mice and elephant traffic. This is crucial because Internet traffic consists of both short and long lived flows. Like the previous subsection, we evaluate *approx-sim* by gradually increase the complexity in topology.

5.2.1 Line topology

We start our experiments with the simple line topology because it is easy to hand-verify our results. Consider a line topology with two nodes A and B as in Figure 4. We vary the link bandwidth C, the mean arrival rate (exponential arrivals) and length of short flows (λ per second and σ kB/s), and the number of long flows E.

First, we observe that both *approx-sim* and *ns-2* get very similar values for aggregate throughput of mice (within 10%). Both simulators predict similar values for elephants as well (within 8.3%). From now, we will focus more on the accuracy of the elephant-flows since in the scenarios we consider, the load of the mice is small compared to the elephants. Finally, these values also match hand calculations as well.

5.2.2 Symmetric Trees

We now move on to validate *approx-sim* on a more complex topology. We choose symmetric trees as they ensure aggregation in the network. Further, these topologies are very simple for hand-verification too. Consider a line topology with two nodes A and B as in Figure 5. We vary the link bandwidth C, the mean arrival rate (exponential arrivals) and length of short flows (λ per second and σ kB/s), and the number of long flows E. The results for this experiment are shown in Figure 12.

We observe that as we increase the the link bandwidth from 1MB/s to 2MB/s, the accuracy of *approx-sim* drops from 5% to 27%. The main reason for this drop is that traffic at the bottleneck link increases due to aggregation and *approx-sim* bounds the maximum throughput to be $(1 + p)C$KB, C is the link capacity and p, the drop probability on that link. But the end-to-end drop probability may be greater than p. But, when we decrease the amount of mice (or the inelastic traffic), there is a higher correlation between the values obtained from *ns-2* and *approx-sim*. Hence *approx-sim* is more accurate with light load.

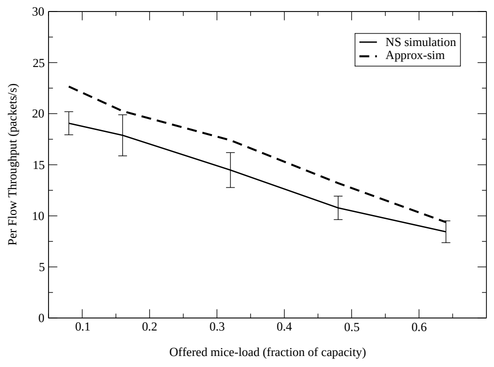

Figure 13: Comparisons between *approx-sim* and *ns-2*: bandwidth achieved by the flows having longer RTT i.e. around 300ms

5.2.3 Asymmetric Trees

Now we compare results for asymmetric trees to avoid symmetry. Figure 13 shows the bandwidth of bulk TCP flows between nodes 2 and 3 (the longest path). We observe that *approx-sim*'s predictions are close to what *ns-2* outputs with an accuracy that varied from 13 to 16%. Figure 14 shows bulk TCP flows between nodes 3 and 4, the short RTT path. Again, we see that *approx-sim* results are similar to those in Fig 13. The accuracy of *approx-sim* varied from 13 to 17% for Fig. 14 and from 6-20% in Fig. 15.

Comparing Figures 13, 14, 15, we observe that for long RTTs *approx-sim* estimates larger throughput than *ns-2* while for shorter RTTs its estimate is lower. For aggregate throughput of short flows, the *ns-2* results are typically 7–10% higher than the those predicted by *approx-sim*. We believe that this difference is related to synchronization in *ns-2* (as described in Section 5.1). Mice provide some level of desynchronization, but some difference between *approx-sim* and *ns-2* remains. We plan to investigate this hypothesis further.

To consider cases with lower load, we also examined scenarios with link bandwidths of 10 and 100Mb/s. We do not report detailed results here due to space constraints, but we observed higher accuracies at lower utilizations as in the all-elephant case (Section 5.1).

Since *approx-sim* and *ns-2* results are quite similar, these experiments suggest that *approx-sim*'s model is appropriate: one can model short flows as inelastic and bulk flows as ”filling out” the rest of the traffic, at least for the traffic loads we consider.

C (MB/s)	# of elephants	# of Mice λ(/s), σ (kB/s)	Elephants *approx-sim* (kB/s)	Elephants *ns-2*, $(std.dev.)$ (kB/s), (kB/s)	Mice *approx-sim* (kB/s)	Mice *ns-2* (kB/s)
1	4	2, 10	29	27 ($<$ 1)	20	22
1	4	2, 20	24	22 ($<$ 1)	40	37
10	4	2, 20	200	199 ($<$ 1)	40	40
45	16	2, 20	200	197 (1.1)	40	40

Figure 11: Comparison of results with drop-tail routers on a Line topology as shown in Figure 4

