META-SIMULATION DESIGN AND ANALYSIS FOR LARGE SCALE NETWORKS

By

David W. Bauer Jr.

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

_____
Dr. Shivkumar Kalyanaraman, Thesis Adviser

_____
Dr. Christopher D. Carothers, Thesis Adviser

_____
Dr. Bülent Yener, Member

_____
Dr. Biplab Sikdar, Member

Rensselaer Polytechnic Institute
Troy, New York

September 2005
(For Graduation December 2005)

META-SIMULATION DESIGN AND ANALYSIS FOR LARGE SCALE NETWORKS

By

David W. Bauer Jr.

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Dr. Shivkumar Kalyanaraman, Thesis Adviser

Dr. Christopher D. Carothers, Thesis Adviser

Dr. Bülent Yener, Member

Dr. Biplab Sikdar, Member

Rensselaer Polytechnic Institute
Troy, New York

September 2005
(For Graduation December 2005)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

Without my wife Sarah, none of this would have been possible. For standing by me and supporting me over the years I will be forever grateful. She has been my constant and unwavering inspiration throughout this entire process.

I would like to thank my advisers Dr. Christopher D. Carothers and Dr. Shivkumar Kalyanmaraman for their support and guidance that helped me through my graduate studies. I have an immense amount of gratitude for the knowledge I gained from their council and guidance. I feel fortunate to have Chris as my advisor, teacher and friend.

I want to express my gratitude to Bülent Yener and Biplab Sikdar for serving on my committee and helping me with my dissertation. In addition, Boleslaw Szymanski, Christopher Carothers and Dr. Richard Fujimoto at Georgia Tech for their kindness in allowing me the usage of their computing resources for performance results. I would also like to thank the Lab Staff in the Computer Science department for all of their help throughout the years in solving so many problems, namely David Cross, Jonathan Chen and Steven Lindsay.

I would also like to thank Garrett Yaun, Murat Yuksel and Shawn Pearce for their help throughout my college career. I enjoyed our many discussions in both our research and my life and have learned much from the both of them.

I appreciate the faculty and staff in the Computer Science Department at Rensselaer Polytechnic Institute who offered me a great study and research environment. I would also like to thank my fellow students for their help in preparing for the qualifiers and their comments on my rehearsals for my candidacy presentation and dissertation defense.

A special thanks to my parents for their belief and support of me in whatever I choose to pursue. I'm grateful for their patience and understanding.

ABSTRACT

Performance analysis techniques are fundamental to the process of network protocol design and operations. A variety of techniques have been used by researchers in different contexts: analytic models (eg: TCP models, web models, self-similar models, topology models), simulation platforms (eg: ns-2, SSFnet, GloMoSim, Genesis), prototyping platforms (eg: MIT Click Router toolkit [9], XORP [10]), tools for systematic design-of-experiments and exploring parameter state spaces (eg: Recursive Random Search, STRESS [13]), experimental emulation platforms (eg: Emulab), real-world overlay deployment platforms (eg: Planetlab), and real-world measurement and data-sets (eg: CAIDA [26], Rocketfuel [27]).

The high-level motivation behind the use of these tools is simple: to gain varying degrees of qualitative and quantitative understanding of the behavior of the system-under-test. This high-level purpose translates into a number of specific lower-level objectives, such as: validation of protocol design and performance for a wide range of parameter values (parameter sensitivity), understanding of protocol stability and dynamics, and studying feature interactions between protocols. Broadly, we may summarize the objective as a quest for general invariant relationships between network parameters and protocol dynamics.

To address these needs, we developed an experiment design platform that will allow us to empirically model and heuristically search for optimizing protocol response. In general the protocol response is a function of a large vector of parameters, i.e., is a response surface in a large-dimensional parameter space (perhaps tens of thousands or more dimensions). We build off recent work at Rensselaer on an efficient search algorithm (called Recursive Random Search) for large-dimensional parameter optimization, and empirical modeling of protocol performance characteristics especially in "interesting" regions of the parameter state space. The result of this work includes a unified search, empirical modeling and optimization framework with demonstrated ability to pose meaningful large-scale network design questions and provide "good" models rapidly.

CHAPTER 1
Introduction

This chapter focuses on the motivation for the research, illustrates the scope of the thesis and presents the main contributions.

1.1 Motivation

Before we can deploy new network protocols or modifications to large scale networks, certain assurances of design, scalability and performance must be met. The networking community lacks performance analysis tools that will help us design, analyze, understand and operate networking protocols on a very large-scale and in heterogeneous deployment contexts. Three reasons why simulating the large-scale networks is difficult are: *scale, heterogeneity and rapid change.* Though tools like ns-2 and SSFnet are popular for small-scale performance analysis, combined with testbed facilities like Emulab and Planetlab, we believe that rapid performance analysis on the large-scale will require scalable simulation combined with meta-simulation tools for large-scale experiment design and empirical modeling of protocol behavior.

Because of the size and scope of such large networks as the Internet or corporate WANs, introducing new network protocols or configurations can be problematic. Problems may arise resulting from unanticipated feature interactions. Simulation plays a crucial role in allowing network engineers to conduct trial and error experiments with new protocols and configurations prior to deployment. Meta-simulation goes a step further by providing a framework for identifying the greater issues that arise from the interactions between new and old technologies. Meta-simulation is built on top of simulation experimentation by creating designs of experiments which lead to more complex problem solving.

Simulation forms the basis for testing and development of new network technologies. But simulation has been hampered by a lack of scalability. Simulation of large scale networks may require simulating tens of thousands to millions of separate

computers running multiple applications and protocols. Because of their size, these networks are rarely statically defined. Rather, they are ever changing, suffering from hardware failures, protocol malfunctions and physical link outages. Scaling the simulation experiment is only one dimension of the problem; beyond scale, we also require models of the dynamic nature of such large-scale networks. Recent work in network simulation has focused generally on scale while neglecting rapid change. Typical models executing in polynomial memory often experience exponential increases in memory consumption when rapid change is modeled.

Beyond simulation, we also require tools for understanding the complex nature of large-scale networks from a high level viewpoint. Determining convergence rates, throughput, link congestion and packet delays and network layer reachability information are all typical examples of performance metrics. Beyond measurement of these metrics, a more detailed analysis includes the degree to which certain network protocol parameters have on these effects. In addition to basic parameter effects, we would also like to characterize the effects between parameters across protocols and applications.

We have developed an experimentation platform called "ROSS.Net" to solve many of these problems. ROSS.Net enables, in the area of simulation modeling, (i) an optimistic parallel simulation engine that leverages memory-efficient reversible computation instead of using traditional state-saving to support rollback recovery (ii) systemic memory-efficient methodology for model construction using a combination of library interfaces to key data structure algorithms. The end goal is for a researcher to combine topology, parameter configuration information and selectively compose a set of experiments that execute space and time efficient models resulting in accurate answers to the questions posed by the model designer.

To address the problems of interpretation of results, we developed *large-scale experiment design tools* that allowed us to model and optimize protocol response. In general the protocol response is a function of a large vector of parameters, i.e., is a response surface in a large-dimensional parameter space. We utilize recent work at Rensselaer on an efficient search algorithm (called Recursive Random Search) for large-dimensional parameter optimization. The result of this work will include a

unified search, optimization and modeling framework with demonstrated ability to pose meaningful large-scale design questions and provide "good" models rapidly.

The overall goal is to provide tools that dramatically improve the way we study network protocols. The platform is capable of simulating full protocol and packet-level dynamics of large networks on commodity uni-processor or multiprocessor hardware.

Fundamental contributions are made in two major areas: scalable simulation and large-scale experiment design. This work also has an immediate impact in applications broadly in the area of computer systems where large-scale design is important (eg: operating systems, distributed systems, computer architecture). However, we also anticipate impact in areas far from computer networking. For example, the large-scale experiment design methodologies could be applied to areas as diverse as industrial quality control, agriculture, protein folding (bioinformatics) and theoretical computer science. These future applications stem from the fast-approximation and modeling features of our methods. We note that this thesis confines its scope to only the computer network domain.

1.2 Contributions

Three major contributions are presented in this thesis. Each contribution completes a major step forward towards the dual goals of scalable network simulation and large-scale experiment design and analysis. These goals address the larger aims of *scale, heterogeneity and rapid change.*

The first contribution is in the area of large-scale experiment design and analysis or, meta-simulation. This contribution provides an analysis tool, called ROSS.Net, which systematically formulates and organizes multiple simulation experiments in the quest of the general invariant relationships between parameters and protocol performance response. ROSS.Net brings together the four main areas of network research: simulation, protocol design, network modeling and measurement, and experiment design. ROSS.Net aims to provide "good" results fast.

ROSS.Net is built upon the second major contribution of this thesis, *ROSS: Rensselaer's Optimistic Simulation System.* ROSS demonstrates for the first time

that stable, highly-efficient execution using little memory above what the sequential model would require is possible for low-event granularity simulation models. The driving force behind these high-performance and low memory utilization results is the coupling of an efficient pointer-based implementation framework, Fujimoto's fast GVT algorithm for shared memory multiprocessors, *reverse computation* and the introduction of *Kernel Processes (KPs)*. KPs lower fossil collection overheads by aggregating processed event lists. This aspect allows fossil collection to be done with greater frequency, thus lowering the overall memory necessary to sustain stable, efficient parallel execution. These characteristics make ROSS an ideal system for use in large-scale networking simulation models.

The third major contribution introduces a new distributed GVT implemented within ROSS, affectionately named the Seven O'Clock GVT algorithm. We formalize the idea of a *network atomic operation (NAO)*, which enables a zero-cost cut mechanism which greatly simplifies GVT computations in a cluster computing environment. We demonstrate its reduced complexity by extending Fujimoto's shared memory algorithm to operate distributed across a cluster of shared-memory multiprocessors.

1.3 Thesis Outline

The thesis is structured as follows:

Chapter 2 illustrates the background of large-scale experiment design and network simulation research which in turn introduces the background for this thesis. An overview of experiment design and black-box optimization is given. Also, an overview of current work in network modeling and simulation is given. This is followed by an assessment of current research and concludes with a discussion in network protocol feature interactions.

Chapter 3 presents the design and implementation of meta-simulation tool, called ROSS.Net. ROSS.Net allows network researchers to flexible study the performance of different network protocols and topologies within a meta-simulation framework. ROSS.Net allows researchers to quantitatively and qualitatively analyze large-scale, dynamic networks. Within this framework is the flexible protocol

subscription model which allows for multiple levels of scalability.

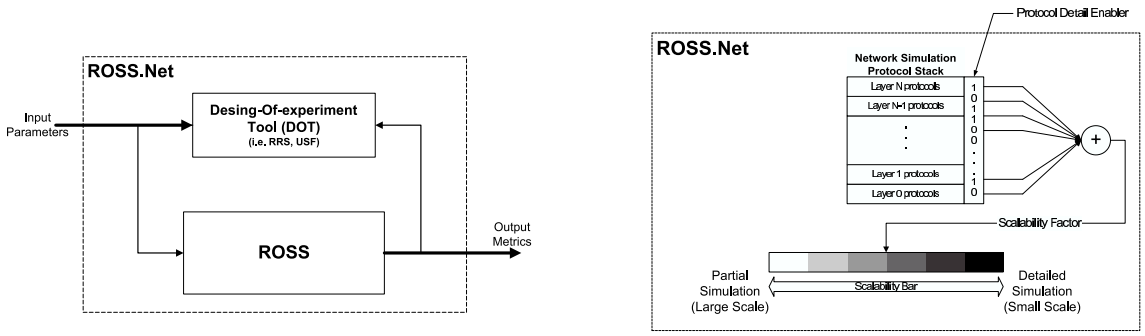
Chapter 4 introduces a major sub-component of the ROSS.Net tool, Rensselaer's Optimistic Simulation System (ROSS). ROSS is a parallel and distribution discrete event simulator which enables the large scale modeling and simulation of logical processes. The goal of this system is to derive as much performance as is possible from commodity (off the shelf) computing equipment. ROSS is a high performance simulator because of its modular design and minimal use of memory. Specifically we focus on the creation of Kernel Processes (KPs) to facilitate fossil collection while introducing a minimum of rollback overhead. The chapter concludes with a performance study which illustrates the efficiency of the design.

Chapter 5 presents a new global virtual time (GVT) algorithm called the Seven O'clock algorithm. This algorithm is unique because it is creates a sequentially consistent distributed memory model based on the synchronization of CPU cycle counters. This synchronization is then used to formulate the generalization of a Network Atomic Operation (NAO). The Seven O'clock algorithm utilizes an NAO to create a zero-cost consistent cut when synchronizing the parallel and distributed simulation system. This chapter concludes with a performance study of the algorithm which demonstrates the efficiency of the algorithm.

Chapter 6 summarizes the current work and provides conclusions for this research.

CHAPTER 2
Historical Review

2.1 Meta-Simulation: Large-Scale Experiment Design, Heuristic Optimization and Empirical Protocol Modeling



(a) Experiment design in ROSS.Net. (b) Multi-Abstraction paradigm of ROSS.Net.

Figure 2.1: ROSS.Net Modeling and Simulation Concepts.

Scaling the simulation platform (described later) is just one dimension of our research. Recall that, beyond mere scaling of simulation platforms, we need meta-simulation capabilities to extract and interpret meaningful performance data from the results of multiple simulations (see Figure 2.1-a). However, there has been little attention by the networking research community on this important space of meta-simulation, i.e., choosing experiments, developing useful interpretations and protocol performance modeling. The purpose of large-scale experiment design is to systematically formulate and organize multiple simulation experiments in the quest of the general invariant relationships between parameters and protocol performance response, and models of protocol behavior in large-dimensional parameter state spaces.

2.1.1 Experiment Design and Empirical Models

Design of Experiments or "experiment design" is a well known branch of performance analysis, specifically, a sub-branch of statistics [4, 11]. It has been used

extensively in areas like agriculture, industrial process design and quality control [11], and has been introduced to the area of practical computer and network systems design by Raj Jain [4]. Statistical experiment design views the system-under-test as a black-box that *transforms input parameters to output metrics*. The goal of experiment design is to *maximally characterize/model* (i.e., obtain maximum information about) the black-box with the *minimum number of experiments*. Another goal is *robust* empirical performance modeling, i.e., one that is minimally affected by external sources of variability and uncontrollable parameters, and can be specified at a level of confidence.

The underlying premise of experiment design is that each experiment (eg: a simulation run, or an Emulab [24], Planetlab [25] test run) has a non-negligible cost. Moreover, the output (i.e., response) corresponding to a vector of parameters in general will have an unknown surface topology, also known as *"response surface"*. The goal of empirical modeling and experiment design is to choose a minimum number experiments to sample and find a curve fitting the unknown response surface. Observe that there are two levels of uncertainty in this problem: a) discover where to sample in the unknown response surface and b) how to best fit a model to the samples that is also representative of the original response surface.

Simple designs like "best-guess" or "one-factor-at-a-time" designs are less favored in complex situations since they do not provide information about the response effects of the interactions between parameters. The next step is to consider models that consider linear parameter interactions (i.e., assume a linear model structure). Designs like full-factorial (eg: 2^n) and fractional factorial (eg: 2^{n-p}) (also called orthogonal designs), appropriately subjected to replication, randomization and blocking fall in this category and are preferred to simple one-factor-at-a-time designs [4, 11]. The usual end-goal of formulating regression models is to efficiently (i.e., with a minimum number of experiments) observe the effects of both individual parameters and linear parameter interactions [4, 11]. Techniques like blocking and analysis of covariance are used to explicitly handle measurable, but uncontrollable (a.k.a. "nuisance") factors [11]. Transforms on data (eg: Box-Cox power-law family of transformations) can effectively aid in producing a family of non-linear regression

models and stabilizing the variance of the response [4, 11].

If the underlying response structure is not linear, one approach is to build lower-order models built with fractional factorial experiments to reach interesting areas where more detailed (higher-order) modeling is done. Well-known small-polynomial-order response surface methods include central composite design (CCD) (i.e., efficient fitting of second order models), Box-Behnken designs (Chapter 11 of [11]). Robust parameter designs (RPDs) for finding settings for controllable variables that minimized the variability transmitted to the response from uncontrolled (or noise-) variables have been proposed by Taguchi [29], that have been credited for triggering a quality-control revolution in the 1980s-90s [11]. Taguchi's RPDs use highly fractionated factorial designs and other fractional designs obtained from orthogonal arrays.

Unfortunately conventional experiment designs do not scale to very large parameter state spaces. Like the area of large-scale data mining, we run into the so called *"curse of dimensionality"* [30], i.e., a hyperspace that becomes prohibitively large to be sampled (and hence modeled) at fine granularity. For example: modeling the performance behavior of intra-domain and inter-domain traffic flows given the settings of hundreds of BGP parameters and OSPF link weights. We have to settle for *"sparse empirical models"* that sample at finer granularity only in selected regions of the state space that are discovered to be "interesting" according to a pre-specified set of metrics.

2.1.2 Heuristic Problem Solving and Black-Box Optimization Approaches

A step weaker than *empirical modeling* (i.e., developing input-output regression models) is *optimization*, i.e., to determine the region of the parameter state space that leads to the "best" performance response. Since the response surface is unknown, this question falls into the broad area of systematic, heuristic problem solving [31]. The main issues in problem solving involves developing an understanding of the difficulty of a problem: size of the search space, accuracy and ease of the evaluation function, and the nature of the problem constraints.

The generalization of the response surface optimization is called *black-box op-*

timization using heuristic search methods. In particular, a variety of network parameter optimization questions can be mathematically formulated (assuming minimization) and solved by such techniques. For example, given a real-valued objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, find a global minimum $\mathbf{x}^*$,

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in D} f(\mathbf{x}) \tag{2.1}$$

where $\mathbf{x}$ is the parameter vector to be optimized, D is the parameter space, usually a compact set in $\mathbb{R}^n$. In these problems, the objective function $f(\mathbf{x})$ is often analytically unknown and the function evaluation can only be achieved through computer simulation or other indirect ways. This type of problems are hence called "black-box" optimization problems where the objective function is modeled as a black-box.

Since little *a priori* knowledge is assumed, these black-box optimization problems are considered very hard to solve. In addition, since the objective functions are often non-linear and multi-modal, this type of optimization are also called *global optimization* in contrast to *local optimization* which has only one single extreme in $f(\mathbf{x})$ and is much easier to solve. Most of black-box optimization problems are NP-hard and can only be solved for near-optimal solutions with heuristic search algorithms. In this class of methods, a variety of "exploration" techniques replace the explicit lower-order modeling mentioned in the prior paragraph. In regions of interest, a local "exploitation" technique is used to find the local extrema.

Many heuristic evolutionary search algorithms have been proposed and demonstrated to be very successful in practice, such as, multi-start hill-climbing [33], genetic/evolutionary algorithms [34] and simulated annealing [35]. However, there has been no consistent report on their performance. In fact, the *No Free Lunch Theorem* [36, 37] has theoretically demonstrated that no matter what performance metric is used, no single optimization algorithm can consistently perform better in all problem classes than the others. The *average* performance of any algorithm is the same over all classes of problems; though there might exist algorithms that perform very well for a large number of classes. In other words, there exists no general all-purpose optimization algorithm. For one specific class of problems, its inherent properties

have to be carefully investigated to perform efficient optimization.

2.1.2.1 RRS: Random Recursive Search

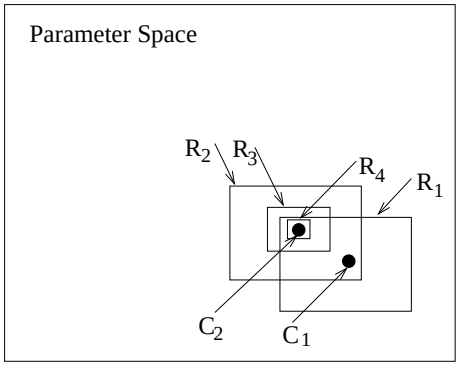
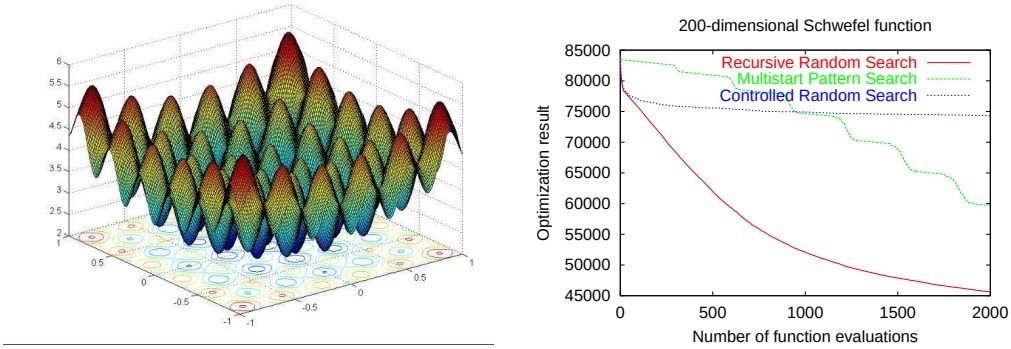


Figure 2.2: RRS' shrink-and-realign process on a 2D parameter space.

Tao Ye and Shivkumar Kalyanmaraman have co-designed a new heuristic search algorithm, Recursive Random Search (RRS) [12], for large-scale network parameter optimization, is based on the initial high-efficiency feature of random sampling (i.e., the results of random sampling improves rapidly during early samples). RRS maintains this initial high-efficiency feature by constantly restarting random sampling with adjusted (i.e., re-scaled) sample spaces. In the "explore" phase, RRS takes N samples (N depends upon a confidence level, eg: 95%) and then successively restarts and rescales the search in the vicinity of the best found result ("exploit" phase) to zoom in to find a local optimum. As shown in Figure 2.2, the best local optimum is then used with future randomized "explore" samples to decide where to re-scale (i.e., exploit) and look for new local optima. The RRS algorithm outperformed traditional search techniques in various benchmarks and has been successfully applied in multiple network management situations using on-line simulation (eg: OSPF, BGP, RED as reported in [12]). For instance, RRS is tested on Schwefel function (which is shown in Figure 2.3-a), and outperformed techniques like Multi-start Pattern Search and Controlled Random Search as shown in Figure 2.3-b. The figures show the average improvement (over several runs, with tight confidence intervals) in the best local optima (i.e., lowest metric value) found as a function of the cumulative number of experiments consumed.



(a) An example objective function: 2D Schwefel function

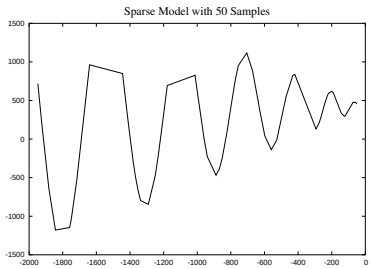
(b) Tests on 200-dimensional Schwefel function

Figure 2.3: Recursive Random Search (RRS) tests on Schwefel function.

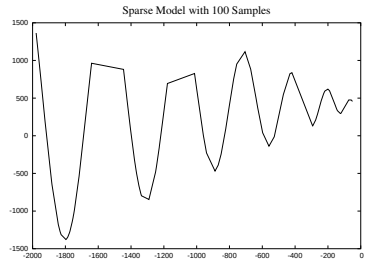
Besides the high efficiency in large-dimensional parameter spaces, the RRS algorithm is *robust to the effect of random noise* in the objective function because of its basis in random sampling and it is able to *automatically exclude negligible parameters* from the optimization process. These features are very important for the efficient optimization of network protocol configuration.

2.2 ROSS.Net Network Modeling & Simulation Platform

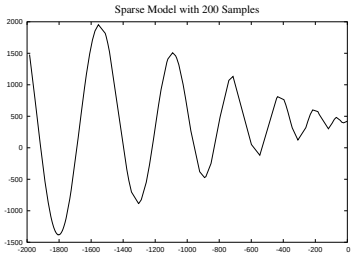
Currently, the networking research community has developed a number of modeling and simulation research projects. Current platforms include SSFNet [7], DaSSF [38], JavaSIM [39], Ns [40] and GloMoSim [8]. There are also interoperability platforms such as PDNS/Backplane [41] and Genesis [42] which combines these existing platforms for the purpose of "model reuse" as well as to enable scalability through distributed execution. Similarly, the "staged" simulation approach (SNS) [43], attempts to reuse computation in the domain of wireless networks. In wired networks, queuing statistics are the predominate computation and not radio interference calculations. Thus, because a queue can change in unpredictable ways, we do not believe this technique will be generally applicable in this domain. So then what else is lacking in these platforms? Our answer to that question suggests that the current state-of-the-art in sequential, parallel or distributed platforms *fails to* address scalability in the following dimensions.



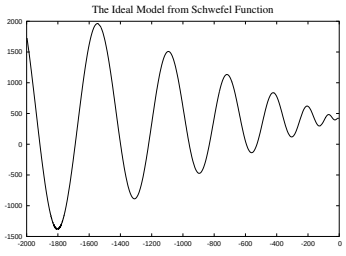
(a) The sparse model with 50 samples.



(b) The sparse model with 100 samples.



(c) The sparse model with 200 samples.



(d) The ideal model, Schwefel function

Figure 2.4: An illustration (not done by RRS) of sparse model development for a Schwefel objective function: The sparse model improves as the number samples from the search space increases. In particular, the regions of interest improves first. In the 100-samples case, the most important region around -1800 improves while the less important regions (like the peaks around -1500 and -1150) show no improvement.

1. **Memory Efficient Parallel Models:** It was report by Nicol in [44], that Ns consumes 93 KB per TCP connection, SSFNet (Java version) consumes 53 KB, JavaSim consumes 22 KB per connection and SSFNet (C++ version) consumes 18 KB for a "dumbbell" model which contains only two routers. Router states in models such as OSPF and BGP are significantly larger than for a single TCP connection. Thus, to support million node topologies, supercomputer size addresses spaces are required to execute large-scale within these platforms.

2. **Scalable Performance Under Realistic Topologies:** While realistic Internet topologies are being used, to date it *has not* been shown that existing platforms are able to scale when real scenarios are used. For example, routing stability studies require the model to dynamically modify the large-scale

network topology (i.e., greater than 1 million nodes) as a consequence of link failures, denial of service attacks or router reboots. Current sequential platforms, such as Ns and JavaSim, cannot execute models much larger than few thousand nodes [44]. The current approaches to parallel execution, such as SSFNet, and DaSSF are tied to the underlying static topology structure. The reason for this is because these algorithms exploit the latency of links as part of the synchronization algorithm. The minimum latency defines the extent to which model parallelism can be exploited. Thus, as links go down these parallel approaches must either block and be readjusted to exclude those failed links in order to obtain optimal synchronization overheads or execute assuming those links are available but at the expense of unnecessary synchronization overheads. In either case, it is unclear how these approaches will scale under dynamic changes in topology.

3. **Flexible Subscription of Model Components:** Current modeling platforms export a static model hierarchy which conforms to the precise specification of application and or protocols including all sub-layers. For example, within Ns, a TCP model will consist of a host abstraction, session abstraction (both receiver and sender sides) as well as an IP layer. While these layers are necessary for emulations, a large-scale simulation does not need all of these abstractions and layers to produce a statistically valid result. Consequently, the static hierarchy results in models that are significantly more heavy weight and fundamentally less scalable.

   More recent approaches include hybrid fluid flow models [22] which allow a far greater amount of background TCP traffic. But these models suffer from an unbounded error rate and cannot be reliably used when performing search optimizations. Typically search optimizations require little or no error in order to generate non-random results, i.e., the "black-box" optimization must be deterministic.

2.2.1 Current Research in Network Modeling & Simulation

We are leading an effort to understand the performance limits of an advanced parallel and distributed discrete-event simulation system. This study is conducted in the context of large-scale network models. In these models, the network elements (i.e., hosts and routers/switches) are divided into a collection of logical processes (LPs). These LPs communicate by exchanging timestamp events messages. Such a message is used to denote the "arrival" of a packet at a router or switch element.

The crucial issue when executing such a model in parallel is ensure that all events are processed in timestamp order. To deal with this synchronization problem there are largely two classes of approaches: *conservative* and *optimistic*. In the context of network models, conservative approaches either barrier synchronize LPs at well defined points in simulated time, exploit the "slack time" between LPs [7] or some combination of two [38]. The synchronization frequency (either barrier or "slack time" based) is derived using the minimum network propagation delay from an incoming link to a host/router node. Thus, as previous noted, all conservative techniques must leverage precise topology structure in order to achieve good performance. Should that structure change, then unnecessary overheads can result.

In our current research, we take a radically different approach which is based on the speculative or optimistic processing of events. Here, processors execute LP events locally in timestamp order. However, should an event from another LP/processor arrive in the destination LP's past, the LP will rollback and "undo" any changes made to its state. Traditionally, the undo operation has been supported by state-saving. Here, an LP will make either a complete or partial copy of state variables as each event is processed and restore the correct version of state in simulated time by rolling back.

The key advantage of an optimistic approach is that it operates independent of the underlying network topology. Thus, it continues to exploit all the available parallelism even during dynamic changes in topology. However, previously the caveat has been that state-saving overheads dominate the computation costs resulting in little or no increase in performance as compared to sequential model execution [45, 46]. To address this problem, we are utilizing a new approach

called *reverse computation* for realizing the undo operation. Under reverse computation, the roll back mechanism in the optimistic simulator is realized not by classic state-saving, but by literally allowing to the greatest possible extent events to execute backward. Thus, as models are developed for parallel execution, both the forward and reverse execution code must be written.

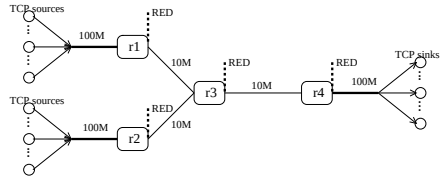
2.2.2 Network Protocol Feature Interactions

Analysis of interactions between inter- and intra-domain routing protocols has been an attractive research topic. In [118], through analysis of data from AT&T's BGP and OSPF traffic measurements, authors showed that majority of BGP updates are because of Hot Potato decision-making practices of ISPs. The major challenge they faced in their analysis was to properly classify Hot Potato BGP updates from the other updates as well as to accurately match internal OSPF link changes as triggering events to external BGP updates. Through a time-based matching algorithm, they counted BGP updates attributable to Hot Potato effects in the ISP's OSPF network. The main difference in our work is that we do not need any matching or estimation technique to determine OSPF-caused BGP updates or vice versa. Since our large-scale simulation environment is fully controlled, we can easily trace the causes of updates.

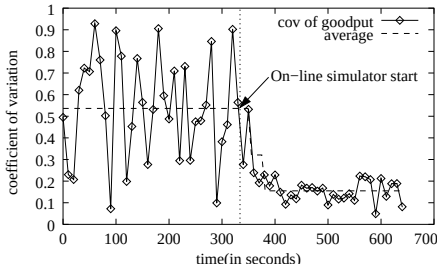
In [119], as a follow-up to their previous work [118], authors modeled sensitivity of BGP (and the network in general) to IGP-caused Hot Potato changes. When building an analytical model for analyzing effects of intra-domain Hot Potato routing changes on inter-domain routing, the authors got inspired by multidimensional data analysis of database systems. More specifically, they modeled BGP's sensitivity to IGP changes by considering all possible IGP changes on a data cube with three dimensions: (i) IP prefixes, (ii) network changes – a particular type of change such as link failure or link weight change, and (iii) locations – the router where the change happened. An element of the cube is set if the corresponding egress point for the IP prefix changes in response to the associated network change at the associated router. Similar to our work, this type of data analysis is indeed another way of expressing all possible parameters that we use in the experiment design framework.

