Coarse-Grained Network Simulation for Wide-Area Distributed Systems

Syam Gadde, Jeff Chase, Amin Vahdat
Department of Computer Science
Duke University
Durham, NC 27708
{gadde,chase,vahdat}@cs.duke.edu

Keywords: Transfer-level simulation, Internet services, TCP/IP

Abstract

Evaluating the performance of large-scale wide-area applications such as content distribution networks and replicated Internet services through simulation presents formidable challenges. Among them is the need to balance competing needs: ensuring accuracy by choosing an appropriate level of detail, and making these large-scale simulations tractable by minimizing required computational resources. As an alternative to packet-level simulation, we evaluate the applicability of coarse-grain *transfer-level* simulation to wide-area networked applications and services, and discuss the tradeoffs inherent to such an approach. We introduce GUTS, a high-level wide-area network simulator whose goal is to enable simulation of realistic Internet-scale topologies, under a range of realistic workloads.

1 INTRODUCTION

The wide deployment of Internet services and rapid growth of Internet usage have been both a great boon to users and a great challenge to system designers. For example, the issues involved in cost-effective provisioning of resources for high end-to-end performance of Web services are complex and not entirely well understood.

Our research focuses on several of these wide-area services including content distribution networks (CDNs), replicated Internet services, grid computing, peer-to-peer file sharing networks, and other wide-area storage infrastructures [2, 7, 9, 13, 16, 23]. These services all share several goals: they are all designed to serve a large population of users, to target a user base that covers a very large network area, and to adapt to changing characteristics of the Internet and unpredictably bursty request traffic.

There are many factors involved in implementing a wide-area service that meets these goals. Simulation is a necessary tool for delineating these factors and evaluating their effect on performance. However, current state-of-the-art network simulators such as `ns` [19] are unable to scale to networks of Internet size due to the computational requirements of fine-grain packet-level simulation and the memory needed to maintain queues at each network link. Existing simulation technologies that do scale to large-scale networks do so only through the use of distributed computational resources, retaining the detail and complexity of the original problem.

Our goal is to enable the simulation and comparative evaluation of wide-area services on large-scale networks using modest resources. We achieve this by developing an abstraction for modeling wide-area networks that is designed explicitly for simulating networked services *at the application level*, rather than at the transport level or below. For the class of services whose traffic consists primarily of bulk transfers over fair, reliable transport protocols such as TCP/IP, we argue that simulating them at a *coarse grain* is practical and efficient, and yields aggregate performance metrics that are accurate enough to compare the relative performance of simulated systems.

We introduce GUTS, a simulator that supports this network model, and which runs more than two orders of magnitude faster than `ns` [19], the most widely used network simulator today. We show that the GUTS network model's asymptotic running time is less than packet-based network simulation models, and that this method still captures relevant performance characteristics for an important class of wide-area services.

In the next section, we motivate coarse-grain simulation by discussing various existing approaches to evaluating network services and the simulation domains they target. Section 3 describes our transfer-level approach to network modeling. Section 4 describes our coarse-grain simulator GUTS and evaluates how well it achieves our goals. Section 5 summarizes our findings and concludes.

	Simulation granularity	Key abstraction technique	Target domain(s)
ns [19]	packet		transport-level protocols and below
Packet trains [1]	groups of packets	treating closely-spaced series of packets as a large packet	transport-level protocols on packet networks
Selective abstraction [12]	packet	selectively move transient details from simulated world to simulator	transport-level protocols and below
Fluid network simulation [15]	flow changes	map network as pipes, packet streams as fluid flows	transport-level protocols
Time-stepped Hybrid Simulation (TSHS) [10]	packet/time steps	packet smoothing	transport-level protocols on packet networks

Table 1: Examples of network simulation abstraction.

