Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

ADVANCED CONCEPTS IN LARGE-SCALE NETWORK SIMULATION

David M. Nicol
Michael Liljenstam

Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, U.S.A.

Jason Liu

Dept. of Math and Computer Sciences
Colorado School of Mines
Golden, CO 80401, U.S.A.

ABSTRACT

This tutorial paper reviews existing concepts and future directions in selected areas related to simulation of large-scale networks. It covers specifically topics in traffic modeling, simulation of routing, network emulation, and real-time simulation.

1 INTRODUCTION

Use of communication networks is pervasive, and is increasing. The larger and more complex these networks become, the harder it is to predict their behavior before deployment. Analytic models often are useful for a coarse level of analysis, but are limited in the problems they can tractably solve. Detailed discrete-event simulations remain a valuable tool for understanding and optimizing network designs. Commercial network simulation tools have wide-spread use, particularly in government. Simulation of large-scale networks poses some severe problems related to scale. In this paper we consider work in three areas that address some of these problems.

We first consider the area of modeling traffic. The problems of scale here are simply that there is so much traffic that it is computationally infeasible to simulate it all at a fine degree of resolution. The work we describe addresses this through abstraction, which (as we will see) brings its own new set of problems to be solved. Next we consider the simulation of routing protocols, particularly the Border Gateway Protocol (BGP). Problems here arise again because of scale—simulating all routers at a high degree of resolution imposes a memory cost that grows in the square of the number of routers. Here again one may use abstraction and approximation to alleviate some of these problems, but again the solutions create additional problems to be considered. Finally we discuss use of simulation in network emulation and real-time simulation. The context here is integration of virtual representation of systems with physically actual systems. One motivation to relieve in part the burden of abstracting and modeling system behavior within a simulator. Emulation gives analysts a means of generating traffic by real applications, have that traffic be managed (in part) by actual networking hardware, but intersperse simulated elements in such a way that the actual elements are unaware of interacting with a simulation. We review current work and problems in that area.

We hope that this exposition of challenging problems in large-scale network serves as a resource for those who by choice or by context need to learn about it.

2 TRAFFIC MODELING IN LARGE-SCALE IP NETWORKS

Models of network traffic drive virtually every conceivable network simulation experiment. We want to use models that faithfully represent behavior of real traffic, but may encounter problems when simulating large networks that carry IP traffic. The most straightforward approach is to represent IP packets individually. Within the simulation an event occurs when a packet arrives at a new device, after having crossed a link. It is straightforward to see the ramifications this has on the simulation workload. Suppose the average link bandwidth in a model is b bits per second, that the average link utilization is p, that an IP packet uses 8000 bits, and that there are N links in the model. Then $N \times b \times u/8000$ is a lower bound on the number of events the simulation executes per simulation second. A large capacity network may have OC-48 links, which carry 2400Gbps. Assuming link utilization of 10%, the lower bound scales in N as $30N$ million events per second. A highly tuned network simulator that runs on a workstation might be able to execute 1M events per second; we see then that a large-scale simulator that models IP packets directly may advance simulation time at a rate that is considerably slower than real-time. This limits the type of simulation experiment that might be performed.

One can address the issue by abstracting the traffic, and consider it as a “flow", not unlike fluid passing through a pipe.

Flow formulations change the way we update the model, and may offer computational advantages when the rate of updates per unit simulation time is significantly smaller (Liu et al. 1999, Liu et al. 2001). It is commonly the case that we would like to use an abstracted traffic model to describe so-called background traffic, and a detailed packet level model to describe "foreground" traffic of particular interest. The background traffic model paints a lower resolution picture of what is happening in the network. Ideally it does so in a way that can be interpreted by the simulator to adequately represent the impact that background traffic has on the foreground traffic behavior.

Some flow models describe how the flow changes in time with differential equations, e.g., see Padhye et al. (1998), Bu and Towsley (2001), Liu et al. (2004), Yan and Gong (1998). The specifics of the equations capture how protocols like TCP affect the offered load, and how things like RED (Random Early Detection queue management) and network bandwidth limitations affect the offered load. These types of models express the behavior in terms of coupled variables whose values are determined by numerical integration of the equations. Some formulations allow one to conceptually aggregate all TCP traffic streams that have the same source and destination as a single mathematical entity, and describe the interactions of those abstracted flows. Models like these have proven their utility most impressively for stationary models of TCP, where the variables are long-term averages. Since TCP control actions occur at the time-scale of a packet's round-trip time across the network, "long-term" means considerably longer than that. For this reason (and others) these types of models are not well-suited for simulations where the metrics of interest in the foreground traffic are sensitive to variability in the background traffic. Time-dependent differential equations can address this issue, but the main limitation with existing approaches is their focus on describing how all traffic is shaped by a specific protocol. The state of the art does not yet support application in a context where traffic is created and shaped by different sources and protocols.

Models based on differential equations inherently take a continuous view of the model state, and describe how that state changes continuously. An alternative flow formulation is inherently discrete event. In this *Discrete Event Fluid Flow* (DEFF) formulation a flow's rate at a point in the network is assumed to have a mathematical form such that given the flow rate at time s, in the absence of discrete events it is simple to compute the flow rate at time t. Examples of simple flow rate functions include constant flow rates, or linear in time. The basic idea has been used for some time, particularly using piece-wise constant flow rate functions, e.g., see Kesidis et al. (1996), Nicol et al. (1999), Nicol and Yan (2004), Nicol et al. (2003). The DEFF allows one to cast flow state computations into the discrete domain, using familiar techniques. For example, if a traffic source injects flow into the network at a linearly increasing rate until the rate exceeds the network ingress point's ability to absorb it, at time s one can compute the time t of saturation and schedule a conditional event at t to deal with the model state change induced by the saturation. Typically the effect of processing such an event is to alter the rate parameters of some flows. Of course, if the model state changes in such a way to invalidate the conditional event at t, then the event can be canceled.

The computational advantage of the DEFF approach can be assessed by considering how many fewer events are needed to maintain the model state. If a discrete-event fluid formulation defines events at time s and t with no intervening events, and if over that epoch the average flow rate is λ packets per second, then the DEFF formulation represents with one event state what the IP packet formulation takes $\lambda \times (t - s)$ events. Clearly the computational savings can be significant.

However, there can be complications which have no parallel in the IP packet formulation. The usual model of flows competing for a given link's bandwidth assumes that if there is congestion (e.g. more demand for bandwidth than capacity), then each flow is allocated bandwidth in proportion to the rate of its arrival to the link. Thus if flow i arrives at rate λ_i, the available bandwidth is μ, and $\Lambda = \sum_{i=1}^{n} \lambda_i$ (assuming n flows), then when $\mu < \Lambda$ the rate of the i^{th} flow through the link is $(\mu/\Lambda)\lambda_i$. However, consider the ramifications. Suppose that flow i is defined by a traffic source, and at some time t an event is executed that changes λ_i. Suppose further that at time t the link into which flow i is fed is congested. The change in λ_i causes a change in Λ, which causes a change in the value of *every* flow into the link. These flow rate changes have to be propagated downstream; therefore the processing of one event may induce many events.