The bottom line is to enumerate all possible Hot Potato IGP changes to perform BGP analysis.

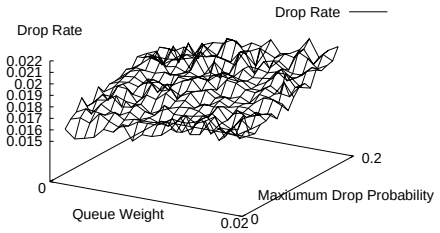
Another major work on analysis of OSPF and BGP interactions was presented in [120]. In contrast to the research direction on analyzing effects of intra-domain changes on inter-domain routing as in [118, 119], the main goal of the work in [120] was to determine if BGP dynamics effect intra-domain routing behavior and in turn effect the traffic engineering of it. The main motivation was to identify if traffic matrix (which is a fundamental component for intra-domain traffic engineering) is affected by BGP updates. The authors considered size of the traffic shifts, in addition to the count of correlated OSPF and BGP updates. They used measurement data from Sprint's BGP routers, e.g. they observed 133 updates/minute for the BGP routing tables of Sprint. An interesting observation they have made is that, compared to the total traffic, the amount of volume shift because of BGP updates is very small. Also, they found that BGP updates are not distributed evenly across the routing table, i.e., IP prefixes. Rather, they observed that longer prefixes (i.e., smaller ASes) receive more of the BGP updates. Similarly, they showed that elephants (or heavy-hitter long traffic flows) are disturbed less than mice from BGP updates.



(a) Linux testbed topology with RED queues.



(b) Optimization of RED's goodput.



(c) RED's empirical objective function.

Figure 2.5: Adaptive tuning of RED parameters with experiment design.

CHAPTER 3
Meta-Simulation: Large-Scale Experiment Design and Analysis

In this chapter we introduce ROSS.Net. We conduct a case study of the OSPFv2 network protocol and investigate interactions between OSPFv2 and the BGP4 network protocols. Our example network focused on AT&T's North American network, which we considered as a single OSPF area containing over 11,000 routers. We compute a full factorial design of experiments that characterizes a great deal of the model space. We compare this with the model generated by the Random Recursive Search (RRS) algorithm. The results of our comparison emphasize RRS's ability to generate good results fast by reducing the number of experiments collected from over 16,000 to 750 with a variance of only 7% in the response plane local minima. We illustrate how, by using ROSS.Net, we were able to compute a critical path analysis of the OSPF network which leads to a 100-fold improvement in the sampling rate.

In our investigation of BGP and OSPF interactions the example network is expanded to 5 small autonomous systems with OSPF as the intra-domain protocol, and BGP as the inter-domain protocol. We define interactions between the two as a correspondence of the effects between the two domains. For example, if a link fails within an AS, does this cause a shift in the reachability information between ASes, and vice versa.

3.1 Experiment Design and Analysis

Performance analysis techniques are fundamental to the process of protocol design and network operations [4, 5, 6]. The high-level motivation of these techniques is simple: to gain varying degrees of qualitative and quantitative understanding of the behavior of a system under-test. A number of specific lower-level objectives include: validation of protocol design and performance for a wide range of parameter values (parameter sensitivity), understanding of protocol stability and dynamics, and studying feature interactions between protocols. Broadly, we may summarize

the objective as a quest for general invariant relationships between network parameters and protocol dynamics [4, 5, 28].

Systematic design-of-experiments [4, 11] is a well studied area of statistics and performance analysis offering guidance in this aspect. A survey of relevant papers in the networking field suggests that such systematic techniques (e.g.: factorial designs, large-scale search) have not been used in the protocol design process or network operations process except possibly by measurement specialists. This ad-hoc approach to organizing simulation or test-bed experiments has worked when we design and examine a small number of features, network scenarios and parameter settings. However, this method is likely to be untenable as we design newer protocols that will rapidly be deployed on a large-scale, or have to deal with a combinatorial explosion of feature interactions in large operational inter-networks. The need for scalable simulation and meta-simulation tools is implicit in Floyd [6]'s statement: *"...we can't simulate networks of that size (global Internet). And even if we could scale, we would not have the proper tools to interpret the results effectively..."*

Beyond mere scaling of simulation platforms, our next need is meta-simulation capabilities, i.e., large-scale experiment design. Statistical experiment design considers the system-under-test as a black-box that *transforms input parameters to output metrics*. The goal of experiment design is to *maximally characterize* the black-box with the *minimum number of experiments*. Another goal is *robust* characterization, i.e., one that is minimally affected by external sources of variability and uncontrollable parameters, and can be specified at a level of confidence. Beyond characterization, the methodology aims to *optimize* the system, i.e., allows one to find the appropriate input parameter vector that elicits the best output response. The underlying premise of experiment design is that each experiment (e.g.: a simulation run) has a non-negligible cost.

While regression models for small dimensional parameter spaces can be built using simple factorial methods [4, 11], these methods do not ramp up to large-scale situations. Usually as the size of a model is increased in either space or number of parameters, the modeler typically retreats to a sub-goal of characterization, and instead focus on optimization alone (a.k.a. black-box *optimization*). As a result,

we replace detailed regression-like characterization with heuristic search methods. In this class of methods, a variety of "exploration" techniques are used to find regions of interest. For example, hill climbing is used to find the local extrema [31]. Many heuristic search algorithms have been proposed such as multi-start hill-climbing [33], genetic algorithms [34] and simulated annealing [35]. While these techniques tend toward the global optima in the limit, they do not have the property of finding good results quickly, i.e., they lack early-stage efficiency. We have utilized an efficient search algorithm (called Recursive Random Search [12]) for efficient large-dimensional heuristic parameter optimization. This approach thus far has yield very positive results in finding "good" global minima with few simulation runs.

Here we apply this meta-simulation technique to examine OSPFv2 convergence times during network link failures. This study includes OSPF optimizations for sub-second convergence, adapted from [53]. Here, *convergence* is defined to be time at which *all* routers in the network have a synchronized routing table or put another way, a consistent view of the routing tables is shared by all routers. We explore the cases , i.e., large-scale experiment design and black-box optimization (i.e., large-dimensional parameter state space search) using realistic topologies with bandwidth and delay metrics to analyze convergence of network route paths in the Open Shortest Path First (OSPFv2) protocol.

By using Recursive Random Search (RRS) approach to design of experiments, we find: (i) that the number of simulation experiments that must be run is reduced by an order of magnitude when compared to full-factorial design approach, (ii) it allowed the elimination of unnecessary parameters, and (iii) it enabled the rapid understanding of key parameter interactions. From this design of experiment approach, we were able to abstract away large portions of the OSPF model that result in a 100 fold improvement in simulation execution time.

The purpose of the large-scale experiment design area of our research is to systematically formulate and organize multiple simulation experiments in the quest of the general invariant relationships between parameters and protocol performance

response. To this end, we being the discussion with an overview of full-factorial design of experiments.

3.2 Overview of Full-Factorial Design of Experiments

Design of Experiments or "experiment design" is a well known branch of performance analysis, specifically, a sub branch of statistics [4, 11]. It has been used extensively in areas such as agriculture, industrial process design and quality control [11] , and has been introduced to the area of practical computer and network systems design by Jain [4]. Statistical experiment design views the system-under-test as a black-box that transforms input parameters to output metrics. The goal of experiment design is to maximally characterize (i.e., obtain maximum information about) the black-box with the minimum number of experiments. Another goal is robust characterization, i.e., one that is minimally affected by external sources of variability and uncontrollable parameters, and can be specified at a level of confidence.

The underlying premise of experiment design is that each experiment has a non-negligible cost. Simple designs like "best-guess" or "one-factor-at-a-time" designs are less favored in complex situations since they do not provide information about the interactions between parameters. Designs like full-factorial and fractional factorial (also called orthogonal designs), appropriately subjected to replication, randomization and blocking are preferred [4, 11]. The usual end-goal of formulating regression models is to observe the effects of both individual parameters and parameter interactions [4, 11]. Techniques like blocking and analysis of covariance are used to explicitly handle measurable, but uncontrollable (a.k.a. "nuisance") factors. Transforms on data (e.g., Box-Cox power-law family of transformations) can effectively aid in producing a family of non-linear regression models and stabilizing the variance of the response [4, 11].

The next step beyond characterization (i.e., developing input-output regression models) is optimization, i.e., to determine the region in the important factors that leads to best-possible response. The output (i.e., response) in general will have an unknown surface topology, also known as "response surface" [1]. The approach

[1] An example response surface is shown in Figure 3.1

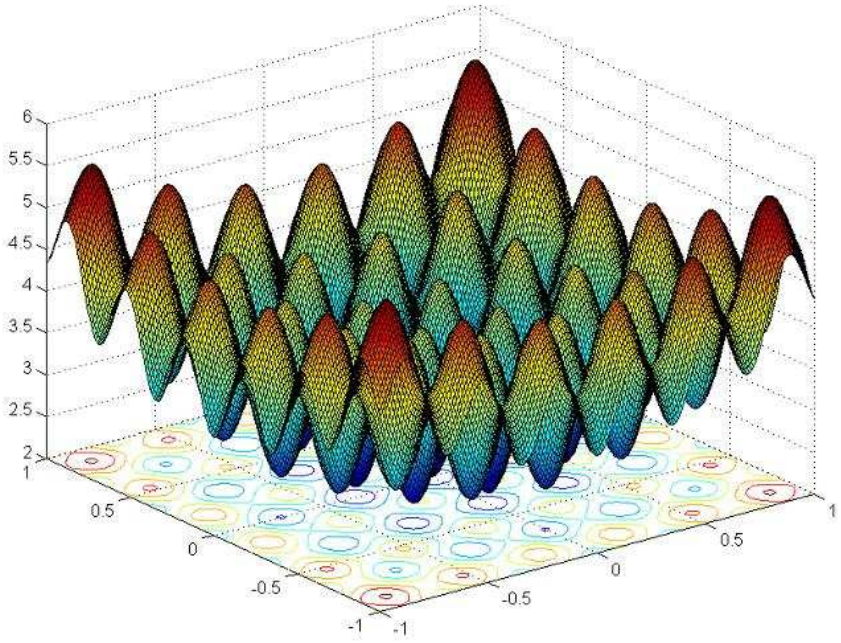


Figure 3.1: The benchmark Schwefel function as the response surface for a parameter space.

typically used involves quickly traversing the surface sequentially (by using lower-order models built with fractional factorial experiments) to reach interesting areas where more detailed (higher-order) characterization is done.

One of the significant drawbacks of the full-factorial approach is the exponential in the number of experiments that must be run as a function of the number of data points per parameter. To vastly reduce the number, search algorithms must be used, such as RRS.

3.3 ROSS.Net

Shown in Figure 3.2, ROSS.Net aims to bring together four major areas of networking research: simulation, protocol design, network modeling and measurement and experiment design. For simulation, at the heart of ROSS.Net is Rensselaer's Optimistic Simulation System (ROSS). ROSS is a discrete event, parallel execution simulation engine that is extremely modular and scalable. Running on top of ROSS is the ROSS.Net simulation model. The ROSS.Net simulation model imposes lay-

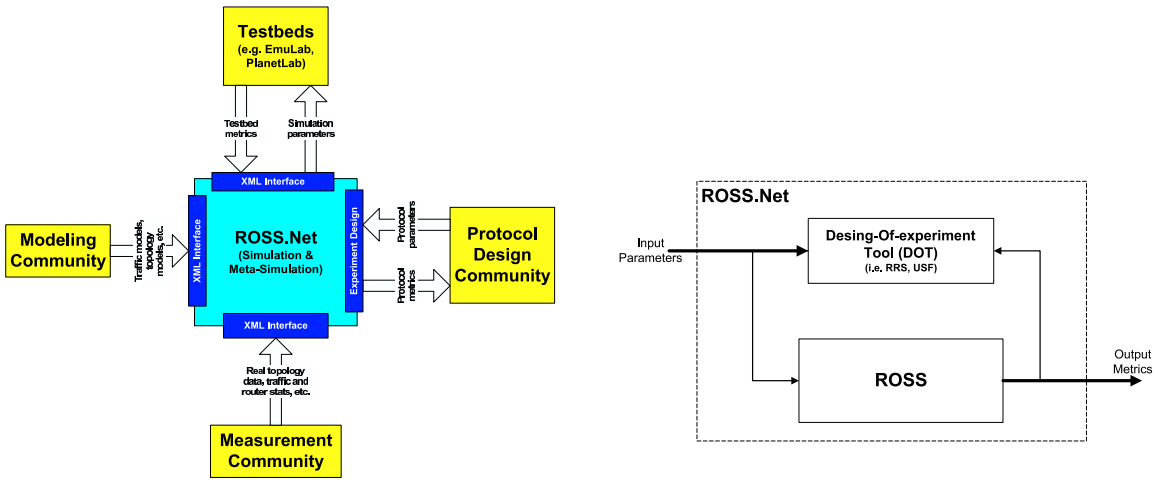


Figure 3.2: ROSS.Net and other major experimentation areas. ROSS stands for Rensselaer's Optimistic Simulation System.

ering constraints on the protocol models, performs the multiplexing operation on streams, and provides an API interface for inter-layer communications. ROSS.Net incorporates a collection of protocol libraries (e.g.: OSPFv2, TCP, UDP, Multicast, etc). It allows for protocol designers to create and examine new and incremental designs within an existing framework of network protocols. We have successfully simulated tens of thousands of OSPFv2 routers [54], tens of thousands of multicast receivers [55], and over a million TCP hosts [47].

ROSS.Net simplifies tasks such as traffic modeling and real topological representation through the use of XML interfaces. Two of the major input parameters to experiments are the topology description and the traffic scenario. These inputs describe additional parameters which control the working nature of protocols, connections, flows, and other various network elements. Our XML schema is an application independent means for describing these inputs and are reusable across multiple meta-simulation systems.

The design of experiments tool (DOT) is a new component which allows for investigators to specify the type of experiment design to generate. This builds on the previous work of the Unified Search Framework (USF) [56]. USF was designed to handle large-scale black-box optimizations. USF provides a general platform on which tailored optimization algorithms can be easily constructed. USF contains different sampling methods as basic building blocks, and creates the search algorithm

though the management of these methods. USF and ROSS.Net together allow us to take advantage of the search algorithms and large-scale network simulations and from them generate an optimized design of experiments tool, the DOT.

The DOT configures and executes the ROSS.Net core with the specified XML input parameter descriptors, simulator engine performance parameters, and protocol dependent parameters and executes the selected search algorithm(s) such as random recursive search, multi-start, or complete random sampling [12].

3.4 The OSPFv2 Design of Experiments

The goal of our design of experiments was to understand the factors determining the amount of wall-clock time required for a network of routers to detect and propagate a link state failure. Our experiment is a simulation of a network of Internet routers all operating the OSPFv2 protocol as described in [58]. For an example router network, we selected the AT&T network, as described by Rocketfuel data [52]. This network description was determined by using various network probing techniques (i.e, traceroute). The AT&T network is challenging because of it's size and complexity.

3.4.1 OSPFv2

OSPFv2 is a link-state routing protocol designed to be run internal to a single Autonomous System. Each OSPFv2 router maintains an identical database describing the internal network's topology (i.e., an Autonomous System (AS)). From this database, a routing table is calculated by constructing a shortest-path tree. OSPFv2 recalculates routes quickly in the face of topological changes, utilizing a minimum of routing protocol traffic. OSPFv2 is classified as an Interior Gateway Protocol (IGP). This means that it distributes routing information between routers belonging to a single Autonomous System. An example of an Autonomous System is the AT&T network, which is AS number 7018. Routing between ASs is handled by an external protocol, such as Border Gateway Protocol (BGP) [59].

The OSPFv2 protocol is based on link-state or shortest-path-first (SPF) technology. In a link-state routing protocol, each router maintains a database describing

the Autonomous System's topology. This database is referred to as the link-state database. Each participating router has an identical database. Each individual piece of this database is a particular router's local state (e.g., the router's usable interfaces and reachable neighbors). The router distributes its local state throughout the Autonomous System via flooding. All routers run the exact same algorithm, in parallel. From the link-state database, each router constructs a tree of shortest paths with itself as the root. This shortest-path tree gives the route to each destination in the Autonomous System [58]. OSPFv2 routers employ the HELLO protocol for establishing and maintaining communications with adjacent routers. Adjacencies are established between two routers when a HELLO protocol packet is received by one of the two routers connected by a link. HELLO packets are then sent at regular intervals between adjacent routers. Upon receiving a HELLO packet from a neighboring router, an inactivity timer is set for that router. If another HELLO packet is not received from that router before the timer expires, then the adjacency is broken and that router should no longer be used to route IP packets.

All of these aspects are modeled in ROSS.Net. However, multiple areas within a single OSPF domain is not currently modeled. In the experiments presented here, we configure OSPF to be a single large area. This was done because there is an interest in determining where OSPF ceases to execute in an efficient manner. This was a sub-goal of our experimentation.

3.4.2 AT&T Network Topology

For our network topology we selected the AT&T network. Figure 3.3 shows the core AT&T network topology which contains 11,964 router nodes and 7,491 links. Internet topologies like the AT&T network are interesting from a modeling prospective because of their sparseness and power-law structure [52]. This structure allows for a greater range of convergence times compared to fully connected networks. The OSPFv2 update packets require multiple hops in order to reach the outer edges of the network.

In performing a breadth-first-search of the AT&T topology, there are eight distinct levels. A number of routers were not directly reachable and thus were

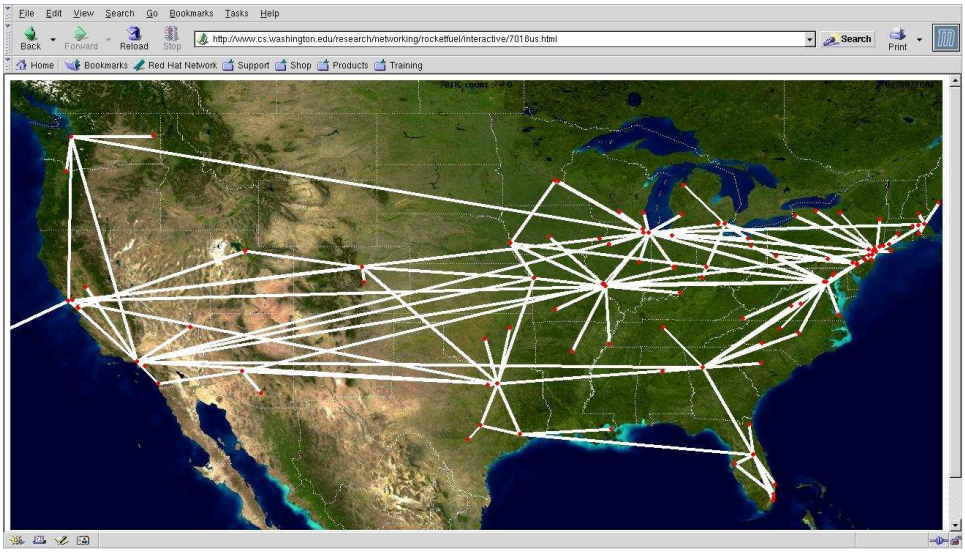


Figure 3.3: AT&T Network Topology (AS 7118) from the Rocketfuel data bank for the continental US. In our initial work, we have simulated TCP and OSPF protocols over this large-scale topology.

removed. Those routers are likely connected by transit routes. In total there are 3,371 backbone routers and at the successive levels there are 8,593 routers. The 4 ms delay that was chosen for the backbone core routers was in-line with the delays that Rocketfuel had associated with the Telstra topology backbone. An order of magnitude higher delay was selected for all lower level routers.

The bandwidth and delay for the AT&T topology is as follows:

- **Levels 0 and 1 routers:** 155 Mb/sec and 4 ms delay
- **Levels 2 and 3 routers:** 45 Mb/sec and 4 ms delay
- **Levels 4 and 5 routers:** 1.5 Mb/sec and 10 ms delay
- **Levels 6 and 7 routers:** 0.5 Mb/sec and 10 ms delay

Our experiments focused on the convergence time metric. We defined convergence to be the time at which all routers on the network have received an update corresponding to a link status change, and have recomputed their forwarding tables. In order to clearly state convergence intervals, in our simulations we have only a single link state failure per simulation and all of the OSPFv2 routers were started

in a converged state. We defined the input plane to this experiment design to be composed of the HELLO interval, HELLO inactivity interval, SPF computation interval, ACK interval and maximum transmission unit. The response plane is the convergence time from the link state failure.

The goal for our design of experiments was to adapt some of the convergence optimizations in [53] for the IS-IS protocol to the OSPFv2 RFC [58] protocol. IS-IS is a link state protocol for Cisco routers. Suggestions for lowering convergence times were: to queue HELLO packets in front of data packets, use a modern shortest-path-first (SPF) algorithm, and to give a higher priority to link state packet (LSP) propagation over SPF computation. We took the following steps to adapt the optimizations.

We did not model the data plane in our OSPFv2 routers so that HELLO packets would always be at the front of the queue. It is still possible for other control plane packets to queue in front of the HELLO packets. To facilitate a higher priority for link state propagation over SPF computation, we remove the LSP propagation timer from the OSPFv2 protocol. Now, LSP propagation will always occur immediately, and the SPF computations will always occur later. In addition, modern SPF computations would only add a small amount of time to overall convergence interval. We modeled this by adding in the amount of time stated by [53] for a topology of our size.

3.4.3 Recursive Random Search Results

As previously discussed, Recursive Random Search (RRS) is a heuristic search algorithm for black-box optimization problems. This algorithm is specifically designed to optimize dynamic network protocol parameterizations with an emphasis on obtaining "good" solutions within a limited time frame. RRS does not attempt to find a full optimization of the parameter space. The RRS algorithm maintains the high efficiency property of random sampling by constantly restarting random sampling but with adjusted parameter spaces. Because it shares this property with random sampling, it is also highly robust to the effect of random noises in the objective function. It also performs efficiently when handling an objective function that

Input Parameter	Minimum Value	Maximum Value
Hello Interval (seconds)	0.5	10.0
Hello Inactivity Interval (seconds)	1.5	8.0
ACK Timer Interval (seconds)	0.5	10.0
Maximum Transmission Unit (bytes)	500	1500
SPF Computation Interval (seconds)	0.5	10.0

Table 3.1: Random Recursive Search Input Plane.

Residuals

Min	1Q	Median	3Q	Max
-15.7286	-1.0311	0.1805	1.4811	11.5511

Coefficients

	Estimate	Std. Error	t value	$Pr(>\|t\|)$
(Intercept)	-19.838684	1.547959	-12.816	$< 2e-16$
HELLO Packet Interval	4.426648	0.121561	36.415	$< 2e-16$
HELLO Inactivity Interval	4.507337	0.169302	26.623	$< 2e-16$
ACK Interval	0.089194	0.113251	0.788	0.432
MTU	-0.001568	0.001146	-1.368	0.173
SPF Computation Interval	0.717926	0.121500	5.909	1.16e-08

Residual standard error: 4.17 on 243 degrees of freedom
Multiple R-Squared: 0.912, Adjusted R-squared: 0.9102
F-statistic: 503.5 on 5 and 243 DF, p-value: $<$ 2.2e-16

Table 3.2: RRS Linear Regression Model Output generated by R.

contains negligible parameters.

For the recursive random search algorithm we chose a wide range of input parameters, as shown in Table 3.1. We allowed RRS to search for 250 experiment runs, specifying a desired confidence level of 99%. RRS generated a convergence minimum after only 7 executions of 4.07 seconds. We fitted a linear regression model to our data using a tool called R [60], and generated the co-efficients shown in Table 3.2. After analyzing the variance on the inputs, we found the parameters that had the greatest impact on the model to be the HELLO packet interval, HELLO inactivity timer, and the SPF computation interval.

After considering the simulation model, we realized that the HELLO polling

interval is set to be the HELLO packet interval multiplied by the HELLO inactivity interval. These two parameters are have an impact on convergence because they determine the time to detect a link state failure. The other factor of convergence time is the time to propagate the link state failure to the remainder of the routers in the network. The update propagation time is defined by flooding packets throughout the network. The router which detects the failure informs all of it's remaining neighbors, who notify their neighbors, and so on, until eventually all routers in the network have received the update. The propagation delay on the updates is bounded by the amount of time it takes for the update to travel across the diameter of the network. Recall also from our definition of convergence that each router must have also recomputed their forwarding tables. So the convergence time is compounded by either how long it takes for the final router to update it's table, or by the longest SPF computation interval.

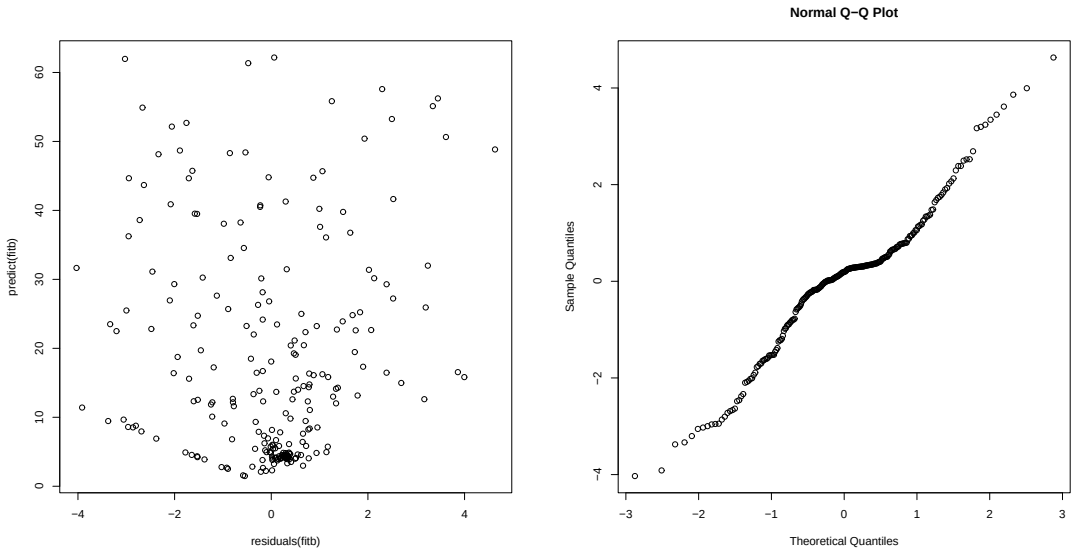


Figure 3.4: RRS Linear Regression Model Scatter and Q-Q Plots

After analyzing the variance we fitted a new linear regression model to the SPF computation interval and the cross between the HELLO packet interval and the HELLO inactivity interval, as shown in Table 3.4. This regression produced an

Input Parameter	Minimum Value	Maximum Value
HELLO Packet Interval (seconds)	0.5	10.0
HELLO Inactivity Timer (seconds)	1.5	8.0

Table 3.3: Re-parameterized Random Recursive Search Input Plane

Residuals

Min	1Q	Median	3Q	Max
-18.6508	-0.6173	0.23916	0.5768	5.3645

Coefficients

	Estimate	Std. Error	t value	$Pr(>\|t\|)$
(Intercept)	-0.31820	0.54756	-0.581	0.5617
HELLO packet Interval	-0.19646	0.11229	-1.750	0.0814
HELLO Inactivity Interval	0.13986	0.13361	1.047	0.2962
SPF Computation Interval	1.0743	0.2125	5.055	5.27e-06
HELLO interval $\times$ inactivity	0.92009	0.02436	37.773	$< 2e-16$

Residual standard error: 1.915 on 245 degrees of freedom
Multiple R-Squared: 0.9853, Adjusted R-squared: 0.9851
F-statistic: 5466 on 3 and 245 DF, p-value: $<$ 2.2e-16

Table 3.4: Re-parameterized Linear Regression Model Output generated by R

adjusted R-squared value of 98%. In order to verify the accuracy and correctness in our model we created a scatter plot of the errors versus predicted responses. The scatter plot did not show any trends in the data. The next step in our verification was to create a quantile-quantile plot of the residual errors. We observe a linear relationship between sample error and theoretical. From Figure 3.4, the linear model assumptions of normality appear to be valid.

In order to gain more detail about the interesting parts of the design, we re-executed ROSS.Net a second time with only those input parameters. Specifically, we used only the HELLO packet interval, and the HELLO inactivity interval in the same ranges as shown in Table 3.3. Again, we allowed RRS to search for 250 iterations and with a confidence interval of 99%. This experiment generated a convergence minimum after only 129 executions of 0.93 seconds.

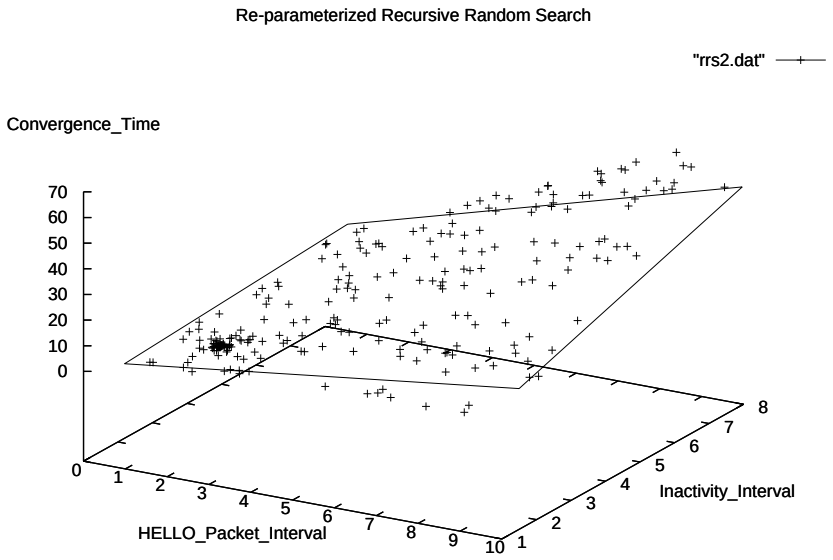


Figure 3.5: Re-parameterized Random Recursive Search. The response plane with a local optimum toward the smaller input parameters.

Input Parameter	Minimum Value	Maximum Value
HELLO Packet Interval (seconds)	0.03	1.0
HELLO Inactivity Interval (seconds)	1.5	2.0

Table 3.5: Re-scaled Random Recursive Search Input Plane

In this series of experiments, a sub-second range for the convergence interval is observed. Figure 3.5 shows that the HELLO packet interval and the HELLO inactivity interval form a plane with one corner tilting downward toward the smaller values. This low corner is anchored by the best convergence result given by RRS. We report a clustering effect occurring on the graph, which is attributed to the RRS algorithm centering upon a given input. It appears that RRS was successful in isolating the HELLO packet interval and the HELLO inactivity interval at the low end of their ranges.