2 EXISTING APPROACHES

There are many ways to evaluate the merits of design decisions in network services. The most direct approach is to implement a prototype system, deploy it on the Internet, then construct experiments to emulate situations that the service is expected to handle, such as diurnal workload patterns or sudden bursts of request traffic. For wide-area Internet applications, the costs and difficulty of evaluating systems directly in this way can be prohibitive. Obtaining resources that are representative of wide-area network effects may not be feasible, generating and coordinating realistic client loads may be very difficult, and evaluating the system under various controlled network conditions may be impossible. In addition, direct evaluation is, by its very nature, limited to current technology.

Network emulation [14, 18, 22] is an invaluable tool that offers many of the advantages of direct Internet evaluation, but eliminates the problem of obtaining remote computational resources and network access. Emulation allows machines running the actual service on a local-area network to experience delays and bandwidth limitations normally imposed by a wide-area network. Emulation achieves this by introducing an intermediary component between end hosts, such as a smart bridging host, that models wide-area network characteristics. However, it – like direct evaluation – requires that the system be implemented at least in a prototype stage, and it also forces experiments to run no faster than real time, which may prove unwieldy for experiments that require a day's worth or more of application execution time on a machine cluster to produce a valid data point.

To address these issues, many researchers have turned to analytical modeling and/or simulation for answers. Several efforts [1, 5, 8, 11, 12, 15, 17] discuss the performance tradeoffs one encounters when mapping real-life environments to a simulated environment and back again. They all sound a common theme – choosing an appropriate level of abstraction is key, both to ensure continued accuracy of results and to enable the simulator to scale to environments of realistic size. Higher levels of abstraction can lead directly to increased simulation speed and decreased resource consumption, but at the cost of some measure of accuracy.

Many examples of abstraction for network simulation exist in the research. Table 1 cites several of the most representative approaches, and notes important characteristics of each. All are used primarily in the literature to evaluate network protocols at the transport level or below, and, with the exception of fluid network simulation, are all based on a packet model. Though the fluid model abstracts streams of packets as fluid flows, it is the changes in flow that generate simulation events and affect simulation time, and for large networks, these can easily approach and even exceed the complexity of packet models.

However, the simulation domains we target in this paper exist at the application level. ns (the most widely-used network simulator today) and other packet-level simulators provide a level of detail that is suitable and necessary for evaluating small-scale systems and transport-level protocols, but they are not ideally suited for simulating the domain of large-scale wide-area networked applications. ns' scalability is limited to thousands of nodes on today's commodity workstations. Even alternative simulation technologies that harness distributed computational resources (Parsec [3], SSF [8]) do not reduce the inherent complexity and cost of packet simulation. This motivates us to consider moving up one level to a coarser-grain model.

3 THE GUTS NETWORK MODEL

The main principle of GUTS network modeling is to focus on maintaining accurate *aggregate* performance metrics that enable us to compare the relative performance of services over a common network infrastructure. So, rather than monitoring the latency of individual transfers over a network, we use aggregate metrics to show, for example, that certain Web content replication strategies have more efficient network utilization and produce lower average client latencies than others. The inaccuracies we introduce by moving to a higher level of abstraction affect all simulated services in the same way, therefore preserving an accurate estimate of the *relative*

performance of simulated services.

3.1 Transfer-level abstraction

For our work, we choose to simulate at a *per-transfer level*, rather than a per-packet level. High-detail low-level network simulators such as ns model congestion using packet queues which can drop or delay packets. Our network model does not use network queues at all. The network model allocates bandwidth for a transfer only once, using the transient state of the network at the beginning of the transfer, and the allocation remains fixed for the transfer's duration. This dramatically reduces the amount of state the simulator must store and the number of events it must process for each transfer.

We justify this level of abstraction by noting particular traits of our target simulated services. First, bulk transfers make up almost all of the per-byte network traffic generated by services like Napster [16] or the Web – transfers are sent in one chunk, as fast as the server CPU and disks can handle it and as fast as the network allows. Second, we assume that these transfers use a "fair," reliable, connection-oriented transport protocol that fairly allocates bandwidth among all transfers simultaneously sharing a link (like an ideal TCP/IP). Lastly, these bulk transfers are on average much larger than MTU/MSS-sized packets (a typical client HTTP workload has average response sizes of greater than 10KB, and the average is much higher for file sharing services such as Napster). Because the majority of traffic in our target applications are relatively long-lived bulk transfers, and because our target services do not need to monitor any transfer's behavior while it is in transit, it is appropriate to amortize the effects of congestion on a transfer over its duration.