One can intermix traffic that is represented by IP packets and traffic represented by flows, in the same model (Nicol, Liu, Liljenstam, and Yan 2003). The issues include determining how each type of representation affects the other. For example, when a packet is scheduled to cross a link, and there are flow representations also competing for that link, the simulation model must determine what latency to ascribe to the packet (latency is composed of queuing time, transit time of the first bit, and bandwidth governing the rate of bits being injected into the link.) The queueing delay can be estimated by retaining a measure of the fluid traffic buffered, and the explicit packets also buffered. The transit time of the first bit is a function of the link itself, and the bandwidth allocated is determined by subtracting the bandwidth consumed by the fluid representation from the link's capacity. Conversely, IP packets can be made to affect DEFF model state. One can estimate the rate of IP packet arrivals at any place in the network, and represent the packet stream as a flow in the computations that determine bandwidth allocation per flow. In this way the behavior of the IP packet representation affects the behavior of the DEFF model.

For a traffic stream between a given source and destination, the rate at which the stream creates events is proportional to the number of links in the path, divided by the average link latency. For the corresponding DEFF formulation it is harder to predict the rate of event generation, because different DEFF flows can cause events for each other by interaction through congested links. Ultimately these interactions are a function of how the traffic load at the edge of the network changes, how these changes propagate through the interior of the network, and the multiplicative effect of these changes through congested links. The natural time-scale of events for a flow model is derived from the time-scale of load changes at the network edge. Congestion related events can reduce that time-scale to that of packets crossing a link, for that is the delay between a congestion induced event and its downstream impact.

A different flow model is formulated to keep the overall time-scale at the level of changes in the offered load (Nicol and Yan 2005). In this model the offered load rate of each flow is periodically and simultaneously updated, say every Δ_t seconds. It is assumed that Δ_t is at a coarser time-scale than the end-to-end network latency; Δ_t needs to be large enough so that one can think of the resulting flows *out* of the network occurring more or less instantaneously. So viewed, the function of the network model exists to transform the offered rate of every flow into a delivered rate. Along the way the transformation needs to identify, for each flow, what the flow rate is at every point in the network. Then IP packet models can be executed in conjunction with those flows in much the same way as they are executed in conjunction with DEFF formulations. The difference is that the flow rates are assumed to be constant over the defining epoch of Δ_t time.

The transformation of offered rates to delivered rates is accomplished by looking at where the rate of a given flow changes. Suppose that a flow i directed through router output port p arrives with rate $\lambda_{i,p}^{(in)}$ and that $\lambda_{i,p}^{(out)}$ defines its rate onto the link fed by p. Furthermore suppose that Λ_p is the sum of all arrival rates into p, and that μ_p is the link bandwidth of port p. Then formalizing what we have said before, we define

$$\lambda_{i,r}^{(out)} = \begin{cases} \lambda_{i,r}^{(in)} & \text{if } \Lambda_p \leq \mu_p \\ (\mu_p/\Lambda_p)\lambda_{i,r}^{(in)} & \text{otherwise} \end{cases} . \quad (1)$$

Now observe that for some router r', $\lambda_{i,r}^{(out)} = \lambda_{i,r'}^{(in)}$. Thus we see that the output flow values we seek are found by solving a large set of non-linear equations. These systems usually have dependency cycles in them.

The solution investigated in (Nicol and Yan 2005) proceeds in three steps. In a first step the problem is aggressively reduced by computing the final value of as many flow variables as possible. This step uses upper bounds on unknown flow variables to identify ports p where it can be determined that $\Lambda_p \leq \mu_p$, even if the exact value of Λ_p is unknown. A port for which this can be established is called “transparent", and it is known that all flows through that port pass through without changes in their flow rates. Once the problem has been reduced the issue is of solving circular dependencies. The dependencies occur between ports that may be congested; output flows (which are unknown because the total rate of inputs to the port are unknown) are inputs to other ports that may be congested, whose outputs are unknown, and so on. The second step engages in an iterative process to resolve the dependencies. Within each iteration each flow variable starts with an estimated value. Equation (1) is applied at any port that maybe congested, but the updated output values are not considered as new input values during that same iteration. At the end of the iteration we ask whether any estimated value has changed enough to warrant iterating again. The solution is considered to have converged when further application of equation (1) does not change any flow value by more than a tiny fraction of a relative percent. The third stage of the algorithm consists of taking the converged values and pushing them through the rest of the network not involved in the cycles.

Experiments with this technique reveal the capability to compute flow rates very fast on large networks. Challenges remain however in effectively parallelizing the technique for very large networks that must be simulated in faster than real time.

3 SIMULATION OF ROUTING

As the size of the simulated network is scaled up, simulation of routing becomes a critical problem. Consider the following. The Internet routing system at the inter-domain level is a distributed system of massive size, consisting at present of more than 16,000 Autonomous Systems (independently administrated networks). Some of these Autonomous Systems (ASes) have hundreds of inter-domain routers (devices participating in the inter-domain routing system). Each AS typically also contains a large number, sometimes thousands, of additional routers for internal routing (intra-domain routing). During periods of rapid expansion of the Internet the growth in size of routing tables has threatened to exhaust the memory available in individual routers, particularly in core routers in the large backbone Internet Service Provider (ISP) networks that store routes to all destination networks in the Internet. Consider then the implications for memory space demands of a simulation consisting of thousands of such routers! Moreover, the computational time spent on generating the routes for a large-scale simulation can quickly become a problem.

To some extent these issues can be alleviated through parallelization of the simulation, mimicking the way that the real system distributes storage space and computations. However, alternative algorithmic techniques to reduce the space and time complexity of simulation of routing can produce large gains, with efficiencies beyond what is possible to achieve through more hardware resources alone. There has been a significant amount of work devoted to dealing with the problem of simulating routing, and this section we will review some of this work.

3.1 Internet Routing

For completeness we provide a very brief review of Internet routing here. However, space constraints preclude more than a high-level treatment. The Internet routing system is hierarchical and operates on two levels. Routes in a local network under a single administrative control (intra-domain routes) are computed based on shortest paths (weighted hop counts) through protocols such as Open Shortest-Path First (OSPF), Enhanced Interior Gateway Routing Protocol (EIGRP), and the Routing Information Protocol (RIP) (Huitema 2000). Link state protocols, such as OSPF, exchange topology information between routers so that all routers have a complete view of the full (local) network topology and can independently compute the shortest paths through the network. In distance-vector protocols, such as RIP and EIGRP, on the other hand, routers exchange information with their neighbors on the shortest known path to a destination at the given time (current preferred choice). The routing computation eventually converges to a stable state as the best choices stabilize. Links can be assigned weights so that the shortest path computation can be controlled to, for instance, steer traffic through higher capacity links, or balance traffic volumes over different links ("traffic engineering").

