Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

STAGED SIMULATION FOR IMPROVING SCALE AND PERFORMANCE OF WIRELESS NETWORK SIMULATIONS

Kevin Walsh
Emin Gün Sirer

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501, U.S.A.

ABSTRACT

This paper describes *staged simulation*, a technique for improving the run time performance and scale of discrete event simulators. Typical wireless network simulations are limited in speed and scale due to redundant computations, both within a single simulation run and between successive runs. Staged simulation proposes to reduce the amount of redundant computation within a simulation by restructuring discrete event simulators to operate in stages that precompute, cache, and reuse partial results. This paper presents a general and flexible framework for staging, and identifies the advantages and trade-offs of its application to wireless network simulations. Experience with applying staged simulation to the ns2 simulator shows that it can improve execution time by an order of magnitude in typical scenarios and make feasible the simulation of large scale wireless networks.

1 INTRODUCTION

The design and evaluation of distributed systems and network protocols relies to a large extent on network simulation. Traditional network simulators, however, do not run efficiently or scale well with increasing simulation size.

A significant source of inefficiency in discrete event simulators is redundant computation. We identify two different classes of redundancy in traditional discrete-event simulators. The first class of redundant computation occurs within a single run of the simulator. Traditional network simulators reevaluate complex functions whenever their results *may* have changed, even though in reality the results may have changed very little, if at all, since the last time they were evaluated. A second class of redundant computation stems from a lack of retained information between multiple runs of the simulator. Executing each simulation independently and without the benefit of past runs leads to computing many functions from scratch in each run. These two sources of redundancy pose significant bottlenecks for wireless network simulations, where network parameters change frequently.

This paper introduces *staged simulation*, a general technique to improve the scale and performance of wireless network simulation by exposing, identifying, and eliminating sources of redundant computation. Staging involves restructuring the events in a discrete-event simulator into an equivalent set of sub-computations, caching their results, and reusing them whenever matches are identified. We introduce three techniques, called *function decomposition*, *refinement*, and *batching* to complement function caching and improve its effectiveness. We apply these techniques both within a single simulation, a technique called *intra-simulation staging*, and between multiple similar runs of the simulator, called *inter-simulation staging*.

We have applied staging to the event processing engine of ns2 (VINT 1995), a well-established simulator whose design is typical of many discrete event simulators. Staging improved execution time by an order of magnitude over the standard ns2 implementation under typical simulation scenarios. As a natural consequence of eliminating redundant computation, staging in ns2 also reduced the running time from $O(n^2)$ in the size of the simulated wireless network to $O(n)$, making feasible large scale simulations with tens of thousands of nodes. Staging maintains strict compatibility with existing simulation scripts and extensions, with no loss in simulator generality or accuracy. More advanced and specialized simulation engines can benefit equally from staging. Specifically, we expect to see a comparable speedup and improvement in scalability in parallel and distributed wireless network simulators.

The contributions of this paper are as follows. First, we identify and expose a general technique for improving discrete event simulator performance. Second, we show how common simulation scenarios can benefit substantially from our optimization techniques. These benefits include drastically reduced simulator run time and good scalability without changing the simulator interface or degrading result accuracy. Finally, we validate our technique through systematic application to wireless simulation in a well-established network simulator.

2 THE STAGING APPROACH

The goal of staging is to eliminate redundant or nearly redundant computations in simulations. Traditional wireless simulators perform many redundant computations within a single run. Examples of common redundancies include sending packets along a particular path or computing neighbor sets. Similarly, across multiple runs of a simulator we find a large overlap in computation, especially when numerous runs of a simulation are made with only slightly varying parameters. For example, studies of proposed ad hoc routing protocols typically call for several sets of simulation runs, each set evaluating the effect of a single protocol or topology parameter (see, for example, Broch et al. 1998 and Royer and Toh 1999). In all, many dozens or hundreds of runs might be executed with very similar input parameters.

The simplest, most fundamental technique for eliminating redundant computations is function caching. This space-for-time trade off involves caching the results of idempotent functions and later reusing those results whenever the same function is invoked with the same inputs. While function caching forms the foundation for staging, it, by itself, is not sufficient to realize performance gains in practice. Typical events in discrete event simulators have time-varying, continuous inputs, which preclude matching function inputs between calls.

Staging significantly improves on function caching by introducing three techniques, called *function decomposition*, *refinement*, and *batching*. These techniques restructure computations such that their results are reusable even when a change in inputs would normally preclude reuse.

Function decomposition splits a large computation is split into several smaller sub-computations that are each dependent on only a subset of the inputs to the original computation. By carefully choosing the decomposition, we can reduce or eliminate the dependency on frequently varying inputs. For example, replacing a function $f(x, y, t)$ with an equivalent, decomposed version $f'(g(x, y), t)$ can allow $g(x, y)$ to be cached and reused even when the parameter t varies between calls.