There are several metrics that are robust for our target service domain when moving from per-packet to per-transfer grain, i.e. those metrics for which the coarsening of simulation detail should not introduce non-monotonic instability. These metrics include average transfer latency, bytes-times-distance (which measures total network utilization), average delay introduced by congestion on the network, network throughput, and others. Any application-level metrics that are directly based on such robust metrics will also be robust. However, as with all abstractions, these coarsened metrics are not appropriate for services such as quality of service (QOS) applications, network feedback-based flow-control, and real-time services, for these applications depend on accurate instantaneous measures, and are not typically designed to handle transient inaccuracies in network state.

3.2 Modeling congestion

Like most network simulators, GUTS models a network as a directed graph with *nodes* that represent either hosts or routers and *links* that represent direct network connections between two nodes. Every link l has two static properties – total capacity C_l and propagation delay D_l – and two dynamic properties – allocated bandwidth A_l and transfers in progress T_l. We now present the basic algorithm GUTS uses for sending a transfer of size s at time t between two nodes:

1. Find the set P of all links on the path from source to sink, and label the link l_b which has the least *available bandwidth* B_{l_b} (defined below) as the *bottleneck* link.
2. For each link $l \in P$, "reserve" bandwidth by incrementing A_l by B_{l_b}.
3. Increment the T_l for each link $l \in P$.
4. Calculate the total duration d of the transfer.
5. Schedule an event to occur at time $t + d$ that, for each $l \in P$, releases bandwidth B_{l_b} and decrements T_l.

Two important properties of this algorithm are that the dynamic state required on each affected link for maintaining transfer information consists of two values ("transfers in progress" and "available bandwidth"), and that this state changes only twice for each transfer (once at the beginning, once at the end).

GUTS defines *available bandwidth* of a link l as the maximum of either $C_l - A_l$, i.e. the unreserved bandwidth on the link, or $C_l/(T_l + 1)$, i.e. the link's total capacity divided by the number of transfers in progress on the link (plus one to include the new transfer). This is a heuristic that allocates full bandwidth to transfers on links that are not under contention, and otherwise allocates a fraction of the capacity depending on how many other transfers are sharing the link.

The transfer duration d is the sum of propagation latencies for all links $\in P$, plus a queuing delay q related to the number of transfers in progress on each link $l \in P$, multiplied by three to account for the two SYNs sent at the beginning of a TCP connection, plus $\frac{s}{B_{l_b}}$ (the actual on-link transfer time).

For the queueing delay q, we assume that every transfer in progress on a link has at least one packet on the link's queue ready to send (and one of those is currently on the link, halfway through on average). We also assume that they are all of MTU size. The start of the new transfer on this link will be delayed by these packets before it gets a chance to use the link. So we define q as $\sum_{l \in P} \frac{MTU}{b_l}(T_l - 0.5)$, for $T_l > 0$.

There are a few aspects of this algorithm that lead to inaccuracy. First, the modeling of a transfer as a constant bandwidth stream is an approximation of real behavior, where the transfer may actually be given smaller or larger portions of network bandwidth depending on changing network conditions. This is, in a sense, an extreme case of "packet-trains" [1], where the granularity of coalescing associated packets within a transfer is very large. Second, because the

algorithm does not have the freedom to change the bandwidth already allocated to transfers in progress, it may "overcommit" a link when a new transfer arrives on a link with no available bandwidth. Likewise, in-progress transfers will not increase their allocated bandwidth in response to a completed transfer releasing its bandwidth, leading to an "undercommited" link. Third, the algorithm does not capture the dynamics of TCP slow-start and the effect of dropped packets on TCP flow control. Though we believe that the long-term effects of dropped packets are captured to some extent by the lower bandwidth we give to transfers sharing bottleneck links, as an area of future work we are investigating the use of throughput-estimation models [6, 20] to augment the heuristics we currently use.