To provide global connectivity, multiple networks (ASes) must collaborate in exchanging traffic. Two ASes that agree to exchange traffic (inter-domain) do so by setting up one or more peering points. Routing at the inter-domain level (across ASes) is performed and controlled differently from intra-domain routing. Traffic exchange between different networks is determined through business agreements, and the routing protocol enforces certain constraints according to those agreements through routing *policies*. For instance, typical business relationships between networks include *customer/provider* and *peer/peer*. A customer buys Internet access from an ISP, implying that all routes learned to other destinations in the Internet by the ISP should be made available (be *advertised*) to the customer. Similarly, routes to the customer networks will be advertised to the rest of the Internet. In a peer/peer relationship, two network operators decide that it would be mutually beneficial to exchange traffic (thus generally without charging). For instance, two regional ISPs, call them A and B, may directly exchange traffic between their customers instead of passing it to their upstream providers. In this situation A advertises routes from its customers to B and accepts routes for B's customers re-advertising them to its customers. However, A does not want to advertise routes learned from its upstream provider to B and does not advertise B's routes to its provider, since this would result in A providing transit service for B. The Border Gateway Protocol (BGP) (Stewart 1998) is the de-facto standard for inter-domain routing in the Internet,

and it allows the specification of routing policies to control route preference, whether or not routes are permitted to be used, and whether they are readvertised. BGP is a *path-vector* protocol and its operation is similar to distance vector protocols in that it works by exchanging information on the preferred routes with its neighbors and has to converge to a stable state. BGP routes thus advertised to neighbors incorporate several *route attributes*, including the full AS path traversed by the route, to be used by route filtering and selection mechanisms in enforcing the configured routing policies. At the inter-domain level, destinations are specified in terms of subnetwork addresses, IP prefixes. The global Internet routing table currently contains more than 160,000 such IP prefixes.

3.2 Incorporating Routing Protocols

Network simulators such as SSFNet (SSFNet 2000) include detailed models of routing protocols like OSPF and BGP. By incorporating all relevant details of the routing protocols, including routing packet format/contents, routing information storage, and the route computation/selection mechanisms and timers in the protocol, a high fidelity model of the routing protocol operations is achieved. Similar to this approach is taking original routing code and porting it to run in a network simulator, as described in (Dimitropoulos and Riley 2003), where the Zebra open source routing software was ported to run in the ns-2 simulator. This brings the simulator to within a very close approximation of the actual system. Such models capture not only the accurate path choices but also the dynamics in adapting to topology changes (e.g., device- or link failures), and the detailed model in SSFNet has been used for instance to study the convergence behavior of BGP (Griffin and Premore 2001, Hao and Koppol 2003).

However, these approaches suffer from exactly those scalability problems previously described. Typical implementations of the BGP protocol require storing not only one routing table, but a complete Routing Information Base (RIB), i.e. a full routing table, for each one of its peers (neighbors) to keep track of routes learned from each one. In fact, it generally requires a second RIB per peer to keep track of routes being advertised to that same neighbor. Consequently, the storage requirements grow approximately as $O(n \cdot p \cdot r)$, where n is the number of BGP routers, p is the average number of peers and r is the number of prefixes in the routing tables. A simulation of the Internet core incorporating thousands of routers, with perhaps ten or more neighbors on average, and with full routing tables, requires storing on the order of $1,000 \cdot 10 \cdot 160,000 = O(10^9)$ routes.

Having a single global memory space for a simulation can be exploited in the implementation to reduce the memory usage. Hao and Koppol (2003) describe modifications to a BGP model, using a global RIB (for all routers in the network) with pointers to shared instances of route representations to remove redundancies in storage of AS path information. In simulations of large AS topology graphs, with a single BGP router representing an AS, they report reducing memory usage by up to $\approx 75\%$ (in converged state) for synthetic topologies of 16,000 ASes.

3.3 Simulation Accuracy: Dimensions

All abstractions, and thus all simulation models, involve some amount of approximations. Several studies have explored algorithmic techniques to reduce the cost of routing in simulations, generally based on some form of approximation. We can consider fidelity along a few dimensions:

Fidelity of paths: or what we might call the *routing semantics*. To what extent do the paths produced in the simulation correspond to paths in reality? For instance: *i)* completely accurate paths incorporating BGP policy decisions at the inter-domain and shortest paths at intra-domain level, *ii)* only shortest paths, or *iii)* approximations of shortest paths. Note that to achieve completely accurate paths it is necessary to not only have accurate models of the routing system, but also of the policies, i.e., the link weights and BGP configuration policies. This information is not generally known, although some efforts have been made to infer it (Mahajan et al. 2002, Gao 2001, Subramanian et al. 2002).

Dynamics: does the model include the convergence behavior of the protocol? That is, does it model when the routes become available (or the existence of transient routes) as well as the resulting stable routes?

Forwarding mechanism fidelity: some models modify the forwarding mechanism used in order to achieve efficiency gains.

3.4 Reducing the Cost of Shortest Path Routes

Several studies have, in particular, started from the assumption of shortest path routes (used exclusively) and considered ways of making this more efficient. If we can assume that: *i)* the topology does not change (e.g., no device or link failures) and *ii)* the routing dynamics are not of interest or have negligible effect, then we may compute equivalent routes through other mechanisms than the exact protocols used in reality. For instance, using global topology knowledge it is straightforward to implement a centralized shortest path algorithm without exchange of routing state packets in the model, which can save considerable simulation time. However, the routing storage space requirements remain high. A flat routing scheme requires $O(N^2)$ storage for shortest path routes between all pairs of nodes. Hierarchical schemes can reduce this, depending on the topology and the number of levels of hierarchy.

Substantial savings can be obtained by recognizing that in most scenarios only a small fraction of all the possible routes that get computed actually get used by traffic. In other words, concurrent all-to-all communication does not occur in the Internet. (Certain kinds of malware propagation result in widespread broadcasts of traffic, but will still be limited by the software configurations that are not vulnerable to a given attack.) Riley, Ammar, and Fujimoto (2000) made this observation and proposed computing routes on-demand to save routing storage space. In their Neighbor-Index vector (NIx-vector) routing scheme, they compute shortest path routes when they are needed and generate a source route consisting of neighbor indexes (essentially pointers to the next outgoing interface), thereby avoiding forwarding table lookup costs. The source routes are cached at the source nodes for reuse. With N nodes, k pairs communicating, and an average node outdegree of d, the routing storage requirements grows as $O(kN \log d)$. Hence, as long as k is small ($k \ll N$) and $d \ll N$ we can get large savings. However, in the worst case, i.e., for all-to-all communication, it results in $O(N^2)$ storage, just as for full flat routing storage. Implementing this scheme in the ns-2 simulator, they report memory savings up to $\approx 90\%$ on random "Transit-Stub" network topologies generated by GT-ITM (Zegura et al. 1996).

If we allow ourselves to use approximations of shortest path routes, other approaches are possible as well. Huang and Heidemann (2001) devised a scheme (*algorithmic routing*) using a k-ary spanning tree, where route lookups are made through a tree walk, to achieve $O(N)$ storage for approximate shortest path routes. Briefly, the network topology is mapped to a k-ary tree through a Breadth First Search (BFS) traversal, during which the node addresses are mapped to addresses giving their position in the tree. This address mapping is the only storage required by the algorithm and what gives it the $O(N)$ space complexity.

