Faster Network Design with Scenario Pre-filtering *

Debojyoti Dutta
Department of Computer Science
University of Southern California
941 W. 37th Place Los Angeles, CA 90089
ddutta@isi.edu

Ashish Goel
Department of Computer Science
University of Southern California
941 W. 37th Place Los Angeles, CA 90089
agoel@pollux.usc.edu

John Heidemann
Information Sciences Institute
4676 Admiralty Way Marina Del Rey, CA 90292
johnh@isi.edu

ABSTRACT

The 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.

Keywords

network simulation, scenario pre-filtering, fixed-point approximation, TCP, RED

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 explored and understood. Network simulation can play an important role in this understanding and in the design of better networks. 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* [28], simulate the network as a series of discrete events, requiring a number of events proportional to the number of packets generated by the network. 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 may require many scenarios with alternate traffic or configuration choices. Often many of these scenarios are not interesting, either they are very overloaded (and so not a sensible operating point), or they are very underloaded (and so not providing insight into the network's performance). Often only a few scenarios are really relevant, providing a balance to define the operating limits of the network. Another fact about simulations with protocols such as TCP [25] 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 problem of how to evaluate a wide range of simulation scenarios when only some are relevant. 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 using analytic means. It employs a user-supplied criteria to *pre-filter* the scenarios, allows uninteresting scenarios to be dismissed quickly (with only analytic evaluation) and the interesting ones to be evaluated in detail (with packet-level simulation).

1.1 Our Contribution

Our approach to building the core of a pre-filtering tool is to solve the network analytically, using a framework based on fixed point approximations. We have designed and implemented a *stand-alone framework to calculate the approximate fixed-point of the network*. Although others have described this general approach [2, 15, 27, 7] (for more details, see the next sub-section), our contribution has been to realize a stand-alone solver that can also been integrated

*The authors are at USC/ISI. This work is supported, in part, by DARPA and the Space and Naval Warfare Systems Center San Diego (SPAWAR) under Contract No. N66001-00-C-8066.

into a packet level simulator. In order to build our tool, *approx-sim*, we have had to develop the following models and techniques.

- In order for the fixed point solver to work in a unified way for different protocols, we need simple models of routers that yield the drop probability as a function of the offered load. We provide such a model for RED gateways; our model is simple and may well be of independent interest. Though other models yield the drop probability as a function of buffer size, they are difficult to incorporate into a modular framework that can solve for both drop-tail and RED routers. To our knowledge, no one has validated their framework for scenarios with both RED as well as drop-tail routers.

- We have developed a new algorithm to promote fast convergence. To promote convergence we apply network-specific knowledge about link scaling, damping oscillations that occur in determining the operating points for drop-tail routers.

We have *integrated* our tool, *approx-sim*, into a packet-level simulator, *ns-2*, and have applied it the problem of *scenario pre-filtering*.

An important additional distinction between *approx-sim* and previous fixed-point models is the expected level of accuracy. Operating alone, fixed-point, steady-state computations of network behavior may suffer from accuracy problems. When applied to pre-filtering and integrated into a packet-level simulator, perfect accuracy is no longer a goal: analysis is used only to select which scenarios are relevant. These scenarios are then simulated with packet-level approaches to get detailed results. Although scenario pre-filtering is our primary goal, *approx-sim* can also run as a stand-alone simulator.

1.2 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.

Parallelism has been used for many years to improve simulation performance [4, 18]. Several parallel network simulators are currently available, such as Parsec [1], SSFNET [6], and parallel versions of *ns-2* [14, 26]. RPI [29] 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. Queuing theoretic approaches have long been used to evaluate network performance (for example, [20]). 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. Recently, there has been extensive work in fluid-flow-based approaches to network simulation [21, 22, 23]. These approaches are promising, and, some such as Misra et al.'s [22] approach can capture the transient behavior.