4 EVALUATION

We have implemented the network congestion algorithm described in Section 3 in GUTS, a portable object-oriented wide-area network simulation package written in C++ that targets evaluation of Internet services at the application level. GUTS provides the infrastructure to implement many common services; those implemented already include content distribution networks and peer-to-peer networks [21]. GUTS also provides a flexible workload construction system, supporting several types of client request patterns and object properties. Because the focus of this paper is the GUTS network model, we will not go into further detail on these components, except as necessary in describing our experiments.

In this section, we are interested in answering several questions. Does the GUTS network model produce aggregate metrics that are representative of a real network when subjected to a realistic workload? Also, for any particular metric, does the model capture a similar range of values to that observed in a real network? And lastly, how much computational time do we save by making coarse-grain abstractions? In the next section, we illustrate the primary characteristics of the GUTS model visually, and then follow with results from experiments to help answer the above questions.

4.1 Visual comparison to "ideal" TCP

Our abstraction of TCP is an attempt to capture similar average behavior to what can be called an "ideal TCP", where all connections share common links fairly and, when possible, use the maximum capacity of each link. As a basis for visual comparison to GUTS' "early reservation" connection modeling, we use an approach similar to the fluid network simulation model [15] to model an ideal high-level abstraction of TCP connections. Figure 1 shows a simple scenario of one link subjected to a contrived workload of several overlapping transfers, both with the GUTS model, and with an "ideal TCP". The black lines indicate the start of a new transfer, and each color represents a different transfer. The x-axis is time,

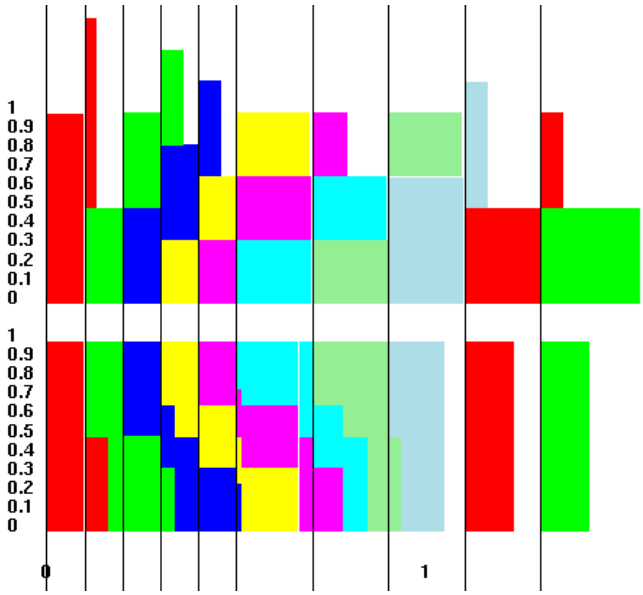


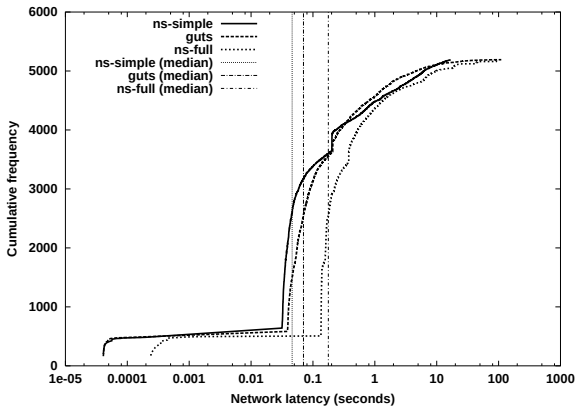
Figure 1: An example of bandwidth allocation to message transfers. Bottom is "ideal" TCP, top is GUTS.

and the y-axis represents bandwidth allocated on the link.