The initial goal of our design was to determine if we could adapt some of

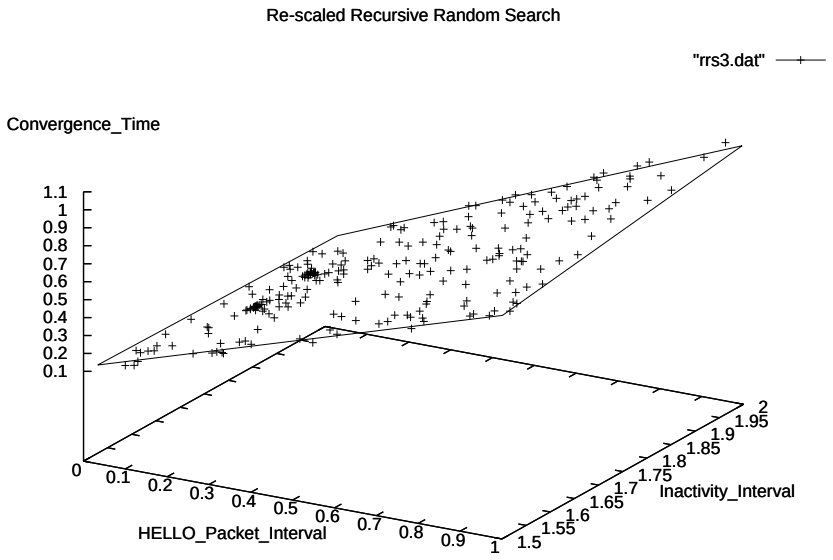


Figure 3.6: Re-scaled Random Recursive Search. The response plane is tilted at a greater angle as we re-scale the input parameters by a factor of 10.

Jacobsen's ideas to the OSPFv2 protocol and achieve convergence times on the order of magnitude in the tens of milli-seconds. Having isolated the effective parameters of the simulation model, we being to see points in the sub-second range. So we re-scaled the experiment into the range of the input values that generated those results. We noticed that the HELLO packet interval was in the range suggested by Jacobsen. We reduced the range for the input parameters as shown in Table 3.5 and Figure 3.6 shows the results of this experiment. All of the convergence values are below a second, and the best values are in range of tens of milli-seconds.

3.4.4 OSPF Model Critical Path Analysis

Berry and Jefferson [61], developed a technique called *Critical Path Analysis* to determine the optimal parallel simulation execution time. Armed with the results from RRS, we apply this technique here to examine what the critical path for OSPF convergence within the context of our model.

Experiment	HELLO Interval	HELLO Inactivity	ACK	MTU	SPF	Non-optimized	Optimized
1	0.5	1.5	0.5	500	0.5	0.895151	0.895151
2	0.5	4.75	0.5	500	0.5	2.707651	2.707651
3	0.5	8	5.25	1000	10	13.520151	13.520151
4	5.25	1.5	0.5	1000	5.25	9.207651	9.207651
5	5.25	4.75	5.25	1000	0.5	23.488901	23.488901
6	5.25	1.5	10	1000	10	13.957651	13.957651
7	5.25	8	10	1500	10	46.770151	46.770151
8	10	4.75	5.25	1000	0.5	44.270151	44.270151
9	10	4.75	0.5	500	5.25	49.020151	49.020151
10	10	4.75	10	1500	10	53.770151	53.770151

Table 3.6: Validation of Optimization using the VSNL (India) Network Topology

After examining the results from our designs, we observed that the two main components of convergence are *detection* and *propagation*. Detection is simply the amount of time that elapses between a link state change and the time at which the inactivity timer fires. The second component, propagation is determined by the longest path the link state update travels through the network. The longest path is not immediately determined by the number of hops between the originating router and the final router to receive the update. It is possible for an update to take many hops over high-speed links and still not be on the longest path. Conversely, it is possible to take only a small number of hops over very low speed links and be on the longest path. Realizing that these two factors have the highest impact on convergence time, **we observed that to accurately model convergence in any network, only the set of nodes which encompass the longest path through the network require simulation.** This observation has been used on other OSPF optimization [62].

Using ROSS.Net, we simulated the AT&T network which contained almost 12,000 routers. The model was instrumented so that each router would keep track of which routers they received link state updates from. Once the update reached the final router in the network, we then backtracked this path to find the longest path in the network.

In order to validate the modeling optimization, we computed a full factorial design of the VSNL, India topology. This topology contained only 291 routers, which allowed us to compare the optimization to the full model simulation results. Table 3.6 shows the results of ten of those experiments runs. The optimization results generated exactly the same output for the optimization as we would have recieved

had we modeled the entire topology.

Simulations on the full AT&T network required anywhere from one half hour to initialize to several hours depending on whether the routing tables need o be computed. In addition to the time required to initialize the simulation, execution time took on average one second of wall clock time to simulate one second of simulated time. In other words, to simulate an hour of OSPFv2 traffic in ROSS.Net required almost one hour of real time.

After determining this optimization, we computed the longest paths through the AT&T network for each of the updates generated from a single link failure. These paths we 11 and 12 hops long respectively, and the paths only varied at a single node. This means that in order to simulate the entire AT&T network for convergence times only required actually simulating 13 total routers. Obviously, this reduced the time required to run the simulation to the order of seconds. At this stage, ROSS.Net only takes a second or so to initialize, and on an average run about 0.0006 seconds to execute. The simulation results presented required only about 12,000 events to generate the convergence times in a simulation of 100 seconds.

Using this optimized model, we are now able to compare the full-factorial experiment approach to RRS.

3.4.5 Analysis and Comparison of RRS to Full Factorial

In the previous section we showed how meta-simulation can reduce the amount of time to acquire meaningful results from our models by employing algorithms such as Recursive Random Search. Using RRS, we were able to generate all of our results in only 750 experiments, or simulation executions. However, RRS had not sampled a large area of the state space, so how confident can we be in the results?

We computed a full factorial model in order to validate the results we gained using ROSS.Net. We used the same 5 input parameters as in the RRS experiments, and 7 levels as shown in Table 3.7. In Table 3.8, we fitted a linear regression model to the data and found that the same three input parameters had the most effect on the model. The full factorial model produced an adjusted R-squared value of 85%.

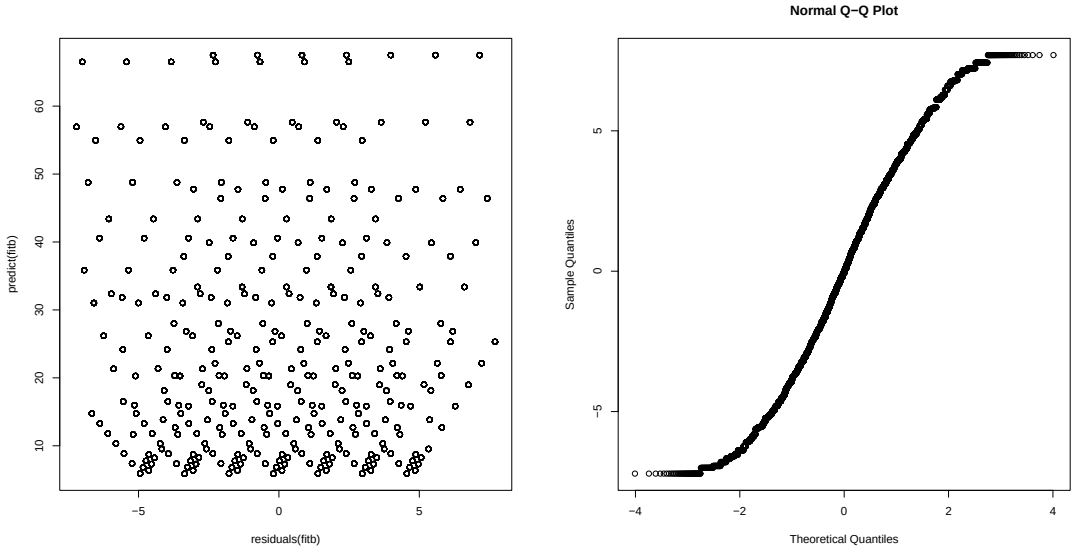


Figure 3.7: Full Factorial Design Scatter and Q-Q Plots

Input Parameter	Minimum Value	Maximum Value
Hello Interval (seconds)	0.05	10.0
Hello Inactivity Interval (seconds)	1.05	8.0
ACK Timer Interval (seconds)	0.5	5.0
Maximum Transmission Unit (bytes)	500	1500
SPF Computation Interval (seconds)	0.5	5.0

Table 3.7: Full Factorial Model Input Plane

We were confident that RRS was properly modeling the same parameter space as we would have explored had we done a more detailed full factorial. Plotting the same two most effective parameters as we modeled in RRS, the scatter plot did not show any trends in the data, as shown in Figure 3.7. The final step in our verification was to create a quantile-quantile plot of the residual errors. We observed a linear relationship between sample error and theoretical. From Figure 3.7, the linear model assumptions of normality appear to be valid for the full factorial model.

This full factorial design generated 16,807 experiment runs, 20 times more than the RRS design, and yielded less information in the areas that we were interested

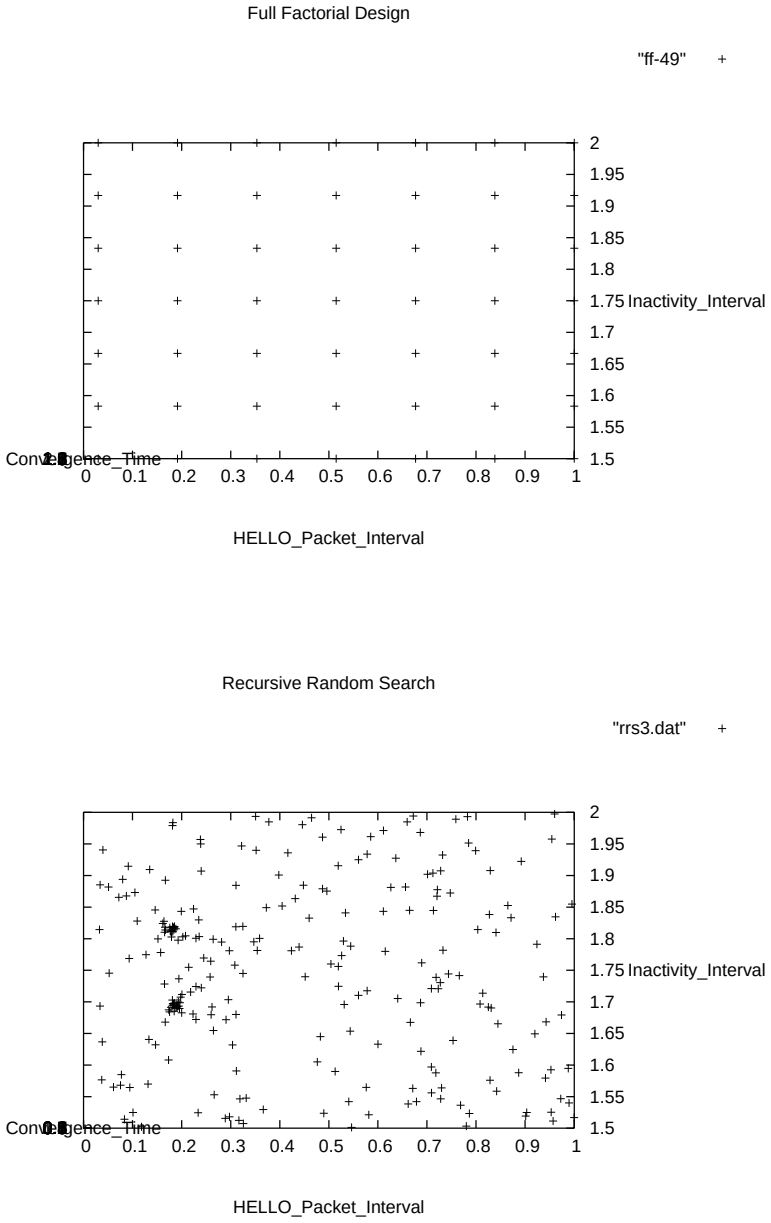


Figure 3.8: Each graph is plotted in 3D. The ζ dimension represents the convergence times collected. Each plot shows the points used to generate the individual response planes.

Residuals

Min	1Q	Median	3Q	Max
-16.567	-4.262	0.523	3.762	15.306

Coefficients

	Estimate	Std. Error	t value	$Pr(>\|t\|)$
(Intercept)	-2.132e+01	2.472e-01	-86.26	$< 2e-16$
HELLO Packet Interval	3.844e+00	1.626e-02	236.49	$< 2e-16$
HELLO Inactivity Interval	4.690e+00	2.376e-02	197.41	$< 2e-16$
ACK Interval	-1.528e-17	1.626e-02	-9.40e-16	1
MTU	-1.335e-19	1.544e-04	-8.64e-16	1
SPF Computation Interval	9.796e-01	1.626e-02	60.26	$< 2e-16$

Residual standard error: 6.674 on 16801 degrees of freedom
Multiple R-Squared: 0.8543, Adjusted R-squared: 0.8543
F-statistic: 1.971e+04 on 5 and 16801 DF, p-value: $<$ 2.2e-16

Table 3.8: Full Factorial Model Output generated by R

in studying. Figure 3.8 illustrates the amount of detail generated by RRS versus the Full Factorial design for convergence times in the sub-second range. RRS also generated a "good" value for a convergence time of 0.11 seconds, which was within 7% of the Full Factorial design best value. While we could have generated a full factorial design using the final RRS input parameter ranges, we would not have had the benefit of the knowing that was in fact the area of interest, beyond our ability to analyze the system. In fact, we purposely chose the full factorial design presented here because we wanted to be certain about the nature of the system. It was necessary to explore a large range in order to validate our results in the RRS design.

3.5 Understanding OSPF and BGP Interactions Using Efficient Experiment Design

Several problems in large-scale networking have been a research challenge [122]. Particularly, understanding of routing protocol dynamics and interactions in large-scale is an important problem due to its immediate affect on current practice of

inter- and intra-domain routing.

Recent research of Internet-scale routing protocol deployments has primarily been focused on measurement data taken directly from the Autonomous Systems (ASes) which exist in today's Internet. These results have been valuable because the measured protocol response has the unique characteristic of *realism* with real routers executing real protocol implementations and routing real data [118, 120, 123, 124, 125, 121, 126]. Measurement data has been limited, however, in that it is difficult to quantify the causes of the measured effects. Typically, to identify causes behind particular routing updates or dynamics, it requires estimation and matching methods [118, 120] which are prone to error. In addition, because of the proprietary nature of the data, it is difficult to measure the external causes of many of the effects. These limitations can lead to a harmful, one-sided view of the Internet which considers only the local domain. *In order to truly understand the nature of the Internet routing, we must consider multiple ISP domains and the effects of the unique management within each ISP domain.*

Network simulation allows us to investigate routing dynamics across all domains because we have complete information. It allows us to consider multiple ASes, and to quantify the possible effects both from within and from outside a particular domain. The trade-off is realism. Minimizing this trade-off, we have developed protocol models which adhere closely to the RFC specifications, and use prevailing topology measurement data such as Rocketfuel [52]. In addition, we model network effects such as link metrics at varying levels of robustness. Realizing that router manufacturers do not always adhere to RFC specifications in their protocol implementations, we attempt to classify routing dynamics between OSPF and BGP in a general form.

Using network simulation and incorporating "realistic" information when possible, we can begin to quantify specific protocol effects and make *general* qualitative statements about the nature of the networks. In particular, we are searching for the underlying invariant relationships between network protocol parameters and protocol performance response.

We focus on characterizing Internet routing protocol performance response

by the number of update messages generated by each routing protocol model (i.e., OSPF and BGP) as a function of protocol timers, variables, and algorithm decisions. Measuring protocol response as a function of update messages generated is important because this is where the interactions between protocols are defined. For example, in a network where route flapping is occurring, routers may converge quickly between the two routes as they change. Measuring convergence as the response would lead us to believe there are no negative effects on OSPF from BGP. Similarly, measuring link congestion does not lead to the negative effect because we may observe only a fractional difference in bandwidth consumption over time. Each route removal/installation can be directly measured within the OSPF domain. By measuring the number of updates generated by the OSPF domain, a clearer picture of the negative effects emerges. Of course these negative effects lead to slower convergence times and greater link utilization, but these are secondary measures. By measuring the interactions directly (i.e., updates messages) we are able to quantify the direct impact on the network without having to separate out other effects. This allows us to begin answering the questions, *does my intra-AS management policy adversely affect my inter-AS policies, and vice-versa?* and *which is the best approach to minimizing negative effects between protocols?*

3.5.1 Why are Protocol Interactions Harmful?

The common approach of hierarchical and layered design of several networking protocols also exist in Internet routing. When designing these protocols the only important matters are dynamics of the protocol under consideration and the assumptions made about the other protocols below or above the hierarchy or the layering. However, this leads to unexpected, typically implicit, interactions between the co-existing protocols.

Network protocol weaknesses are not fully understood until they have been deployed in large-scale production environments. There is probably no better example of this than the BGP protocol. Clear limitations of this protocol have been illustrated since its introduction (e.g. BGP storms [127, 128], the stability problem [129, 130]), and several solutions (e.g. route reflection) have been proposed and

implemented to overcome them.

These investigations have typically focused on the individual effects of the parameter settings, and neglected the external effects on protocol performance. The problem that we see is that there are two conflicting views of the network: intra-domain and inter-domain routing. Our concern is that decisions made to efficiently route data *within* a domain are directly affecting the ability of the network to route data *across* the domain.

One immediate cause for concern is **Hot Potato Routing** [119], though some researchers have similarly voiced concerns over Cold Potato routing [131]. Hot Potato routing is interesting because it allows a router which does not necessarily contain an up-to-date view of the internal network to make decisions about how to route traffic through that network. As a practical example, the BGP protocol makes a decision about which routes to install based on the distance of each competing intra-domain route. The problem arises when this information is not stable. BGP routers typically are responsible for generating large flows of traffic data into and out of the network. The major concern is that a *small* degree of unstable routing information may inversely impact a *large* amount of network traffic.

Traffic shifts because of OSPF-BGP interactions happen typically at ASes with multiple paths to another ISP. More than half of the non-ISP ASes have such multiple paths to a tier-1 ISP [120]. Previous work indicates that Hot Potato changes can cause major shifts in routing and network traffic. In addition, hot-potato routing may add to the degradation of forwarding plane convergence and generate temporary forwarding plane loops. Finally, Hot Potato routing leads to measurement inaccuracies in probes of the forwarding plane, and the external visibility of BGP routes.

3.5.2 Contributions

Our main goal is to minimize the number of negative interactions between the OSPF and BGP protocols in a multi-AS environment. In particular, the number of OSPF updates caused by BGP protocol dynamics, and the number of BGP updates caused by OSPF protocol characteristics should be *minimized*. Past investigations

relying on measurement data have been constrained to one-way analysis of either BGP dynamics on the OSPF protocol, or OSPF dynamics on the BGP protocol, and then only for a single AS.

In addition, our experiment design framework includes an analysis of both high and low level protocol dynamics. In addition to the effects of the BGP decision algorithm on protocol performance, we also study the effects of BGP and OSPF protocol timers and variables. Example timers include: liveness timers such as HELLO/Inactivity in OSPF and KeepAlive/Hold in BGP, and filtering timers such as FloodInterval and MinRouteAdvertisementInterval (MRAI).

Our work also allows the unique ability to turn on/off certain parameters within the BGP decision process. It would be difficult to imagine re-configuring an actual AS in order to turn Hot Potato routing on or off, and re-testing the effects under identical conditions. This is an automatic benefit with simulation. With simulation we can search protocol timers, variables and major algorithmic details in an on/off fashion to quantify the effects in the face of varying network topology robustness. Measurement data is restricted to the actual (or "real") network stability over the time the measurement is made. Network simulation allows us to vary topology robustness by having more/less aggressive management strategies.

Our major contributions in this area can be itemized as follows:

- *A framework to optimize OSPF and BGP protocol response:* Based on a controlled large-scale simulation of OSPF and BGP, we present a framework to optimize a particular protocol performance metric over possible parameter search space through a heuristic search algorithm. We particularly use the number of updates with various original causes (i.e., OSPF-caused, BGP-caused) as metrics to optimize several OSPF and BGP parameters used in practice. Instead of measurement-based estimation and matching methods, we leverage a controlled simulation environment to trace exact causes of each routing update in the system.

- *Experiment design approach to understand OSPF and BGP interactions:* We devise a systematic design of experiments methodology to investigate particular effects of three classes of protocol parameters in the total number of

negative interactions between OSPF and BGP. We measure the negative interactions as the total number of OSPF-caused BGP updates and BGP-caused OSPF updates. We investigate three classes of parameters as factors into the negative interactions: (i) OSPF timers, (ii) BGP timers, and (iii) BGP decision making attributes.

- *Large-scale realistic OSPF and BGP simulation:* We present large-scale simulation of OSPF and BGP in a single model. Our simulation model uses realistic inter- and intra-domain topology generated from Rocketfuel [52] measurement data and nearly complete RFC implementations of the OSPF and BGP protocols.

3.6 Network Protocol Simulation Models

The network protocols simulated include: BGP4, OSPFv2, and IPv4. For scalability, the TCP layer was not simulated for the BGP routers. The assumptions made for scalability and time available for development are addressed in this section.

3.6.1 OSPFv2 Model

OSPFv2 is a link-state routing protocol designed to run internal to a single AS. Each OSPFv2 router maintains an identical database describing the AS network's topology. From this database, a routing table is calculated by constructing a shortest-path tree. OSPFv2 recalculates routes quickly in the face of topological changes, utilizing a minimum of routing protocol traffic.

The OSPFv2 protocol model is capable of simulating three types of LSAs: Router, Summary and AS External. Router LSAs represent the network layer reachability information of individual routers. There is one Router LSA per OSPF router in the topology. Summary LSAs represent reachability information for Area Border Routers and describe the connections to multiple areas. AS External LSAs describe reachability information external to the OSPF area and are created by the BGP model. When executed alone, the OSPF model only creates and floods Router and Summary LSAs. When executed in conjunction with the BGP model, AS external LSAs are created by the BGP protocol directly into the OSPF model executing

on the same router. Router LSAs are only advertised by the owning router, while Summary LSAs may advertise the same reachability information across multiple destinations. According to OSPF RFC [58], when two summaries advertise the same information, the decision is to add the LSA with the lower advertising router ID. We developed our OSPFv2 simulation model as a nearly complete RFC implementation. The only exception is that Network LSAs were not modeled, and so the topology was not configured with stub networks.

Several input parameters are available within the OSPF model. Parameters affecting the HELLO protocol include the Hello Timer and the InactivityInterval. The FloodInterval, SPFInterval and ACKInterval affect the convergence times of the OSPF model. The flood timer allows for aggregation of the LSAs being flooded, and the SPFInterval allows for multiple LSA updates to be aggregated into one shortest path first computation. Finally, the ACKInterval delays the acknowledgment of LSAs received by a router.

3.6.2 BGP4 Model

BGP4 speakers exchange network reachability information with each other. The network reachability information includes the list of ASes that reachability information traverses as well as source route information. This information is used to construct a graph of AS connectivity among BGP speakers in separate ASes. Two routers must form a transport protocol connection between one another before BGP begins exchanging routing information. We have chosen not to simulate the TCP layer to increase the scalability of the simulation. Therefore, the connection is modeled directly in the BGP model by sending a CONNECT message prior to exchanging OPEN messages. The BGP model updates routing tables when new information about the BGP topology changes. The BGP model periodically sends KeepAlive messages to ensure the viability of the underlying connection. If a neighboring BGP speaker becomes unreachable, then the routing table is modified and routes advertised by the neighbor are withdrawn. BGP does not prescribe periodic refresh messages of the routing table, and so the only information available from currently connected speakers is contained in the Routing Information Table (RIB)

BGP Decision Algorithm
1. Highest Local Preference
2. Lowest AS Path Length
3. Lowest Origin Type (0 iBGP, 1 eBGP, 2 Incomplete)
4. Smaller MED (iff next hops equal)
5. Lowest IGP Cost
6. Lowest Next Hop
7. Lowest BGP Identifier
8. Vendor-dependent Tie Break

Table 3.9: Stages of the BGP decision algorithm for route selection.

[57].

Our BGP4 model simulates both eBGP and iBGP. External peers were defined by connections to BGP routers outside the current AS. Internal peers were connected as a full mesh between all eBGP routers in a given AS. With the exception of the error notification details, we have nearly fully modeled the BGP4 protocol per the RFC specification. In BGP, the decision making process occurs in three phases. The first phase is related to calculating route preferences according to owner-defined policies. The second phase selects the best route to each destination based upon the route attributes, and installs those routes into the local RIB (Loc-RIB). Phase 3 involves route aggregation and dissemination, which we did *not* model since route aggregation and information reduction are not described in the RFC and are optional and commercially dependent. Routes are disseminated as appropriate for eBGP and iBGP, and control traffic minimized by using the MinRouteAdvertisementInterval (MRAI). The specification simply calls for MRAI seconds to elapse between successive route updates between any two BGP speakers.

We have modeled all other parts of the decision making algorithm, illustrated in Table 3.9, and the default decision if a tie exists at all levels is to keep the existing route. Conversely, if no route exists, the route in question is always added to the RIB. In our model the vendor dependent tie breaking decision is to keep the existing

route.

In order to reduce the complexity of the design of experiments, input parameters are configurable at the AS level. Some of the variables in the BGP decision stage are simply on/off, such as MED and Hot Potato routing, which signify that these features are either enabled or disabled in the AS throughout the simulation. Other variables can take a value in a range, such MED, path padding and Local-Pref, which indicate the values of each feature, if enabled. So if MED is enabled in an AS, we can also select a value for it's policy. Conversely, if MED is disabled, then the AS will have no MED attribute set. Additional parameters involve the timers in BGP model, such as the MRAI which significantly affects the total number of update events in the system. KeepAlive messages are affected by the interval at which KeepAlives are sent; the HoldInterval, or just Hold, determines how many KeepAlive messages can be missed before a connection is disabled.

3.6.3 IPv4 Model

The IP simulation model we developed is very simple and only responsible for keeping statistical data such as packets forwarded, dropped or completed.

The main function of the IP model is to determine the destination port for each packet in the system. The IP model determines which port should be used by first determining if a link to the destination exists for the packet. If not, then a routing table lookup is done. If this also fails, then the packet is dropped by the network.

3.7 Difficulties in Large-Scale Network Simulation

3.7.1 Generation of Inter- and Intra-domain Topology

We used the topological data measured by the University of Washington's Rocketfuel project [52]. In totality, the Rocketfuel data identifies internal AS topologies for 10 major ISPs which cover a large area of the world. Our simulation included 5 of these ISP topologies. We chose the ISPs based upon reachability, ISP size and number of external routers detected. For example, for scalability purposes, the AT&T and Sprint topologies were excluded from this study, both having greater

than 10,000 routers. The Verio map was not used because the high number of external routers would have required greater than 3.5 million iBGP connections. Finally, the Telstra and VSNL maps covering India and Australia were not used because they could not be connected to the remaining ISPs from the available data.

Rocketfuel data lists routers as having both internal and external ISP connections. We established an OSPFv2 router at each router in the topology. Also, we determined that routers which contained one or more external connections to be BGP4 routers in addition to an OSPFv2 router. This means that some routers were running only OSPF, while others modeled both BGP and OSPF protocols. Each ISP was configured as a single AS, and within routers were broken down into two additional levels: areas and subnets. To determine the areas and the subnets, we used the IP addresses of the individual machines. For example, if two machines shared the same class A, B and C address, then we placed them into the same subnet. Areas were determined in the same way using the class A and B prefixes.

Because the Rocketfuel data relies on traceroute to determine the ISP topologies, certain limitations had to be addressed. Rocketfuel data does not define the bandwidth, speed or delay of the links. Link bandwidth and delay classes were defined for the different Rocketfuel-determined router levels. The bandwidth and delay for the topology is as follows:

- **Level 0 routers:** 9.92 Gb/sec and 1 ms delay
- **Level 1 routers:** 2.48 Gb/sec and 2 ms delay
- **Level 2 routers:** 620 Mb/sec and 3 ms delay
- **Level 3 routers:** 155 Mb/sec and 50 ms delay
- **Level 4 routers:** 45 Mb/sec and 50 ms delay
- **Level 5 routers and below:** 1.55 Mb/sec and 50 ms delay

Table 3.10 outlines the details of the multiple AS topology. BGP routers within an AS are fully connected to form the iBGP domain. The degrees are listed for each AS to every other AS, and the total number of BGP connections is listed along the diagonal.

ISP	iBGP Conn	AS0	AS 1	AS2	AS3	AS4
AS0: AboveNet	2,500	199	8	12	18	161
AS1: EBONE	16,384	8	38	6	12	12
AS2: Exodus	50,176	12	6	53	9	26
AS3: Tiscali	441	18	12	9	50	11
AS4: Level 3	7921	161	12	26	11	210

Table 3.10: Rocketfuel ISP Topology Parameters

3.7.2 Realism in Network Simulation

While this investigation lacks many of the details which currently exist in today's Internet, we have attempted to incorporate realism when possible. Our use of Rocketfuel ISP topologies is one example. Researchers in [117] point out that Rocketfuel topologies over estimated the numbers of routers and links in the Sprint topology. This required us to go back to the raw traceroute data and resolve aliases and to find the actual links between AS domains. In our own use of the data, we have found it difficult to map the link statistics to the ISP maps, and so have generated a reasonable link hierarchy based on current technology.

Also, we note that link metrics are typically related to link length, and link status changes are not typically uniformly random. The ROSS.Net tool has the facilities for modeling individual link failures and metric updates in a variety of ways, but for our purposes a general model suffices. In fact, while we have the capability of defining each router individually, we have chosen to apply router settings on a per AS level.

We have attempted to incorporate some degree of realism, however, at some point a generalization must be made. This generalization is still valuable because it allows us to determine casually what the effects are within large-scale networks. What is unique is that we are attempting to answer these questions within a simulation framework at all, and that we are capitalizing on the fact that in simulation you have a complete view of the model. We use this feature here to not only determine interactions between the protocols, but also to investigate the effects of different management perspectives, such as, *what if all ISPs shared information to reduce update messages in the control plane?*

3.7.3 Initialization of Protocols

Typically a simulation start state must be defined for the model. For a large-scale network of several ISPs like we have, it is too complex to determine the start state of the model algorithmically. Summary LSAs and BGP routes are determined by the execution of the routing protocols within complex networks. Also, it is too costly (not to mention unrealistic) to start the simulation as though each machine has been powered on at time zero. We wanted the network to begin in a converged state, before measuring the changes that occur as we dynamically modify link states over time in our experiments.

In order to achieve a converged state among multiple protocols we allowed each model to converge separately, in two stages: (i) allow OSPF to converge for each intra-domain ISP network, and (ii) using the previously converged OSPF domain, allow BGP to converge for the inter-domain topology. It is important to converge the intra-AS domains first, because the BGP domain relies on it, and so that the AS External LSAs can filter through as much of an AS as possible.

The first stage allows the OSPF model to converge and sets up the intra-AS routing topology. Converging Router LSAs is simple enough to perform algorithmically. Which summary LSAs should be installed per router is more difficult to determine because of loops in the topology. During this stage summary LSAs are flooded through each network. Once the flooding comes to an end, the network is in a converged state for OSPF. At this point, the LSA database contains the correct converged information for Router and Summary LSAs. Each router writes their individual LSA database to disk and uses it during the initialization phase for each successive execution.

The second stage involves converging the BGP4 protocol model. It is possible that certain configurations of the BGP model may not lead to a converged state. Convergence was possible here because we did not model BGP policies and because we selected model settings which broke ties consistently across all ASes. For consistency, the converged state must reflect the input parameters to be studied by the design of experiments, therefore multiple BGP start states were computed. If the design of experiments includes the BGP decision algorithm stage for hot-potato

routing, then the hot-potato routing parameter was also used to determine the converged state. Each BGP router sends the appropriate OPEN messages to its peers. Once all routes have been disseminated throughout both the BGP and OSPF models, and there are no outstanding Adj-RIB-out entries to be sent or OSPF LSAs to be flooded, the model is said to be converged. Please note that we did not add special code to guarantee BGP convergence and that each converged state was genuinely converged. Finally, each BGP router writes their individual RIB to disk and simply reads it in during the initialization phase for each future execution.