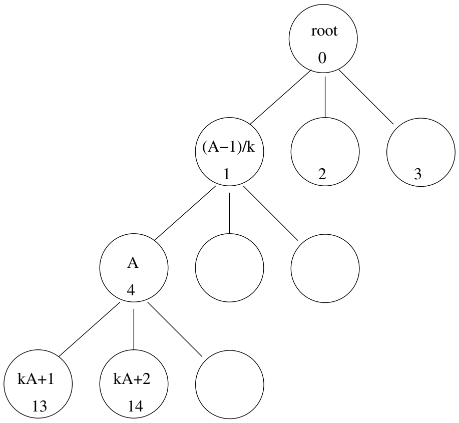

Figure 1: Example of k-ary tree for next hop computation.

One can then compute the next hop in the tree based only on the new addresses of the current node and the destination in $O(\log_k N)$ time. Figure 1 illustrates a tertiary (3-ary) tree. From an arbitrary node A, any destination node B is reached through next hops $(A-1)/k$ (up towards the root), $kA+1$ (down left), $kA+2$ (down middle), or $kA+3$ (down right). Starting from B, iteratively compute its parent as $(b-1)/k$, with initially $b=B$, moving towards the root. If passing one of A's children before reaching the root, this is the next hop. Otherwise route towards the root. Hence, each next hop is computed by traversing the tree. Using random network topologies of up to 500 nodes generated by GT-ITM, they report significant savings in overall memory and simulation time in experiments, with path length inflation errors below 10% for more than 80% of the routes. However, the approximation occasionally leads to significant errors.

By storing multiple spanning trees for a topology and combining it with a negative cache for "exception routes", Chen et al. (2004) are able to avoid approximation errors while still making large savings compared to $O(N^2)$ storage. They use a heuristic based on regression modeling from experimental results to determine the appropriate number of trees resulting in a good balance between tree storage and negative cache size. Using their ModelNet network emulator, they report routing memory savings of up to 90% for power-law network topologies generated by the BRITE generator's B-A algorithm implementation (Medina et al. 2001). They use pre-computed source routes to eliminate forwarding table lookup costs and, to also keep route lookup cost low they added a positive route cache which reduced the memory savings somewhat. The flat routing structure and use of source routes and caches thus resembles some aspects of the implementation of the NIx-vector scheme.

3.5 Dealing with Inter-domain Routing Policies

As already described, using shortest path routes exclusively in a model is an approximation of the two level hierarchical routing system used in the Internet. Several Internet measurement studies have studied the impact of policy based routing in terms of the *path inflation* it causes, i.e. essentially the sub-optimality in path length compared to shortest hop-count routes (Tangmunarunkit et al. 2001, Spring et al. 2003) at the router adjacency level. They report that path inflation due to inter-domain policies is significant, inflating some 20% of the routes by 50% or more. A more complex question is what the ultimate impact might be on the output of a simulation from routing approximation, i.e. how do path inaccuracies translate into errors in the output. For instance, an unrealistic route leading to deviations in the load on certain bottleneck links might be a greater concern than differences in the length of the paths chosen. However, this will depend on the particular scenario and since it is difficult to predict the result of this type of deviations it is desirable to avoid them.

BGP theory: Computing BGP (policy based) routes is significantly more costly and a fundamentally different problem from computing shortest path routes. The computation performed by a collection of BGP speaking routers is fundamentally different because *i)* path ranking is not based on any universal cost function and *ii)* policies may reject paths (including the shortest path) (Griffin et al. 2002). Instead, Griffin, Shepherd, and Wilfong (2002) model the computation performed in BGP through what they call the "*Stable Paths Problem*". Varadhan, Govindan, and Estrin (1996) showed that there exists combinations of policies that can cause BGP to diverge, i.e. cause persistent route oscillations. Griffin, Shepherd, and Wilfong (2002) provided conditions under which policy configurations are guaranteed to converge, but also showed that determining whether a given policy configuration will converge is an NP-complete problem.

Consequently, if we try to compute routes on-demand, for instance, divergence of the routing system would lead to non-termination of the on-demand route computation algorithm; clearly an unacceptable situation. However, Gao and Rexford (2001) created a set of guidelines for the choice of route preferences in BGP policies and showed that if these guidelines are followed, convergence of the BGP routing system will be guaranteed. They argued that their guidelines conform to preferred policies from an economic perspective, and conjectured that most network operators thus follow the guidelines and this might be why there have not been any major instability events observed in the Internet. A later measurement study (Wang and Gao 2003) found support for this conjecture by inferring policies from collected BGP routing data.

Simulation: Liljenstam and Nicol (2004) proposed constraining the permitted policy configurations according to Gao and Rexford's guidelines and exploit the resulting properties for efficient on-demand computation of BGP consistent routes in simulations. Specifically, by using properties resulting from the guidelines it is possible to compute the final converged BGP routes (on-demand) without emulating the whole BGP convergence process. We briefly describe their *Policy-Aware On-demand Routing* (PAO-routing) scheme, which has been implemented in the SSFNet (SSFNet 2000) and RINSE (iSSFNet) (Liljenstam et al. 2005) network simulators.

As described in Section 3.1, customer/provider and peer/peer are typical peering relationships between ASes exchanging traffic. The simplest version of Gao and Rexford's guidelines, *Guideline A*, states that an AS must always prefer to route traffic directly to its customer ASes before its peers or providers. Consider the Directed Acyclic Graph (DAG) formed by connecting customer ASes with directed edges to their providers, as in Figure 2. Starting from a destination customer AS, a in the figure; if route selections are done in one AS at a time in an order that obeys the partial order of the DAG, it can be guaranteed that the chosen routes are stable and the choice not need to be changed. In the example graph this means traversing ASes a, c, d. Once the top-level provider(s) have been reached, this is the end of phase 1 (outline in the figure). In phase 2, Peer/peer relationships (undirected edges) and provider/customer directed edges are traversed from provider to customer (backwards), so that the routing information will reach all ASes. The second phase makes use of the same partial order, but in the opposite direction. Peer/peer relationships have certain properties related to their traffic restrictions (Section 3.1) that make it possible to incorporate them in the scheme although they contribute undirected edges.

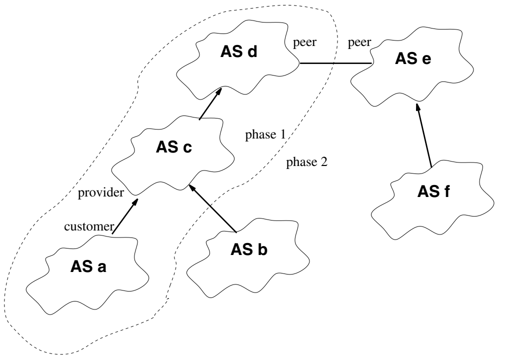

Figure 2: Example of AS relationship DAG.