Height of tree	C (MB/s)	# of elephants per leaf	# of Mice λ(conn/s), σkB/s	Elephants *approx-sim* (kB/s)	Elephants *ns-2* (kB/s), $std.dev.$(kB/s)
1	1	4	2, 10	11	11.653 (1.44)
2	1	4	2, 5	6.01	6.3 ($<$ 1)
2	1	4	2, 10	3.25	4.25, ($<$ 1)

Figure 12: Comparison of results with drop-tail routers on a symmetric tree topology as shown in Figure 5

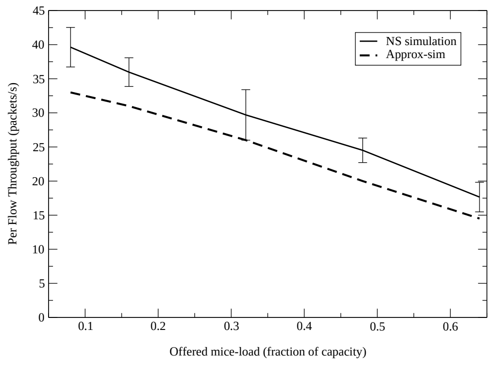

Figure 14: Comparisons between *approx-sim* and *ns-2*: bandwidth achieved by the flows having short RTT i.e. around 200ms

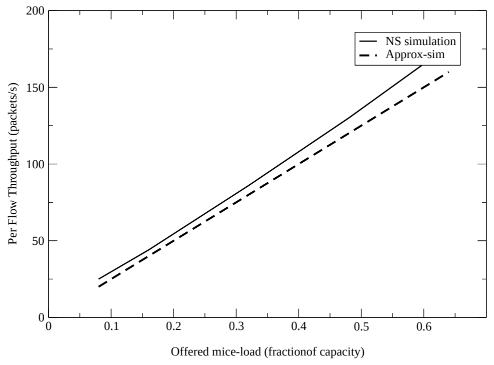

Figure 15: Comparisons between *approx-sim* and *ns-2*: Aggregate short flow throughput

5.2.4 Circular topologies

We next considered the ring topology shown in Figure 16. Between each alternate node (eg. between A, C), we vary the link bandwidth C, the mean arrival rate (exponential arrivals) and length of short flows (λ per second and σ kB/s), and the number of long flows E. Since there is overlapping traffic, we believe that this scenario provides a more difficult case for convergence in *approx-sim*.

Figure 17 presents the throughput of short and long flows between nodes A and C. Again, we observe a good match between *approx-sim* and *ns-2*, with the bulk flows within 6.1%. More importantly, even with this circular topology *approx-sim* converges within 5 iterations.

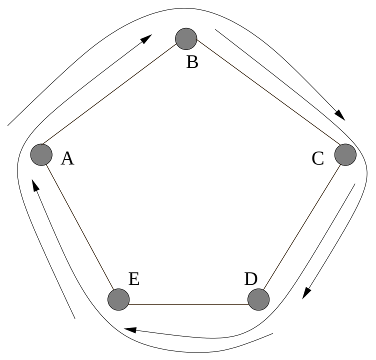

Figure 16: The ring topology

5.3 Experiments with RED

Finally we evaluate routers with RED queuing policies. We have examined some scenarios of each of the topologies (line, symmetric and asymmetric tree, and the circle) with RED, but here we summarize only the line and circle topologies. In each topology we consider RED routers with the parameters $(min_{th}, max_{th}, p_{max}) = (5, 15, 0.1)$ with the thresholds in 1KB packets. We make no claims

C	# of	# of Mice	Elephants	
(MB/s)	elephants	λ(conn/s), σkB/s	*approx-sim* (kB/s)	*ns-2* (kB/s), $std.dev.$(kB/s)
45	16	20, 10	100	98.13 (< 1)
45	16	20, 20	100	97.98 (< 1)
45	16	20, 40	100	97.57 (1.07)
45	32	20, 10	86.1	81.79 (6.06)
45	32	20, 20	79.86	78 (9.34)
45	32	20, 30	73.61	73.68 (12.87)

Figure 17: Comparison of results with drop-tail routers on a circular topology as shown in Figure 16

about these parameters being ideal (in fact, there is some evidence that it is quite difficult to "tune" RED [5]), they are merely the defaults in our simulator.

If we look at Figure 18, we see that with light load, *approx-sim* is again very accurate while accuracy decreases with load. This further justifies our claim of *approx-sim* being suitable for approximate pre-filtering. Although our preliminary evaluation of *approx-sim* with RED routers is promising, a more thorough examination is needed and in progress.

6 Conclusion

Our method solves for the approximate operating point of many TCP flows using analytical techniques. To achieve this, we use a combination of existing and new models for network elements and TCP flows coupled with an approximate fixed point iteration algorithm.

Our work led us to some nice observations about modeling and simulation of networks. S. Ben Fredj et al. [12] claim that for a single congested link with a drop-tail router, short flows (mice) add to the load and that the elephants adjust according to the available bandwidth. Our results show that these results are true even in complicated networks and also in the presence of RED gateways.