In each convergence stage, completion of route dissemination and LSA flooding was determined by setting the end time of the model arbitrarily high. No link attribute changes were performed in the convergence executions, so the topology remained statically defined. We also turned off the HELLO and KeepAlive protocols so that only events related to routing would be created. We determine that a model converged if and only if the simulation ended prematurely (ran out of events to process). If a simulation does not terminate prematurely when a high end time is defined, then routing events are not "settling down" and the models are not converging. This can happen in BGP if route flapping occurs and does not terminate. If this occurs in the OSPF model, then an error has occurred because it is guaranteed to converge.

Once the converged state has been created, it can then be used in subsequent experiments where HELLO and KeepAlive messages are again turned on.

3.8 OSPF and BGP Interactions

The goal of our experiment design is to investigate feature interactions between the OSPF and BGP protocols on a "realistic" network topology. In order to quantify the effects of possible interactions, our simulation experiments measured the number of update messages generated by each protocol. Network performance analysis typically focuses on the behavior of a single protocol at a time, and from within a single AS. By optimizing the response for multiple protocols and ASes simultaneously we can analyze the performance of the network overall as well as from various perspectives.

Towards this goal we have generated 3 designs of experiments. The first design investigates the effects of various network management policies on the response. Broadly, there are two approaches to network management at the AS level: greedy and cooperative. We define a greedy strategy as one in which the management policy promotes efficiency within the AS without consideration of the effects on the surrounding ASes. By contrast, a cooperative strategy is one in which the efficiency goal is considered across all of the ASes first.

The second design investigates the effects of cold versus Hot Potato routing in the BGP protocol model. This is significant because Hot Potato routing in BGP relies on information from the OSPF protocol. Because there is a direct correlation between the models, we expect there to be a direct feature interaction as well.

The final design considers the effects of the parameters on the response with varying degrees of network robustness. Here we perform a full-factorial on the topology parameters: link stability and link weight changes. The goal is to determine the range of the parameters and their interactions under varying network conditions.

3.8.1 Response Surface

Our response surface is defined by the number of network topology update messages exchanged by the BGP and OSPF protocols in the control plane. There are four types of update messages possible:

- OSPF caused OSPF updates (OO)
- BGP caused BGP updates (BB)
- OSPF caused BGP updates (OB)
- BGP caused OSPF updates (BO)

There are two types of changes which may occur in the topology: link status changes and link weight changes. The OSPF protocol detects link status changes via the HELLO protocol, and the BGP protocol via the KeepAlive Timer. Link weight changes are only detected by the OSPF protocol and are detected directly. When the OSPF protocol detects a change in the topology, it creates new LSAs

appropriate for the cause and floods them throughout the OSPF domain. As the new LSAs are flooded they are accounted for in the "OSPF caused OSPF updates" statistic. The same is true for BGP caused BGP updates, and we do not distinguish between eBGP and iBGP route updates.

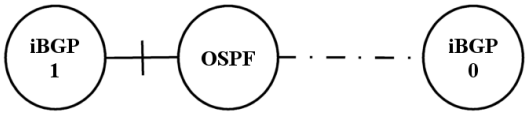


Figure 3.9: OSPF Caused BGP Update: When the link between the OSPF router and iBGP 1 goes down, the iBGP connection between 0 and 1 is also broken. The OSPF network between iBGP nodes fails to route BGP KeepAlive messages, iBGP 0 removes routes learned via iBGP 1. The cause of the BGP update WITHDRAW messages is said to be "OSPF caused".

Illustrated in Figure 3.9, OSPF caused BGP updates are measured when the connection between two iBGP peers changes. This signals a change in the underlying OSPF network between the peers, and so the cause of the subsequent updates are attributed to the OSPF protocol. For example, a link which was previously down in the intra-AS domain becomes available again, and the OSPF network rebuilds the corresponding routing tables. The new routing tables allow BGP KeepAlive messages to suddenly start getting through again, and reachability information is exchanged via update messages.

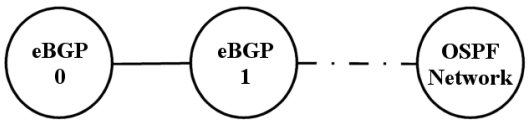


Figure 3.10: BGP Caused OSPF Update: When the link between eBGP 0 and eBGP 1 becomes available, eBGP 1 creates an AS external LSA, and floods it into the connected OSPF network. The cause of the resulting flood of OSPF LSA update messages is said to be "BGP caused".

BGP caused OSPF updates are measured when an eBGP router creates or installs a new route to a destination IP prefix, shown in Figure 3.10. The AS External LSA created by the IGP domain is tagged as being caused by BGP and at

every hop throughout the flood is measured as such. Not all AS External LSAs are caused by BGP. OSPF routers must exchange their entire LSA database when a link becomes available, and these LSAs must be flooded throughout a domain according to the OSPF RFC.

$$BGP\ caused\ OSPF\ +\ OSPF\ caused\ BGP\ Updates \tag{3.1}$$

Because we are specifically interested in feature interactions between the OSPF and BGP protocols, our main response surface is defined in Equation (3.1). Also, because the interactions are implicit in the models, specific code had to be added to the models to detect and mark updates as to their cause, and tracked throughout the system for quantification purposes.

3.8.2 Network Topology Stability

Recall that our network protocol models start in the converged state for each experiment generated by the optimization. In steady state, no control plane update messages are exchanged, other than periodic OSPF LSA refreshing. BGP does not require refreshing of the RIB. In order to generate update messages in the system, two types of network events were modeled: link status changes and link weight changes.

Link statuses are either *up or down* and occur with a uniform random probability over the simulation endtime. These events can model either link congestion in the data plane or actual link availability on a given timeline. The probability that a link status may change in the given simulation endtime is varied to model different levels of network topology stability. The stability levels are: 1%, 10% and %15 over runtime. While it was shown in [100] that some links fail far more frequently than others over a given interval, generalizing link failures uniformly allows us to investigate varying degrees of network topology stability. While the system has the capability of modeling individual links, creating a more "realistic" link failure model is beyond the scope of this investigation.

Link weight changes follow the same uniform random probability over the simulation endtime, but rather than act as up/down events, they affect the network

by varying the metric on the links. Also, link weight change events are delivered directly to the affected OSPF routers and are modeled as network administration events which occur through either human contact or programmatically. Each router originating an LSA containing the affected link refreshes the LSAs containing the link in question. Each new link metric is chosen randomly over the ranges: ± 10, 25, and 50 units.

3.9 Design of Experiments

We present here three investigations with the goal of generally characterizing the system under test in variety of conditions. The first experiment design considers varying network management perspectives. These perspectives each attempt to minimize the response as related to either a global or local perspective. One example of a local perspective is optimizing the OSPF domain without considering the impact on the BGP domain. The global perspective implies all ISPs working together to reduce control plane traffic.

The second design investigates cold- versus hot-potato routing policies within an AS. This investigation focuses on the BGP attribute, MultiExitDiscriminator (MED) for cold-potato routing and the IGP hop count for hot-potato routing.

Design 3 analyzes the performance of protocol models under varying degrees of network stability and link weight management. Network stability is determined by the frequency and duration of link outages in the network.

For each experiment conducted, an efficient RRS search was performed for the given response value, and each RRS search generated 200 simulation samples. We then performed a multiple linear regression on the results of the RRS search. Please note that only $AdjustedR^2$ results are shown because experiments may have different input parameters. The $AdjustedR^2$ value indicates the degree to which the input parameters are related to the response. In each experiment the P value was always < 0.0001, indicating in each case that the regression model predicted the response in a statistically significant manner. In other words, in each experiment the predictions of the model are better than chance alone. In addition, the Degrees of Freedom are not reported per experiment. In each experiment the degrees of freedom was high,

> 100. Finally, multi-collinearity was not observed to be a problem in any of the experiments (i.e., all R^2 with other X values were < 0.75.

3.9.1 Input Parameter Classes

Input Parameter Classes	Min, Max, Step	Defaults
OSPFv2 Timer Class:		
OSPF Hello Interval	[1,4,1] secs	2
OSPF Inactivity Interval	[2,5,1] multiplier	4
OSPF Flood Interval	[1,4,1] secs	1
BGP4 Timer Class:		
BGP KeepAlive Interval	[25,35,2] secs	30
BGP Hold Interval	[36,56,4] secs	45
BGP Min Update Interval	[20,40,4] secs	30
BGP Policy Routing Class:		
MED	ON/OFF	ON
Hot Potato	ON/OFF	ON
BGP Decision Algorithm Class:		
Local-Pref	{low, med, high}	low
MED	{low, med, high}	low
AS-PATH Padding	{low, med, high}	low

Table 3.11: Detail of parameter space for the large-scale OSPF and BGP experiment designs.

The system under test can be characterized as different classes of input parameters. The four classes shown in Table 3.11 represent timers for OSPF and BGP, the BGP route selection policies and the BGP decision algorithm. Each class is defined at the AS level and the values generated are determined by the efficient search algorithm, RRS.

The BGP and OSPF timer classes represent router timers and the values they may have during each simulation run. The specified ranges and steps for each timer value determines the search sample space. The defaults shown are the values used per AS when a given class is not searchable within a given design.

The BGP Policy Routing Class allows hot- and cold-potato routing to be enabled/disabled within an AS. The ROSS.Net framework allows any of the stages

in the BGP decision algorithm to be disabled, however these are the two of interest in this paper.

The BGP Decision Algorithm Class provides specific values for the AS routes. For example, if cold-potato routing is enabled within an AS, then the MED value is defined for routes created by that AS. In this paper we investigate cold-potato routing so must define MED values for those ASes where cold-potato routing in enabled. Values are *low, medium and high* and correspond to varying levels of aggressiveness within each AS. Recall that during the BGP decision algorithm, stage 1, we install the route with the higher Local-Pref value, so each AS must define this attribute for each route created. When these stages are enabled, but not searched by the experiment design, the default values are used.

When all of the input parameter classes are searched the sample space is greater than 14 million. Heuristic search algorithms such as RRS allow us to search this sample space efficiently, i.e., using a proportionally small number of experiments, while still achieving highly correlated results (high $AdjustedR^2$ values).

3.9.2 Experiment Design 1: Management Perspective

Our first investigation focuses on the role of network management perspectives in the response plane. We identify two disparate approaches to network management: local and global. The local approach involves performance tuning an AS domain without knowledge or concern for the impact on neighboring ASes, or even other protocols within the AS. The global approach attempts to optimize all of the ASes simultaneously and is semantic to optimal performance with respect to the inter-network as a whole. Here information about each neighboring AS is openly available and the optimization goal is across all ASes. BB, OO, OB, BO and BO+OB are considered local policies and the global policy is the addition of all update messages (BB+OO+OB+BO).

This design focuses on multiple response surfaces, as shown in Table 3.12 and optimizes across all input parameter classes. Each Experiment conducted generates a unique response plane corresponding to a network management perspective. For example, Experiment 1 generates a response plane where OSPF caused OSPF

Design 1: Management Perspectives									
	Response Surfaces								
Experiment	BB	OO	OB	BO	**Optimal Response**	**BO+OB**	*Adj* R^2	**Effects: optimal values**	
0	+	-	-	-	1,938	77,024	0.30	Inactivity: 3	Keep: 26
1	-	+	-	-	27,424	59,429	0.88	Hello: 3	Flood: 4
2	-	-	+	-	52,700	52,945	0.88	Flood: 1	Keep: 34
3	-	-	-	+	18	75,499	0.18	MRAI: 34	
4	-	-	+	+	52,959	52,959	0.91	Keep:34	Hold: 45
5	+	+	+	+	83,261	52,727	0.52	Flood: 1	Keep: 34
Sample Space Size: 14,348,907								*+ = searched*	

Table 3.12: Design 1: Search varying network management perspectives. The optimal response column relates to the specific management goal searched. The BO+OB column represents the interactions between protocols that occurred.

updates were minimized. The optimal response column indicates that of the 200 simulation runs, the minimum number of OO updates obtained was 27,424. The BO+OB column indicates the number of interactions that occurred between the OSPF and BGP protocols. A value of 59,429 indicates that minimizing OO updates does not greatly increase the number of updates between OSPF and BGP when compared to the other perspectives. The $AdjustedR^2$ value of 88% indicates that the search parameters highly correlated to the response, and the optimal values were 3 seconds for the OSPF HELLO timer, 4 seconds for the OSPF Flood timer, and 56 seconds for the BGP HOLD interval.

Exp	ΣBB	ΣOO	ΣBO	ΣOB	$\Sigma BO + OB$	$\Sigma Global$
0	**1,938**	27,624	20	77,004	77,024	106,586
1	9,565	27,864	245	**52,700**	52,945	90,374
2	2,507	27,672	**18**	75,481	75,499	105,678
3	2,574	**27,424**	20	59,409	59,429	89,427
4	8,619	27,888	211	52,748	**52,959**	89,466
5	2,687	27,847	24	52,703	52,727	**83,261**

Table 3.13: Variation in the optimization of different perspectives. This table illustrates the tradeoffs made for each particular optimization. Bold values are optimization results.

How efficient is each management perspective? Table 3.13 lists the results from each of the experiments. We see that the lowest number of interactions occurred in Experiment 1 where OSPF caused BGP updates were optimized. Because we were optimizing OB updates, and OB updates account for greater than 99% of

BO+OB updates, this result make sense. Experiment 5 generated the least number of updates overall and was 7-27% better than the local perspectives. Not only does optimizing globally lower the number of overall updates, it also lowers the number of interactions between the protocols, within ¡1% of the best case. So it is clear that maintaining privacy between ISPs leads to an increase in the amount of update messages in the network.

Each row of the table represents the optimal value generated by the Experiment. Each column indicates the total number of each type of update message generated for those parameters. Experiments 1, 3 and 4 generated the least number of updates overall locally, and the least number of interactions. Each of these local policies where within 7% of global.

Of the different types of update messages, BB and BO were insignificant in respect to the global number of updates. Conversely, OO and BO were a significant fraction of all update messages, but the OO updates varied little. This leaves OSPF caused BGP (OB) update messages as the significant response to optimize when attempting to minimize both feature interactions and the overall number of update messages in the network.

Which protocol parameters effect the response? If we choose to minimize the number of updates and/or interactions in the network by minimizing OB updates, then Table 3.12 suggests settings for the OSPF Hello interval and Flood interval be set high. In our search, settings of 3 seconds for the Hello interval and 4 seconds for the Flood interval suggest that OSPF convergence times be lengthened in order to minimize overall updates. Generally, slow convergence is not a desirable feature in OSPF networks as it can lead to losses in the data plane. However, slower detection in OSPF may reduce the effects of highly unstable links.

An alternative is to optimize for one of the other local perspectives which prescribe aggressive OSPF convergence settings. In Experiments 2, 4 and 5 the important parameter appears to be the BGP KeepAlive timer. In each case, this timer is set to a high value. Since iBGP connections far outweigh eBGP connections, it makes sense then that by setting the KeepAlive timer to a high value would minimize the effects of highly unstable links in the path between iBGP neighbors.

3.9.3 Experiment Design 2: Cold vs Hot Potato Routing

Design 2: Cold vs Hot Potato Routing				
	BGP Decision Classes			
Experiment	Hot Potato	MED	**Optimal Response**	*Adj* R^2
0	-	-	52,722	0.91
1	+	-	52,494	0.91
2	-	+	52,675	0.91
3	+	+	52,908	0.91
	Sample Space Size: 14,348,907			*+ = searched*

Table 3.14: Design 2: Specifically analyze performance of protocol models under competing goals of Hot and Cold Potato routing.

When two otherwise equal routes are being considered for addition to the BGP RIB, and those routes are both from iBGP peers, the route selected should be from the nearest peer. To determine which peer is the shortest distance away, the IGP hop count path is considered. This is the definition of Hot Potato routing, and was highlighted as a potential cause of many OSPF caused BGP updates in [118]. In that study it was noted that it was not possible to quantify the causes of the updates through measurement data. Also, protocol timer settings in routers throughout the network were not known. Simulation allows us to have a global view of the network, and complete topological information. Searching the sample space allows us to quantify the causes of the updates as well as determine the effects of any potentially influential protocol parameters.

Now that we have a validation that the OSPF domain adversely impacts the BGP domain, we can begin to focus our experiments on the hypothesized cause of the interruptions. In Table 3.14 we investigate the effects of cold versus hot potato routing. In this design we perform a simple full-factorial of RRS optimizations, turning Hot Potato routing on/off, and the MED on/off within the BGP decision algorithm.

If the goal of Hot Potato routing is to transit data through the network by the shortest paths possible, the goal of cold potato routing is the opposite. Cold potato routing is employed when end-to-end quality of service is of importance to an ISP. By carrying data longer in the network, an ISP can exert more control over the data

before handing it off to another ISP. The MED accomplishes this goal by advertising to an AS the *preferred* routes data should take. *Preferred* is a term which is open to interpretation, but in this sense it implies "highest quality" ingress points to a neighboring AS [132]. An ISP implements cold potato routing by setting the MED parameter.

BGP Decision Algo	**Hot Potato**	**MED**	**Neither**	**Both**
Local-Pref	**6383**	**1,714**	**767**	**885**
AS Path	**15,251**	**5,503**	**2,240**	**5,874**
Origin	1	8	50	204
MED	OFF	4	OFF	0
Hot Potato	199	OFF	OFF	1,229
Next Hop	123	369	175	113
Default	476	778	272	635
Total	22,433	8,376	3504	12,444
% Hot Potato	0.8	-	-	9
% MED	-	$\ll 1$	-	0

Table 3.15: This table illustrates the steps used in the BGP decision algorithm for route updates. Each entry illustrates how many times a particular step resulted in a tie-breaking event.

Which steps in the BGP decision algorithm are most important? Table 3.15 quantifies the tie-breaking steps in the BGP decision making algorithm. We expected MED and Hot Potato to play a larger role in the algorithm, based on previous work [118, 120]. In our model it appears that Local-Pref and AS Path Padding play a much larger role in the decision process. In practice, these parameters may not be implemented in some or all ISP networks. Clearly, these parameters do play an important role in dampening the effects of both Hot and Cold Potato routing.

While our statistical models show a high correlation between the input parameters and the response ($AdjustedR^2 = 91\%$), we believe that this design is only an initial step towards systematic questioning of the BGP decision algorithm. For example, when hot-potato routing only is enabled, the number of times the AS Path length was the tie-breaker increased from about 2,000 to over 15,000. Clearly, hot-potato routing is generating longer AS Path lengths in the routes. But it is unclear why there would be a corresponding 10-fold increase in the number of times

Design 3: Network Robustness

	Topology Parameters						
Experiment	Link Stability (LS)	Link Weight (LW)	**Opt Resp**	*Adj* R^2	**Effects: optimal values**		
0	1%	± 10	50,450	0.89	Keep: 34		
1	1%	± 25	54,254	0.91	Keep: 32	Hold: 46	MRAI: 33
2	1%	± 50	52,959	0.91	Keep: 34	Hold: 45	
3	10%	± 10	73,196	0.87	Keep: 34	Hold: 39	Inactivity: 4
4	10%	± 25	75,819	0.88	Keep: 34	Hello: 4	Inactivity: 5
5	10%	± 50	76,346	0.87	Keep: 34	Hold: 55	
6	15%	± 10	99,564	0.78	Keep: 35	Flood: 2	MRAI: 28
7	15%	± 25	100,493	0.78	Keep: 34		
8	15%	± 50	110,009	0.900	Keep: 34	Hello: 4	

Sample Space Size: 14,348,907

Table 3.16: Design 3: Analyze performance of protocol models under varying degrees of network stability and link weight management.

the Local Pref tie-breaker was used. When just cold-potato routing was employed, these tie-breakers only doubled, which indicates that cold-potato routing has the same problem, but to much less a degree. More importantly, Table 3.15 indicates that when both policies are enabled, cold-potato routing can dampen the negative effects of hot-potato routing.

We did not expect this policies to have such a large effect on the other stages in the BGP decision algorithm. More insight into these results may be gained by future designs which takes this into account.

3.9.4 Experiment Design 3: Network Robustness

Table 3.16 illustrates our third design. The purpose of this experiment design is to ascertain the effects of network robustness on our characterization of the system under test. Network robustness is varied in two dimensions: link stability and link weight changes. Link stability was varied randomly over the intervals 1, 10 or 15% and link weights randomly over the intervals 10, 25 or 50 units. The design computes a full factorial over the two parameters of network robustness.

Which parameters were most important in reducing interactions? We report that the liveness timers are the important parameter settings and are related to minimizing OSPF caused BGP updates (OB). KeepAlive is maximized in BGP, and the InactivityInterval is maximized in OSPF. In OSPF, the flood timer, when important is optimized to a value of 2 leading to slower convergence. As the network becomes less stable however, we begin to see that other parameters are having more of an impact on the response. In OSPF we begin to see the Hello frequency

becoming more important, and maximized. This is interesting because delaying detection allows OSPF to aggregate (implicitly) more changes into a single LSA update, which would act to minimize the overall number of updates generated. This implicit aggregation is occurring in the BGP domain as well by setting the KeepAlive interval to 34 seconds and the Hold Interval to 45-55 seconds. By detecting fewer link status changes the models are generating fewer control plane update messages.

LW/LS	Optimal	Avg BO+OB	Avg Global	Defaults
±10,1%	50,450	17%	19%	18%
±10/10%	73,196	17%	8%	
±10/15%	99,564	36%	34%	
±25/1%	54,254	10%	13%	18%
±25/10%	75,819	17%	17%	
±25/15%	100,493	22%	21%	
±50/1%	52,959	19%	14%	20%
±50/10%	76,346	17%	19%	
±50/15%	110,009	18%	17%	

Table 3.17: Improvements over average BO+OB, Global in design 3. The last column illustrates the improvement over using the default protocol parameters.

Table 3.17 shows that we continue to receive consistent improvements in the response over the average regardless of the robustness in the network. We see that the optimal simulation experiments are simply setting the link failure detection parameters in either protocol to their slowest convergence settings. By not detecting link status changes quickly, the number of updates generated can most effectively be minimized. The table compares the amount of improvement over the average cases of BO+OB and the global response, as well as over the default settings. Generally, this approach to minimizing updates yielded a 20% improvement over the average. This figure is primarily related to the intervals chosen for the protocol parameters. In the future we could relate the improvements to the rate of convergence, which would be a more meaningful representation of the trade-off.

From the table we also see that the response is independent of the link weight changes. Each link weight interval varies by ¡5% for each fixed link stability interval.

This is surprising since aggressive link weight policies are known to produce routing loops among other problems. While aggressive changes impact the OSPF domain internally, those updates do not appear to be propagating into the BGP domain via OSPF. We theorize that the link status changes have a much greater impact on the OB response because they have a direct impact on the iBGP connections which dominate the model.

3.10 Summary

We have demonstrated the efficacy of the Recursive Random Search (RRS) technique when applied to large-scale meta-simulation of complex OSPFv2 routing networks. We found that:

1. the number of simulation experiments is reduced by an order of magnitude when compared to full-factorial design approach,
2. this approach enabled the rapid elimination of unnecessary parameters, and
3. RRS enabled the rapid understanding of key parameter interactions.

RRS enabled us to make a critical path analysis and make the interesting observation that when modeling only OSPF control-plane dynamics we were able to shrink the number nodes down to that subset that was only needed for determining convergence times. This reduction resulted in models that execute 100 times faster than their full topology counterparts while at the same time gaining greater detail in the areas of interest. ROSS.Net provided "good" results fast for our selected response plane: OSPF router convergence.

Also, we have used an experiment design approach to characterize OSPF and BGP behavior in combination as well as their interactions. Based on the Rocketfuel data repository, we have developed a realistic large-scale simulation of these two dominant inter- and intra-domain routing protocols. We, then, employed an efficient experiment design framework, ROSS.Net [114, 51], to search for best protocol parameter settings. The protocol parameters we investigated included OSPF timers, BGP timers and BGP decision making attributes. We defined the number

of routing updates as the metric to minimize in our heuristic search for the best parameter settings. We also classified the routing updates into four categories to help design our experiments more flexibly.

We believe that systematic experiment design approaches can be leveraged in several large-scale networking problems. Future work includes investigation of the steps of BGP decision making algorithm.

CHAPTER 4
ROSS: A High-Performance, Low Memory, Modular Time Warp System

We introduce a new Time Warp system called *ROSS: Rensselaer's Optimistic Simulation System*. ROSS is an extremely modular kernel that is capable of achieving event rates as high as 1,250,000 events per second when simulating a wireless telephone network model (PCS) on a quad processor PC server. In a head-to-head comparison, we observe that ROSS out performs the Georgia Tech Time Warp (GTW) system by up to 180% on a quad processor PC server and up to 200% on the SGI Origin 2000 . ROSS only requires a small *constant* amount of memory buffers greater than the amount needed by the sequential simulation for a constant number of processors. ROSS demonstrates for the first time that stable, highly-efficient execution using little memory above what the sequential model would require is possible for low-event granularity simulation models. The driving force behind these high-performance and low memory utilization results is the coupling of an efficient pointer-based implementation framework, Fujimoto's fast GVT algorithm for shared memory multiprocessors, *reverse computation* and the introduction of *Kernel Processes (KPs)*. KPs lower fossil collection overheads by aggregating processed event lists. This aspect allows fossil collection to be done with greater frequency, thus lowering the overall memory necessary to sustain stable, efficient parallel execution. These characteristics make ROSS an ideal system for use in large-scale networking simulation models. The principle conclusion drawn from this study is that the performance of an optimistic simulator is largely determined by its memory usage.

4.1 The Time Warp Protocol

For Time Warp protocols there is no consensus in the PDES community on how best to implement them. One can divide Time Warp implementation frameworks into two categories: *monolithic* and *modular* based on what functionality is directly

contained within the event scheduler. It is believed that the *monolithic* approach to building Time Warp kernels is the preferred implementation methodology if the absolute highest performance is required. The preeminent monolithic Time Warp kernel is *Georgia Tech Time Warp (GTW)* [88, 92]. One only needs to look at GTW's 1000 line "C" code `Scheduler` function to see that all functionality is directly embedded into the scheduling loop. This loop includes global virtual time (GVT) calculations, rollback, event cancellation, and fossil collection. No subroutines are used to perform these operations. The central theme of this implementation is *performance at any cost*.

This implementation approach, however, introduces a number of problems for developers. First, this approach complicates the adding of new features since doing so may entail code insertions at many points throughout the scheduler loop. Second, the all-inclusive scheduler loop lengthens the "debugging" process since one has to consider the entire scheduler as being a potential source of system errors.

At the other end of the spectrum, there are *modular* implementations which break down the functionality of the scheduler into small pieces using an object-oriented design approach. SPEEDES is the most widely used Time Warp system implemented in this framework [111, 112, 113]. Implemented in C++, SPEEDES exports a *plug-and-play* interface which allows developers to easily experiment with new time management, data distribution and priority queue algorithms.

All of this functionality and flexibility comes at a performance price. In a recent study conducted on the efficiency of Java, C++ and C, it was determined that "C programs are substantially faster than the C++ programs" (page 111) [108]. Moreover, a simulation of the National Airspace System (NAS), as described in [115], was originally implemented using SPEEDES, but a second implementation was realized using GTW. Today, only the GTW implementation is in operation. The reason for this shift is largely attributed to GTW's performance advantage on shared-memory multiprocessors. Thus, it would appear that if you want maximum performance, you cannot use the modular approach in your implementation.

Another source of concern with Time Warp systems is memory utilization. The basic unit of memory can be generalized to a single object called a *buffer* [87].

A buffer contains all the necessary event and state data for a particular LP at a particular instance in virtual time. Because the optimistic mechanism mandates support of the"undo" operation, these buffers cannot be immediately reclaimed. There have been several techniques developed to reduce the number of buffers as well as to reduce the size of buffers required to execute a Time Warp simulation. These techniques include infrequent state-saving [81], incremental state-saving [94, 113], and most recently reverse computation [84].

Rollback-based protocols have demonstrated that Time Warp systems can execute in no more memory than the corresponding sequential simulation, such as Artificial Rollback [99] and Cancelback [96], however performance suffers. Adaptive techniques [87], which adjust the amount of memory dynamically, have been shown to improve performance under "rollback thrashing" conditions and reduce memory consumption to within a constant factor of sequential. However, for small event granularity models (i.e., models that require only a few microseconds to process an event), these adaptive techniques are viewed as being too heavy weight.

In light of these findings, Time Warp programs typically allocate much more memory than is required by the sequential simulation. In a recent performance study in retrofitting a large sequential Ada simulator for parallel execution, SPEEDES consumed 58 MB of memory where the corresponding sequential only consumed 8 MB. It is not known if this extra 50 MB is a fixed constant or a growth factor [110].

We introduce a new Time Warp system called *ROSS: Rensselaer's Optimistic Simulation System*. ROSS is a modular, C-based Time Warp system that is capable of extreme performance. On a quad processor PC server ROSS is capable of processing over 1,250,000 events per second for a wireless communications model. Additionally, ROSS only requires a small *constant* amount of memory buffers greater than the amount needed by the sequential simulation for a constant number of processors. The key innovation driving these high-performance and low memory utilization results is the integration of the following technologies:

- pointer-based, modular implementation framework,
- Fujimoto's GVT algorithm [90],

- reverse computation, and
- the use of *Kernel Processes(KPs)*.

KPs lower fossil collection overheads by aggregating processed event lists. This aspect allows fossil collection to be done with greater frequency, thus lowering the overall memory necessary to sustain stable, efficient parallel execution.

As a demonstration of ROSS' high-performance and low memory utilization, we put ROSS to the test in a head-to-head comparison against one of the fastest Time Warp systems to date, GTW.

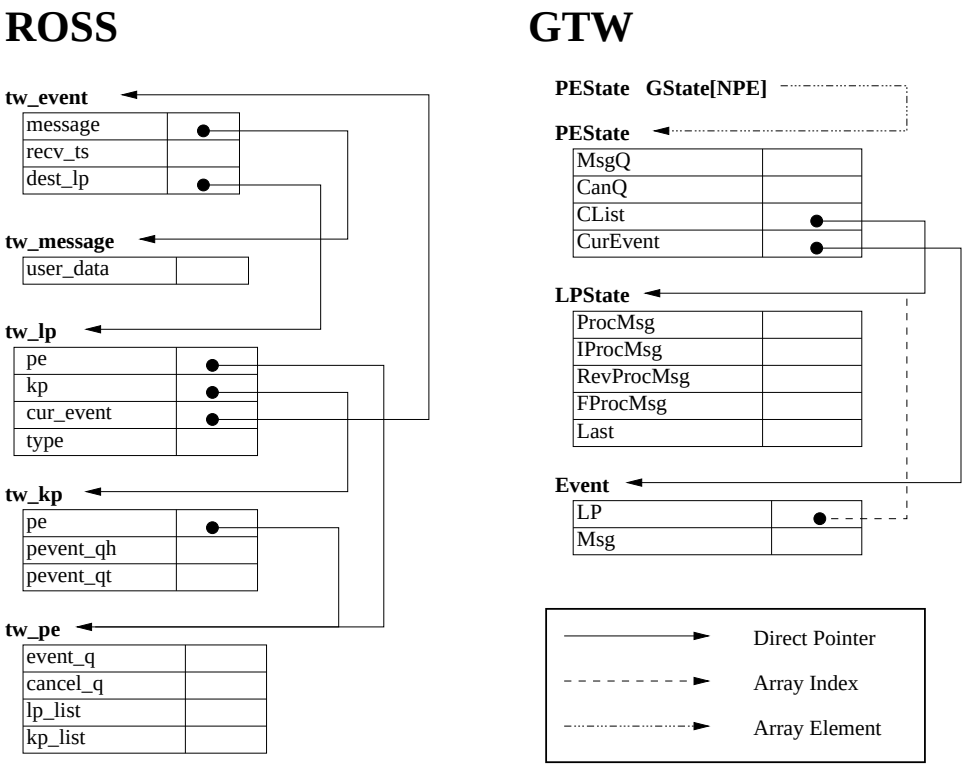


Figure 4.1: Data Structure Comparison: ROSS vs. GTW.