The PAO-Routing algorithm exploits the partial ordering to choose stable routes immediately, thus avoiding emulating BGP convergence. However, there are many additional details that are outside the scope of this paper. In addition to traversing the ASes, the algorithm must compute intra-domain routes on the fly as routing information is propagated through the graph, and the internal exchange of BGP routes inside ASes must be handled. Intra-domain routes are computed using Dijkstra traversals of each AS graph from the origin (destination) device or from each BGP router (entry point) where external BGP routes enter the AS. Thus, the internal routing distances can be computed as the BGP routing information is propagated through the graph.

The algorithm was validated by comparing the routes it produced with routes produced by SSFNet's detailed BGP model in AS topologies sampled from a model of the real AS topology build from BGP routing data (University of Oregon Route View Project 2004). Order of magnitude reductions on memory and simultaneous gains in simulation time were reported in experiments using sampled AS topologies and a model of the U.S. Internet backbone (Liljenstam et al. 2003). As with other on-demand routing schemes the savings depend on the traffic pattern (preferably significantly sparser than all-to-all), and route caching techniques need to be employed to avoid redundant recomputations.

3.6 Some Remaining Issues

Several of these techniques were based on the assumption that the network topology is constant, i.e. there are no events such as device or link failures that alter the connectivity in the network during the simulation. This assumption can be relaxed somewhat. Riley, Ammar, and Fujimoto (2000) describe the possibility of using topology version numbers in the NIx-vector scheme to keep track of which topology a cached route was referring to. As topology changes take place the topology version is updated and routes that are found to be outdated need to be recomputed. However, this essentially amounts to starting over with a clean slate whenever a change occurs. If a link or device goes down it is not difficult to determine which source routes make use of the affected device and should thus be invalidated and recomputed. On the other hand, if a previously "down" device (or link) comes back up it is very hard to determine which routes should be recomputed. Essentially, it requires determining which routes should use a new more preferred path. The PAO-routing mechanism in iSSFNet operates accordingly. It selectively invalidates cached route entries when devices go down, but invalidates all routes when a device comes back up. Finding a more efficient way of dealing with topology changes appears non-trivial and is an open problem.

A related problem concerns accounting for routing dynamics in models that compute routes without simulating the complete operations of the routing protocols. There will be a delay from a topology change event until the routing protocols can converge on new routes. In the meantime traffic may be lost as it is forwarded along the old path. If it is possible to estimate the convergence time of the protocol it could be useful in modeling the time until new paths become available and the effects it has on the traffic. However, efficient methods for coming up with such estimates is currently also an open problem.

4 NETWORK EMULATION AND REAL-TIME SIMULATION

Simulation provides a convenient test bench for future and existing network protocols and services. However, several important issues must be addressed properly for large-scale network simulation to become an easily adopted tool for network researchers. The first and foremost issue is to contain the complexity of the simulation models. Although this problem is partially solved by most network simulators nowadays offering software modules that implement various common network protocols and services, it is nonetheless a daunting task when it comes to implementing new network protocol or modifying an existing model. Creating a network model that can deal with subtle treatment of protocol specifications and provide a proper mix of various implementations as in the real world (e.g., different TCP versions) is a non-trivial issue, letting alone that an implementation of an advanced network protocol (such as the Border Gateway Protocol) is typically quite an involved process itself. The complexity of the models further aggravates the concern of validity of those used in simulation. Validation of large-scale network simulation models tends to be an elusive undertaking—there are many model parameters and design nuances that could potentially spoil the final simulation outcome. Another important issue is the inspection of the simulation result. As large network models generate large amount of state information, it is extremely important to allow the modelers to efficiently plough through the data and filter out information necessary to make prompt decisions on further simulation investigations.

Emulation and real-time simulation (with distinctions described below) offer a solution. The major difference between network simulation and emulationis that the former is purely virtual, whereas the latter focuses on real-time interactions with real applications. A network simulation contains only software modules representing various network entities (routers, links, protocols, etc.). Network operations, such as packet forwarding, are logical operations, as opposed to physically moving network packets from router to router. More important, the time advancement in simulation bears no direct relationship to the wall-clock time. Network emulation or real-time simulation, on the other hand, focuses on interactions with real applications, such as distributed network services, web servers and web clients, and distributed database systems. These real applications, for example, can generate real traffic on the virtual network, where packet delays and losses are *calculated* as a function of the simulated network condition. Because the virtual network interfaces with real applications that operate in real time, the network emulation and real-time simulation must execute in real time.

Network emulation and real-time simulation provide a certain degree of realism because a subsection of the system involves real code, which in part lifts the concern about the validity of simulation. In addition, as the system progresses in real time, the modelers are given the opportunity to interact with the network model on-line, by observing the system state, tuning the model parameters, and further readjusting the system from feedbacks—all in real time. This interactive process is of important value to both network researchers and network operators, who, for instance, want to study the network responses to cascading network events such as the effect of Internet worm propagation and the strength of security counter measures.

4.1 Current State

Over the years, we have seen many network emulators with varying degrees of complexity, ranging from single-link traffic modulators to full-scale network emulation testbeds (Ahn et al. 1995, Carson and Santay 2003, Davies et al. 1995, Herrscher and Rothermel 2002, Huang et al. 1999, Peterson et al. 2002, Raychaudhuri et al. 2005, Rizzo 1997, Vahdat et al. 2002, White et al. 2002, Zheng and Ni 2003). In general, the network emulators can be divided into three categories: link-centric, node-centric, and network-centric, although the borderline between them is not a clear-cut. The link-centric network emulators are traffic modulators particularly designed for a single communication link or a small set of links. For example, in dummynet (Rizzo 1997), a link is represented as a finite queue that imposes bandwidth limitations and communication delays. Packets are intercepted at the protocol stack of the target host and pushed through the finite queue to simulate desired link state—packets may be reordered and dropped at random to model the unreliability of the target link. Node-centric network emulation focuses on kernel virtualization. For example, ENTRAPID (Huang et al. 1999) provides a protocol development environment, in which a “virtualized” networking kernel (including all essential networking services of a BSD Unix kernel) is implemented in the user space. In essence, ENTRAPID is a “network in a box”: protocols are implemented in the user space interacting with the standard network services for communication. In a similar vein, Netbed (White et al. 2002), which is a descendent of Emulab, provides an integrated environment for network experimentation with necessary support for coordinating and dedicating heterogeneous resources (including both computers and network resources) to individual users for distributed experiments. This kernel virtualization idea is further extended in PlanetLab (Peterson et al. 2002), in which a distributed resource management scheme is applied to select a set of distributed nodes (called a *slice*) that form a global overlay network for each distributed application. In addition, a resource control mechanism on each node ensures the application can receive the promised computation, communication, and storage resources in a multiplexed operating environment. Using a similar idea for wireless mobile networks, ORBIT (Raychaudhuri et al. 2005) provides a software infrastructure to support experimental research on an array of physical radios, deployed either as an indoor radio grid or an outdoor field network. Network-centric emulation focuses on the virtualization of the network. For example, ModelNet (Vahdat et al. 2002) is an extension of the dummynet that targets scalable emulation of a large-scale network. Unmodified network applications are multiplexed and mapped to physical edge nodes, which are connected to the core nodes that emulate network operations.