Refinement further expands the applicability of function caching by taking advantage of the continuity of the physical model underlying the computation. When a small change in inputs is expected to lead to little or no change in the computed results, computing bounds then refining them to precise results can be more efficient than computing the same result from scratch. For instance, computing upper and lower bounds on node mobility may allow the simulator to eliminate costly computations to determine neighborhoods. In this case, the upper and lower bounds are computed such that they are valid for a range of inputs and so can be cached and reused even when inputs vary slightly between calls.

The third staging technique, *batching*, reorders the computations within the simulator so that many independent, fine-grained computations can be executed more efficiently in a single pass. Function decomposition and refinement both transform the event stream in a simulator into an equivalent, but much finer grained, sequence of computations. Many of these computations are not time dependent, and so can be reordered without affecting simulation accuracy. Batching groups related computations together, and replaces them with a single computation which computes all the needed results efficiently in a single pass. Batching not only allows the utilization of more efficient global algorithms instead of independent local computations, but can also improving processor and memory cache performance by improving locality.

Staging fundamentally involves a space-time trade off. For staging to be worthwhile, the target computation must be more expensive than the cost of storing and fetching cached results from a potentially large table. Additionally, the cached results will likely increase the amount of memory required for the simulation, due to the cost of storing the cached results. Although this increase in memory use may increase virtual memory paging by increasing the working set, it may conversely reduce the working set by eliminating memory intensive computations.

The remainder of this paper illustrates the use of staging in a widely used network simulator under typical usage scenarios. We give examples of existing, ad hoc applications of staging in current state of the art simulators, identify new opportunities for staging, and evaluate the effectiveness of both intra- and inter-simulation staging in a ubiquitous and mature network simulation engine.

3 TRADITIONAL WIRELESS SIMULATION

Efficient and scalable wireless network simulators are critical to network research, but present unique challenges in their implementation. They differ from other simulators in several key ways, each of which introduces redundant computation at runtime. As a result, many commonly used wireless simulators are slow and do not scale gracefully with network size.

The fundamental reason redundant computation is prevalent is that wireless mobile networks have highly dynamic characteristics, which imply that simulation state must be recomputed dynamically and often. As nodes move about a simulated field, the network-level topology may change rapidly. Link characteristics, routing information, and network topologies must be maintained and recomputed during the simulation, and mobile nodes must continually update their positions in order to provide accurate information to the network model. In addition, complex physical models make wireless simulation expensive. Since wireless is a broadcast medium, a straightforward simulation approach treats the network as a single broadcast LAN, incurring $O(n^2)$ run time in a network with n active nodes.

Existing wireless network simulators address some of the challenges of wireless networks. These range from general-purpose simulators, such as ns2 and OpNet (Chang 1999), to special-purpose and custom simulators including SWiMNet (Boukerche et al. 1999), MobSim++ (Liljenstam, Rönngren, and Ayani 2001), DaSSF (Liu et al. 2001), and GloMoSim (Zeng, Bagrodia, and Gerla 1998). These simulators have widely varying designs, including parallel or distributed event engines and specialized language features. Distributed simulators achieve scalability and performance by recruiting multiple simulator hosts. Even in such systems, each simulator host may perform a large amount of redundant computation that can be eliminated to improve efficiency.

We chose to study wireless simulation in the ns2 network simulator because it is widely used in academic research, and because it is has a well-established and validated set of protocols. The protocol implementations in ns2 total over 150,000 lines of code, and provide accurate models for node mobility, wireless energy consumption, radio propagation and MAC-layer protocols.

Ns2 tends to be slow and scale poorly with increasing number of nodes. As we show in the following sections, staged simulation can drastically reduce the amount of work required to simulate a wireless system by reducing redundant computation. These results are not specific to ns2, but can be applied likewise to more advanced simulation engines as well.

4 STAGED SIMULATION IN NS2

In the baseline ns2 implementation, the wireless physical layer and mobility models are the largest consumers of processing time in typical simulation scenarios. These components pose the most significant bottlenecks to efficiency and scaling. Consequently, we focus on staging computations related to node mobility and the wireless physical layer.

We incrementally describe four different types of staging, each employing a different approach to eliminating redundant computation. The first is an example of reusing common intermediate results across function calls. The second demonstrates the use of restructuring to enlarge the overlap in computation across calls. The third optimization illustrates precomputation as a staging technique, and the final one demonstrates inter-simulation staging by reusing results across multiple runs of the simulator.

4.1 Grid-Based Neighborhood Computation

For staging to be effective, redundant computations need to be readily identifiable. The monolithic structure of the default ns2 implementation, however, obscures the redundant computations it performs at runtime. Specifically, ns2 in particular, and wireless network simulators in general, perform numerous calculations to ultimately determine the set of nodes that will receive a given packet. These calculations depend on the positions of sending and receiving nodes, packet transmission and detection power levels, geography, and radio and antenna models. We note that many of these inputs will be identical or similar across computations, and show in Section 5 that the resulting redundant operations are significant and lead to non-linear scaling with network size.