4.2 Data Structure and System Parameter Comparison

4.2.1 Algorithm and Implementation Framework

GTW is designed to exploit the availability of shared-memory in a multi-processor systems. With that view in mind, a global structure called `GState` is the

backbone of the system as shown in Figure 4.1. This array represents all the data used by a particular instantiation of a `Scheduler` thread, which is executed on a distinct processor.

Inside each `GState` element is a statically defined array of LP pointers, locks for synchronizing the transfer of events between processors, pointers to manage the "free-list" of buffers, and timers for performance monitoring. To obtain the pointer for LP i, the follow access is required:

$$LP\_Ptr \;=\; GState[TWLP[i].Map].CList[LPNum[i]];$$

where, i is the LP number, $TWLP[i].Map$ is the processor on which the LP resides and $LPNum[]$ array specifies to which slot within a processor's `CList` array the LP's pointer was located (see Figure 4.1).

Now, using these data structures, GTW implements an optimistic time management algorithm that throttles execution based on the availability of memory. On each processor, a separate pool of memory is created for each remote processor. When the application requests a free memory buffer, the owning processor will use the LP destination information provided in the `TWGetMsg` routine to determine which processor's pool to allocate from. If that pool is empty, the *abort* buffer is returned and no event is scheduled. When the current event has completed processing, the `Scheduler` will rollback (i.e., abort) that event and attempt to reclaim memory by computing GVT. This arrangement is called *partitioned buffer pools* [91]. The key properties of this approach are that over-optimism is avoided since a processor's forward execution is throttled by the amount of buffers in its free-list and the false sharing of memory pages is lessened since a memory buffer is only shared between a pair of processors.

To implement GVT, GTW uses an extremely fast asynchronous GVT algorithm that fully exploits shared memory [90]. To mitigate fossil collection overheads, an "on-the-fly" approach was devised [90]. Here, events, after being processed, are immediately threaded into the tail of the appropriate free-list along with being placed into the list of processed events for the LP. To allocate an event, the `TWGetMsg` function must test the *head* of the appropriate free-list and make sure

that the time stamp of the event is less than GVT. If not, the abort buffer is returned and the event that is currently being processed will be aborted. As we will show in Section 4.3, "on-the-fly" fossil collection plays a crucial roll in determining GTW's performance.

ROSS' data structures, on the other hand, are organized in a bottom-up hierarchy, as shown on the left panel of Figure 4.1. Here, the core data structure is the `tw_event`. Inside every `tw_event` is a pointer to its source and destination LP structure, `tw_lp`. Observe, that a pointer and not an index is used. Thus, during the processing of an event, to access its source LP and destination LP data only the following accesses are required:

$$\begin{aligned} my\_source\_lp &= event->src\_lp; \\ my\_destination\_lp &= event->dest\_lp; \end{aligned}$$

Additionally, inside every `tw_lp` is a pointer to the owning processor structure, `tw_pe`. So, to access processor specific data from an event the following operation is performed:

$$my\_owning\_processor = event->dest\_lp->pe;$$

This bottom-up approach reduces access overheads and may improve locality and processor cache performance. Note that prior to adding Kernel Processes (KPs), the `tw_kp` structure elements were contained within the `tw_lp`. The role of KPs will be discussed in Section 4.3.4.

Like GTW, ROSS' `tw_scheduler` function is responsible for event processing (including reverse computation support), virtual time coordination and memory management. However, that functionality is decomposed along data structure lines. This decomposition allows the `tw_scheduler` function to be compacted into only 200 lines of code. Like the scheduler function, our GVT computation is a modular implementation of Fujimoto's GVT algorithm [90].

ROSS also uses a memory-based approach to throttle execution and safeguard against over-optimism. Each processor allocates a *single* free-list of memory buffers.

When a processor's free-list is empty, the currently processed event is aborted and a GVT calculation is immediately initiated. Unlike GTW, ROSS fossil collects buffers from each LP's processed event-list after each GVT computation and places those buffers back in the owning processor's free-list. We demonstrate in Section 4.3 that this approach results in significant fossil collection overheads, however these overheads are then mitigated through the insertion of Kernel Processes into ROSS' core implementation framework.

4.2.2 Performance Tuning Parameters

GTW supports two classes of parameters: one set to control how memory is allocated and partitioned. The other set determines how frequently GVT is computed. The total amount of memory to be allocated per processor is specified in a configuration file. How that memory is partitioned for a processor is determined by the `TWMemMap[i][j]` array and is specified by the application model during initialization. `TWMemMap[i][j]` specifies a *ratioed* amount of memory that processor j's free-list on processor i will be allocated. To clarify, suppose we have two processors and processor 0's `TWMemMap` array has the values 50 and 25 in slots 0 and 1 respectively. This means that of the total memory allocated, 50 buffers out of every 75 will be assigned to processor 0's free-list on processor 0 and only 25 buffers out of every 75 buffers allocated will be assigned to processor 1's free-list on processor 0.

To control the frequency with which GVT is calculated, GTW uses *batch* and $GVT_{interval}$ parameters. The *batch* parameter is the number of events GTW will process before returning to the top of the main event scheduling loop and checking for the arrival of remote events and anti-messages. The $GVT_{interval}$ parameters specifies the number of iterations through the main event scheduling loop prior to initiating a GVT computation. Thus, on average, $batch \times GVT_{interval}$ is the number of events that will be processed between successive GVT computations.

ROSS, like GTW, shares a *batch* and $GVT_{internal}$ parameter. Thus, on average, $batch * GVT_{interval}$ events will processed between GVT epochs. However, because ROSS uses the fast GVT algorithm with a conventional approach to fossil collection, we experimentally determined that ROSS can execute a simulation model efficiently

in:

$$C \times NumPE \times batch \times GVT_{interval}$$

more memory buffers than is required by a sequential simulation. Here, $NumPE$ is the number of processors used and C is a constant value. Thus, the additional amount of memory required for efficient parallel execution only grows as the number of processors is increased. The amount per processor is a small constant number.

The intuition behind this experimental phenomenon is based on the previous observation that memory can be divided into two categories: *sequential* and *optimistic* [87]. Sequential memory is the base amount of memory required to sustain sequential execution. Every parallel simulator must allocate this memory. Optimistic memory is the extra memory used to sustain optimistic execution. Now, assuming each processor consumes $batch \times GVT_{interval}$ memory buffers between successive GVT calculations, on average that is the same amount of memory buffers that can be fossil collected at the end of each GVT epoch. The multiplier factor, C, allows each processor to have some reserve memory to schedule new events into the future and continue event processing during the asynchronous GVT computation. The net effect is that the amount of *optimistic* memory allocated correlates to how efficient GVT and fossil collection can be accomplished. The faster these two computations execute, the more frequently they can be run, thus reducing the amount of optimistic memory required for efficient execution. Experimentally, values ranging from $C = 2$ to $C = 8$ appear to yield the best performance for the PCS model depending on the processor configuration.

GTW is unable to operate efficiently under the above memory constraints because of "on-the-fly" fossil collection. This aspect will be discussed in more throughly in Section 4.3.3.

4.3 Performance Study

4.3.1 Benchmark Applications

There are two benchmark applications used in this performance study. The first is a personal communications services (PCS) network model as described in [85]. Here, the service area of the network is populated with a set of geographically distributed transmitters and receivers called *radio ports*. A set of radio channels are assigned to each radio port, and the user in the *coverage area* sends and receives phone calls using the radio channels. When a user moves from one cell to another during a phone call a *hand-off* is said to occur. In this case the PCS network attempts to allocate a radio channel in the new cell to allow the phone call connection to continue. If all channels in the new cell are busy, then the phone call is forced to terminate. For all experiments here, the *portable-initiated* PCS model was used, which discounts *busy-lines* in the overall call blocking statistics. Here, *cells* are modeled as LPs and PCS subscribers are modeled as messages that travel among LPs. PCS subscribers can travel in one of 4 directions: north, south, east or west. The selection of direction is based on a uniform distribution. For both, GTW and ROSS, the state size for this application is 80 bytes with a message size of 40 bytes and the minimum lookahead for this model is *zero* due to the exponential distribution being used to compute call inter-arrivals, call completion and mobility. The event granularity for PCS is very small (i.e., less than 4 microseconds per event). PCS is viewed as being a representative example of how a "real-world" simulation model would exercise the rollback dynamics of a optimistic simulator system.

The second application is a derivative of the rPHOLD [92, 93] synthetic workload model called *rPHOLD*. Here, the standard rPHOLD benchmark is modified to support "reverse-computation". We configure the benchmark to have minimal state, message size and "null" event computation. The forward computation of each event only entails the generation of two random numbers; one for the time stamp and the other for the destination LP. The time stamp distribution is exponential with a mean of 1.0 and the LP distribution is uniform, meaning that all LPs are equally like to be the "destination" LP. Because the random number generator (RNG) is perfectly reversible, the reverse computation "undoes" an LP's RNG seed state by

computing the perfect inverse function as described in [84]. The message population per LP is 16. Our goal was to create a pathological benchmark which has a minimal event granularity, yet produces a large numbers of remote messages (75% in the 4 processor case), which can result in a large number of "thrashing" rollbacks. To date, we are unaware of any Time Warp system which is able to obtain a positive speedup (i.e., greater than 1) for this particular configuration of rPHOLD.

4.3.2 Computing Testbed and Experiment Setup

Our computing testbed consists of two different computing platforms. The first is a single quad processor Dell personal computer. Each processor is a 500 MHz Pentium III with 512 KB of level-2 cache. The total amount of available RAM is 1 GB. Four processors are used in every experiment. All memory is accessed via the PCI bus, which runs at 100 Mhz. The caches are keep consistent using a snoopy, bus-based protocol.

The memory subsystem for the PC server is implemented using the Intel NX450 PCI chipset [95]. This chipset has the potential to deliver up to 800 MB of data per second. However, early experimentation determined the maximum obtainable bandwidth is limited to 300 MB per second. This performance degradation is attributed to the memory configuration itself. The 1 GB of RAM consists of 4, 256 MB DIMMs. With 4 DIMMs, only one bank of memory is available. Thus, "address-bit-permuting" (ABP), and bank interleaving techniques are not available. The net result is that a single 500 MHz Pentium III processor can saturate the memory bus. This aspect will play an important roll in our performance results.

The second computing platform is an SGI Origin 2000 [97] with 12, 195 Mhz R10000 processors. This architecture, unlike the PC server, is distributed memory and has non-uniform memory access times, yet is still cache-coherent via a directory-based protocol. To compensate for large local and remote memory access delays, each processor has a 4 MB level-2 cache.

For first series of PCS experiments, each PCS cell is configured with 16 initial subscribers or *portables*, making the total event population for the simulation 16 times the number of LPs in the system. The number of cells in the system was

varied from 256 (16x16 case) to 65536 (256x256 case) by a factor of 4.

Here, $GVT_{interval}$ and *batch* parameters were set at 16 each. Thus, up to 256 events will be processed between GVT epochs for both systems. These settings where determined to yield the highest level of performance for both systems on this particular computing testbed. For ROSS, the C memory parameter was set to 2. In the best case, GTW was given approximately 1.5 times the amount of memory buffers required by the sequential simulations for large LP configurations and 2 to 3 times for small LP configuration. This amount of memory was determined experimentally to result in the shortest execution time (i.e., best performance) for GTW. Larger amounts of memory resulted in longer execution times. This performance degradation is attributed to the memory subsystem being a bottleneck. Smaller amounts of memory resulted longer execution times due to an increase in the number of aborted events. (Recall, that when the current event being processed is unable to schedule a new event into the future due to a shortage of free memory buffers, that event is aborted (i.e., rolled backed) and re-executed only when memory is available).

GTW and ROSS use precisely the same priority queue algorithm (Calendar Queue) [82], random number generator [98] and associated seeds for each LP. The benchmark application's implementation is identical across the two Time Warp systems. Consequently, the only performance advantage that one system has over the other can only be attributed to algorithmic and implementation differences in the management of virtual time and memory buffers.

4.3.3 Initial PCS Performance Data

The data for our initial performance comparison between GTW and ROSS using the quad processor PC server is presented in Figure 4.2. Here, the event rate as a function of the number of LPs is shown for ROSS, GTW and GTW-OPT. "GTW" represents the Georgia Tech Time Warp system without proper settings of the `TWMemMap` array (i.e., $TWMemMap[i][j] = 1 \forall i, j$). "GTW-OPT" uses the experimentally determined optimal settings for `TWMemMap`.

For GTW-OPT, this setting was determined to be 50 when i and j are equal

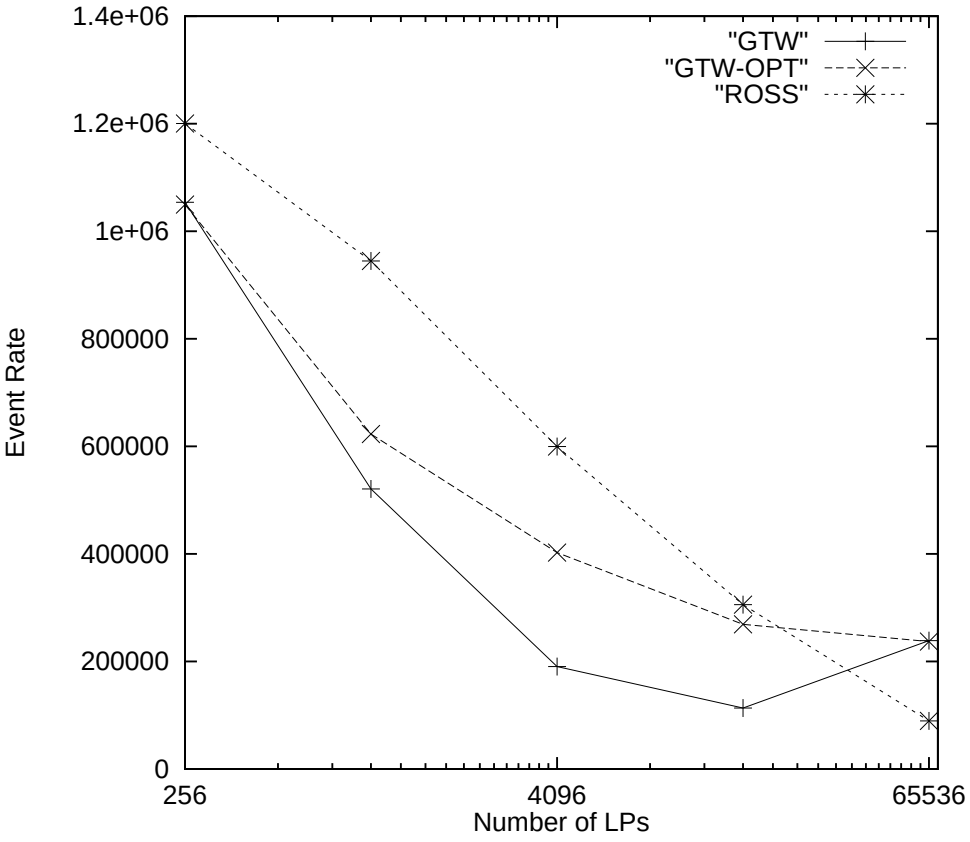


Figure 4.2: Quad PC Server Performance Comparison: GTW vs. ROSS. The "GTW" line indicates GTW's performance without optimized memory pool partitioning. "GTW-OPT" indicates GTW's performance with optimized memory pool partitioning.

and 5 for all other cases. This allocation strategy is very much inline with what one would expect for this *self-initiated* simulation model [107]. This ratio for memory allocation was used for all cases.

We observe that in the comparison, GTW-OPT out performs GTW in all cases. In the 64x64 case, we see a 50% performance gap between GTW-OPT (400,000 events per second) and GTW (200,000 events per second). These results underscore the need to find the proper parameter settings for any Time Warp system. In the case of GTW, the local processor's free-list (i.e., `TWMemMap[i][i]` was not given enough memory to schedule events for itself and a number of aborted events resulted. This lack of memory caused a severe performance degradation.

	Mem Usage in Buffers	Amt Relative to Seq
GTW-OPT 16x16 case	11776	287%
ROSS 16x16 case	6144	150% (seq + 2048)
GTW-OPT 32x32 case	31360	190%
ROSS 32x32 case	18432	113% (seq + 2048)
GTW-OPT 64x64 case	93824	143%
ROSS 64x64 case	67584	103% (seq + 2048)
GTW-OPT 128x128 case	375040	143%
ROSS 128x128 case	264192	100.8% (seq + 2048)
GTW-OPT 256x256 case	1500032	143%
ROSS 256x256 case	1050624	100.2% (seq + 2048)

Table 4.1: Event Buffer Usage: GTW-OPT vs. ROSS. The buffer size for both GTW and ROSS is 132 bytes.

Now, when GTW-OPT is compared to ROSS, it is observed that ROSS out performs GTW-OPT in every case except one: the 64K LP case. For ROSS, the biggest win occurs in the 4K LP case. Here, a 50% performance gap is observed (600,000 events per second for ROSS and 400,000 for GTW-OPT). However, in the 16K LP case, the gap closes and in the 64K LP cases GTW-OPT is outperforming ROSS by almost a factor of 4. Two major factors are attributed to this performance behavior.

For both GTW-OPT and ROSS, the under powered memory subsystem is a critical source of performance degradation as the number of LPs increase. The reason for this is because as we increase the number of LPs, the total number of pending events increase by a factor of 16. This increase in memory utilization forms a bottleneck as the memory subsystem is unable to keep pace with processor demand. The 4K LP case appears to be a break point in memory usage. ROSS, as shown in Table 4.1 uses significantly less memory than GTW. Consequently, ROSS is able to fit more of the free-list of events in level-2 cache.

In terms of overall memory consumption, GTW-OPT is configured with 1.5 to 3 times the memory buffers needed for sequential execution depending on the size of the LP configuration. As previously indicated, that amount of memory was experimentally determined to be optimal for GTW. ROSS, on the other hand, only allocates an extra 2048 event buffers (512 buffers per processor) over what is

required by the sequential simulation, regardless of the number of LPs. In fact, we have run ROSS with as little as 1024 extra buffers ($C = 1.0$, 256 buffers per processor) in the 256 LP case. In this configuration, ROSS generates an event rate of over 1,200,000. These performance results are attributed to the coupling of Fujimoto's GVT algorithm for shared memory multiprocessors with memory efficient data structures, reverse computation and a conventional fossil collection algorithm, as discussed in Section 4.2.

However, this conventional approach to fossil collection falls short when the number of LPs becomes large, as demonstrated by 64K LP case. Here, GTW-OPT is 4 times faster than ROSS. The culprit for this sharp decline in performance is attributed to the overwhelming overhead associated with searching through 64,000 processed event-lists for potential free-event buffers every 256 times though the main scheduler loop. It is at this point where the low-overhead of GTW's "on-the-fly" approach to fossil collection is of benefit.

To summarize, ROSS executes efficiently so long as the number of LPs per processor is kept to a minimum. This aspect is due to the ever increasing fossil collection overheads as the number of LPs grow. To mitigate this problem, "on-the-fly" fossil collection was considered as a potential approach. However, it was discovered to have a problem that results in a increase in the amount of memory required to efficiently execute parallel simulations.

The problem is that a processors ability to allocate memory using the "on-the-fly" approach is correlated to its rollback behavior. Consider the following example: suppose we have LP A and LP B that have been mapped to processor i. Assume both LPs have processed events at $TS = 5, 10$ and 15. With GTW, processor i's free-list of event buffers for itself (i.e., `GState[i].PFree[i]`) would be as follows (with the head of the list being on the left):

$$5.0_A, 5.0_B, 10.0_A, 10.0_B, 15.0_A, 15.0_B$$

Note how the free-list is ordered with respect to virtual time. Suppose now LP B is

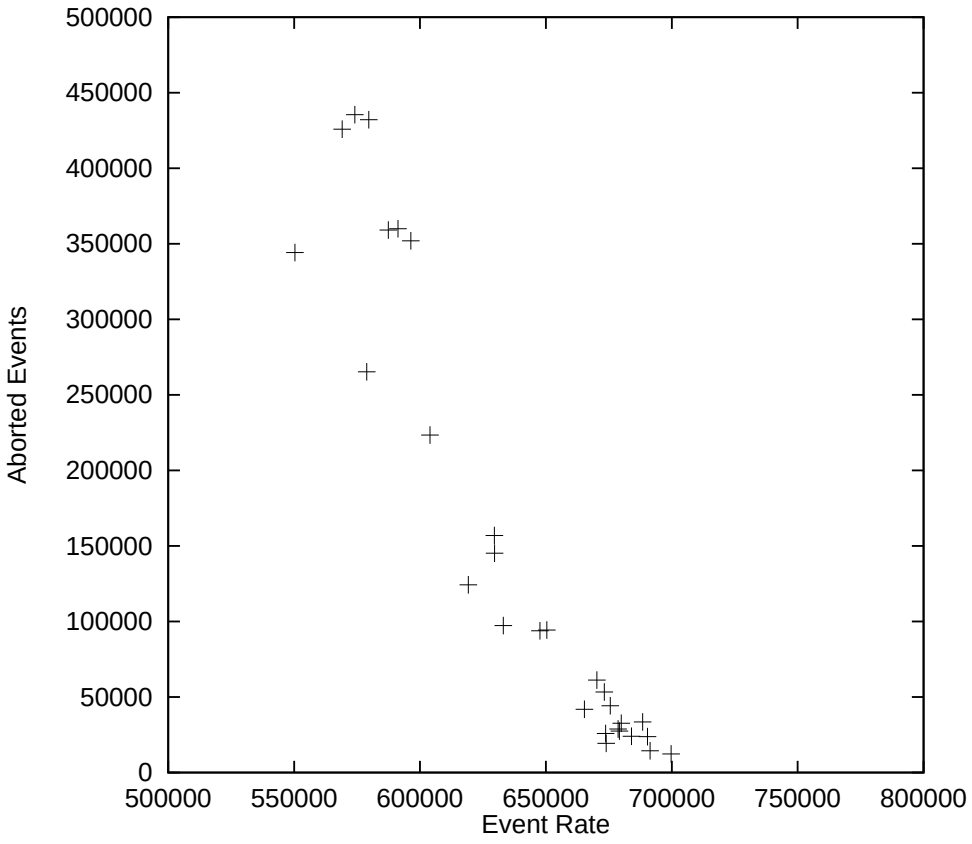


Figure 4.3: The impact of *aborted* events on GTW event rate for the 1024 (32x32 cells) LP case.

rolled back and re-executes those events. The free-list will now appear as follows:

$$5.0_A, 10.0_A, 15.0_A, 5.0_B, 10.0_B, 15.0_B$$

Observe that because LP B has rolled back and re-executed forward, the free-list is now unordered with respect to virtual time. Recall that after processing an event it is re-threaded into the tail of the free-list. This unordered free-list causes GTW to behave as if there are no free buffers available, which results in events being falsely aborted. This phenomenon is caused by the event at the head of the free-list not being less than GVT, yet deeper in the free-list are events with a timestamp less than GVT.

On-the-fly fossil collection under tight memory constraints can lead to large variations in GTW performance, as shown in Figure 4.3. Here, the event rate as

it correlates to the number of aborted events for the 1024 LP case is shown. We observe the event rate may vary by as much as 27%. This behavior is attributed to the rollback behavior increasing the "on-the-fly" fossil collection overheads as the free-list becomes increasingly out-of-order, which leads to instability in the system. To avoid this large variance in performance, GTW must be provided much more memory than is required for sequential execution. This allows the free-list to be sufficiently long such that the impact of it being out-of-order does not result in aborted events and allows stable, predictable performance.

A solution is to search deeper into the free-list. However, this is similar to aborting events in that it introduces a load imbalance among processors who are rolling back more than others (i.e., the more out-of-order a list becomes, the longer the search for free-buffers). In short, *the fossil collection overheads should not be directly tied to rollback behavior*. This observation lead us to the creation of what we call *Kernel Processes (KPs)*.

4.3.4 Kernel Processes

A Kernel Process is a shared data structure among a collection of LPs that manages the processed event-list for those LPs as a single, continuous list. The net effect of this approach is that the `tw_scheduler` function executes forward on an LP by LP basis, but rollbacks and more importantly fossil collects on a KP by KP basis. Because KPs are much fewer in number than LPs, fossil collection overheads are dramatically reduced.

The consequence of this design modification is that all rollback and fossil collection functionality shifted from LPs to KPs. To effect this change, a new data structure was created, called `tw_kp` (see Figure 4.1). This data structure contains the following items: *(i) identification field, (ii) pointer to the owning processor structure,* `tw_pe`*, (iii) head and tail pointers to the shared processed event-list and (iv) KP specific rollback and event processing statistics.*

When an event is processed, it is threaded into the processed event-list for a shared KP. Because the LPs for any one KP are all mapped to the same processor, mutual exclusion to a KP's data can be guaranteed without locks or semaphores.

In addition to decreasing fossil collection overheads, this approach reduces memory utilization by sharing the above data items across a group of LPs. For a large configuration of LPs (i.e., millions), this reduction in memory can be quite significant. For the experiments done in this study, a typical KP will service between 16 to 256 LPs, depending on the number of LPs in the system. Mapping of LPs to KPs is accomplished by creating sub-partitions within a collection of LPs that would be mapped to a particular processor.

While this approach appears to have a number of advantages over either "on-the-fly" fossil collection or standard LP-based fossil collection, a potential drawback with this approach is that "false rollbacks" would degrade performance. A "false rollback" occurs when an LP or group of LPs is "falsely" rolled back because another LP that shares the same KP is being rolled back. As we will show for this PCS model, this phenomenon was not observed. In fact, a wide range of KP to LP mappings for this application were found to result in the best performance for a particular LP configuration.

4.3.5 Revised PCS Performance Data

Like the previous set of experiments, ROSS utilizes the same settings. In particular, for all results presented here, ROSS again only uses 2048 buffers above what would be required by the sequential simulator.

In Figure 4.4, we show the impact of the number of kernel processes allocated for the entire system on event rate. This series of experiments varies the total number of KPs from 4 to 256 by a factor of 2. In the 4 KP case, there is one "super KP" per processor, as our testbed platform is a quad processor machine. We observe that only the 256 (16x16) and the 1024 (32x32) LP cases are negatively impacted for a small number of KPs. All other cases exhibit very little variation in event rate as the number of KPs is varied. These flat results are not what we expected.

If we look at the aggregate number of rolled back events, as shown in Figure 4.5, for the different LP configurations, we observe a dramatic decline in the number of rolled back events as the number of KPs is increased from 4 to 64. So, then why is performance flat? The answer lies in the fact that we are trading rollback

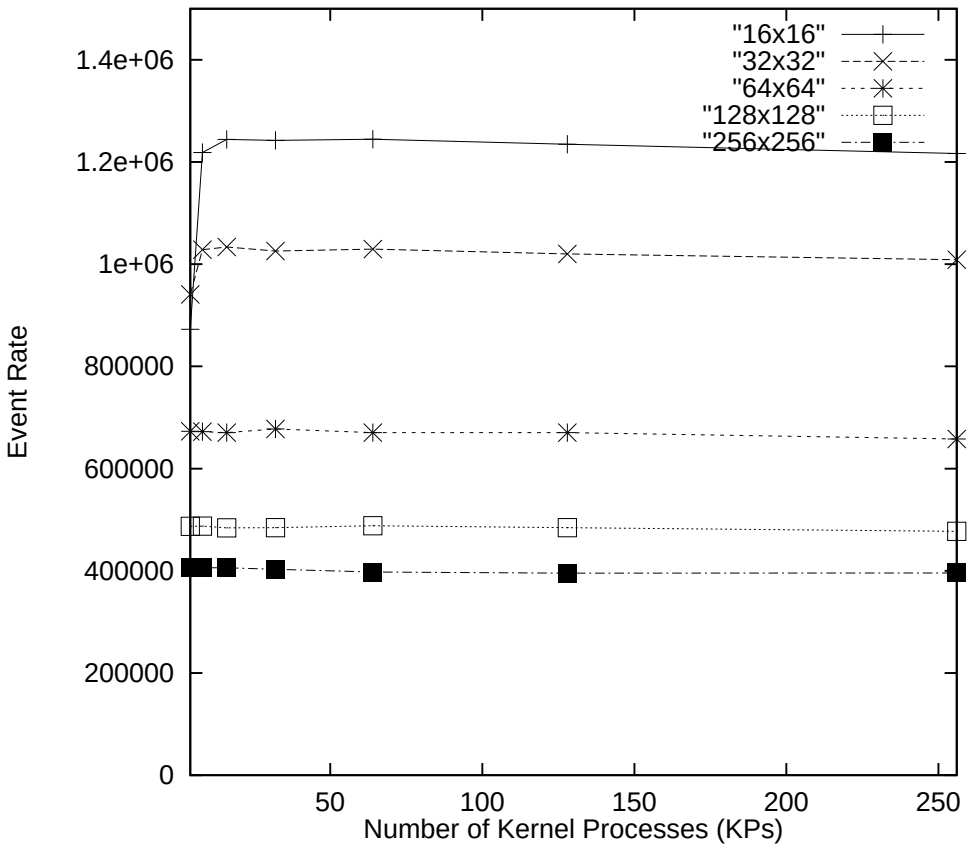


Figure 4.4: Impact of the number of kernel processes on ROSS' event rate.

overheads for fossil collection overheads. Clearly as we increase the number of KPs, we increase fossil collection overheads since each processor has more lists to sort through. Likewise, we are also reducing the number of "false rollbacks". This trade-off appears to be fairly equal for KP values between 16 and 256 across all LP configurations. Thus, we do not observe that finding the *absolute best* KP setting being critical to achieving maximum performance as was finding the best `TWMemMap` setting for GTW. We believe this aspect will allow end users to more quickly realize top system performance under ROSS.