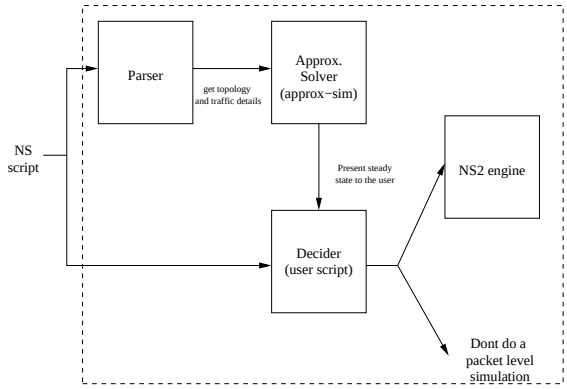

Figure 1: The structure of the pre-filtering tool

Accurate analytical models for the steady state of bulk TCP [24] exist. Short lived TCP flows have been modeled by Cardwell et al. [3] and Huang et al. [17]. Ben Fredj et al. [13] describe short flows as inelastic traffic and demonstrate that simple queuing models like $M/G/1$ are reasonably accurate for modeling drop-tail routers. RED has been studied in detail [11, 8, 10]. Hollot et al. presented a control theoretic model of RED [16].

Several frameworks based on fixed point approximations have been studied by several groups. In [2], Bu et. al. compute the fixed point for a single congested RED router with reasonable accuracy but they do not have complex scenarios with both RED and drop-tail routers, and, with both long and short lived TCP flows. Gibbens et. al. [15] define a theoretical framework for long term TCP flows. In Roughan et. al. [27], the framework cannot handle short flows and drop-tail routers. In [7], the authors present a similar analytical framework such as ours. Their approach works if the scenario can be reduced to just one bottleneck link and their results for multiple bottleneck link network is validated for only long term TCP and UDP traffic. In general, other researcher have suggested similar frameworks. However none of them have validated their framework for complex scenarios with both RED and drop-tail routers, and, with both long and short lived TCP flows.

In the rest of this paper, we will first describe our iterative process, and explain why it requires models of network routers which give drop probability as a function of throughput. We then present our simple model of RED, and our iterative fixed point module, with special attention paid to the damping step for convergence. Finally, we present simulation results that show that our tool is very fast, has reasonable accuracy and it can be used within *ns-2* to pre-filter uninteresting scenarios based on user criteria.

2. STRUCTURE OF APPROX-SIM

This section first describes the user interface of *approx-sim* and how it can be used to pre-filter scenarios. Then we briefly describe the analytical engine of *approx-sim* along with the analytical models, algorithms and techniques used. This section describes our main contribution.

2.1 The User Interface

To rapidly calculate the steady state of the network, we use a fixed point approximation technique. The structure of the tool is shown in Figure 1. 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 parses the *ns-2* script and populates the internal data structures of *approx-sim*. The user can invoke the *approx-sim* module by the following simple Tcl command from his *ns-2* script. If we use the stand alone version of *approx-sim*, we need to run *ns-2* and instruct it to dump the internal data structure of *approx-sim* corresponding to that scenario into a file that can be read by *approx-sim*.

Let us look at the user interface in more detail. In this subsection we will describe how the user can use *approx-sim* to pre-filter undesired scenarios. To demonstrate how the embedded version works, consider the following script. This script shows the ease with which a pre-filtering tool can be constructed with our *approx-sim* tool. Note that we need to add only a few lines to an existing *ns-2* script to make the tool. Also our tool is easily customizable.

```
$ns useasim_
# N=no of nodes
for {set i 0} {$i<$N} {incr i} {
   set node_($i) [$ns node]
}
addlink $node_(2) $node_(0) 1Mbps 10ms
addlink $node_(3) $node_(1) 1Mbps 10ms
..........
# define other topological info and traffic
$ns asim-run  # start analytic engine, find steady state
# determine whether scenario is interesting
set l [$ns link $node_(0) $node_(1)]
set bneck [$ns asim-getLinkTput $l] # get approx. results
set asimbw [expr $bneck * 8 / 1000]
puts "Asim bandwidth = [format "%.2f" $asimbw] Mbps "
# check whether prefilter criterion is met
if {$asimbw < 7} {
   puts "  *** UNINTERESTING scenario, exiting ..‘‘
   exit 0
}
puts "  *** Potentially INTERESTING scenario ... "
$ns run # do detailed packet level simulation
```

The above script invokes the analytic engine which yields the steady state of the network which is then available to the user, to be used for pre-filtering scenarios. Based on the available information, we can choose to pre-filter the current scenario (as shown above) or add new traffic agents before running the detailed simulation.