To expose parts of this redundancy, we first apply a very simple grid-based staging approach where we reuse previously computed power levels for nearby nodes. We first divide the coordinate space into a grid of buckets, with each bucket holding a list of nodes positioned within the corresponding grid rectangle. This data structure can then be used to quickly determine if a group of nodes falls entirely outside the possible transmission range of a node, thereby eliminating the need to perform individual calculations for each node. Nodes in the remaining buckets, which may or may not be in range, are checked individually as before. In order to maintain the grid as nodes move during the simulation, we compute all of the times at which a node will cross a grid boundary, scheduling events at these times to update the grid as needed.

While grid-based decomposition in simulators is not novel, it serves as an initial application of staging that enables us to identify and eliminate other redundant applications through more advanced applications of staging in the subsequent sections. Nevertheless, grid-based neighborhood computation employs staging in two distinct ways. First, by grouping nodes into buckets, the simulator can reuse a single computed result for all nodes within the bucket. Furthermore, since the grid data structure will remain fixed across many packet transmissions, we can share and reuse a single global grid structure. We assume here, as is typical typical in ad hoc network research, that all nodes use uniform and constant transmission and reception parameters. This assumption does not present a limitation of the staged simulation approach, but simplifies our examples considerably.

4.2 Neighborhood Caching

Variations on the grid approach allow more advanced applications of staging using auxiliary computations to reduce redundancy in computation across packet transmissions. In typical simulation scenarios, inter-packet spacing is very short in comparison to the speed at which nodes move. Depending on node mobility and traffic patterns, many hundreds or thousands of packets may be transmitted from a single node before nodes move a significant distance. That is, we should expect the inputs to, and hence the results of, the neighborhood computation for a node to be reusable across many packet transmissions.

Since inputs will vary slightly, we should not expect the neighborhood set to be identical to that computed during the previous packet transmission. However, a conservative upper-bound, or superset, of the neighborhood set will remain valid for some time after it is computed, depending on the amount of node mobility and the tightness of the bound. This holds similarly for a lower-bound or subset of the neighborhood set. We therefore restructure the neighborhood set computation to first compute upper and lower bounds on the result, then refine these bounds into an exact result. After restructuring the computation, intra-simulation staging is used to cache and reuse the common intermediate results, the two bounds, across many packet transmissions.

This restructuring introduces one additional parameter, Δt, to control the caching policy. This parameter fixes the desired epoch duration for which the bounds on the neighborhood set will be valid. If s_{max} is the maximum possible node speed in the movement scenario, then the maximum change in distance between two nodes in an epoch is just $\Delta r = 2 s_{max} \Delta t$. If two nodes are within distance $r - \Delta r$ at some time, then they will remain within range r for Δt seconds into the future. Similarly, nodes beyond distance $r + \Delta r$ need not be considered at all for Δt seconds into the future.

We maintain a cache to capture the upper and lower bounds on the neighborhood set of each node. At most one cache entry is maintained for each node in the network. A cache entry, illustrated in Figure 1, is composed of an expiration time and two sets, $\mathcal{N}_{r-\Delta r}$ and $\mathcal{N}_{r\pm\Delta r}$, containing lists of the nodes within a ball of radius $r - \Delta r$ and those in the annulus with radii $r \pm \Delta r$. During packet transmission, the cache manager computes the set of nodes within range of a given node by first looking for a valid cache entry. Finding an entry that has not yet expired, it can immediately consider all nodes in the list $\mathcal{N}_{r-\Delta r}$ to be within range. The second list $\mathcal{N}_{r\pm\Delta r}$ is then scanned, and each node found to be within range is appended to the final result. At the same time, it can cheaply but conservatively update the lists, moving some nodes from $\mathcal{N}_{r\pm\Delta r}$ to $\mathcal{N}_{r-\Delta r}$ and eliminating others from $\mathcal{N}_{r\pm\Delta r}$ entirely. If, on the other hand, no cache entry is found during packet transmission, the cache manager consults the underlying mobility (grid) manager and constructs a cache entry with expiration Δt seconds into the future.

In the above caching scheme, there is some additional overhead during cache misses, when computing $\mathcal{N}_{r\pm\Delta r}$, since a larger radius is considered than previously necessary.

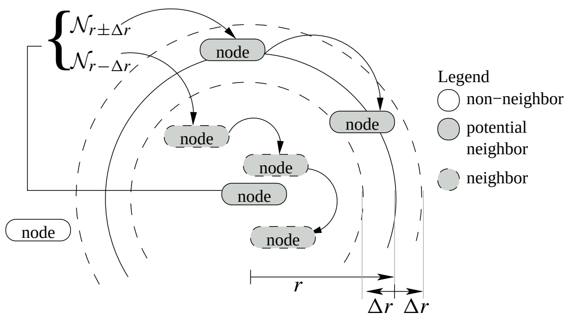

Figure 1: Computing Bounds on Node Movement Enables the Simulator to Examine Only the Nodes Located in an Annulus $\mathcal{N}_{r\pm\Delta r}$ During Packet Transmission by Node at Center

This overhead is controlled directly with the parameter Δt, which fixes the longevity and the accuracy of cache entries. In addition, there is overhead associated with scanning the list of nodes in $\mathcal{N}_{r\pm\Delta r}$ during each cache hit, but this is also limited by appropriately choosing the Δt parameter. We analyze these overheads in Section 5.