Looking deeper into the rollback behavior of KPs, we find that most of the rollbacks are primary, as shown in Figures 4.6 and 4.7. Moreover, we find that as we add KPs, the average rollback distance appears to shrink. We attribute this behavior to a reduction in the number of "falsely" rolled back events as we increase

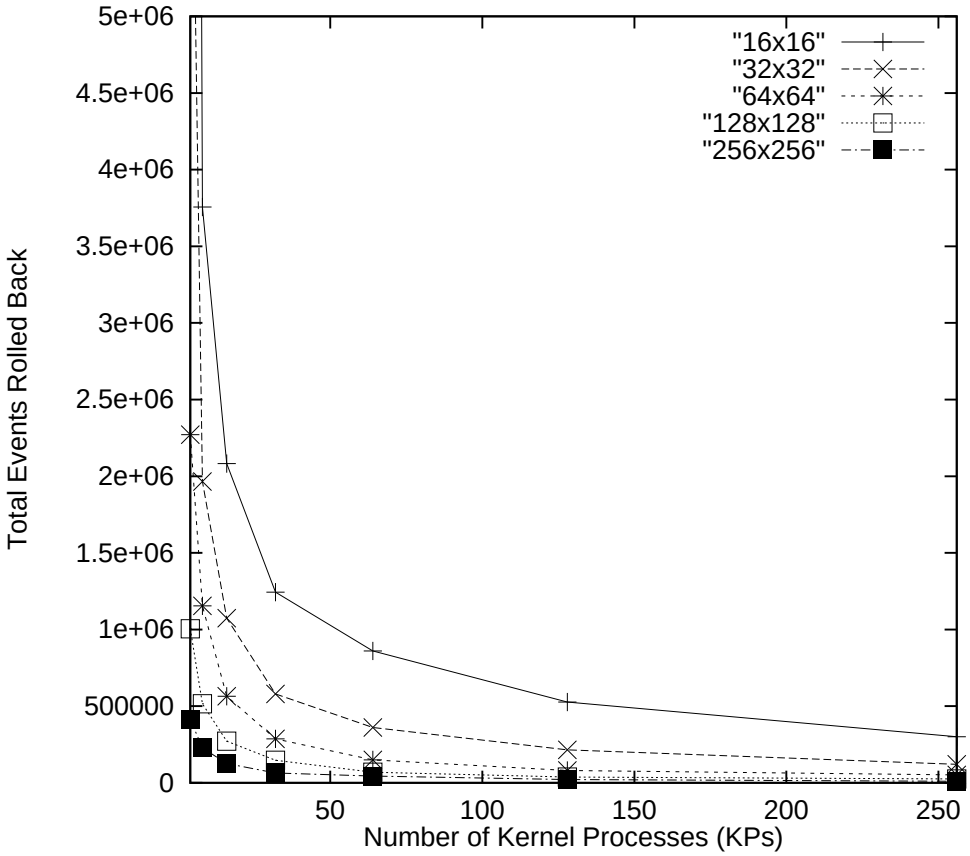


Figure 4.5: Impact of the number of kernel processes on events rolled back.

the number KPs.

As a side note, we observe that as the number of LPs increase from 256 (16x16 case) to 64K (256x256 case), the event rate degrades by a factor of 3 (1.25 million to 400,000), as shown in Figure 4.4. This performance degradation is due to the sharp increase in memory requirements to execute the large LP configurations. As shown in Table 4.1, the 64K LP case consumes over 1 million event buffers, where the 256 LP case only requires 6,000 event buffers. This increase in memory requirements results in higher cache miss rates, placing a higher demand on the under-powered memory subsystem, and ultimately degrades simulator performance.

The performance of ROSS-OPT (best KP configuration) is now compared to that of GTW-OPT and ROSS without KPs in Figure 4.8. We observe that ROSS-OPT outperforms GTW-OPT and original ROSS across all LP configurations, thus

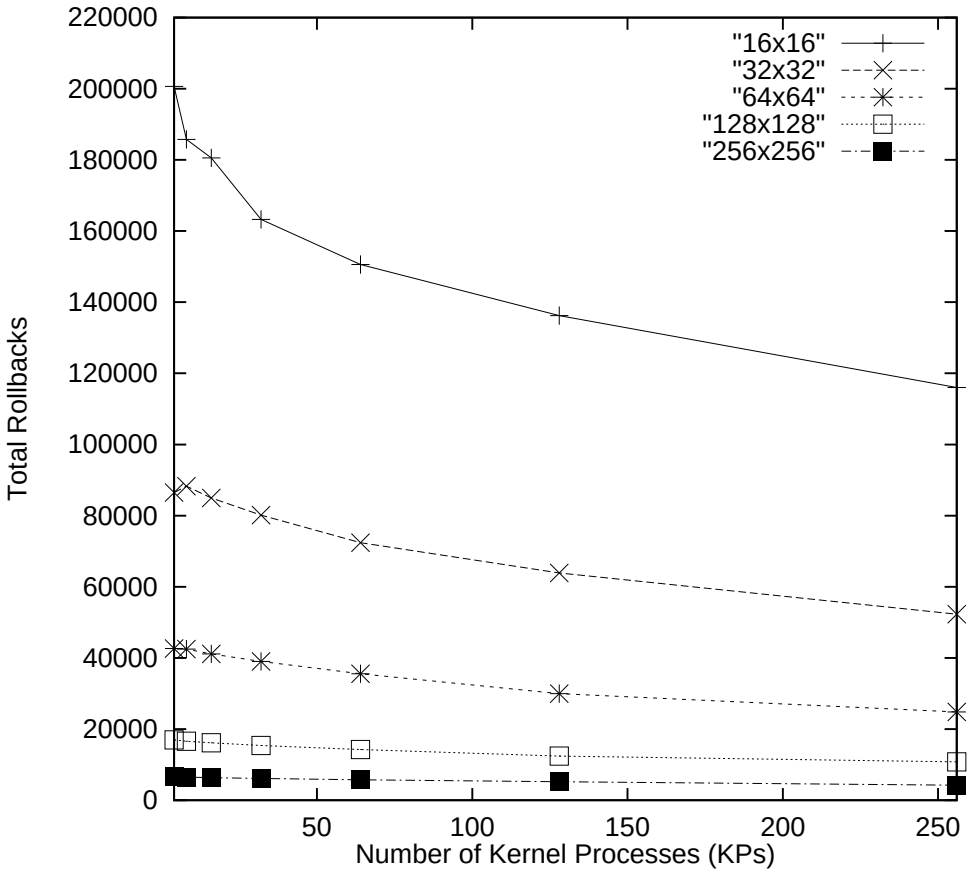


Figure 4.6: Impact of the number of kernel processes on total rollbacks.

under scoring the performance benefits of Kernel Processes. In the 64K (256x256) LP case, ROSS-OPT using 256 KPs has improved its performance by a factor of 5 compared to original ROSS without KPs and is now 1.66 times faster than GTW-OPT. In the 16K (128x128) LP case ROSS-OPT using 64 KPs is 1.8 times faster than GTW-OPT. These significant performance improvements are attributed to the reduction in fossil collection overheads. Moreover, KPs maintain ROSS' ability to efficiently execute using only a small constant number of memory buffers per processor greater than the amount required by a sequential simulator.

4.3.6 Robustness and Scalability Data

In the previous series of experiments, it is established that ROSS (with KPs) is capable of efficient execution and requires little optimistic memory to achieve that

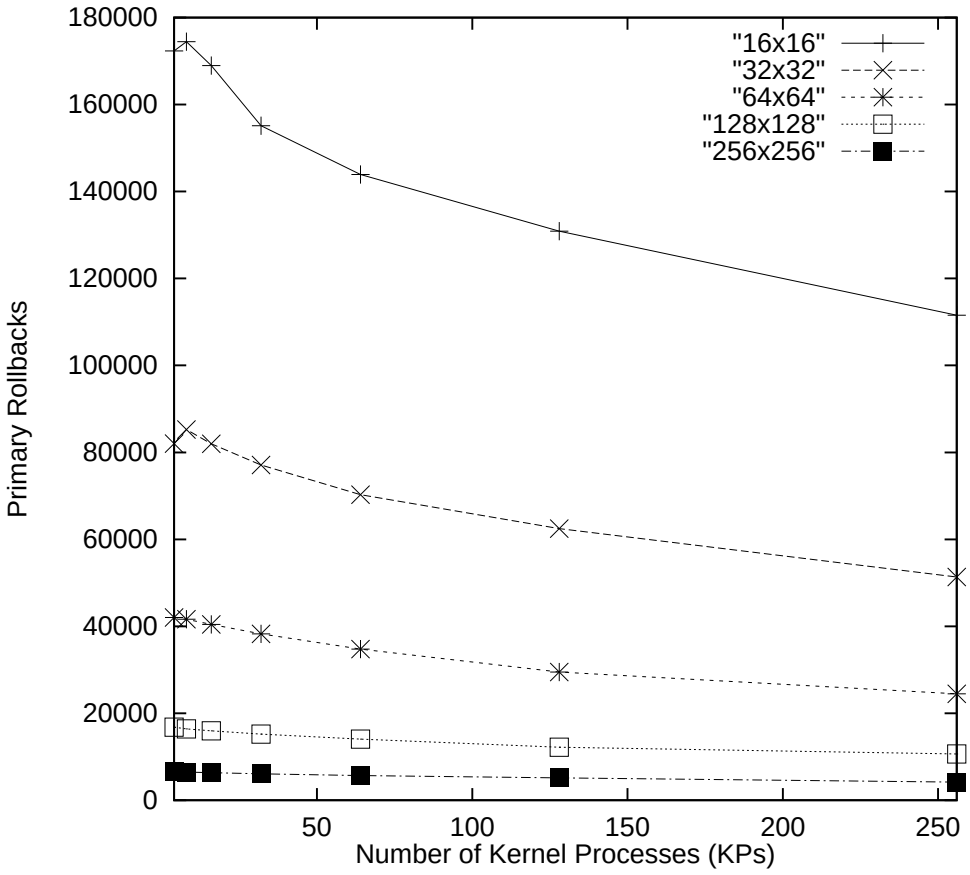


Figure 4.7: Impact of the number of kernel processes on primary rollbacks.

level of performance. However, the PCS application is a well behaved and generates few remote messages. Moreover, the last series of experiments only made use of a 4 processor system. Thus, two primary questions remain:

- Can ROSS with little optimistic memory execute efficiently under "thrashing" rollback conditions?
- Can ROSS' performance scale as the number of processors increase?

To address these questions, we present the results from two additional series of experiments. The first series examines the performance of ROSS under almost pathological rollback conditions using the rPHOLD synthetic benchmark on the quad processor PC server. The second series examines scalability using the PCS application on the Origin 2000 multiprocessor. Here, the performance of GTW

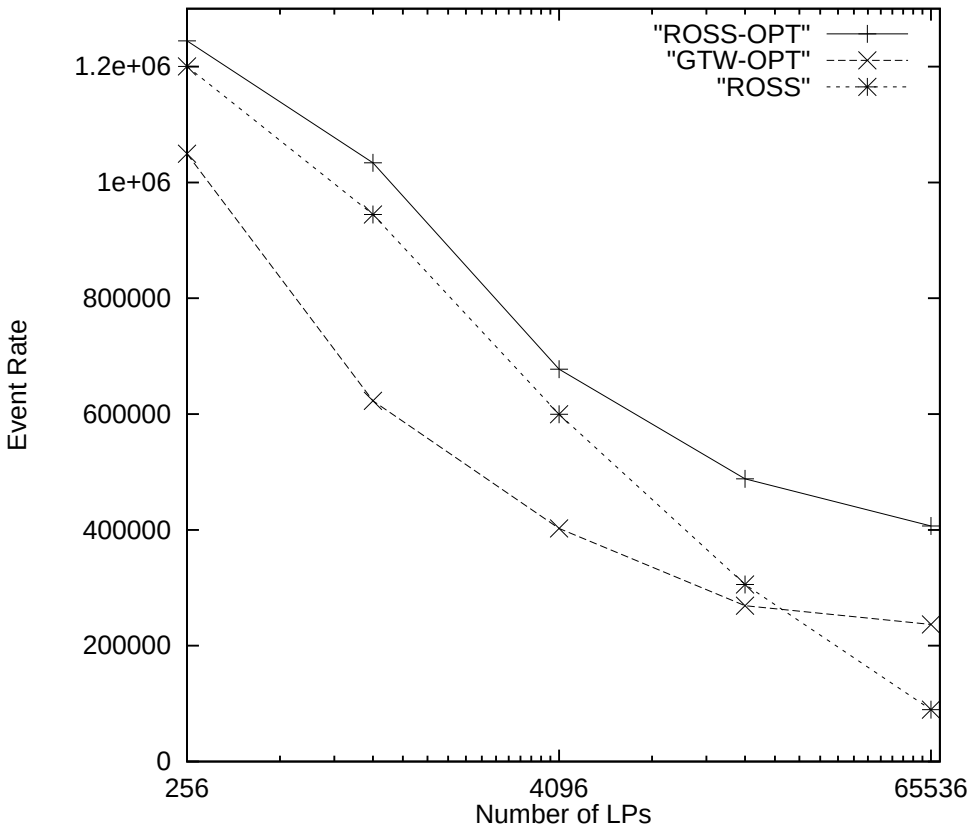


Figure 4.8: Performance Comparison: ROSS-OPT with KPs (best among those tested), "GTW-OPT" indicates GTW's performance with optimized memory pool partitioning, and "ROSS" indicates ROSS's original performance without KPs.

from [3] is used as a metric for comparison. We begin by presenting the rPHOLD results.

For the rPHOLD experiments, the number of LPs vary from 16 to 16K by a factor of 4. Recall that the number of messages per LP is fixed at 16. For the 16 LP case, there is 1 LP per KP. For larger LP configurations, up to 16 LPs were mapped to a single KP. $GVT_{interval}$ and *batch* parameters vary between 8, 12 and 16 simultaneously (i.e., (8, 8), (12, 12) and (16, 16) cases). Thus, the number of events processed between each GVT epoch ranged between 64, 144 and 256. $C = 4$ determined the amount of optimistic memory given to each processor. Thus, in the (8, 8) case, 256 optimistic memory buffers were allocated per processor.

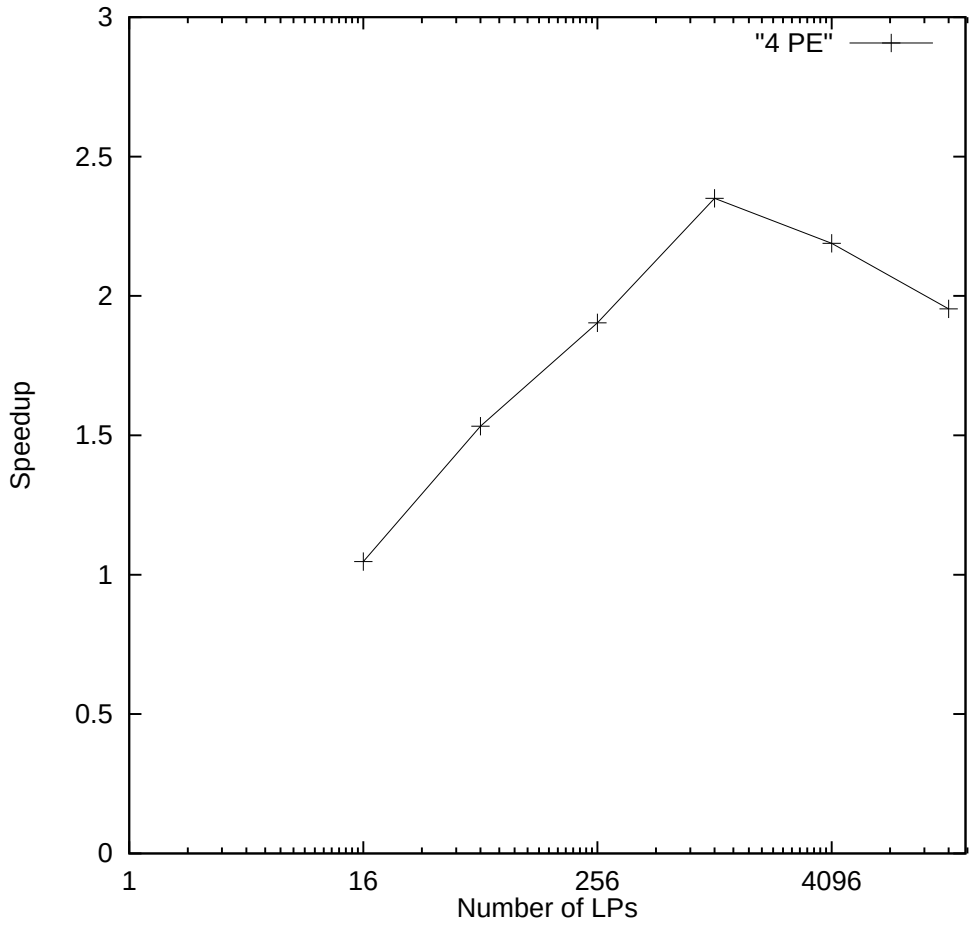


Figure 4.9: rPHOLD Speedup for ROSS-OPT. Quad processor PC server used in all data points.

Figure 4.9 shows the best speedup values across all tested configurations as a function of the number of LPs. For configurations as small 16 (4 LPs per processor), a speedup of 1.1 is reported. This result was unexpected. As previously indicated, to the best of our knowledge, no Time Warp system has obtained a speedup on this pathological benchmark configuration. The number of remote messages is so great (75% of all events processed were remote) combined with the small event granularity and high-speed Pentium III processors that rollbacks will occur with a great frequency.

As the number of LPs increase to 1024, we see a steady increase in overall speedup. The largest speedup is 2.4. However, for the 4K and 16K LP cases, we see a decrease in speedup. This behavior is attributed to the under powered memory

subsystem of the PC server not being able to keep pace with the increased memory demand caused by the larger LP configurations. For example, the 1024 LP case has only 16K messages whereas the 16K LP case has 256K messages or 16 times the event memory buffer requirements. As previously indicated, this server only has 300MB/second of memory bandwidth.

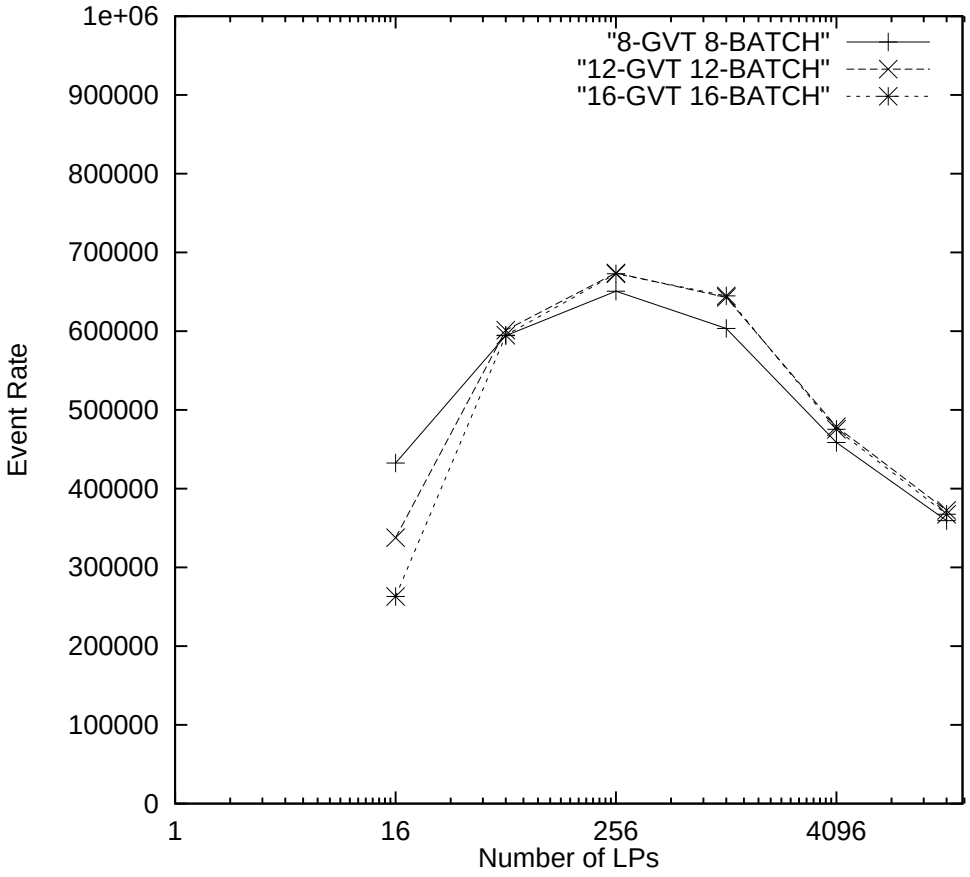


Figure 4.10: rPHOLD Event Rate for ROSS-OPT across different *GVT-interval* and *batch* parameter settings.

The limited memory bandwidth problem aside, these overall speedups are due to lower $GVT_{internal}$ and *batch* settings reducing the probability of rollback. As shown in Figure 4.10, we observe that event rate improves as the $GVT_{internal}$ and *batch* parameters are reduced from values of 16 to 8 for the 16 LP case. Here, performance improves by almost 60%. The reason performance improves for lower $GVT_{internal}$ and *batch* settings is because by reducing these settings the frequency with which the scheduler "polls" for rollback-inducing positive and anti-messages

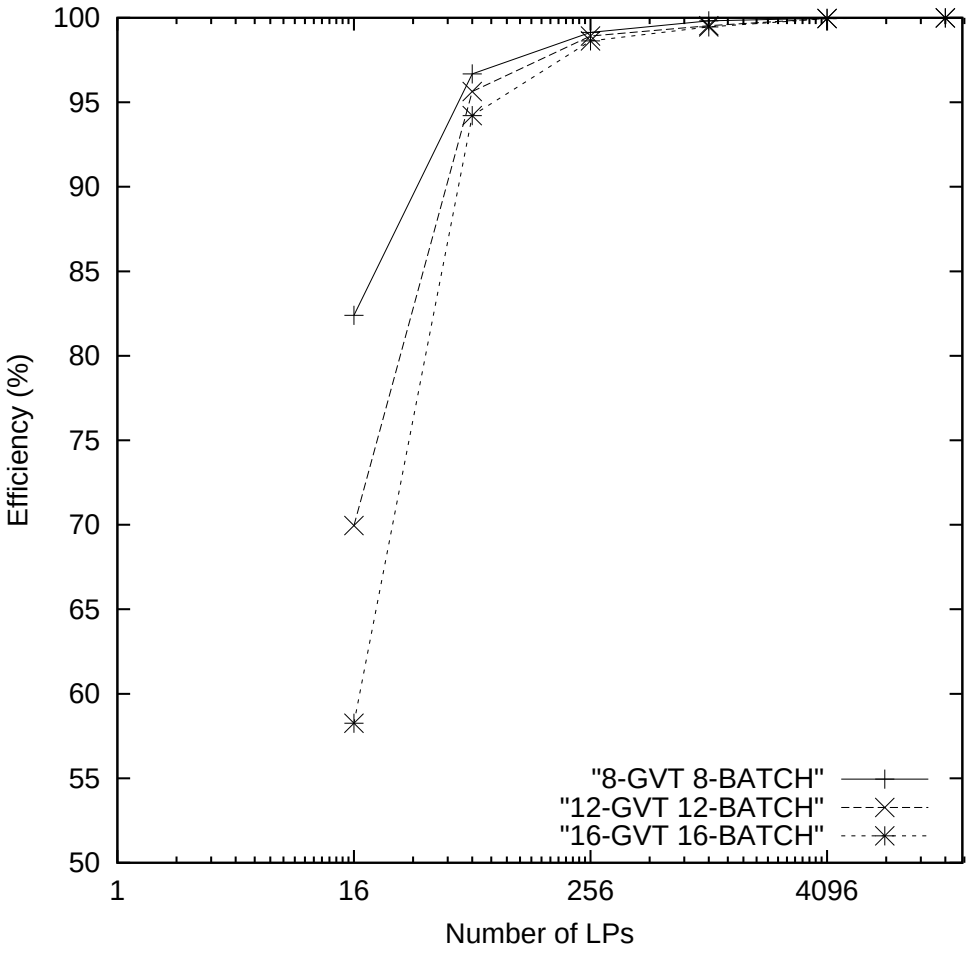


Figure 4.11: rPHOLD efficiency for ROSS-OPT across different *GVTinterval* and *batch* parameter settings. Efficiency is the ratio of "committed" (i.e., not rolled back) events to total events processed, which includes events rolled back.

increases. Thus, by checking more frequently, a potential rollback is observed by the Time Warp system much sooner or even prevented, which ultimate increases simulator efficiency, shown in Figure 4.11. Here, we see that for the (16, 16) case, efficiency is under 60% for 16 LPs, but raises to over 80% in the (8, 8) cases, which is a 33% increase in simulator efficiency.

Looking at the 64 LP case, we observe the different $GVT_{internal}$ and *batch* settings fail to yield any difference in event rate. This phenomenon is due to an even trade-off between increasing rollbacks and the overheads incurred due to the increased frequency with which GVT computation and fossil collection are done. If

we look at Figure 4.11, we see a variance of 3% in simulator efficiency among the 3 parameter configurations, with the (8, 8) case being the best. However, this slight increase in efficiency comes at the price of computing GVT and fossil collection more frequently.

As the LP configurations grow to 256 and 1024 LPs, peak speedup is obtained (Figure 4.9). Here, we observe a 95+% efficiency (Figure 4.11). The reason for this increase in efficiency and speedup is because each processor has a sufficient amount of work per unit of virtual-time. This increase in work significantly lowers the probability a rollback will occur.

For the larger 4K and 16K LP configurations, event rate, like speedup, decreases, however, the efficiency in these configurations is almost 99%. So, if rollbacks are not the culprit for the slowdown in performance, what is? Well, again, for these large configurations, the demand for memory has overwhelmed the under powered PC server, thus, the processors themselves are stalled waiting on memory requests to be satisfied.

In this final series of experiments, ROSS' ability to scale on larger processor configurations is examined. Here, we compare the performance of ROSS to GTW from what was reported in [3] on the SGI Origin 2000. For this scalability comparison, the PCS application is used. This time, PCS is configured with 14400 LPs (120x120 cell configuration). This size allows an even mapping of LPs to processors across a wide range of processor configurations. Here, we report our findings for 1, 2, 3, 4, 5, 6, 8, 9 and 10 processors. The number of subscribers per LP is increased to 100, making the total message population over 1.4 million. Again, both GTW and ROSS are using reverse computation.

In terms of memory utilization, GTW allocates 360 MB of memory to be used for event processing. This was determined to be experimental the best configuration for GTW. Because of the large amount of memory allocated, GTW had to be compiled such that a 64 bit address was available. ROSS is configured with $GVT_{interval} = 16$ and $batch = 16$ and $C = 8$ for all runs. We increase C to compensate for the additional time it would take to compute GVT for larger processor configurations. In total, between 0.3% and 1.4% extra memory is allocated over

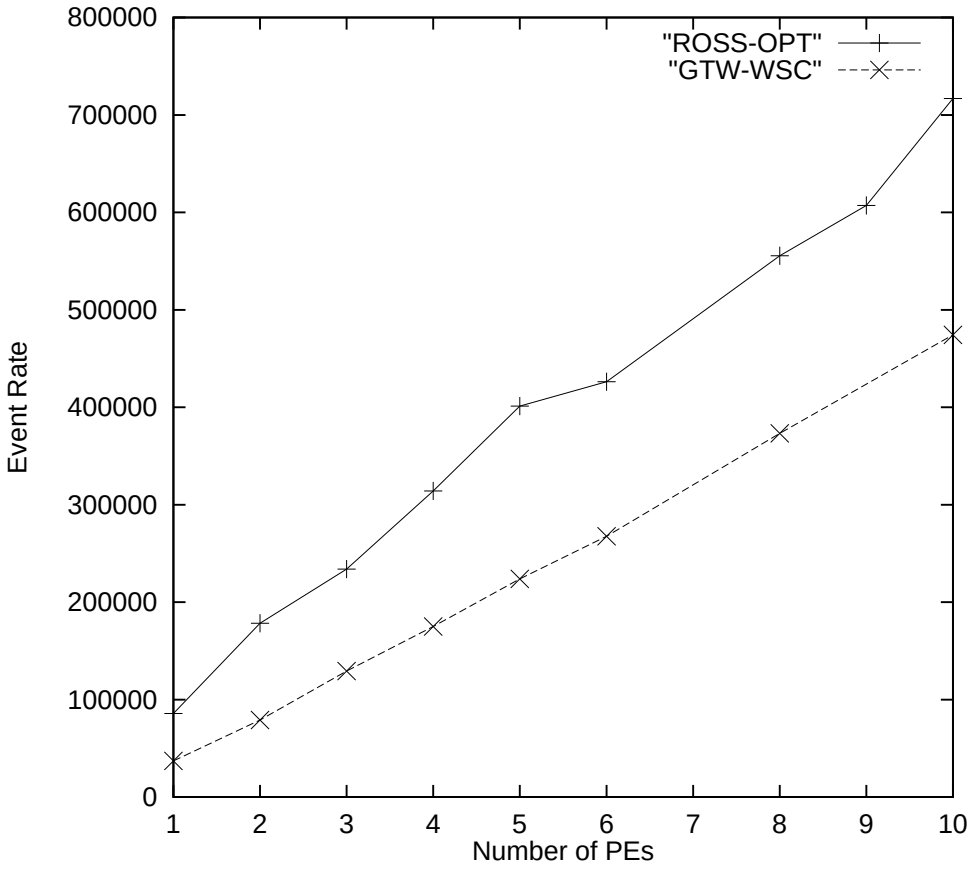


Figure 4.12: Performance Comparison on the SGI Origin 2000 using PCS: "ROSS-OPT" indicates ROSS with the best performing KP configuration, and "GTW-WSC" indicates GTW's performance as determined from [3] using optimized memory pool partitioning.

what is required by the sequential simulator to support optimistic processing. This yields a maximum memory usage of only 176 MB for the 10 processor configuration and is less than half the memory given to GTW. Because ROSS' memory consumption is significantly lower, the simulation model fits within a 32 bit address space.

We note that the GTW's performance data was collected on an Origin 2000 with 8 MB of level-2 cache. ROSS' performance data was collected on an SGI Origin 2000 with only 4 MB of level-2. Thus, the machine used by GTW has some performance advantage over the Origin 2000 used for ROSS' experiments. This advantage, however, is negated due to GTW being compiled for 64-bit address space (i.e., loads and stores and address calculations are done on 64 bits and not

32).

In Figure 4.12, the event rate for both ROSS and GTW are reported as a function of the number of processors used. Here, we find a similar picture to what was seen on the quad processor PC server. ROSS is significantly faster than GTW across all cases. In the optimized sequential case, ROSS is over 2 times faster. Here, both GTW and ROSS' event schedulers have been reduced to only support sequential execution. All GVT, fossil collection functionality has been striped away. Events are committed as soon as they are processed. Consequently, the only difference between these two are they way in which each system manipulates its core data structures. As previously described, ROSS implements a bottom-up, pointer-based design which facilitates better data locality. We believe this increase in locality is the cause for the increase in performance over GTW.

For the multiprocessor runs, ROSS is able to scale to a speedup of 8.4 on 10 processors and consistently outperforms GTW in every case. The 5 processor case is where the largest gap in performance occurs between ROSS and GTW. Here, ROSS is 2 times faster than GTW. In the 10 processor case, ROSS is 1.6 times faster than GTW.

Finally, we observe in the multiprocessor cases, GTW's partitioned buffer scheme [91] does not appear to be a factor on the Origin 2000 architecture. This scheme was originally designed to avoid the "false" sharing of memory pages on the KSR ALL-CACHE shared-memory multiprocessor system. The Origin's distributed memory is organized completely different and provides specialized hardware support for memory page migration [97]. ROSS only utilizes a single free pool of memory and those buffers can be sent to any processor, which on the KSR would cause a great deal of false sharing. ROSS, however, on the Origin 2000 appears to scale well and does not appear to suffer from unnecessary memory management overheads. We believe efficient, hardware assisted page migration combined with the directory based cache-coherence and memory pre-fetching is allowing ROSS to achieve scalable performance without having to resort to more complex buffer management schemes, which can complicate a parallel simulator's design and implementation. However, more experimentation is required before any definitive conclusions can be drawn.