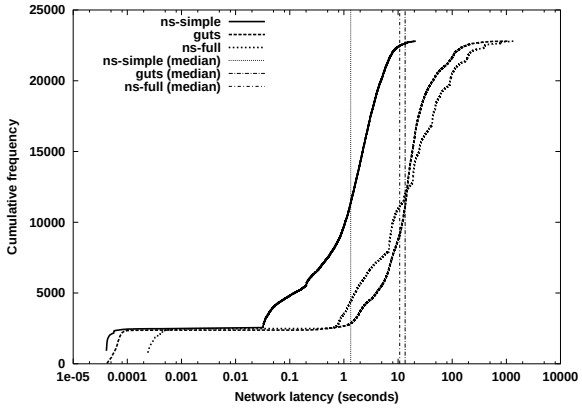
This figure illustrates two hallmarks of the GUTS abstraction, link overcommit and link undercommit. Both GUTS and ideal TCP allocate full bandwidth to the first transfer initially because there is no competing traffic. Meanwhile a second transfer starts to use the link. Whereas ideal TCP now halves the original bandwidth allocation of the first transfer, GUTS allows it to use that full bandwidth until its completion. It also assigns half of the capacity of the link to the second transfer, even though the previous transfer is still occupying the link at full capacity. These link overcommits are balanced by link undercommits later on, when transfers do not take advantage of bandwidth freed when flows disappear from the link. These inaccuracies are the cost of making an inexpensive one-time bandwidth allocation decision and forgoing the use of packet queues to share the link at a finer granularity. In the next section, we show that the effect of undercommits and overcommits does not have a major impact for our target applications on a class of realistic workloads.

4.2 Comparing to a "realistic" TCP

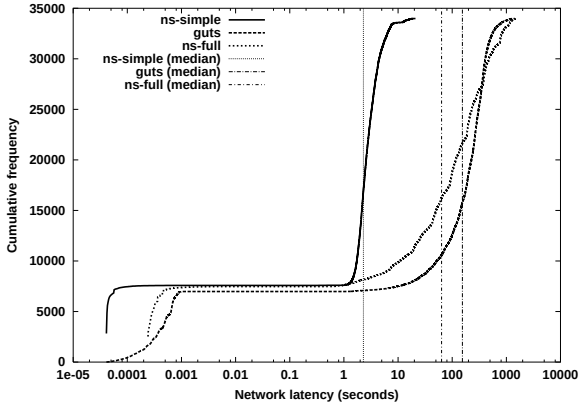
We can show that the GUTS network model produces similar results to much more detailed simulators, but with a dramatic reduction in computational cost. We ran several simple scenarios through an unmodified GUTS, then replaced the GUTS network module with an interface to ns. This enables us to validate the results we generate using the GUTS network model, at least to the network sizes to which ns can scale. We used ns version 2.1b7, and made modifications to GUTS



(a) 5 client sites



(b) 25 client sites



(c) 100 client sites

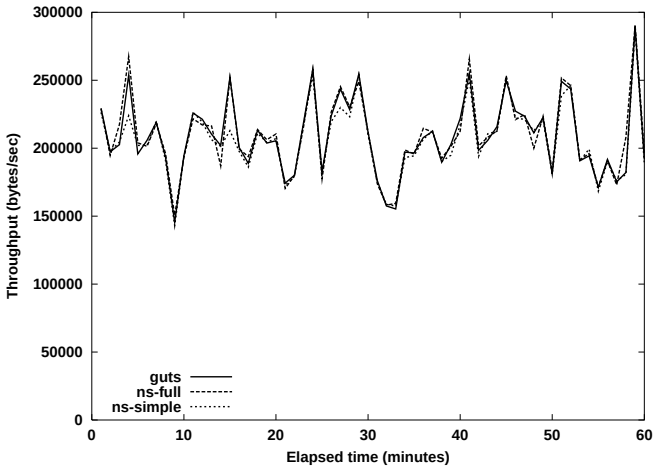
Figure 2: Client latency CDFs reported by GUTS and ns. Note log scale on x-axis.