4.3 Perfect Caching

There is a large overlap in computation when constructing cache entries for nodes using the neighborhood caching scheme. We use precomputation to address this redundancy by computing many cache entries simultaneously. When constructing a cache entry, a node normally examines all nodes within a potentially large radius. If many nodes in a reasonably dense network are active, and each periodically construct cache entries on-demand and independently, each pair of nodes will eventually be considered twice.

A staged simulation approach, which we term *perfect caching*, eliminates redundancy by precomputing all cache entries simultaneously. This approach maintains the same data-structures as neighborhood caching. But, rather than calculating cache entries on-demand, it precomputes all cache entries at the beginning of every Δt epoch. All normal queries for neighborhood information are then guaranteed to hit the cache. There are several possible advantages to precomputation. First, we only need to examine each pair of nodes at most once, rather than twice, to compute all of the entries. Second, the positions of all nodes can be updated a single time at the start of the generation process. Previously, it was necessary to update the positions of all nodes within range of the sender during each cache miss. Finally, memory locality should improve when precomputing all entries simultaneously as compared to individually on-demand.

The overhead of this technique is a scheduled event during each Δt epoch, and possibly some wasted computation if some nodes do not send packets during an epoch, and thus do not use their cache entries. In a sparse or quiet network, perfect caching might construct more entries than needed during the simulation. This problem can be addressed directly by appropriately choosing the Δt epoch parameter.

4.4 On-Disk Caching

A final inter-simulation staging application improves on perfect caching, and demonstrates how staging can be applied across multiple similar runs of the simulator. The intra-simulation examples above reduce the amount of computation significantly, but also add some additional events to the event queue leading to more work in the event scheduler. Event queue management is a well-studied problem, especially in the particular case of the Calendar Queue used in ns2. However, we can eliminate the work done by many events by looking at a set of simulation runs together. This application of inter-simulation staging therefore builds on the previous optimizations by reducing the number of scheduled events generated by the grid manager and the cost of constructing neighborhood cache entries in the perfect caching scheme.

First note that, by itself, perfect caching generates strictly more events than the on-demand caching approach, and may actually compute results that are not used in any particular simulation run. But, also observe that perfect caching will perform identical work during multiple simulation runs using the same mobility scenario. In the second and subsequent runs of the simulator we can eliminate these extra events, as well as most cache maintenance, by writing all cache entries to disk every Δt seconds during the first simulator run. Subsequent runs can obtain cache entries from disk rather than maintaining an underlying cache manager or grid. This technique then introduces two phases. The *generation-phase* is identical to perfect caching except that all cache entries are spooled to disk. The *use-phase* does not maintain a grid, does not need to track changes to node positions, and requires no scheduler events. Instead, cache entries are read from disk serially as needed during packet transmission. A set of runs with the same mobility model will use the more expensive generation-phase for the first run, and the less expensive use-phase for all remaining runs.

5 EVALUATION

We have implemented each of the optimizations detailed in Section 4 in the ns2 simulator. We find that even the simplest application of staging reduces the run time of the simulator significantly, and allows for practical simulation of much larger network sizes than previously feasible. We show that more advanced intra-simulation techniques improve stability and robustness of the simulator, while the application of inter-simulation staging improves performance yet further. With the latest staged implementation, we regularly simulate networks of over 1000 nodes in the time it previously took to simulate networks of hundreds of nodes.

In addition to evaluating total simulation run time using our techniques, we also characterize the effect of each parameter we have introduced. For staging to be effective, it must be possible to easily or automatically find near-optimal choices for these parameters and, at the very least, avoid parameter choices that would lead to run time behavior worse than the default, non-staged implementation. We first describe our test environment and changes required to add staging to the simulator, then present the results of our staging techniques.

5.1 Evaluation Platform and Environment

We take as our baseline a modified ns2 version 2.1b9a simulator. All simulations were completed on a single-processor machine equipped with 1.7GHz Pentium 4 processor and 256MB of physical memory. Physical memory is an important constraint in ns2; more generous machines can simulate proportionally larger networks before becoming memory-limited. Before implementing our staging techniques, we made a few non-standard modifications to improve the baseline ns2 code. Most notably, we disabled all unused packet headers to reduce packet sizes and improve memory locality, and implemented more efficient packet tracing. This improved run time by 85% for a 250 node network. The performance results detailed in the this paper are computed relative to this optimized ns2 baseline implementation.