2.2 The essential components of Approx-sim

In this section, we will describe the basic models and algorithms of the analytic engine. Consider the flowchart of the analytic engine of *approx-sim* as shown in Figure 2. In step 1, the engine (*approx-sim*), in its *simulate* procedure, first calculates the drop probability and the queuing delays at each router from the previous iteration or from the initial conditions (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

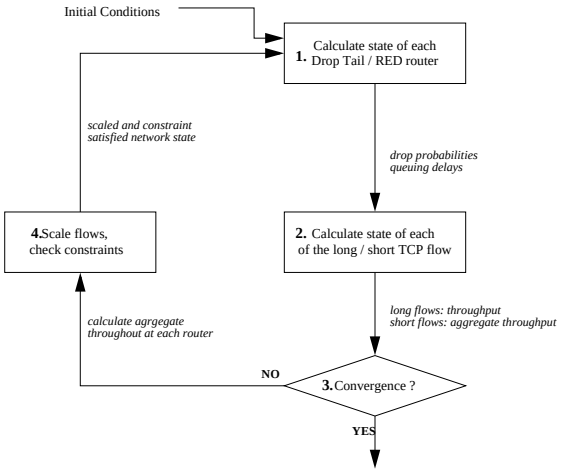

Figure 2: The structure of the *Approx-sim* simulator

checks whether new network state has converged. 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 (M/M/1/K model, see [19]) for drop tail routers and we propose a very simple analytical model for RED [11] gateways that yields the drop probability and the delay in terms of the offered load. 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 using the well known equations in [24]. For short lived flows, we calculate their aggregate throughput. We discuss this issue of convergence in Section 2.5.

2.3 Modeling RED

RED represents a common AQM policy implemented in routers. 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

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

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 1. *The steady state average value of the queue-*

length with utilization ρ and drop probability p is given by

$$l_q \approx \frac{\rho(1-p)}{1-\rho(1-p)} \quad (2)$$

PROOF. The quantity $\rho(1-p) < 1$ since it is the goodput. The result follows from M/M/1 model. □

Note that we have not used the M/M/1/K model which would have been more accurate. But in that case, the closed loop solution would have been much more complex. Again we stress on simplicity of our solution.

LEMMA 2. *In the steady state, the average queue length will never be larger than max_{th}*

PROOF. If the queue length increases over max_{th}, the packets are dropped. Hence the queue. □

THEOREM 1. *If the drop probability seen at a RED 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*

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

Using the above theorem we obtain a quadratic equation in p. Hence, given a value of λ (or ρ), we find a drop probability p. Then, using the value of p and Lemma 1, we 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 in Lemma 1 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 1 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 by $p = \frac{\rho - \rho_{max}(1-p_{max})}{\lambda}$. 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. Thus, we are trying to find the average state. If $\rho_{min} < \rho < \rho_{max}$, the drop probability can be computed using Theorem 1. 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 [9].

2.4 Modeling mice

In this section, we present a simple model for the aggregate of short lived TCP flows. Short lived TCP flows, or *mice*, have been extensively studied in [3] and [17]. 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 to help us find the approximate fixed point very quickly. We have refined Ben Fredj et al. [13] model of *mice* to incorporate the drop probabilities along the path. 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. From [13], we have the following: If the average 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 average rate of short flow traffic between the same source-destination pair is given by

$$\lambda_{mice} = \lambda_{arrival} \times \sigma \quad (4)$$

For now, we assume a simple exponential arrival pattern for the mice. Also, each connection transfers a constant amount of data. Then, we have the following.