to allow it to use ns' event scheduler, to implement GUTS "message actions" as TcpApp callbacks, and to convert GUTS network descriptions into equivalent ns topologies. We used the two TCP models supplied by ns that TcpApp supports, FullTcp and SimpleTcp. FullTcp implements TCP Reno, while SimpleTcp has no congestion control, fragmentation, or link sharing and essentially implements a first-in first-out message queue (not a packet queue) on every link. SimpleTcp was designed for the simulation of Web cache traffic [24], and, like GUTS, sacrifices some measure of accuracy for speed because the desired metric granularity is coarser than that given by FullTcp.

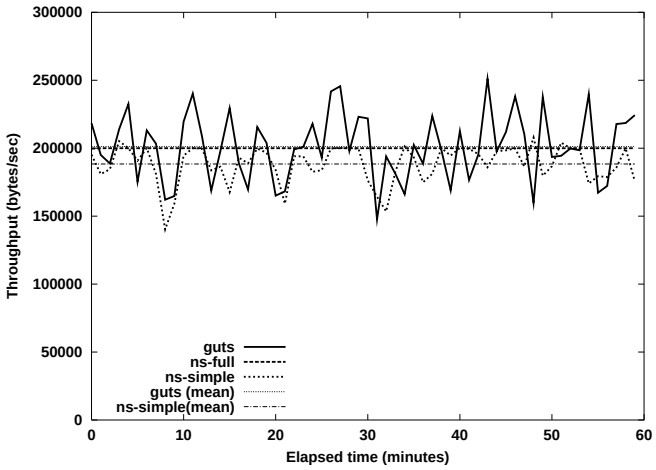
Our experiments simulate a number of "client sites", which represent clusters of clients requesting objects from any of 10 Web server replicas, chosen at random for each object. Clients generate a Surge-like [4] workload, but with client off times independent of server response time (allowing a reproducible workload). We use a simple topology with 10 subnetworks of 10 nodes each plus a gateway, with all gateways connected in a star topology. We place clients and servers randomly on network nodes. We assign very high bandwidth and very low latency to all links except we assign T1 bandwidth to the shared broadcast star, which forces this to be the bottleneck and the sole point of contention for any traffic that travels between subnetworks. The purpose is to isolate and stress the congestion models of the simulators.

Figure 2 shows CDFs of network latency per request for GUTS and ns when subjected to a half-hour of various client loads. Though slow-start effects (which neither SimpleTcp nor GUTS models) cause GUTS to show results closer to SimpleTcp than FullTcp for short-duration transfers on lightly loaded systems (Figure 2(a)), GUTS latencies track the shape of the FullTcp curve much more closely than SimpleTcp does for both moderately and highly loaded systems (Figures 2(b) and 2(c)), where congestion is more likely to result in dropped packets and diminishing bandwidth. Because SimpleTcp does not discretize transfers into packet-size chunks, and because it does not allow transfers to share links simultaneously, large transfers, once scheduled, can monopolize a link until they are completed. Therefore, simultaneous transfers will not interfere with large transfers in SimpleTcp, whereas it is these long transfers that are most likely to experience dropped packets and interference in FullTcp. The GUTS heuristic, on the other hand, scales down bandwidth for large transfers automatically when links are highly congested. This explains the large difference between SimpleTcp and FullTcp at the high end of the latency scale, and why GUTS does capture the high-latency transfers that FullTcp does.

Though the client latency CDFs show the distributions that individual transfers follow, we are also interested in metrics, such as throughput, that capture the aggregate behavior of interacting connections. So, we repeated the experiment, but this time monitoring the throughput of the bottleneck link, running once at T1 speed, and again with a gigabit link. Figure 3(a) shows that on a well-provisioned (1Gb) link, all three approaches show very similar throughputs. But on a



(a) On a link with bandwidth to spare



(b) On a congested link

Figure 3: Throughput on a representative link as reported by GUTS and ns.

congested link (Figure 3(b)), FullTcp uses the total capacity consistently, while SimpleTcp and GUTS are more bursty. While the burstiness of SimpleTcp is restricted to below the FullTcp line, the GUTS model allows the burstiness to push the throughput above and below the capacity of the link. However, the average (mean) throughput of GUTS for the entire run, shown by the line labeled "guts (mean)," is almost equivalent to (and nearly hidden by) that of FullTcp.