Staging can impact the performance of a simulator by introducing fine-grain events and changing the event distribution observed by the event scheduler. Calendar queue schedulers are particularly sensitive to such perturbations (Oh and Ahn 1999). To counteract the sensitivity of the calendar queue scheduler to the event distribution, we modified the calendar queue event scheduling algorithm to re-optimize the event queue after 30 seconds of simulated time, effectively avoiding occasional mis-predictions by the scheduler.

Overall, our simulation runs closely resemble those discussed in Broch et al. (1998), a very common setup. We used standard CMU Monarch mobility and communication model generators from the standard ns2 distribution. As an exemplar of typical wireless network research, we chose the AODV ad hoc routing protocol implementation included with ns2. Our results are not specific to these choices of application, mobility model, or communication pattern. These system parameters, summarized in Table 1, closely follow the standard values used in ad hoc networking literature. Although the nominal reception radius for our antenna model is only 250 meters, we use the transmission detection radius of 551 meters for all optimizations in order to properly account for interference effects.

Table 1: Default Simulation Parameters for Experiments

Network load	
model	Constant bit rate
concurrent data streams	30
packet size & rate	512 bytes $\times$ 8 packets/s
Node mobility	
model	random-waypoint
maximum node speed	5 m/s
pause time	10 s
field density	$\approx$ 31 nodes / km^2
Simulation	
routing protocol	AODV
simulation time	400 s

5.2 Simulator Performance

We first examine how the different applications of staging affect total simulation execution time using a 1000 node network. In this experiment, we fix grid granularity at 250 meters and Δt at 2 seconds, and later describe their selection and the sensitivity of staging to these parameters. We run our simulations with various applications of staging enabled, as shown in Table 2. For each level of staging, we run the simulator on five randomly generated networks and present the average of the execution times. The sample standard deviation for each data point is less than 0.2%.

Table 2: Levels of Ns2 Optimization for Experiments

Level	Optimizations
L_0	Ns2 baseline: improved tracing and packet size
Intra-simulation staging	
L_1	L_0 + Grid-based
L_2	L_1 + Caching
L_3	L_2 + Perfect caching
Inter-simulation staging	
L_{4a}	L_3 + On-disk caching (generation)
L_{4b}	L_3 + On-disk caching (use)

The speedup achieved by increasing levels of staging relative to the baseline simulator is shown in Figure 2. These results, obtained using a 1000 node network, highlight especially the benefits of the simplest intra-simulation staging L_1 technique and of the inter-simulation staging technique. Optimization level L_{4b}, the second phase inter-simulation staging approach, improves simulation run time significantly in comparison to using only intra-simulation techniques. Also, the one-time cost of the first phase, L_{4a} is no worse than the best possible intra-simulation technique L_3. Thus, in this case inter-simulation staging imposes no additional cost during the first run of a series, but offers a significant speedup during subsequent runs.

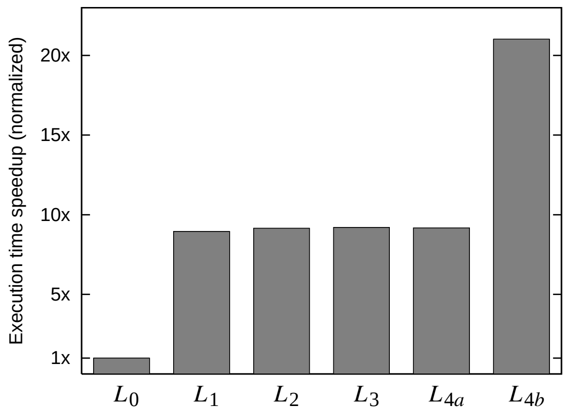

Figure 2: Speedup in Execution Time with Increasing Staging Relative to Baseline Ns2 Implementation using a 1000 Node Network

5.3 Scaling with Network Size

In order to evaluate how staging affects simulation scale, we simulated networks with varying number of nodes while holding the application-level load constant and increasing the field size to maintain a constant node density.

Figure 3 shows that staging can improve the scalability of wireless simulators by reducing redundant computations. This experiment also demonstrates the benefits of inter-simulation staging, which achieves 56% improvement over the intra-simulation staging techniques in 1000 node networks. Although the different intra-simulation staging approaches show similar performance in this experiment, they exhibit different behaviors as optimization parameters or network characteristics change. As we show in the next two sections, the more advanced optimizations offer increased robustness and stability, an advantage not evident in Figure 3.

Additional experiments indicate similar performance benefits using networks of varying density, up to more than twice the density used above. Very dense networks, however, expose a trade-off in our disk-based inter-simulation optimization. In our implementation, cache entries are stored on disk during the first simulator run, and must be read from disk and processed during each subsequent run. While most of these disk accesses are easily pipelined and dispatched in the background, there is still a non-negligible CPU cost for dispatching and processing data stored on disk. As network density increases, the cache entries grow larger and cache processing may become more expensive than simply recomputing results from in-memory data.