LEMMA 3. *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*

$$B_{mice} = \frac{\lambda_{mice}}{1-p}, \text{ where } p\text{=drop probability} \quad (5)$$

PROOF. The total short lived traffic that arrives at a router is given by Equation 4. Now, if drop probability is p, the traffic generated by short flow request in one second is given by

$$B_{mice} = \lambda_{mice} \times (1 + p + p^2 + p^3 + ...) \quad (6)$$

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 [13]. 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. Note we can also calculate the bandwidth of the CBR flows (over UDP) and add it to the *load*.

2.5 Convergence

It is not clear whether a fixed point exists between the router characteristics and the flow characteristics in all network scenarios but Bu et al. [2] prove the existence for a single congested link. Since we find the *approximate fixed point*, we have been able to reach convergence by using an iterative scheme that tolerates errors.

Our algorithm requires us to start with reasonable 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.

Initially we assume that that the load on the links are due to the presence of the mice traffic alone. Also, 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 throughput of the elephants are according to the classic TCP equation in [24] with low drop probabilities. That may result in window limited or large throughput. Now we run the algorithm and 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 times an overload factor. 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.

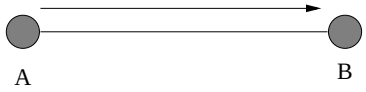

Figure 3: The line topology

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

THEOREM 2. *The worst case running time, T, of the scaling algorithm on a network of size N connections, M links is given*

$$T(M, N) = \theta(M \log M + N.H) \quad (7)$$

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. Now, the procedure can take up to M steps. Hence the theorem. □

3. 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, 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.

3.1 Elephant traf£c alone

First we consider results that we obtained for the experiments with elephant-only traffic. We evaluated *approx-sim* on the line topology (Figure 3) as well as symmetric (Figure 4) and asymmetric trees (Figure 7). This gradual in-

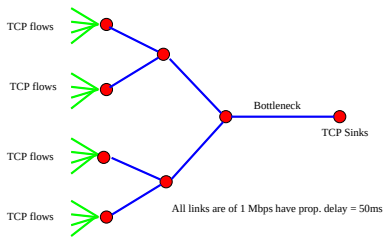

Figure 4: The symmetric tree topology: a sample binary tree of height 2 with four clients at each leaf.

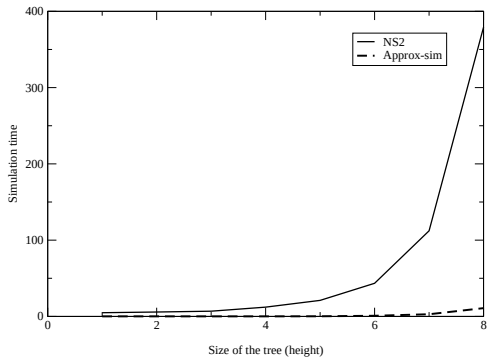

Figure 5: Running time comparisons between *approx-sim* and *ns-2* for the symmetric tree topology with elephant traffic only

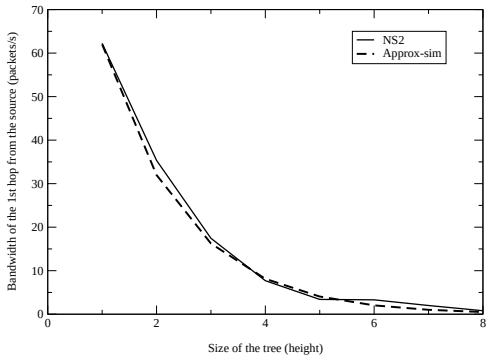

Figure 6: Accuracy of *approx-sim* throughput compared to *ns-2* for the symmetric tree topology with elephant traffic only

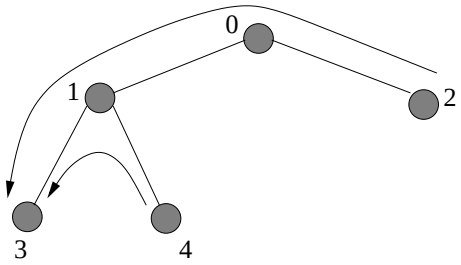

Figure 7: The asymmetric tree topology

crease 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 4 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 5 shows the run-time performance of the stand-alone version of *approx-sim* compared to packet-level simulation with *ns-2* for symmetric trees as a function of tree height. Comparing the embedded version of *approx-sim* is difficult, as some time is spent in populating the internal structure of *ns-2*. Note that the input file for the stand-alone version is generated from the *approx-sim* embedded in *ns-2*. *Approx-sim* is 10-300× 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 6 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 7) to avoid biases in evaluation due to symmetry. We examine asymmetric trees of varying heights. Figure 7 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 7) had very low throughput while short RTT flows (eg. between 4, 3 in Fig-