We now look at the efficiency of the simulators using the same experiments. Although execution time is a valuable metric, it alone does not suggest anything about the efficiency of the underlying models used by both simulators. For example, an execution profile of both ns runs indicate that at least 90% of ns' time is spent in interpreted Tcl code rather than C++ code. Execution time does not tell us whether implementing the ns model completely in a compiled language would eliminate the differences in execution time. To this end, we also use the implementation-independent metric *event rate*, which refers to the number of basic operations the model performs per unit of simulated time. This tells us the fundamental cost of increased accuracy in ns and other simulators that simulate at the same granularity. The event rate is closely related to asymptotic running time, because we can generally treat each event as a constant-time operation. We calculate the event rate by counting the number of events dispatched by the scheduler.

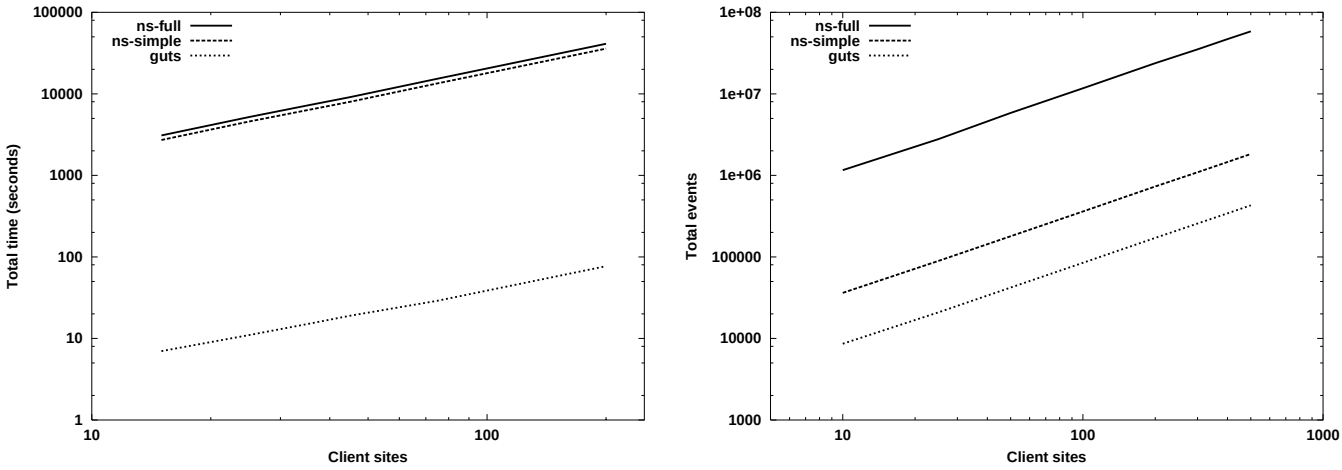
Intuitively, we expect the GUTS model to have much lower event rates than ns. The GUTS network model introduces two events for every transfer, one to calculate routes, allocate bandwidth, and calculate propagation time, and one when the transfer arrives at its destination, to deallocate bandwidth that the transfer was using. The number of events packet-level simulators generate is strictly larger, because every packet in a multi-packet transfer will require an additional event for every link it traverses. Even the SimpleTcp model in ns is likely to generate more events because transfers are scheduled on every link in the path.

Validating this hypothesis, Figure 4(a) plots the execution time and Figure 4(b) shows the event rate of GUTS and ns as we vary the number of client sites. We see that the ratio between ns' and GUTS' execution times remains constant, with GUTS consistently running more than 2 orders of magnitude faster than ns. The ratio of ns' to GUTS' event rates likewise remains constant as we vary the intensity of the workload, with SimpleTcp and FullTcp respectively generating 5 times and 100 times more events than GUTS.