This trade-off is present to some extent in any result caching scheme, and designers must be careful that cache overhead is less than the cost of recomputation. But in practice we find that only the disk-based caching optimization might impose a significant processing overhead, for certain

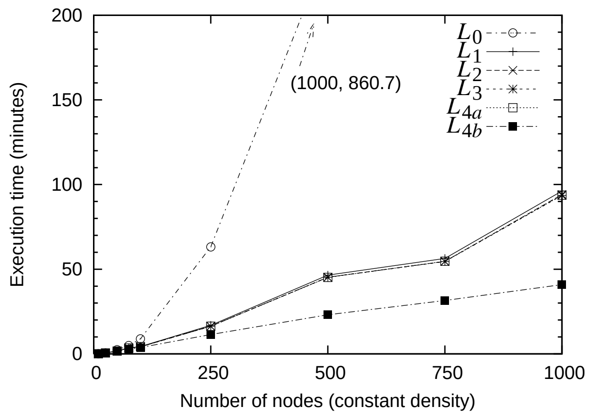

Figure 3: Effect of Network Size on Total Simulation Run Time Holding Node Density Constant

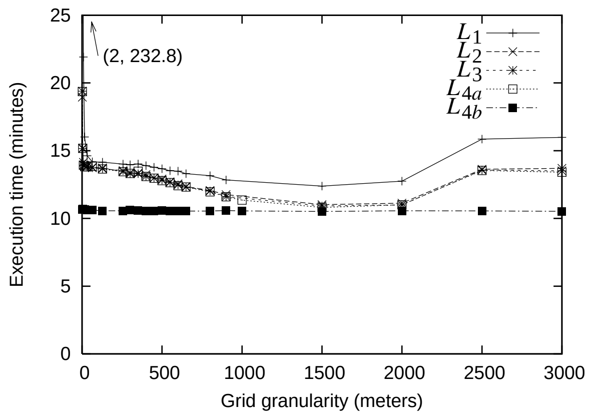

Figure 4: Effect of Varying Grid Granularity on Simulation Run Time

networks, and that the optimization offers a net improvement in performance for networks of reasonable density.

5.4 Optimization Parameters

It is important to characterize the effect of any new simulation parameters introduced by our optimization techniques. We study simulation performance under various choices for optimization parameters and examine the robustness and stability of the different optimization levels. Recall that the grid-based intra-simulation approach introduces a granularity parameter, and the caching intra-simulation approach a Δt lookahead parameter.

We first evaluate the effect of varying grid granularity on each level of staging. Intuitively, it is clear that a very fine granularity will give rise to many grid-crossing events as nodes move about in the topology, and also leads to more work in packet transmission, as many empty bins will be scanned for nodes. Conversely, a very coarse granularity reduces to a single bucket and, essentially, a scan over all nodes during each packet transmission or cache miss. A reasonable choice is to use the node transmission radius, which requires a scan of roughly nine buckets during each transmission or cache miss.

We run the simulator on a single 250 node network with the same configuration as before and Δt fixed at 2 seconds, but vary the grid granularity. Figure 4 verifies our intuitive description of the effects of grid granularity. Interestingly, we find that any choice of granularity other than the two extremes yields a substantial improvement in run time under L_1 staging, with only minor variation between 500 and 2000 meters, with the optimum choice approximately 1500 meters.

In this experiment, even the right-most extreme performs much better than the ns2 baseline implementation since we avoid creating events and copies of the packet for nodes outside the transmission range. Further, much of the degradation due to a poor choice in granularity is mitigated by the use of the higher levels of staging. In these cases, the poorly-tuned grid is consulted only in the rare case of a cache miss.

The choice of grid granularity depends on the particular choice of node mobility and load patterns. In practice, we find that the optimal choice of granularity can be as low as 250 meters, but is rarely higher than 2000 meters. In all cases we have examined, the trends are similar to those presented above, making automatic tuning a feasible approach.

5.5 Caching Lookahead Parameter

The overhead of constructing cache entries is controlled by the Δt parameter to the neighborhood caching routine. Recall that Δt specifies the desired expiration time when constructing a cache entry. A larger value means that a larger radius must be examined to build a cache entry, leading to a larger data structure, but allowing the cache entry to remain valid for longer. We set up our simulator as the previous experiment, but fix the grid granularity at 250 m.

Figure 5 shows how Δt controls the cache hit rate (top), and the sizes of the two neighborhoods sets $\mathcal{N}_{r-\Delta r}$ and $\mathcal{N}_{r\pm\Delta r}$ stored in cache entries (bottom). We only show the results for L_2 caching; those for L_3 perfect caching and the first phase L_{4a} of intra-simulation staging are identical. For reference, the actual average neighbor set size for queries is shown as constant $\mathcal{N}_r$.