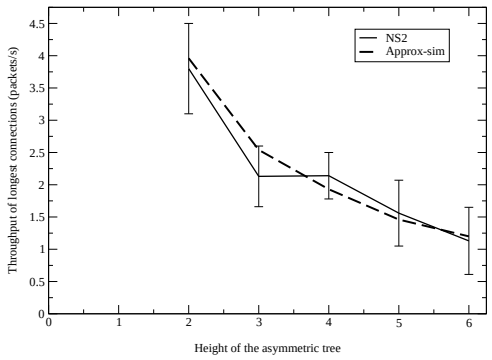

Figure 8: Comparison between *approx-sim* and *ns-2*: throughput of the longest TCP connections in several asymmetric trees - all links have a capacity of 1MB/s

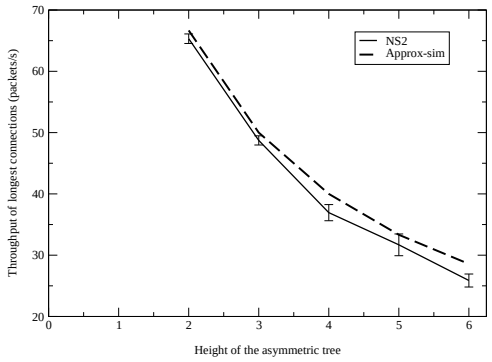

Figure 9: Comparison between *approx-sim* and *ns-2*: throughput of the longest TCP connections in several asymmetric trees - all links have a capacity of 45MB/s

ure 7) had high throughput. Although it is well known that TCP is unfair to flows with different RTT and [24] 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 [13].

[1]Although at coarse scales, some protocol synchronization has been observed [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 8 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* to be more accurate when the network is less loaded. We therefore also considered the same scenario with 45Mb/s-bandwidth links (See Figure 9). 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.

3.2 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.

3.2.1 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 3. 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.

3.2.2 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 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. The results for this experiment are shown in Figure 11.

We observe that as we increase the the link bandwidth from 1MB/s to 2MB/s, the accuracy of *approx-sim* drops from

Capacity (MB/s)	# of elephants	# of Mice λ(/s), σ (kB/s)	Elephants		Mice	
			approx-sim (kB/s)	*ns-2*, (*std.dev.*) (kB/s), (kB/s)	*approx-sim* (kB/s)	*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 10: Comparison of results with drop-tail routers on a Line topology as shown in Figure 3

Height of tree	Capacity (MB/s)	# of elephants per leaf	# of Mice λ(conn/s), σkB/s	Elephants	
				approx-sim (kB/s)	*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 11: Comparison of results with drop-tail routers on a symmetric tree topology as shown in Figure 4

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.

3.2.3 Asymmetric Trees

Now we compare results for asymmetric trees to avoid symmetry. Figure 12 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 13 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 12. The accuracy of *approx-sim* varied from 13 to 17% for Fig. 13 and from 6-20% in Fig. 14.

Comparing Figures 12, 13, 14, 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 3.1). 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 3.1).

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.