4.4 Related Work

The idea of Kernel Processes is very much akin to the use of clustering as reported in [80, 83, 89], and [109]. Our approach, however, is different in that it is attempting to reduce fossil collection overheads. Moreover, KPs , unlike the typical use of clusters, are not scheduled in the forward computation and remain passive until rollback or fossil collection computations are required.

Additionally, while low memory utilization is experimentally demonstrated, we do not consider KPs to be an adaptive approach to memory management, as described by [86] and [87]. KPs is a static approach that appears to yield similar reductions in memory requirements when combined with an efficient GVT algorithm.

In addition to "on-the-fly" fossil collection, Optimistic Fossil Collection (OFC) has been proposed [116]. Here, LP states histories are fossil collected early without waiting for GVT. Because we are using reverse computation, complete LP state histories do no exist. Thus, this technique will not immediately aide in ROSS' approach to fossil collection.

4.5 Summary

The design and implementation of a stable, highly-efficient new Time Warp system is presented. For the first time, it is shown that this system generates excellent performance using only the minimal additional memory required to sustain efficient optimistic execution. This high-performance, low-memory system is the result of combining several key technical innovations:

- pointer-based, modular implementation framework,
- Fujimoto's GVT algorithm [90],
- reverse computation, and
- the introduction of *Kernel Processes(KPs)*.

It was shown that KPs lower fossil collection overheads by aggregating processed event lists. This aspect allows fossil collection to be done with greater frequency,

thus lowering the overall memory necessary to sustain stable, efficient parallel execution.

In the performance demonstration, two applications are used: PCS and a pathological synthetic workload model, *rPHOLD*; and two different parallel computing platforms are used: quad processor PC server and the SGI Origin 2000. For PCS, it was shown that ROSS is a scalable Time Warp system that is capable of delivering higher performance using little optimistic memory. For rPHOLD, ROSS demonstrates that even under harsh rollback and memory limited conditions, good speedups are obtainable. These characteristics make ROSS an ideal system for use in large-scale networking simulation models.

In re-examining the performance data from a higher level, it appears that low-memory utilization and high-performance are no-longer mutually exclusive phenomenon on Time Warp systems, but instead complement one another. On today's almost infinitely fast microprocessors, the parallel simulator that "touches" the least amount of memory will execute the fastest. For the experiments conducted in this study, ROSS "touched" much less memory than GTW due to its structural and implementation organization. We anticipate the trend of memory utilization determining system performance to continue until new architectural and software techniques are developed that breaks down the "memory wall".

In an attempt to reduce the number of systems parameters, we have artificially linked GVT computation and fossil collection frequency to "loops" through the main scheduler via the $GVT_{internal}$ parameter. This parameter, inconjunction with *batch*, also determines how frequently the incoming messages queues are "polled". The performance data shows that more frequent polling of these queues can greatly increase simulator efficiency when the rollback probability is high. In the future, we would like to explore the decoupling of GVT and fossil collection computations from these parameters and instead make them completely dependent on the availability of memory. Our belief is that simulator performance will increase for lower $GVT_{interval}$ and *batch* settings over what has been presented here.

CHAPTER 5
Seven O'Clock: A New Distributed GVT Algorithm Using Network Atomic Operations

At the heart of an optimistic parallel simulation system is the ability to reclaim memory from its virtual time past and re-use it to schedule future events as well as support state-saving operations as part of speculative event processing. Global virtual time (GVT) defines a lower bound on any unprocessed event in the system and defines the point beyond which events *should not* be reclaimed. Thus, it is imperative that the GVT computation operate as efficiently as possible.

Global virtual time (GVT) algorithms must solve two key problems. The first is *the transient message problem*. Here, a message is delayed in the network and neither the sender nor the receiver consider that message in their respective GVT calculation. Thus, a GVT algorithm must account somehow for all messages scheduled. The second problem is called *simultaneous reporting*. This problem arises "because not all processors will report their local minimum at precisely the same instant in wall-clock time" [73]. Here, the underlying assumption is that event processing is allowed to continue asynchronously during the GVT computation which enables better overall parallel performance.

Asynchronous GVT algorithms rely on the ability to create a "cut" across the distributed simulation that divides events into two categories: past and future [73, 74]. GVT is then defined by the lower bound on unprocessed events in the "past" of the cut, which is in effect an estimate of the true GVT since events are being processed during its computations. Creating a cut can be done in several ways depending on the architecture of the machine(s) being used. For distributed computing platforms, the primary method of creating a "cut" is via message-passing as was defined by Mattern [74]. Here, messages are sent such that at most two cuts are made. The first cut signals the "start" of the GVT computation. The second cut, if needed based on message counts computed in the first cut, consider any transient messages discovered from the first cut. Please note, that these cuts

need not be *consistent*. A *consistent cut* is defined as a cut where there is no message that was scheduled in the future of the sending processor but received in the past of the destination processor. These messages can be ignored because by definition they must be scheduled at a time that is greater than GVT computed using a consistent cut. The "cuts", while not consistent, effectively divide past from future in a causally consistent manner to solve the simultaneous reporting problem. Additionally, because of the use of message counts, it is able to determine if a second "cut" round is needed and traps any transient messages.

In contrast, a shared memory multiprocessor greatly simplifies the GVT algorithm. Fujimoto's GVT algorithm [75] generates a cut by setting a global flag positive. In a shared memory system this operation is observed on all processors in a causally correct order because the underlying hardware memory management mechanism ensures that no two processors will observe different orderings of memory references to a shared variable. Shared memory multiprocessor systems that adhere to this memory ordering model are called *sequentially consistent* [76]. The impact this memory model has on a GVT algorithm are that: (i) *no* messages are lost which prevents the transient message problem, and (b) the simultaneous reporting problem is solved because all processor effectively "observe" the start of the GVT calculation at the same instant of wall-clock time.

The motivation behind our research here is the question: *Is there a method to achieve some of the benefits of sequentially consistent shared memory but in a loosely coordinated, cluster computing environment?* The answer turns out to be *yes*. We formalize the idea of a *network atomic operation (NAO)*, which enables a zero-cost cut mechanism which greatly simplifies GVT computations in a cluster computing environment. We demonstrate its reduced complexity by extending Fujimoto's shared memory algorithm to operate across a cluster of shared-memory multiprocessors (SMP).

5.1 Fujimoto's GVT Algorithm and NAOs

We begin with an overview of Fujimoto's GVT algorithm, as shown in Algorithms 1 – 5. Please note, there are minor modifications from the original algorithm

Algorithm 1 Fujimoto's Shared Memory GVT Algorithm: Variable Definitions.

Global Variables

```
int gvt_flag;
lock_t gvt_lck; /* mutual exclusion variable */
int gvt_interval; /* number of time thru batch loop */
virtual_time_t lvt[npe]; /* LVT of each processor */
virtual_time_t gvt;
```

Processor Private Variables

```
virtual_time_t send_min_ts;
```

Algorithm 2 Fujimoto's Shared Memory GVT Algorithm: Initiate GVT

Steps to Initiate GVT within Scheduler Loop

```
if(this processor is the MASTER) {
  gvt_cnt++;
  if( gvt_cnt >= gvt_interval AND
  processor status is GVT_NORMAL)
  {
    gvt_cnt = 0;
    if(gvt_flag == -npe)
    {
      lock(&gvt_lck);
      gvt_flag = npe;
      unlock(&gvt_lck);
    }
    set processor status to GVT_COMPUTE;
  }

} else if (gvt_flag > 0 AND processor status is GVT_NORMAL) set
processor status to GVT_COMPUTE;
```

Algorithm 3 Fujimoto's Shared Memory GVT Algorithm: Receive Events

Steps to Receive Events and Anti-messages within Scheduler Loop

```
move positive messages from shared memory message queue to
processors priority queue. process any rollbacks. remove
anti-messages from shared memory ‘‘cancel’’ queue. process any
message cancellations and rollbacks.
```

presented in [75], but the correctness and efficient execution is preserved. These 5 parts are described as follows:

1. **Variables (Algorithm 1):** The key shared variable is the `gvt_flag`, which contains a mutual exclusion variable, `gvt_lck`.

Algorithm 4 Fujimoto's Shared Memory GVT Algorithm: Compute GVT

Steps to Compute GVT Once Initiated within Scheduler Loop

```
if( processor status is GVT_COMPUTE ) {
  set processor status to GVT_WAIT
  lvt[my_pe] = min(send_min_ts, smallest event in priority queue);
  lock(&gvt_lck);
  gvt_flag--;
  if( gvt_flag == 0)
  {
    gvt = min( lvt[0] ... lvt[npe-1] );
    gvt_flag = -1;
    set processor status to GVT_NORMAL;
    reset send_min_ts to max time value;
    unlock(&gvt_lck);
    collect processed events and state < GVT;
    if( gvt > end time of simulation ) goto DONE;
  } else
  {
    unlock(&gvt_lck);
    set processor status to GVT_WAIT;
  }
} else if( processor status is GVT_WAIT AND gvt_flag < 0 ) {
  lock(&gvt_lck);
  gvt_flag--;
  unlock(&gvt_lck);
  if( gvt > end time of simulation )
    goto DONE;
  collect processed events and state < GVT;
  set processor status to GVT_NORMAL;
  reset send_min_ts to max time value;
}
```

2. **Initiate GVT (Algorithm 2):** The algorithm is initiated when the "master" processor iterates through the event scheduler loop `gvt_interval` times before setting the `gvt_flag` equal to the number of processors (i.e., `npe`). Here is where the algorithm exploits the sequentially consistent memory properties. Every processor will "observe" the start of the GVT at the same instant in wall-clock time. More precisely, once the flag has been set, any other messages sent, are the responsibility of the sender, as shown in Algorithm 5. This provides a true separation between events in the logical past and logical future. When

Algorithm 5 Fujimoto's Shared Memory GVT Algorithm: Process and Schedule Events.

Steps to Process and Schedule Events within Scheduler Loop

```
  Process smallest event in priority queue;
  if( sending a new event during event processing )
  {
    enqueue message on destination processor's receive queue;
    if( gvt_flag > 0 and processor status is NOT GVT_WAIT )
        send_min_ts = min( send_min_ts, time-stamp of new event);
  }

DONE:   compute parallel simulation stats and exit.
```

another processor observes the start of a GVT computation, it changes its status to "needs to compute an local virtual time (LVT)" This is denoted by `send_min_ts`.

3. **Receive Events (Algorithm 3):** Here, new events and anti-messages are processed from arrival queues shared between processors. Each processor has its own externally exported queue that all other processors use to send events. A mutual exclusion lock is used around the queue to correctly serialize the arrival of either new events or anti-messages. Because of sequentially consistent memory, no message can be lost "in the network" and so the transient message problem is intrinsically solved. Observe that this "receive" and process rollbacks and anti-messages is a necessary step prior to computing any part of the GVT. It is also a normal step in every iteration through the scheduler loop.

4. **Compute GVT (Algorithm 4):** Once the new events and anti-messages are processed, each processor computes its local virtual time (LVT) value, which is the smallest unprocessed event that it is "aware" of, which includes any events it sent after the `gvt_flag` was set. This is denoted by `send_min_ts`. The last processor to compute its LVT also computes the minimum among all LVTs, which becomes the new GVT value. To inform other processors that the new GVT value is available, the `gvt_flag` is set to negative one. The last

Algorithm 6 `gvt_interval` is a predefined number of iterations, and gvt_count is the current number of iterations.

```
CPU 0:
   a. If(gvt_interval == gvt_count)
   b.       set gvt_flag positive

CPU 1:
   c. if(gvt_flag)
   d.     start processing LVT
```

processor never "waits" for the GVT value and skips that state, while all other processors move to the "asynchronously waiting for GVT" state.

5. **Process Events (Algorithm 5):** In this last step, forward event processing commences. Here, the smallest event is removed from the pending event set and processed. If a new event is scheduled and the `gvt_flag` has been set greater than zero and the LVT value has not been reported (i.e., the processor should not be in the "wait" state), then it means *this* processor must consider this event in its LVT computation.

5.2 Network Atomic Operations

To directly extend Fujimoto's GVT algorithm to a network of machines, would require a sequentially consistent distributed memory model, similar to what is provided in a shared memory system. As we described above, each processor observes the "start" of the GVT computation at the same wall clock time because of sequentially consistent memory. In reality, a processor attempting to read the flag may be stalled while the underlying system updates the local cache with the correct value of the flag. Consider the following abstracted view of the algorithm run in parallel on an SMP machine as shown in Algorithm 6.

Running this code on a single processor defines the sequential consistency. The statements could be ordered on a single CPU as $O = \{a, b, c, d\}$. In this case both CPU 0 and 1 would begin computing GVT. However, if we change the interleaving of instructions to $O = \{a, c, b, d\}$ then the GVT computation would only begin on CPU 0. CPU 1 would begin processing more events, but when an event is sent,

Algorithm 7 Here, `gvt_interval` redefined as a measure of time usually in clock cycles, and not defined as the number of batch round through the scheduler.

```
CPU 0:
   a. if(local clock time >= gvt_interval)
   b.       start computing GVT

CPU 1:
   c. if(local clock time >= gvt_interval)
   d.     start computing GVT
```

the `gvt_flag` would be checked per the algorithm to ensure the sender correctly accounts for events during the GVT computation. The instruction, call it e, would then be accounted for by CPU 1 because e occurred after instruction b, and the consistent cut is properly formed.

NAOs provide a similar functionality in a distributed system, however they are clock-based and not memory or state-based. **The general concept is that an operation may occur atomically within a network of machines if all machines "observe" the event at the same instant of wall clock time.** This functionality can be implemented on modern processors because most now provide a time-stamp counter, or clock-cycle counter for performance measuring, such as the `rdtsc` instruction on all x86 series processors [49]. So we can compute wall-clock time based off of each processor's time-stamp counter and synchronize these counters to a common view of wall clock time. Calls to reading the CPU clock adhere to the principles of a sequentially consistent memory model because wall clock time is consistent across all processors. Consider the following clock-based approach shown in Algorithm 7.

If we again attempt to create a sequential ordering of instructions, it becomes obvious that any permutation is guaranteed to be consistent. Consistency is guaranteed because instructions a and c in Algorithm 7 will evaluate to true if and only if the same instance of wall clock time has passed for each CPU. Because we can only read the current wall-clock time (as measured in clock cycles), any permutation of the possible orderings is valid because wall clock time is *assumed* to be same for all processors. There are limitations which we will discuss later in Section 5.3.3.

So NAOs may be characterized as a subset of the possible operations provided by a complete sequentially consistent memory model. For example, NAOs can only occur at predefined intervals, not dispersed throughout the timescale of the running application. Not only must NAOs occur at agreed upon intervals, but they must also take on a specific meaning or value. In the case of the GVT algorithm, we use NAOs to generate a consistent cut. The system is either in a GVT computation, or it is not. Further, GVT computations occur at a predefined frequency through the runtime of the application. An NAO cannot be used to give some global variable any value, because the only global variable in an NAO is wall clock time. However, any sequence of operations can be performed once the clock has been read.

5.2.1 Clock Synchronization

The heart of a network atomic operation is the assumption that all processors share a highly accurate, common view of wall-clock time. For this to occur, each processor's time-stamp or cycle-counter must be synchronized in some fashion. This is a well researched problem in distributed computing. The most recent, relevant result for our operating environment is by Ostrovsky and Patt-Shamir [32]. Here, the present provably optimal clock synchronization scheme where the clocks have drift and the message latency may be unbounded. Previous to this result, all other optimal results were based on non-drifting clocks. Moreover, they suggest that operational clock synchronization algorithms *need not be general* and "that they should work for the particular system in which they are deployed". We take this view here. In particular, because of the time-scale of the clock is 1,000 times greater than message sends (i.e., nanoseconds vs. microseconds), clock drift rates can largely be ignored here.

Additionally, since our contribution is not about clock synchronization algorithms, we used a simplified approach. To synchronize the clocks across all processors, we use a *network barrier*. Here, a master time keeper sends a synchronization message to each node, which responds back with its local time-stamp-counter. The master time keeper then sends a message to each processor with an appropriate time-stamp counter value that would be when in real-time measure in cycles the

first GVT is to occur. Upon receipt of that message, each processor is released from the barrier and begins processing events. We recognize that for a large 1000 processor cluster, this approach has some scalability limitations. For such an operating environment, we would implement Ostrovsky and Patt-Samir algorithm [32].

5.3 Seven O'Clock GVT Algorithm

We can now give a definition of a network atomic operation:

Definition: *An NAO is an agreed upon frequency in wall-clock time at which some event is logically observed to have happened across a distributed system.*

Each processor in the system uses the NAO to determine the current state of the system depending upon the logical meaning of the NAO. For example, we use an NAO to determine if a GVT computation has been started. There is no actual global variable, such as the `gvt_flag` in Algorithm 2, which signals the start of the GVT. Instead, we simply compute GVT every n units of wall-clock time.

We have affectionately call this algorithm the *"Seven O'clock Algorithm"*. Seven O'clock comes from the idea that if we could have synchronized wall clocks on each network node, then we could simply compute GVT at well-defined intervals, i.e., every minute starting from seven o'clock, where seven o'clock is simply the start time of the scheduler. During the discussion of this algorithm we assume that each processor's timestamp counter is perfectly synchronized with all of the other counters. We introduce the complexity of clock drift and jitter at the end.

As we previously indicated, if one were to open up the underlying hardware implementation of a sequentially consistent shared memory system, a number of messages would be observed being passed over the memory bus between memory and cache modules. Also, any memory reads to a shared location could be blocked while waiting for the memory address to be made consistent. In particular, these consistency messages would either be well synchronized in time over a memory bus or acknowledged over a network depending on the architecture [77, 78]. Because we assume a distributed message passing system without acknowledgments, we need some additional information about the communications environment to avoid the transient message problem (i.e., events lost in the network). This prob-

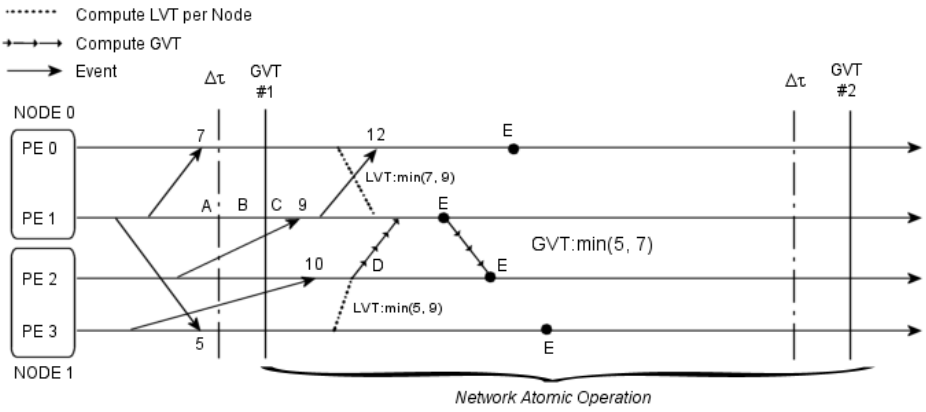


Figure 5.1: States of the Seven O'Clock Algorithm:

lem can be overcome by adding a small amount of time to the NAO expiration, *max_send_delta_t*.

Definition: *max_send_delta_t is a worst case bound on the time to send an event through the network.*

This delta allows for the sending processor to account for remotely sent events which may cross the cut boundary. Note that this is not the same as delta causality which allows for excessively old events to be discarded. While it may seem unreasonable to assume such a value can be determined in practice, current cluster computing networks rely on high-speed switching fabrics. These fabrics typically have extremely low loss probabilities ($1e-12$ or less) and can typically support the full bandwidth of all ports. Consequently, the worst case is experimentally computable and does not vary greatly from the average case.

The states of the Seven O'Clock algorithm are shown in Figure 5.1 and are discussed below.

- **State A:** Events are processed normally and not accounted for in GVT computation. This is no different from Fujimoto's GVT Algorithm.
- **GVT "start":** The NAO signals the consistent start of the GVT to all processors, just as setting the `gvt_flag` in Fujimoto's algorithm.
- **State B:** Events sent during the `max_send_delta_t` interval are accounted for on the sending side. This is similar to how Fujimoto's Algorithm uses

`gvt_flag` to capture events the receiving processor might not consider in the sender's LVT computation. Here, our algorithm reads the processor's local cycle counter, determines if it is within the `max_send_delta_t time` of a GVT computation. If it is, then it captures this event's timestamp against the minimum of any previously scheduled events during the `max_send_delta_t` time.

- **State C:** Processors compute LVT by taking min(unprocessed events, events sent during B). This is identical to Fujimoto's GVT Algorithm.
- **State D:** Node 1 is first to complete local GVT algorithm and propagates this value to other nodes.
- **State E:** Node 0 completes it's local GVT algorithm and receives Node 1's. It takes the minimum of the two and retransmits this value back to Node 1. The other processors check for the new GVT value at some point in the future and read the value from the shared memory. The processors return to state A.

The only other differences between Fujimoto's GVT algorithm and Seven-O'Clock are: (a) physically receiving messages and (b) physically sending messages. In the receiving step (Algorithm 3), all remotely sent new event messages and anti-messages are read from a communications channel, such as a socket. This is in addition to the shared memory queuing structures. From the standpoint of the GVT algorithm, it captures messages sent over either communications medium. Likewise, when a message is sent, the algorithm does not differentiate between which events are shared memory or off-system messages.

5.3.1 Proof of Correctness

As previously noted, in order for a GVT algorithm to operate correctly, it must solve the transient message and simultaneous reporting problems.

First, we assume all processors clocks are perfectly synchronized and there are no clock drift or jitter problems. We will relax this constraint later.

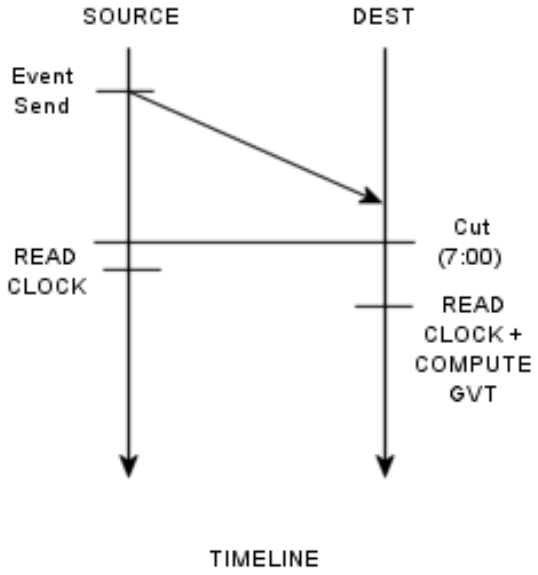


Figure 5.2: Simultaneous Reporting Problem: Source and destination processors see the cut being created at the same instant in wall clock time, so there is no opportunity for an event to "slip between the cracks".

Proof: To prove that a transient message cannot occur, assume that a transient message occurs in the scheduler. Then the time to send the event must be greater than the *max_send_delta_t*. But by definition *max_send_delta_t* is a worst-case bound on the time to send an event. This leads to a contradiction because the transient event took longer to send than the worst-case bound.

Next, the simultaneous reporting problem occurs when all processors do not begin computing their local minimum at the same instant. In fact, this is exactly what happens in the Seven O'clock algorithm. Because the consistent cut is generated using an NAO, each processor in the distributed system begins accounting for messages at precisely the same instant in wall clock time. Therefore this problem does not occur in this GVT algorithm as shown in Figure 5.2.

Theorem: *The simultaneous reporting problem cannot occur in a system where a consistent cut is defined across all processors at precisely the same instant in wall clock time.*

Proof: *Assume to the contrary that the simultaneous reporting problem can*

occur.

CASE 1: Assume each processor's clock is perfectly synchronized with all other processor clocks. For the simultaneous reporting problem to occur, at least two processors must have different views of wall clock time. This is a contradiction because there only exists one notion of wall clock time.

CASE 2: Each processor's clock is synchronized with some degree of error, which is bounded by *epsilon*. In order for the problem to occur now, at least two processors must have different views of wall clock time, which differs by at most *epsilon*. This means that in *epsilon* time between two processors, a message was sent which was not accounted for. This is a contradiction because it is not possible to send a message in *epsilon* time and not account for the event being sent. Consider the steps for remote sending of events:

1. *Send the event.*
2. *Read local time-stamp clock.*
3. *If time_now + max_send_delta_t >= gvt_interval, then account for the event.*

Since *epsilon* is a value far less than *max_send_delta_t*, we must always account for the sent event. In the event that *epsilon* >= *max_send_delta_t*, we simply change our *max_send_delta_t* to have a larger value to overcome *epsilon*.

5.3.2 Problems with Clock-Based Algorithms

While the discussion of clock synchronization and its associated problems are outside of the scope of this thesis, the Seven O'clock algorithm does provide a mechanism to solve each of these problems. Three problems arise in any clock-based algorithm: drift, jitter and synchronization error. During the course of a simulation, clocks may drift together, or apart. In the later case, this can lead to a gradual disparate view of time. Clock jitter occurs when a time is discretized and the width of the time units is not uniform. This can be a cause of clock drift over time depending on the frequency and size of the jitter. Finally, it is difficult to synchronize clocks to a high degree of granularity. This can lead to two processors

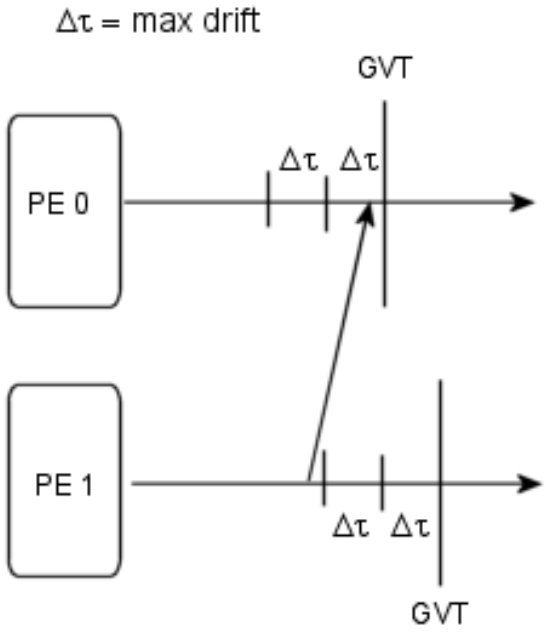


Figure 5.3: Eliminating Clock Drift and Clock Synchronization Errors. The two processors are maximally misaligned. The remotely sent event cannot cross the cut boundary without being accounted for by the sender.

being synchronized, but off by an indeterminate amount. Each of these problems can be dealt with in the Seven O'Clock algorithm by adjusting the definition of *max_send_delta_t*. Recall that this value is a worst-case bound on the time to send an event between two processors.

We now redefine max_send_delta_t as the maximum of:

1. worst-case bound on events sends
2. two times the synchronization error
3. two times the maximum clock drift during execution

The *max_send_delta_t* parameter is simply an adjustment to the NAO to compensate for in-transit events that the sender must account for if they will not be received prior to the NAO expiring at the receiver. It can also be used to overcome the above mentioned problems when they are larger than of the maximum send

time. In Figure 5.3, two processors have drifted as far apart as is possible for a given run. As long as *max_send_delta_t* is twice the maximum drift, it is possible for the sending processor to account for the event. In practice these values are several orders of magnitude smaller than the maximum time to send an event through the network, and can be safely ignored.

5.3.3 Limitations

Two issues arise from the introduction of the Seven O'clock algorithm that do not exist in other GVT algorithms. Both stem from the general problem which is that the Seven O'clock GVT algorithm cannot be "forced". First, GVT must advance for a simulation to determine that the simulation must end. In the ROSS parallel scheduler this happens quickly because events to be scheduled past the end time are not processed and the GVT interval counter climbs quickly so that GVT may be computed when the system is effectively out of events. The Seven O'clock algorithm cannot be forced because each node must wait for the NAO to expire. At the end of a simulation, the ROSS system has no events scheduled for processing, and is simply waiting for the CPU to compute the next GVT interval. This situation cannot be aborted early because each network node is unaware of other nodes still actively processing events. The time wasted is bounded in the worst case by the size of the GVT interval. It is reasonable to expect this value to be small in relation to the overall time spent in execution, and so we do not consider this to be a major loss because it can be effectively amortized away.

The fact we cannot force a GVT computation leads to the second limitation to the system. When all of the free events were consumed in the ROSS parallel scheduler, we were previously able to "jump" the GVT interval counter and force a GVT computation to occur. Refreshing the GVT value meant that we could reclaim at least one additional event in the system and continue forward processing. When free events are exhausted in the ROSS distributed scheduler we can no longer force GVT, and so must simply wait for the next GVT interval to pass. This problem occurs when we do not have sufficient optimistic memory to continue forward execution. The solution to this problem is to simply add more optimistic memory, *or*

more network nodes, further distributing the model so that this does not occur. An indirect cause of this problem is speculative execution. In this case, stalled waiting for GVT to pass can act as a throttle on the faster nodes in the network such that they cannot overly speculatively execute, thereby creating the potential for long rollbacks. This problem is best solved by tuning the GVT interval to more closely match the amount of available memory.

It is possible to force a GVT computation without disrupting the agreed upon NAO by simply sending a round of "force" messages. These messages would indicate that the NAO has pre-maturely expired and effectively changing the next NAO interval to *immediate*. Because we have not implemented this feature we cannot prove it's correctness, however, we have not found it difficult to choose reasonable NAO settings. In the performance section of this paper we compute a design of experiments to determine the best setting for this variable.

5.3.4 Uniqueness

	Fujimoto	7 O'clock	Mattern	Samadi
Cut Calculation Cost	O(1)	O(1)	O(N) or O(log N)	O(N) or O(log N)
Parallel/Distributed	P	P+D	P+D	P+D
Global Invariant	Shared Memory Flag	Real Time Clock	Message Passing	Message Passing
Independent of Event Memory	N	Y	N	N

Table 5.1: Comparison of major GVT algorithms to Seven O'clock.

The Seven O'clock GVT algorithm is unique because it uses synchronized Real-Time Clocks as the global invariant. No message passing is required to communicate the current view of time among the processors. While algorithms such as Mattern's or Samadi's can be made scalable, there is no lower cost than reading the timestamp counter on a CPU for constructing a consistent cut. This fact leads us to believe that the Seven O'clock algorithm must be the most efficient cut algorithm possible. This is equivalent to the cost of Fujimoto's algorithm for shared memory processors. It is difficult to imagine a smaller cost cut algorithm that the setting of a global variable.

Table 5.1 compares the Seven O'clock algorithm to other widely known algorithms. The most important difference with the Seven O'clock algorithm is that it is the only known algorithm which is entirely independent of the available event

memory. Seven O'clock relies on a real-time clock frequency that allows us to begin to view event processing in terms of the frequency domain rather than the spatial domain. Each of the other GVT algorithms execute $GVTinterval * Batch$ events between GVT epochs. In our algorithm, we are allowed $NAOInterval$ time between GVT epochs, and so the number of events processed is indeterminate based on the interval frequency and speed of the individual CPU(s). While outside the scope of this paper, performing a spectral analysis is useful in determining the power of a given model. Spectral analysis may also lead to automatic filtering and compression of model spaces when computing design of experiments.