The overheads associated with caching are limited by the cache hit rate and $\mathcal{N}_{r\pm\Delta r}$. A very small value for Δt leads to many cache misses, each of which is potentially expensive. Conversely, a large value for Δt forces both cache hits and misses to process a larger set $\mathcal{N}_{r\pm\Delta r}$. The cache is effective for reasonable values of Δt, roughly 2 to 4 seconds, with high hit rate but still reasonably sized $\mathcal{N}_{r\pm\Delta r}$. The curves for the neighborhood set sizes can be explained geometrically based on the known transmission radius, and the average number of neighbors of transmitting nodes. The cache hit rate is a function of the average inter-

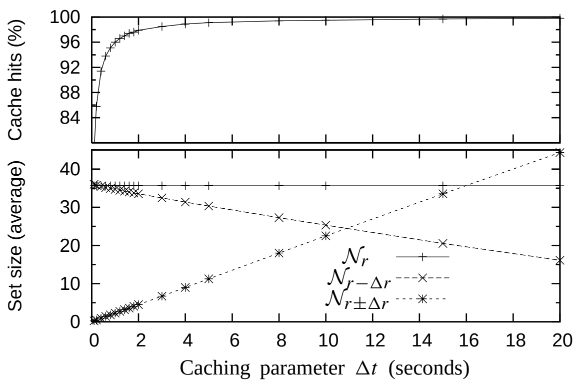

Figure 5: Effect of Varying Caching Parameter Δt on Cache Hit Rate and Neighborhood Sizes

packet spacing. While our implementation does not pick Δt automatically, the figure shows that a near-optimal value for parameter Δt can be computed as a function of the packet rate, node density, and transmission radius.

Surprisingly, even with such varying cache behavior there is very little overall change in total simulation run time. Further experiments indicate that over the entire range of values in Figure 5, run time varies by at most 5% over the range of Δt values shown. The L_2, L_3, and L_{4a} staging levels all perform similarly, while the second phase L_{4b} inter-simulation approach improves run time by approximately 30% as compared to L_3, independent of the Δt parameter. As with the grid granularity, nearly any reasonable choice of parameter will work well for the highest levels of staging.

6 RELATED WORK

Several important examples of staging can be found in existing simulators. In our analysis of the ns2 implementation, we identified applications of staging, but find that the technique of staging is not widely applied in the implementation or recognized in the literature. There has been no prior recognition or development of the technique of staging as a general approach to simulation optimization.

The NixVector (Riley, Ammar, and Fujimoto 2000) approach improves wired-network routing efficiency in the ns2 simulator by computing and caching routes on demand rather than maintaining a complete routing table. This approach has not been applied between multiple runs of the simulator, nor does it eliminate redundancy when inputs vary slightly between computations.

A second example from ns2 is a grid implementation very similar to our L_1 staging. A key difference is that we expose and explore the parameter space of grid granularities, while the previous attempt uses a hard-coded granularity of 1 meter. In typical scenarios, this choice leads to performance worse than the baseline. Similarly, Wu and Bonnet (2002) propose an alternative packet transmission routine for ns2, essentially equivalent to our L_1 staging with granularity parameter ∞. Again, we have shown that this choice of granularity is particularly inefficient as compared to nearly any other choice. These examples illustrate the importance of properly characterizing staging parameters and relating them to system variables such as the transmission radius and expected number of neighbors.

In the context of discrete event simulators, we find occasional use of staging or similar techniques to improve performance. Splitting (Glasserman, Heidelberger, Shahabuddin, and Zajic 1996), cloning (Hybinette and Fujimoto 1997), and updateable simulations (Ferenci et al. 2002) are three related techniques which eliminate identical computations in multiple runs of the simulator. These techniques do not exploit redundant computations within a single run of the simulator, nor do they address computations which are similar but not identical.

Boukerche et al. (1999) propose a two-phase design for Personal Communications System (PCS) network simulation using SWiMNet. This design is used to facilitate various lookahead optimizations in a parallel simulation engine, rather than to eliminate redundant computation or optimize multiple runs of the simulator.

A popular technique for improving scale and performance uses distributed simulation (for example Boukerche et al. 1999, Liu et al. 2001, and Liljenstam, Rönngren, and Ayani 2001), sometimes combined with specialized language features (for example Zeng, Bagrodia, and Gerla 1998). These approaches are complimentary to our optimizations, since staged simulation can be applied equally well to both distributed and centralized designs. Other techniques are used to reduce simulation run time, such as model abstraction and approximation (Huang, Estrin, and Heidemann 1998, Gadde, Chase, and Vahdat 2002). Our approach differs from model abstraction in that we do not alter in any way the final result of computations. Additionally, abstraction may not be possible if the system of interest has not yet developed stable or well-understood models.