Most of these network-centric emulators are time-driven, which we simply refer to in this paper as *network emulation*, as opposed to real-time simulation we describe below. For example, ModelNet stores the network packets in "pipes"—the packets move through the pipes representing the network paths taken by the packets to travel from the source of the transmission to the destination. These pipes are sorted in the system according to the scheduled delivery time of the earliest packets. The scheduler is invoked periodically (once every 100 μs) to emulate the packets passing through the network. Some of these emulators, such as ModelNet (Vahdat et al. 2002) and EMPOWER (Zheng and Ni 2003), even support scalable emulation on parallel computer platforms.

There are three main drawbacks associated with the time-driven approach. First, the accuracy of the network emulation depends on the time granularity of the scheduler. The scheduler, which must be executed periodically to model network operations, is limited by the resolution (or time quantum) of system interrupts, which in turn depends on the target machine and the operating system. Second, network emulation cannot be extended easily to model behaviors of low-layer protocols, such as MPLS on ATM. All network emulators we encounter treat the IP packet from real applications as an indivisible transmission unit. The time-driven approach does not provide sufficient support for the fine-grained time advancement necessary to model the lower protocol layers. Last, to our knowledge, so far there has not been a good model for background traffic in network emulation. Background traffic has a strong effect on the global network behavior and plays an important role in determining the end-to-end behavior of the (foreground) applications. The lack of a good background traffic model can severely impair the accuracy of a network model.

Real-time network simulation, or simulation-based emulation, which we define as an event-driven network-centric emulation, offers both the flexibility of running detailed network models and the capability of interfacing with real applications. This technique enables us to study both application and network behaviors with more realism. In the same framework, one can both study the performance of network applications, such as a high-performance middleware that interconnects distributed applications, under controllable and repeatable networking conditions, and evaluate the characteristics of network protocols and services under realistic application traffic patterns. Existing real-time network simulators include NSE (Fall 1999), IP-TNE (Bradford et al. 2000, Simmonds and Unger 2003), MaSSF (Liu et al. 2003), and Maya (Zhou et al. 2004). NSE is an emulation extension of the popular *ns-2* simulator (Breslau et al. 2000) with added support for connecting with real applications and scheduling real-time events. IP-TNE is an emulation extension of a parallel network simulator called IP-TN. IP-TNE is the first simulator we know that applies parallel discrete-event simulation techniques for large-scale network emulations. MaSSF is built on DaSSF (Liu and Nicol 2002) with emulation support for the grid computing environment. Maya is an emulation extension of the commercial QualNet simulator for wireless mobile networks (Scalable Network Technologies 2000).

Both network emulation and real-time simulation need to resolve two important and related issues: responsiveness and timeliness. Responsiveness dictates that the emulation system must be able to interact with real applications in time. That is, the system interface should be able to receive real-time events promptly and output them according to the real-time deadlines. Timeliness refers to the system's ability to keep up with the wall-clock time. That is, the simulation must be able to characterize the behavior of the large-scale network, potentially with millions of network entities and with representative network traffic, in *real time*. Failing to do so will introduce timing faults, where the simulation fails to process events before designated deadlines. An elevated occurrence of timing faults will cause the simulator to become less responsive when interacting with real applications. In the remainder of this section we briefly describe techniques developed to factor out these issues.

There are several ways to incorporate real applications into the emulation environment, the decision of which to use largely depends on where the interactions take place. Several techniques can be applied to run unmodified software, including kernel virtualization (Huang et al. 1999), packet capturing (Bradford et al. 2001), dynamic linking library (Liu et al. 2003), and binary executable modification (Varadarajan 2004). In certain cases, moderate software modifications are necessary to allow direct execution. For example, Nsclick directly executes the Click Modular Router inside the *ns-2* network simulator (Neufeld et al. 2002). Dimitropoulos and Riley (2003) incorporate a public-domain implementation of BGP from the routing software called Zebra into simulation. Liu et al. (2004)

manage to run several routing algorithms inside a parallel wireless network simulator called SWAN.

The performance of a large-scale network simulation must be able to keep up with the wall-clock time so that the system can interact with real applications in real time. To this end, two techniques have been exploited: parallel and distributed simulation (Liljenstam et al. 2005, Simmonds and Unger 2003, Zhou et al. 2004) and multi-resolution modeling (Kiddle et al. 2003, Nicol et al. 2003, Zhou et al. 2004). Simmonds and Unger (2003) apply parallel discrete event simulation to emulation and they observe that only the components interacting with real applications are subject to real-time constraints and the rest of the system does not have to run as fast. The observation is important as it leads to better synchronization algorithms that can take advantage of the inherent asynchrony in the system. Our RINSE simulator (Liljenstam et al. 2005) extends this idea by enacting a priority-based scheduling policy to distinguish between real-time and regular event processing in the parallel simulation kernel. The multi-resolution modeling technique can also be used to achieve real-time performance and is described in more detail in section 2.

4.2 RINSE: A Real-Time Simulator for Network Security Exercises

RINSE stands for Real-time Immersive Network Simulation Environment and is being developed for large-scale network security exercises and training purposes (Liljenstam et al. 2005). The idea behind RINSE is to have a virtual common network playground for assessing community preparedness and training against network failures and malicious attacks. Participating organizations, which most likely are geographically distributed, monitor and control portions of the virtual network. They can deploy security defense measures against simulated network anomalies, such as a distributed denial of service attack. This application highlights the need for a scalable real-time simulation of large-scale networks: the players participating in the security exercise and training must be provided with a live feed of the network state, while decisions must be made in real time to alter operations (such as activating specific packet filters on routers) in the virtual network.

In many aspects, RINSE epitomizes the state of real-time network simulations. RINSE is built on a parallel discrete-event simulation kernel, which is a C++ implementation of the Scalable Simulation Framework (Liu and Nicol 2002). The parallel simulation kernel handles synchronization and communication between instances of the simulator running on a parallel and distributed environment. To enable interaction with real applications running at the client sites (one for each participating organization of the exercise), the simulation kernel is augmented to include real-time simulation support. In SSF, `inChannel` and `outChannel` objects are defined as communication end-points between entities: `inChannel` is used to receive timestamped events from other entities, while `outChannel` is used to send events to other entities. We extend the concept to allow the simulator to send and receive events from external physical devices. Packets are intercepted from a real network application running at the client site (using packet filters (McCanne and Jacobson 1993)) and sent to the machine running the simulator. A reader thread converts the packets to simulation events and injects them into the simulator's event-list using a well-defined function for the `inChannel` object. When a network packet needs to be exported to a real network application at the client, the simulator hands the corresponding simulation event to the `outChannel` object, which is responsible for converting the event to a network packet and sending the packet back to the client site upon a designated real-time deadline. The real-time deadline is calculated from the virtual time of the event and the emulation throttling speed, used for regulating simulation execution speed with respect to the wall-clock time. A writer thread is designated to deliver the packet upon the deadline.

Inside the parallel simulation kernel, we use a priority-based scheduling algorithm to process events—events with real-time constraints (those that are imported from or about to be exported to external physical devices) are given higher priority over regular simulation events. Whenever a real-time event is scheduled for execution, the scheduling algorithm for logical processes interrupts the normal event processing and makes a context switch to load the logical process containing the real-time event and execute it.