A very important observation is that scenarios with TCP flows in packet level simulators (such as *ns-2*) can easily be dragged into synchronizations. Such phenomenon is very misleading and can give us unrealistic results. One must take adequate care and interpret these simulation results. Tools such as *approx-sim* can used to identify scenarios that are prone to such phenomenon. If the results of *approx-sim* are fairly close to the *ns-2* results, we can be confident that such synchronizations were not seen in the *ns-2* simulations. Also, one can remove such synchronization by adding some background random traffic. Another approach would be to add short term TCP flows between each source destination pair.

Since *approx-sim* is fast and scalable, we feel that it may be used for a wide variety of applications other than filtering of scenarios. One possible application is to help in converging on a correct SLA between 2 network providers.

There is a lot of work that needs to done. One direct extension would be get a more accurate model of the network elements and flows without compromising on the simplicity. Then we need to ensure that our *approx-sim* results are accurate for networks with thousands of nodes. Also, the integration of our *approx-sim* engine with *ns-2* is under development (currently, we have a simple module in *ns-2* that outputs a description that *approx-sim* can read and populate its data structures).

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H. Y. Song. PARSEC: A parallel simulation environment for complex systems. *IEEE Computer*, 31(10):77–85, October 1998.

[2] T. Bu and D. Towsley. Fixed point approximation for TCP behavior in an AQM network. In *Proceedings of the ACM SIGMETRICS*, June 2001.

[3] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *Proceedings of the IEEE Infocom*, page to appear, Tel-Aviv, Israel, March 2000. IEEE.

[4] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–205, April 1981.

[5] 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, September 2000. ACM.

[6] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the global internet. *Computing in Science & Engineering*, pages 30–36, January 1999.

[7] S. Floyd. RED: Discussions of setting parameters, 1997.

[8] S. Floyd. Recommendation on using the gentle variant of RED, http://www.aciri.org/floyd/red.html, 1999.

[9] S. Floyd. RED (random early detection). http://www.aciri.org/floyd/red.html, 2001.

Topology, C (MB/s)	# of elephants	# of Mice λ(conn/s), σkB/s	Elephants *approx-sim* (kB/s)	*ns-2* (kB/s), $std.dev.$(kB/s)
Line, 1	4	2, 10	23.56	22 (< 1)
Line, 45	4	2, 10	200	199.9 (< 1)
Line, 45	16	2, 10	200	199 (< 1)
Circle, 45	16	20, 10	100	97 (< 1)
Circle, 45	32	20, 10	81	68.3 (5.48)

Figure 18: Comparison of results with RED routers

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

[11] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *ACM/IEEE Transactions on Networking*, 2(2):122–136, April 1994.

[12] S. Ben Fredj, T. Bonald, A. Proutiere, G. Rgni, and J. W. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. In *Proceedings of the ACM SIGCOMM*, pages 111–122. ACM, 2001.

[13] M. H. Ammar G. F. Riley, R. M. Fujimoto. A generic framework for parallelization of network simulations. In *Proceedings of the Seventh International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, (MASCOTS) College Park, MD. October 1999*, October 1999.

[14] C. Hollot, V. Misra, D. Towsley, and W. Gong. A control theoretic analysis of RED. In *Proceedings of the 2001 IEEE Infocom*, April 2001.

[15] P. Huang and J. Heidemann. Capturing TCP burstiness in light-weight simulations. In *Proceedings of the SCS Conference on Communication Networks and Distributed Systems Modeling and Simulation*, pages 90–96, Phoenix, Arizona, USA, January 2001. USC/Information Sciences Institute, Society for Computer Simulation.

[16] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[17] L. Kleinrock. *Queueing Theory Volume 1*. John Wiley & Sons, Inc., 1967.

[18] L. Kleinrock. *Queueing Theory Volume 2*. John Wiley & Sons, Inc., 1967.

[19] V. Misra., W. Gong, and D. F. Towsley. Stochastic differential modellingand analysis of TCP window size behavior. In *ECE-TR-CCS-99-10-01*, October 1999.

[20] V. Misra, W. Gong, and D. F. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *SIGCOMM*, pages 151–160, 2000.

[21] D. Nicol, M. Goldsby, and M. Johnson. Fluid-based simulation of communication networks using SSF. In *Proceedings of the European Simulation Symposium*, Erlangen-Nuremberg, Germany, October 1999.

[22] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM*, Vancouver, Canada, September 1998. ACM.

[23] J. Postel. Transmission control protocol. Internet Request for Comments RFC 793, September 1981.

[24] G. Riley, M. Ammar, R. Fujimoto, D. Xu, and K. Perumalla. Distributed network simulations using the dynamic simulation backplane, 2001.

[25] UCB/LBNL/VINT. The NS2 network simulator, available at http://www.isi.edu/nsnam/ns/.

[26] T. Ye and S. Kalyanaraman. An adaptive random search alogrithm for optimizing network protocol parameters. Technical report, Rensselaer Polytechnic Institute Computer Science Department, June 2001.