Finally, we note that staging as a concept is a general technique, employed most notably in compilers and iterative programming. Chambers (2002) discusses a staged compilation technique that combines partial precompiling of code coupled with dynamic optimizations at runtime. Iterative programming is a general framework for describing computation. Like staged simulation, it relies on reusing results, intermediate values, and extraneous values from previous iterations. Liu, Stoller, and Teitelbaum (1996) discuss methods for automatically extracting this information using program and data-flow analysis. We find this particular approach unsuitable for large and complex simulator implementations, where data-flow and simulation behavior depend very heavily on the particulars of a simulation run. Additionally, the use of multiple languages compounds the difficulty of low-level automatic program analysis.

7 CONCLUSIONS

We propose a general technique, termed staged simulation, for reducing the run time of discrete event simulators. The central idea is to eliminate redundant or partially-redundant computations typical in simulations by caching and reusing the results of computations. The technique consists of identifying redundant computation both within single runs as well as across consecutive runs of the simulator. Staging then relies on precomputing, caching and reusing partial results to eliminate redundant computation. Our technique is general and applicable to a wide range of designs, including parallel and distributed simulation engines.

We show that staging is an effective technique for reducing simulation run time without loss of accuracy, and is effective in a wide range of simulation scenarios including varying mobility and communication patterns, network sizes, and node densities. We implement three levels of intra-simulation staging and one level of inter-simulation staging in the ns2 wireless networking simulation system. Simple intra-simulation optimizations are found to reduce simulator run time by a factor of 9 and to improve simulator scalability from networks of hundreds of nodes to networks of ten thousand nodes. An application of inter-simulation staging can reduce run time even further to a factor of 21 over the non-staged implementation. We find that the techniques are robust in the choice of parameters, and these parameters appear easy to estimate automatically as a function of other simulation variables and observed runtime behavior.

REFERENCES

Boukerche, A., S. Das, A. Fabbri, and O. Yildiz. 1999. Exploiting model independence for parallel PCS network simulation. In *Workshop on Parallel and Distributed Simulation*, 166–173.

Broch, J., D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. 1998. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *ACM/IEEE Intl. Conference on Mobile Computing and Networking*, 85–97.

Chambers, C. 2002. Staged compilation. *ACM SIGPLAN Notices* 37 (3).

Chang, X. 1999. Network simulations with OPNET. In *Winter Simulation Conference*, ed. P. Farrington, H. Nembhard, D. Sturrock, and G. Evans, 307–314. Piscataway, NJ: IEEE Press.

Ferenci, S., R. Fujimoto, M. Ammar, K. Perumulla, and G. Riley. 2002. Updateable simulation of communications networks. In *Workshop on Parallel and Distributed Simulation*, 107–114.

Gadde, S., J. Chase, and A. Vahdat. 2002. Coarse-grained network simulation for wide-area distributed systems. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*.

Glasserman, P., P. Heidelberger, P. Shahabuddin, and T. Zajic. 1996. Splitting for rare event simulation: Analysis of simple cases. In *Winter Simulation Conference*, ed. J. Charnes, D. Morrice, D. Brunner, and J. Swain, 302–308. Piscataway, NJ: IEEE Press.

Huang, P., D. Estrin, and J. Heidemann. 1998. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *MASCOTS*, 241–248.

Hybinette, M., and R. Fujimoto. 1997. Cloning: a novel method for interactive parallel simulation. In *Winter Simulation Conference*, ed. S. Andradóttir, K. Healy, D. Withers, and B. Nelson. Piscataway, NJ: IEEE Press.

Liljenstam, M., R. Rönngren, and R. Ayani. 2001. MobSim++: Parallel simulation of personal communication networks. *IEEE DS Online* 2 (2).

Liu, J., L. Perrone, D. Nicol, M. Liljenstam, C. Elliott, and D. Pearson. 2001. Simulation modeling of large-scale ad-hoc sensor networks. In *European Simulation Interoperability Workshop*.

Liu, Y., S. Stoller, and T. Teitelbaum. 1996. Discovering auxiliary information for incremental computation. In *ACM SIGPLAN-SIGACT Symposium*, 157–170.

Oh, S., and J. Ahn. 1999. Dynamic calendar queue. In *Annual Simulation Symposium*.

Riley, G. F., M. H. Ammar, and R. Fujimoto. 2000. Stateless routing in network simulations. In *MASCOTS*, 524–531.

Royer, E., and C. Toh. 1999. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications* 6 (2): 46–55.

The VINT Project. 1995. Ns-2 network simulator. Available at: `<http://www.isi.edu/nsnam/ns>` [accessed July 1, 2003].

Wu, S., and C. Bonnet. 2002. An alternative packet transmission procedure for mobile network simulation. In *Intl. Symposium on Performance Evaluation of Computer and Telecommunication Systems*.

Zeng, X., R. Bagrodia, and M. Gerla. 1998. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, 154–161.

AUTHOR BIOGRAPHIES

KEVIN WALSH is currently a Ph.D. candidate in Computer Science at Cornell. His e-mail address is `<kwalsh@cs.cornell.edu>`.

EMIN GÜN SIRER is currently an Assistant Professor in Computer Science at Cornell. His e-mail address is `<egs@cs.cornell.edu>`.