Since real applications are running at the client side, which are most likely distributed geographically, the network latencies from the physical connections between the real applications and the simulator must be properly accounted for, especially for those applications (such as the `ping` command) that are sensitive to such latencies—the latencies may contribute a significant portion of the total round-trip delay. Our solution is to hide the latencies *inside* the network model. In simulation, delays are imposed upon simulated network packets moving from one router to another along the forwarding path. We modify the queuing behavior at the link layer to compensate for the latencies by shortening the packet's expected queuing delay. If the latency cannot be absorbed completely at the source router, the packet continues to carry the remainder of the latency as a deficit and tries to offset it at the next hop. The process continues until either the deficit is reduced to zero or the packet reaches its destination. Note that this scheme is an approximation, because we do not roll back an erroneous execution (caused by the simulator's sending a packet before the packet with a deficit arrives) if the deficit cannot be compensated at the first hop. The method, however, plays an important role for those latency-sensitive applications that require real-time interaction with the simulator. Future work is needed to quantify the effect of such inaccuracies to the overall simulation result.

We conclude this section by reflecting on a couple of future directions for network emulation and real-time simulation research. The ability of successfully scheduling real-time tasks hinges upon the quality of the workload prediction of various models in simulation, particularly those running on parallel platforms. This direction requires closer inspection of the models, which may include both packet-level and fluid traffic representations, as well as the fix-point calculations of network background traffic. Another research direction is to make the system more robust and more resilient in terms of keeping up with the real time. One idea is to dynamically change the mixture of modeling abstraction levels to avoid timing faults during transient system overloads or computing resource shortfalls. The ultimate goal of an emulation system like RINSE is to provide an immersive network simulation testbed that captures important characteristics of the global network and provides seamless interaction with distributed real network applications.

5 SUMMARY

This tutorial paper highlights three problem areas of current research interest in large-scale network simulation. We discussed approaches to abstracting network traffic, with attendant problems and solutions. We showed why routing simulations have scaling problems, and discussed some ways of alleviating the memory burden they impose. Finally, we discuss problems in integrating simulated components with real components. In all these areas we point out directions for future research.

ACKNOWLEDGMENTS

This resesearch was supported under Award number 2000-DT-CX-K001 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security or the Science and Technology Directorate. In addition this research was supported by SPAWAR contract N66001-04-C-6013. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

REFERENCES

Ahn, J. S., P. B. Danzip, Z. Liu, and L. Yan. 1995, August. Evaluation of TCP Vegas: emulation and experiment. In *Proceedings of the 1995 ACM SIGCOMM Conference*, 185–195.

Bradford, R., R. Simmonds, and B. Unger. 2000, August. A parallel discrete event IP network emulator. In *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'00)*, 315–322.

Bradford, R., R. Simmonds, and B. Unger. 2001. Packet reading for network emulation. In *Proceedings of the 9th International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, 150–157.

Breslau, L., D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. 2000. Advances in network simulation. *IEEE Computer* 33 (5): 59–67.

Bu, T., and D. Towsley. 2001, June. Fixed point approximations for TCP behavior in an AQM network. In *Proceedings of ACM SIGMETRICS 2001*. Cambridge, Massachusetts.

Carson, M., and D. Santay. 2003. NIST Net: a Linux-based network emulation tool. *SIGCOMM Computer Communication Review* 33 (3): 111–126.

Chen, J., D. Gupta, K. Vishwanath, A. Snoeren, and A. Vahdat. 2004. Routing in an Internet-scale network emulator. In *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

Davies, N., G. S. Blair, K. Cheverst, and A. Friday. 1995. A network emulator to support the development of adaptive applications. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location Independent Computing*, 47–55.

Dimitropoulos, X., and G. Riley. 2003. Creating realistic BGP models. In *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

Fall, K. 1999, July. Network emulation in the Vint/NS simulator. In *4th IEEE Symposium on Computers and Communications (ISCC'99)*, 244–250.

Gao, L. 2001, Dec. On inferring automonous system relationships in the Internet. *IEEE/ACM Transactions on Networking* 9 (6): 733–745.

Gao, L., and J. Rexford. 2001, Dec. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking* 9 (6): 681–692.

Griffin, T., and B. Premore. 2001, November. An experimental analysis of bgp convergence time. In *9th International Conference on Network Protocols (ICNP)*.

Griffin, T., B. Shepherd, and G. Wilfong. 2002. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Transactions on Networking* 10 (2): 232–243.

Hao, F., and P. Koppol. 2003. An Internet scale simulation setup for BGP. *Computer Communication Review* 33 (3): 43–58.

Herrscher, D., and K. Rothermel. 2002, October. A dynamic network scenario emulation tool. In *Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN'02)*, 262–267.

Huang, P., and J. Heidemann. 2001. Minimizing Routing State for Light-weight Network Simulation. *9th International Symposium on Modeling, Analysis and Simulation on Computer and Telecommunication Systems (MASCOTS)*, 108–116.

Huang, X. W., R. Sharma, and S. Keshav. 1999. The ENTRAPID protocol development environment. In *Proceedings of the IEEE INFOCOM 1999*, 1107–1115.

Huitema, C. 2000. *Routing in the internet, 2nd ed.* Prentice Hall.

Kesidis, G., A. Singh, D. Cheung, and W. W. Kwok. 1996, November. Feasibility of fluid-driven simulation for atm network. In *Proceedings of IEEE Globecom'96*. London, GB.

Kiddle, C., R. Simmonds, C. Williamson, and B. Unger. 2003. Hybrid packet/fluid flow network simulation. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS'03)*, 143–152.

Liljenstam, M., J. Liu, and D. Nicol. 2003. Development of an Internet Backbone Topology for Large-Scale Network Simulations. *2003 Winter Simulation Conference*, ed. Smith, Peters, White, Wilson, 694–702. New Orleans, LA.

Liljenstam, M., J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier. 2005. Rinse: the real-time immersive network simulation environment for network security exercises. In *Proceedings of the 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS)*.

Liljenstam, M., and D. Nicol. 2004. On-demand computation of policy based routes for large-scale network simulation. In *Proceedings of the 2004 Winter Simulation Conference*, ed. R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 215–223. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Liu, B., D. R. Figueiredo, Y. Guo, J. Kurose, and D. Towsley. 2001. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Proceedings of IEEE Infocom'01*. Anchorage, Alaska.

Liu, B., Y. Guo, J. Kurose, D. Towsley, and W. Gong. 1999. Fluid simulation of large scale networks: Issues and tradeoffs. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*. Las Vegas, Nevada.

Liu, J., and D. M. Nicol. 2002. Dartmouth Scalable Simulation Framework (DaSSF). <`http://alamode.mines.edu/~xliu/projects/dassf/index.html`>.

Liu, J., Y. Yuan, D. M. Nicol, R. S. Gray, C. C. Newport, D. F. Kotz, and L. F. Perrone. 2004. Simulation validation using direct execution of wireless ad-hoc routing protocols. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, 7–16.