5 CONCLUSION

This paper explores techniques for efficient simulation of large-scale Internet services. The most important decision is choosing the appropriate level of simulation detail. Packet-level simulation as implemented in several current network simulators offers enough detail to accurately model existing and new network protocols. We provide intuition why this is excessive and unnecessary when simulating a large class of wide-area networked services whose traffic is largely composed of bulk transfers. This is especially true when comparing simulated services to each other – inaccuracies that affect one will likely affect the others in the same way, preserving measurements of *relative* performance.

We have developed a simple, coarse-grain algorithm for modeling transfers over fair, reliable, connection-oriented network protocols. It allocates a fixed bandwidth to each transfer based only on transient network state at the time the



(a) Execution time

(b) Event rate

Figure 4: Speed of GUTS and ns. Note log scale on both axes.

transfer is scheduled. This algorithm requires only four state variables per network link and generates at most two simulation events per message transfer.

We have implemented this algorithm in GUTS, a network simulator designed for evaluating large-scale Internet services. We show that GUTS offers significant speed benefits over packet-level simulation, but with appropriate levels of accuracy under the workloads with which we expect to test our target applications.

References

[1] Jong Suk Ahn and Peter B. Danzig. Packet network simulation: Speedup and accuracy versus timing granularity. *IEEE/ACM Transactions on Networking*, 4(5):743–757, October 1996.

[2] Akamai Technologies, Inc. `http://www.akamai.com/`.

[3] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu an Chen, Xiang Zeng, Jay Martin, and Ha Yoon Song. PARSEC: A parallel simulation environment for complex systems. *IEEE Computer*, 31(10):77–85, October 1998.

[4] Paul Barford and Mark E. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of Performance '98/ACM SIGMETRICS '98*, pages 151–160, June 1998.

[5] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John S. Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, 2000.

[6] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling TCP latency. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[7] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[8] James H. Cowie, David M. Nicol, and Andy T. Ogielski. Modeling the global internet. *Computing in Science and Engineering*, 1(1):42–50, January/February 1999.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[10] Yang Guo, Weibo Gong, and Don Towsley. Time-stepped hybrid simulation (TSHS) for large scale networks. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[11] John Heidemann, Nirupama Bulusu, Jeremy Elson, Chalermek Intanagonwiwat, Kun chan Lan, Ya Xu, Wei Ye, Deborah Estrin, and Ramesh Govindan. Effects of detail in wireless network simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–11, January 2001.

[12] Polly Huang, Deborah Estrin, and John Heidemann. Enabling large-scale simulations: selective abstraction approach to the study of multicast protocols. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 241–248. IEEE, July 1998.

[13] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, November 2000.

[14] Jay Lepreau et al. Emulab.Net: The Utah network testbed. `http://www.emulab.net/`.

[15] Benyuan Liu, Daniel R. Figueirido, Yang Guo, Jim Kurose, and Don Towsley. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. *Proceedings of IEEE Infocom 2001*, April 2001.

[16] Napster Inc. Napster. `http://www.napster.com/`.

[17] David Nicol, Michael Goldsby, and Michael Johnson. Fluid-based simulation of communication networks using SSF. In *Proceedings of the 1999 European Simulation Symposium*, October 1999.

[18] Brian Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, October 1997.

[19] *The Network Simulator – ns-2*. `http://www.isi.edu/nsnam/ns/`.

[20] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM '98*, September 1998.

[21] Nadia Rehman. Comparative analysis of peer-to-peer storage and retrieval systems using GUTS. Master's thesis, Duke University, 2001.

[22] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1), January 1997.

[23] Speedera Networks, Inc. `http://www.speedera.com/`.

[24] Haobo Yu, Lee Breslau, and Scott Shenker. A scalable Web cache consistency architecture. In *ACM SIGCOMM'99*, pages 163–174, 1999.