3.2.4 Circular topologies

We next considered the ring topology shown in Figure 15. Between each alternate node (eg. between A, C), we vary

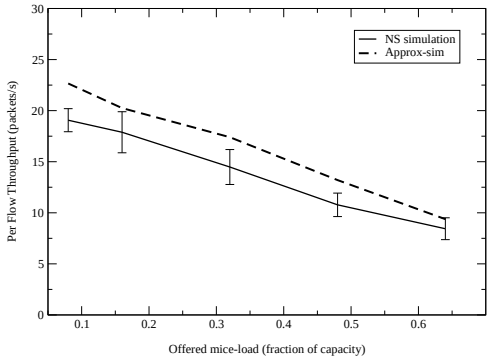

Figure 12: Comparisons between *approx-sim* and *ns-2*: bandwidth achieved by the flows having longer RTT i.e. around 300ms

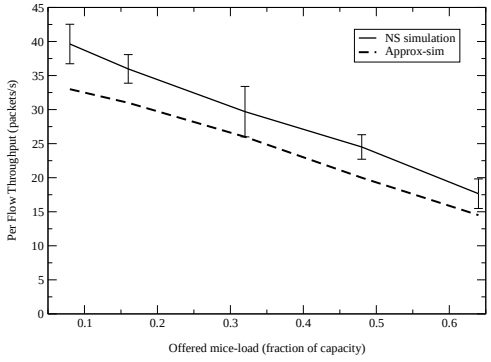

Figure 13: Comparisons between *approx-sim* and *ns-2*: bandwidth achieved by the flows having short RTT i.e. around 200ms

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 16 presents the throughput of short and long flows

Capacity (MB/s)	# of elephants	# of Mice λ(conn/s), σkB/s	Elephants *approx-sim* (kB/s)	Elephants *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 16: Comparison of results with drop-tail routers on a circular topology as shown in Figure 15

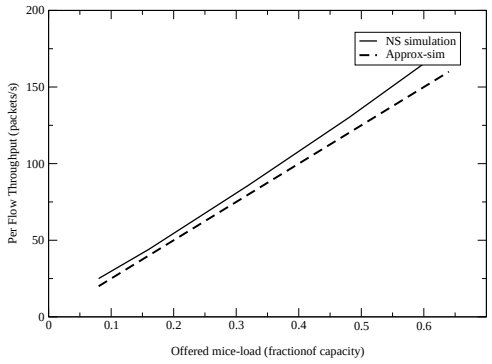

Figure 14: Comparisons between *approx-sim* and *ns-2*: Aggregate short flow throughput

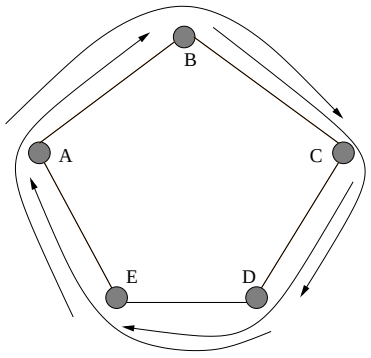

Figure 15: The ring topology

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.

3.3 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 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 17, 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.

4. 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. [13] 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. Another way is to use some algorithmic enhancements to reduce the complexity of the scaling algorithm. Then we also need to ensure that our *approx-sim* results are accurate for networks with thousands of nodes. Like any software, we need to fine tune and improve the performance of both the stand alone as well as the embedded version of *approx-sim*.

Topology, Capacity (MB/s)	# of elephants	# of Mice λ(conn/s), σkB/s	Elephants *approx-sim* (kB/s)	Elephants *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 17: Comparison of results with RED routers

5. 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] Victor Firoiu, Ikjun Yeom, and Xiaohui Zhang. A framework for practical performance evaluation and traffic engineering in ip networks. In *Proceedings of the IEEE International Conference on Telecommunications*, 2001.

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

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

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

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

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

[13] 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.

[14] 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.

[15] R.J. Gibbens, S.K. Sargood, C. Van Eijl, F.P. Kelly, H. Azmoodeh, R.N. Macfadyen, and N.W. Macfadyen. Fixed-point models for the end-to-end performance analysis of ip networks. In *Proceedings of the 13th ITC Specialist Seminar: IP Traffic Measurement, Modeling and Management, Sept 2000, Monterey, California*, 2000.

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

[17] 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.

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

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

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

[27] Matthew Roughan, Ashok Erramilli, and Darryl Veitch. Network performance for tcp networks part i: Persistent sources. In *Proceedings of ITC'17, Brasil*. ACM, Sepetember 2001.

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

[29] 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.