Liu, X., H. Xia, and A. Chien. 2003. Network emulation tools for modeling grid behavior. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03)*.

Liu, Y., F. L. Presti, V. Misra, D. Towsley, and Y. Gu. 2004. Scalable fluid models and simulations for large-scale ip networks. *ACM Transactions on Modeling and Computer Simulation* 14 (3): 305–324.

Mahajan, R., N. Spring, D. Wetherall, and T. Anderson. 2002. Inferring link weights using end-to-end measurements. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 231–236. New York, NY: ACM Press.

McCanne, S., and V. Jacobson. 1993. The BSD packet filter: a new architecture for user-level packet capture. In *Proceedings of the 1993 Winter USENIX Conference*, 259–269.

Medina, A., A. Lakhina, I. Matta, and J. Byers. 2001. Brite: An approach to univeral topology generation. *9th International Symposium on Modeling, Analysis and Simulation on Computer and Telecommunication Systems (MASCOTS)*.

Neufeld, M., A. Jain, and D. Grunwald. 2002. Nsclick: bridging network simulation and deployment. In *Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM'02)*, 74–81.

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

Nicol, D. M., J. Liu, M. Liljenstam, and G. Yan. 2003. Simulation of large-scale networks using SSF. In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 650–657. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Nicol, D., and G. Yan. 2004. Discrete event fluid modeling of background TCP traffic. *ACM Transactions on Modeling and Computer Simulation* 14:1–39.

Nicol, D., and G. Yan. 2005. Simulation of network traffic at coarse time-scales. In *Proceedings of the 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 141–150.

Padhye, J., V. Firoiu, D. Towsley, and J. Kurose. 1998. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM'98*. Vancouver, CA.

Peterson, L., T. Anderson, D. Culler, and T. Roscoe. 2002. A blueprint for introducing disruptive technology into the Internet. In *Proceedings of the 1st Workshop on Hot Topics in Networking (HotNets-I)*.

Raychaudhuri, D., I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. 2005. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2005)*.

Riley, G., M. Ammar, and R. Fujimoto. 2000. Stateless Routing in Network Simulations. *8th International Symposium on Modeling, Analysis and Simulation on Computer and Telecommunication Systems (MASCOTS)*, 524–531. San Francisco, CA.

Rizzo, L. 1997. Dummynet: a simple approach to the evaulation of network protocols. *ACM SIGCOMM Computer Communication Review* 27 (1): 31–41.

Scalable Network Technologies 2000. <http://scalable-networks.com/>.

Simmonds, R., and B. Unger. 2003. Towards scalable network emulation. *Computer Communications* 26 (3): 264–277.

Spring, N., R. Mahajan, and T. Anderson. 2003. Quantifying the Causes of Path Inflation. Proceedings of ACM SIGCOMM'03.

SSFNet 2000. Available online at <www.ssfnet.org> [accessed July 10, 2005].

Stewart, J. 1998. *BGP4: Inter-Domain Routing in the Internet*. Reading, MA: Addison-Wesley.

Subramanian, L., S. Agarwal, J. Rexford, and R. Katz. 2002. Characterizing the Internet Hierarchy from Multiple Vantage Points. *IEEE INFOCOM*, 618–627.

Tangmunarunkit, H., R. Govindan, S. Shenker, and D. Estrin. 2001. The Impact of Routing Policy on Internet Paths. IEEE INFOCOM, 736–742.

University of Oregon Route View Project 2004. <http://www.routeviews.org/>.

Vahdat, A., K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. 2002. Scalability and accuracy in a large scale network emulator. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*.

Varadarajan, S. 2004. The Weaves runtime framework. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 10*, 197b.

Varadhan, K., R. Govindan, and D. Estrin. 1996. Persistent Route Oscillations in Inter-domain Routing. Univ. of Southern California Information Sciences Institute, Marina del Rey, CA, ISI Tech. Rep. 96-631.

Wang, F., and L. Gao. 2003. Inferring and Characterizing Interent Routing Policies. *ACM SIGCOMM Conference on Internet Measurement*, 15–26.

White, B., J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. 2002. An integrated experimental environment for distributed systems and networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, 255–270.

Yan, A., and W. B. Gong. 1998. Time-driven fluid simulation for high-speed networks with flow-based routing. In *Proceedings of the Applied Telecommunications Symposium'98*. Boston, MA.

Zegura, E. W., K. Calvert, and S. Bhattacharjee. 1996. How to model an internetwork. *IEEE INFOCOM*.

Zheng, P., and L. M. Ni. 2003. EMPOWER: a network emulator for wireline and wireless networks. In *Proceedings of the IEEE INFOCOM 2003*, Volume 3, 1933–1942.

Zhou, J., Z. Ji, M. Takai, and R. Bagrodia. 2004. MAYA: integrating hybrid network modeling to the physical world. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 14 (2): 149–169.

AUTHOR BIOGRAPHIES

DAVID M. NICOL is Professor of Electrical and Computer Engineering at the University of Illinois, Urbana-Champaign, and member of the Coordinated Sciences Laboratory. He is co-author of the textbook *Discrete-Event Systems Simulation*, and served as Editor-in-Chief at ACM TOMACS from 1997-2003. He was the General Chair of the 2005 Conference on Principles of Advanced and Distributed Simulation, and the General Chair of the 2006 Winter Simulation Conference. From 1996-2003 he served as the Editor-in-Chief of the ACM *Transactions on Modeling and Computer Simulation*. He has a B.A. in mathematics from Carleton College (1979), an M.S. (1983) and Ph.D. (1985) in computer science from the University of Virginia. His research interests are in high performance computing, performance analysis, simulation and modeling, and network security. He is a Fellow of the IEEE. His e-mail address is <nicol@crhc.uiuc.edu>.

MICHAEL LILJENSTAM is a Visiting Research Assistant Professor at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. From 2000–2003 he was a Research Associate at the Institute for Security Technology Studies and Computer Science Department, Dartmouth College. He has served on the program committee for the International Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) and the Conference on Principles of Advanced and Distributed Simulation (PADS). His research interests include large-scale network simulation, se-

curity, routing, and modeling and simulation of wireless networks. He received his M.Sc. (1993) and Ph.D. (2000) from the Royal Institute of Technology, Stockholm, Sweden. His e-mail address is <mili@crhc.uiuc.edu> or <michael.liljenstam@acm.org>, and his web page is <www.crhc.uiuc.edu/~mili>.

JASON LIU has been Assistant Professor of Computer Science at the Colorado School of Mines since 2004. Prior to that he was a post-doctoral student at the Coordinated Sciences Laboratory, at the University of Illinois, Urbana-Champaign. His research focuses on parallel discrete-event simulation, performance modeling and simulation of computer systems and communication networks, and large-scale simulation of wireless ad hoc networks. He received the B.A. in Computer Science from Beijing Polytechnic University in China in 1993, M.S. in Computer Science from College of William and Mary in 2000, and Ph.D. in Computer Science from Dartmouth College in 2003. His e-mail address is <xliu@mines.edu>.