5.4 Performance Study

There are two benchmark models used in this performance study. The first is a synthetic workload model called PHOLD. This commonly used benchmark has been modified to support reverse-computation and is configured to have minimal Logical Process (LP) state, message sizes and event processing. The forward computation of events involves computing three random numbers: one for computing if a remote event should be created, one used to compute the time-stamp and one used for the destination LP. The reverse computation involves "un-doing" an LP's random number generator (RNG) in order to restore it's state. Because the RNG is perfectly reversible, the reverse computation restores seed state by computing the perfect inverse function as described in [1]. The destination LP is determined by calling a uniformly distributed random number generator in the range of 0 to 100. If the generated value is less than the specified percent of remote messages allowed, we choose the destination LP over an exponential distribution of all LPs, otherwise, the event will be sent to the source LP. The third call determines the offset timestamp for events and is exponentially distributed with a mean of 1.0, with the model completing at timestamp 100. For all experiment runs, we mapped LPs to Processing Elements (PE) in a round robin fashion. Each simulation run contained 1 million LPs, and the number of Kernel Processes (KP) was determined as $N_{cpu}^2 * multiplier$. The $multiplier$ was experimentally determined, as is explained in the next section. The message population per LP is 16. This model is a patholog-

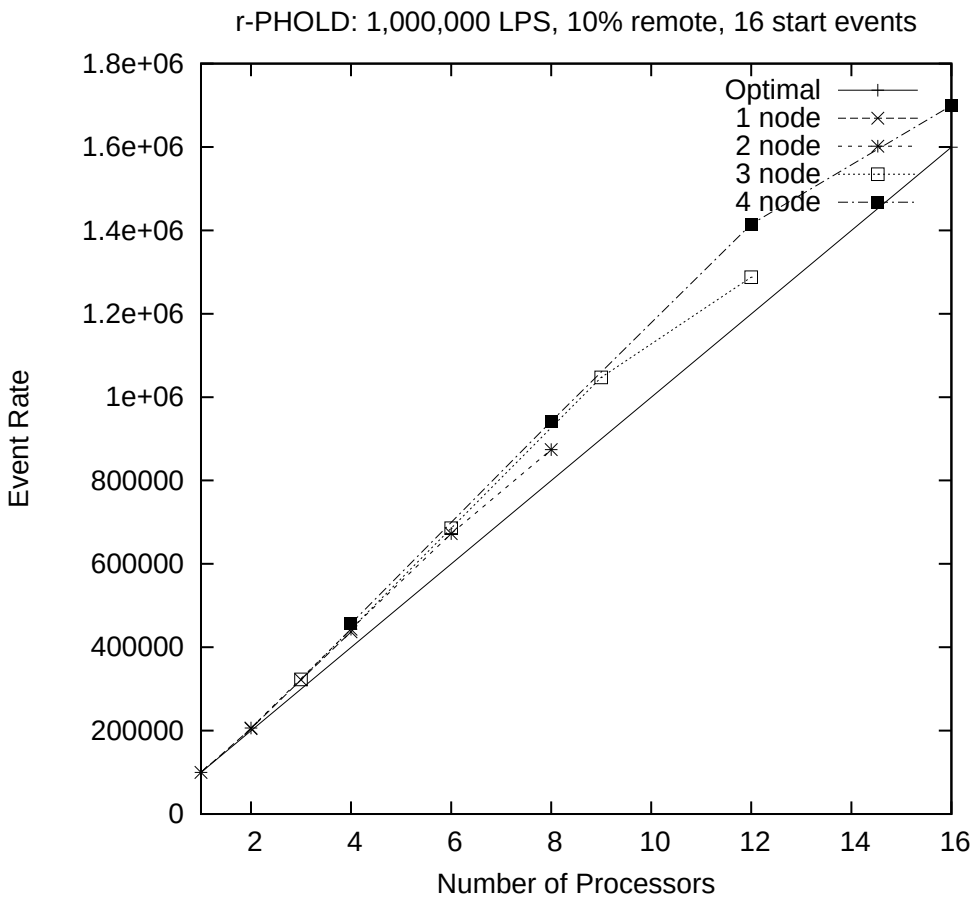


Figure 5.4: PHOLD results in the distributed cases with 1,2 or 3 processors utilized per node. The 4 processor cases are clearly affected by context-switching in the operating system for I/O operations.

ical benchmark which has minimal event granularity while producing a configurable number of remote events which can result in "thrashing" rollbacks. In [1], KPs where introduced as an aggregation structure for reducing LP fossil collection and rollback. One modification to this model was the ability to constrain the number of remote events created by the LPs. This allows us to analyze the system under varying workloads.

The second application is a model of TCP and this implementation follows the Tahoe specification [2]. There are three main data structures in this model: the data packet which is sent between hosts in the forwarding plane, the network router LPs which maintain queuing information and forward packets through the network and the host LPs which keep statistical information on the transferring of data. For

detailed model design and implemented, we refer the interested reader to [47].

For all TCP experiments we use the campus network defined in [14] which has been widely used to benchmark many network simulations in the past. We create a large-scale topology from this small network in the same approach used by Perumulla in [15], that connects multiple campus networks together to form a ring.

5.4.1 Computing Testbed and Experiment Setup

Initial results were collected on 4 quad-processor Itanium servers. The Itanium-2 processor [48] is a 64 bit architecture based on *Explicitly Parallel Computing (EPIC)* which intelligently bundles instructions together that are free of data, branch or control hazards. This approach enables up to 48 instructions to be in flight at any point in time. Current implementations employ a 6-wide, 8-stage deep pipeline. A single system can physically address up to 2^{50} bytes and has a full 64-bit virtual address capability. The L-3 cache comes in a 3 MBs configuration and can be accessed at 48 GBs/second which is the core bus speed. TCP over Gigabit Ethernet was used as the interconnection network.

The Netfinity cluster at RPI is a Red Hat Linux 9.0 cluster consisting of 40 machines, for a total of 80 processors. Each machines is a Symmetric Multi-Processor (SMP) machine with two 800MHz Pentium III processors. The 2 CPUs of each machine share 512 MB of RAM. The 40 SMP machines are connected to each other via a gigabit ethernet switch.

The Sith cluster computing platform at Georgia Tech is a Linux cluster consisting of 30 machines, for a total of 60 CPUs. Each machine is a Symmetric Multi-Processor (SMP) machine with two 900MHz Itanium processors. The 2 CPUs of each machine share 4 GB of RAM. The 30 SMP machines are connected to each other via a gigabit ethernet switch with EtherChannel aggregation.

All simulators have several input parameters which determine how major facilities such as the GVT computation and fossil collection are performed. For example, most GVT algorithms define $GVTInterval$ batch loops be performed between successive GVT computations. Determining the best values for any simulator is difficult because every model is different and it is difficult to know which settings yield the

best performance for a given model. Even within a single execution of a given model the performance of the model may change. For example, in the TCP model we present here, the TCP LPs which originate the data in the network have a much higher event granularity than do the internal IP routers which simply forward data traffic events through the network. The Seven O'clock algorithm increases this complexity because the time between GVT computations is not based on the number of events processed, but on the $NAOInterval$ alone. This means that between any two GVT computations a variable number of events will be processed.

In order to determine the settings that will yield the best for the PHOLD model, we use the Unified Search Framework (USF) [16] and the heuristic algorithm Random Recursive Search [12]. **By using Recursive Random Search (RRS) approach to design of experiments, we find: (i) that the number of simulation experiments that must be run is reduced by an order of magnitude when compared to full-factorial design approach, (ii) it allowed the elimination of unnecessary parameters, and (iii) it enabled the rapid understanding of key parameter interactions**. From this design of experiment approach, we were able to experimentally determine the settings which yielded the highest performance for the PHOLD model in a relatively small number of experiments. The best parameter settings for the TCP model were determined ad-hoc because of time limitations.

Optimistic event memory was computed in each case from the following formula: $OptimisticMemory = ceil(g\_tw\_nlp/g\_tw\_npe) * phold\_start\_events * multiplier$ where $multiplier$ was fixed at 2. Here, g_tw_npe is the number of processors used within the cluster configuration, and g_tw_nlp is the total number of LPs. During distributed execution, each processor consumes approximately, per GVT epoch, $NAOTime$ / $AvgTimeToComputeOneEvent$.

5.4.2 PHOLD Performance Data

5.4.2.1 Measuring Performance

In [23], we configured PHOLD with 10% remote messages, 16 seed events per LP and used the Myrinet network. Figure 5.4 illustrates super-linear results on a

NumCPU Voluntary CS	Involuntary CS	Total CS
2	0	12
3	0	14
4	46	1123

Table 5.2: PHOLD context switching in the OS degrades performance when the simulation "conflicts" with other system processes.

small cluster of 4 Itanium-II machines. Comparing the results in the parallel and distributed case is complicated because we have results based both on the number of processors used and based on the number of nodes used. We start by considering two nodes maximizing the processors before increasing the number of nodes. Then we consider three nodes, and finally four nodes.

Performance begins to degrade once a configuration becomes overly parallelized. In each node case, when we added the final processor per node, performance degraded due to context switching in the operating system for I/O operations. We informally measured the amount of context switches which occurred in each case and found that the four processor case typically generated greater than 100 times the number of context switches, as shown in Table 5.2.

Measuring speedup is an attempt to determine the cost of synchronization in a parallel and distributed environment. Typically Amdahl's law [101] and Gustafson's law [104] are applied by first performing the sequential simulation of the model, and then comparing each subsequent parallel execution to that. An examination of these laws is outside of the scope of this paper, but it is widely accepted that this laws may lead to confusion [102] or even abused depending on how they are used [103, 105, 106]. Our interest in introducing a new measurement is two-fold. We would certainly like to be open and honest about our performance results, and eliminate any possible sources confusion. Second, we would like to be able to easily compare against previous work.

In performing large-scale simulation it is not always possible to generate a sequential case for speedup comparison. One constraint of Amdahl's law is that the same instructions be processed for the sequential case, as are processed during the parallel case. One approach to overcome this problem is to use the results from

the smallest configuration possible as the basis for the sequential time. This is problematic because, while this minimizes the costs of parallelization, it does not reduce them entirely, leading to an under-estimation of optimal performance. The benefit of this approach however, is that the model being measured is exactly the model used to generate the sequential time.

In [15], the extrapolation was to generate a smaller model running on a single machine from the largest execution, and to "grow" those results up to the full model size. This approach leads to an over-estimation of the optimal because the model used to generate the sequential case is not the actual model being measured. While this is a highly accepted scientific method used for determining performance improvements, we show here how the physical nature of the hardware can fluctuate from those results, creating anomalies which are unexplainable by this method. These anomalies can only be explained by taking the actual model being measured into account when computing the sequential case. We note that while this approach was appropriate for the model in [15], it does not work in the general case.

Clearly a more precise approach is needed for measuring the performance of parallel and distributed systems. A new approach should take into account all of the particular details of the underlying hardware, and provide an optimal solution which cannot be superseded by a super-linear result. Besides context switching, other problems may arise such as memory bus overloading or serialized memory references [79, 78, 50], which limit the possible speedup due to parallelism. By maximizing the number of nodes used in a simulation it is possible to avoid or reduce these problems by de-coupling the hardware systems which are limiting performance, and this too must be taken into account when determining the optimal performance characteristics for a given model.

In the following sections we outline a general approach for measuring the optimal performance of a parallel and distributed simulation that considers not only the hardware used but also the model used, and the LP mapping of that model to the underlying hardware.

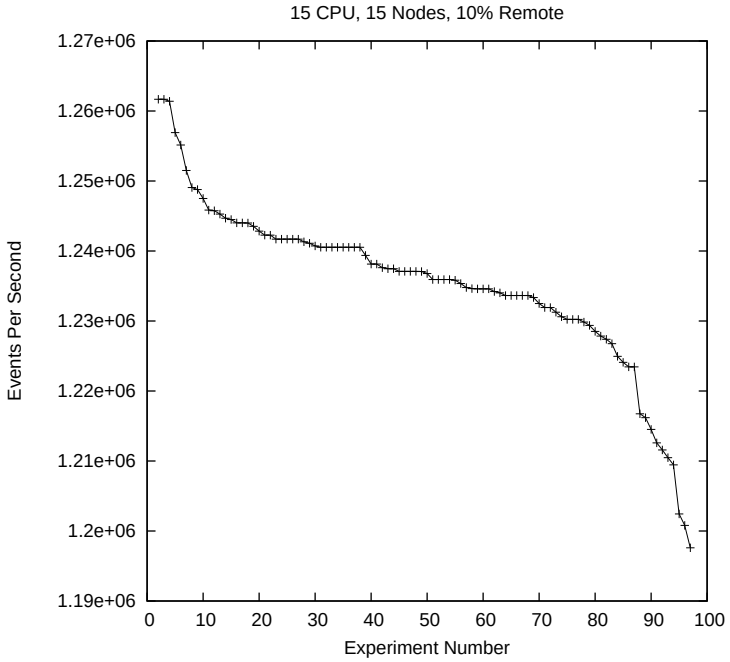


Figure 5.5: Simulator variation measured by design of experiments using Random Recursive Search.

Parameter	Min	Max	Step
Batch Loop Size	16	4096	32
NAO Interval (seconds)	0.01	0.50	0.01
Number of KPs	450	9000	2

Table 5.3: Input parameters to simulator used in design of experiments

5.4.2.2 Experiment Design

With several input parameters which affect the raw performance of the simulation, it is necessary to compute a design of experiments to quantitatively categorize the system under test. For this purpose, we utilized the Unified Search Framework (USF) [16] to create a design of experiments. Because we are only interested in the highest degree of performance, we used the Random Recursive Search algorithm [12] within USF to quickly determine the best settings for a particular configuration of the PHOLD model. The need for a design of experiments is illustrated in Figure 5.5. There is a variation of approximately 5% from the best to the worst experiment, varying the input parameters: NAO interval, size of the batch loop, and the number of KPs used. Table 5.3 shows the variance on the input parameters. These

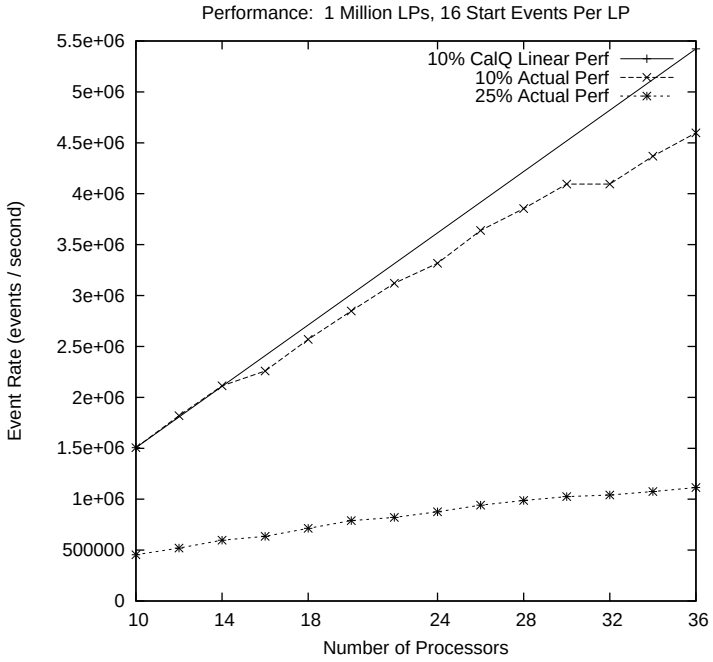


Figure 5.6: Results for 10% and 25% remote messages using increasingly more Netfinity nodes. Linear speedup for the 25% remote model was nearly identical.

input parameters were chosen because of their effect on the major simulator facilities, including GVT, event processing and fossil collection. The input parameter for the number of KPs is actually a multiplier and the number of KPs was selected as $N_{cpu}^2 * KP\_Multiplier$.

Selecting from the best input parameter settings, a full node configuration was computed using the Netfinity cluster. The best settings for 10% remote events were a batch size of 2531, and NAO of 0.36 seconds and a KP multiplier of 2. The best settings for 25% remote events were a batch size of 886, an NAO of 0.24 seconds and a KP multiplier of 12. The 25% model response was also approximately 5%. Results shown in Figure 5.6 indicate excellent linear performance for 10 and 25% remote events. Super-linear performance was achieved on the Itanium cluster primarily because of the highly optimized memory hardware and because we were able to generate a "true" sequential execution time. Linear performance on the Netfinity cluster was determined by using the LP mapping for the 36 node, 10% remote event, case, and extrapolating what the sequential case would have been. This is an

accepted scientifically accepted approach for calculating linear performance speedup, but it should be noted that this approach over-states the potential speedup possible because the model used for the sequential case is not the same model being measured.

5.4.2.3 Measuring Optimal Performance: Distributed

Computing the sequential case for performance comparison is not always possible for large-scale models. The purpose of computing the model sequentially is to determine the optimal performance without overhead due to parallelization. In this section we outline a novel approach for determining the optimal performance of the simulator software given a specific model, hardware configuration and model LP mapping to the hardware.

Whenever we make improvements to simulator engine facilities, we are attempting to improve performance. At some point, there must exist an *achievable* optimal performance beyond which further improvements to the software would not be able to generate additional speedup. In the past others have determined a sequential case by running large-scale models on the minimum hardware required. But this approach understates the optimal case because it contains code necessary for resolving contention and communication issues. In addition, this approach also leads to super linear performance when enough CPU L2 caches are added to the configuration. Super linear performance occurs because as more compute nodes are added to the hardware configuration, more of the model will fit into the L2 cache, and a speedup is gained by the nature of the hardware.

Another problem in determining a base case for performance is in the mapping of the model to simulator entities (PEs or KPs). Radically different performance can be measured within the same model when LPs are mapped differently to PEs because the mapping largely determines the communication and contention load in the system.

To overcome these problems, we asked ourselves if there was a way in which we could generate a sequential case over all of the CPUs used per simulation run. In fact, we wanted to be able to run one instance of the sequential scheduler within ROSS per CPU. In order to do this however, LPs could no longer send events to LPs

scheduled on remote CPUs. In order to implement this approach, it was necessary for us to be able to modify the models in such a way that this did not occur. Using this approach we were able to take into consideration the hardware configuration, and the specific model and its mapping of LPs to the hardware.

To determine the optimal performance for the Netfinity cluster, we simply re-compute the PHOLD model, but this time each event's destination LP is always the sending LP. When we constrain the model in this way we no longer require the parallel scheduler, since we know there will be no contention issues. So, we replace the parallel scheduler with the sequential scheduler and re-execute the model for the desired node configurations. It is important to note that by using the sequential scheduler, we eliminate simulator facilities such as the GVT computation and fossil collection. Also, because each event is now sent in "loopback" mode, the network connecting the nodes is not utilized during the simulation. A key observation is that the model continues to commit the same instructions and events that will be committed during normal execution. For example, even though each event in the optimal case is sent in loopback mode, we continue to compute the destination LP. This means that we continue to execute the same lines of code as in the model being measured. Comparing the PHOLD model used to determine the optimal performance to the PHOLD model measured, there was a difference of approximately 3,000 events out of over 161 million events computed. This represents about 2.2e-5% of variance, but more importantly, the amount of work computed by the model can be considered equal.

Figure 5.7 shows the actual results plotted against both the Linear Speedup curve as well as the new Optimal Performance curve. This is done to illustrate the difference between the two measurements. This figure illustrates the problem with computing speedup in the typical way. Clearly the linear performance overstates the actual performance gained because it could not take into consideration this model's LP mapping and the hardware used.

We believe that our approach to measuring performance for large-scale models more clearly illustrates the performance of the simulation executive software because it takes into account not only the model, but also the underlying hardware

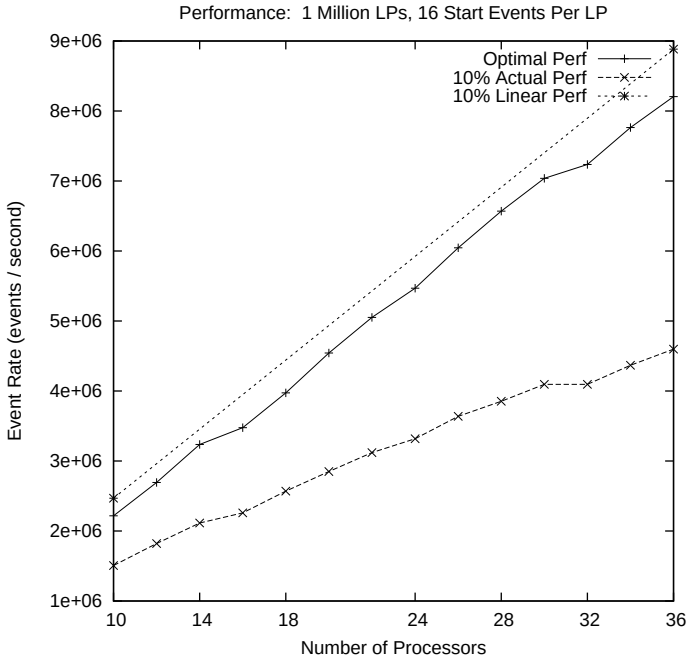


Figure 5.7: Results for PHOLD configured with 10% remote events, compared to the Optimal Performance possible for the available hardware. The linear performance curve remains for comparison.

limitations. For example, as the number of nodes used increases, the model should perform increasingly better because more and more of the model fits into the L2 caches. The optimal performance measurement accounts for this behavior, while the linear speedup curve does not. To properly use the old measurement also requires computing the sequential case *for each cluster configuration*, otherwise the performance for the smaller configurations will be under-stated. The optimal performance measurement is correct for each cluster configuration and is therefore a more realistic measurement of the model for the given hardware and mapping. Most importantly, the optimal performance measurement is only *theoretically achievable* and super-linear results beyond this are unlikely because even if overhead due to parallelization were eliminated entirely, you would be left with the optimal case.

The performance curves illustrated in Figure 5.7 clearly dip slightly around 14 nodes used, and again at 32 nodes. We changed the machines used to collect these results several times and found that the curves overlapped with less than 0.1%

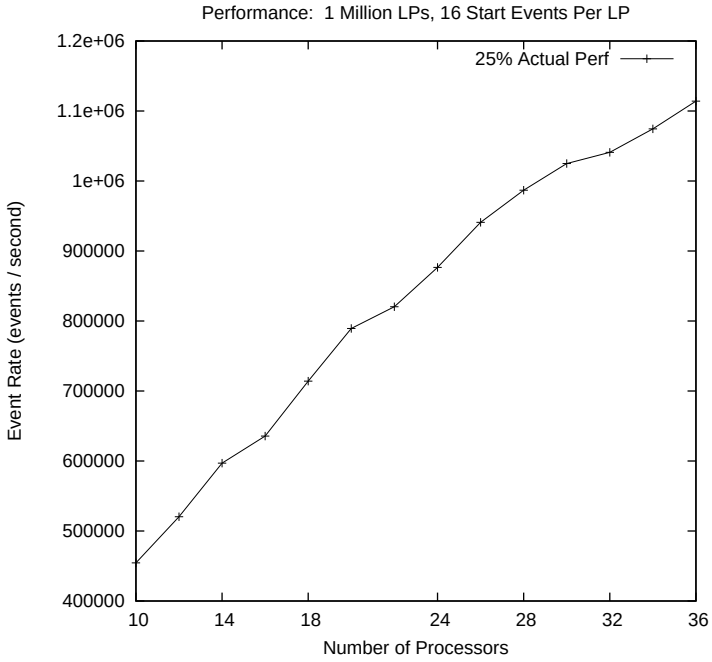


Figure 5.8: We see the same effects of the LP mapping in the 25% remote event PHOLD model. (The optimal performance curve is not shown due to scaling.)

variation. If the performance is not dropping due to the underlying hardware or network used, then it must be the model which is causing the performance anomaly. In addition, we can rule out major simulator facilities such as GVT or fossil collection as the cause because they do not exist in the optimal performance curve. The LP mapping must be the cause for these irregularities because it is the only constant for both curves which could account for the observed behavior. This is an important distinction because this clearly illustrates not only what the optimal performance should be for the given model and hardware, but also given the way in which the model is mapped for parallelization. Figure 5.8 shows that PHOLD with 25% remote messages contain the same anomaly, because we have maintained the LP mapping. In order to clearly (and honestly) represent performance results, it is necessary to plot the optimal or linear curves to produce a proper scaling. The results in Figure 5.8 do *not* illustrate good linear improvement. The improvement for 36 nodes is only 2 times that of 10 nodes. We address this special case in the subsequent section.

Clearly one limitation to this approach is that it may not be possible to gen-

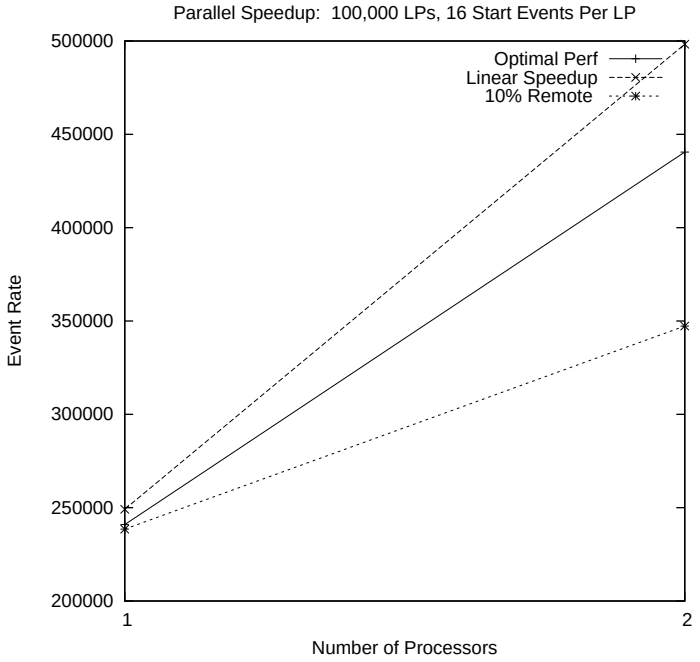


Figure 5.9: For 100,000 LPs, ROSS' parallel scheduler generates a speedup of 1.4. However, when we compare to the measure of optimal performance, we see that we are within 25% of optimal.

erate the same critical path workload distributed across multiple sequential schedulers for every model. However, if we want to determine as honestly as possible the speedup achieved by improvements within the software via either engine improvements or LP mappings, it is clear that this approach is viable when possible.

5.4.2.4 Optimal Performance: Parallel and Distributed

In [23] we observed a performance drop whenever the last CPU on a cluster node was mapped for the simulation. We have identified the cause of this behavior to be a high overhead due to context switching within the OS for I/O operations. In order to quantify the amount of loss due to context switching requires that we look at the speedup available in parallelization alone.

Figure 5.9 shows both the linear speedup and optimal performance achievable on a single Netfinity cluster node. This illustrates that linear speedup is not achievable through software alone on this hardware. This is likely due to limitations on the memory bus between the two processors, as we have measured in the past [1].

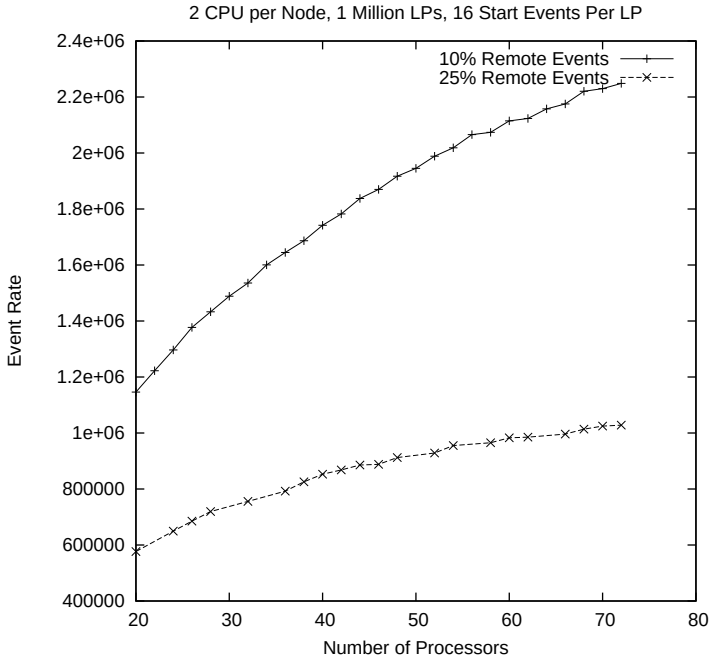


Figure 5.10: Results for PHOLD on the Netfinity cluster. A speedup of approximately 2 is achieved from 20 to 72 processors.

It is helpful to know what speedup the hardware will allow when measuring the performance of the simulator software.

Taking the hardware into consideration, ROSS performs within 25% of the optimal performance achievable through software alone. Figure 5.10 shows the distributed and parallel results remain far below this level for 2 CPU per node configurations. The primary reason for the poor performance on the Netfinity cluster is the context switching overhead introduced to handle the network I/O operations, and not attributable to ROSS. Figure 5.4 does not indicate the same slowdown until we add the fourth and final processor to the configuration.

In this section we have shown how measuring linear performance is imprecise for large-scale distributed simulation because it is difficult if not impossible to maintain the hardware restrictions for fair comparison or to compute the costs of associated with the hardware. We have proposed a new measurement called Optimal Performance, and shown how to collect the result. Optimal performance measures exactly how fast the given hardware can execute events for a given model. This new measurement formalizes the performance level of the actual simulator performance,

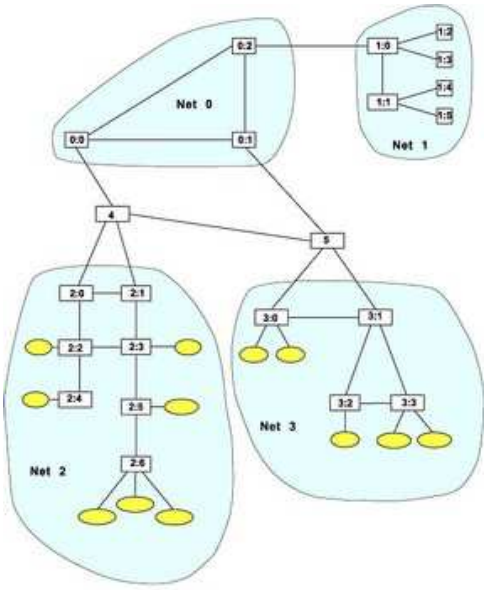


Figure 5.11: Campus Network [14]

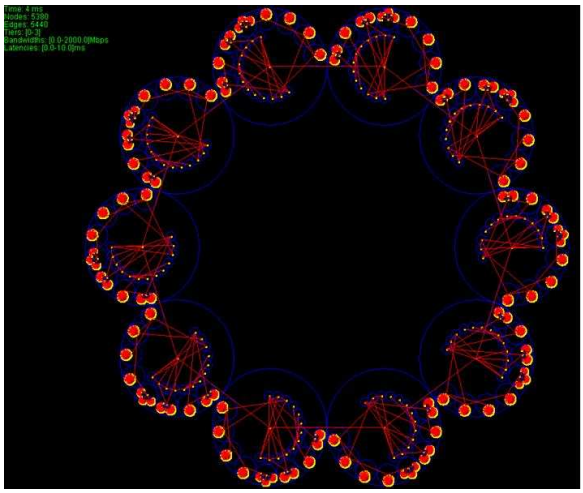


Figure 5.12: Ring of 10 Campus Networks. [15]

regardless of model size or LP mapping. Intuitively, simulator performance should not exceed the optimal case. We believe that this measurement will greatly simplify performance analysis of parallel and distributed simulation.

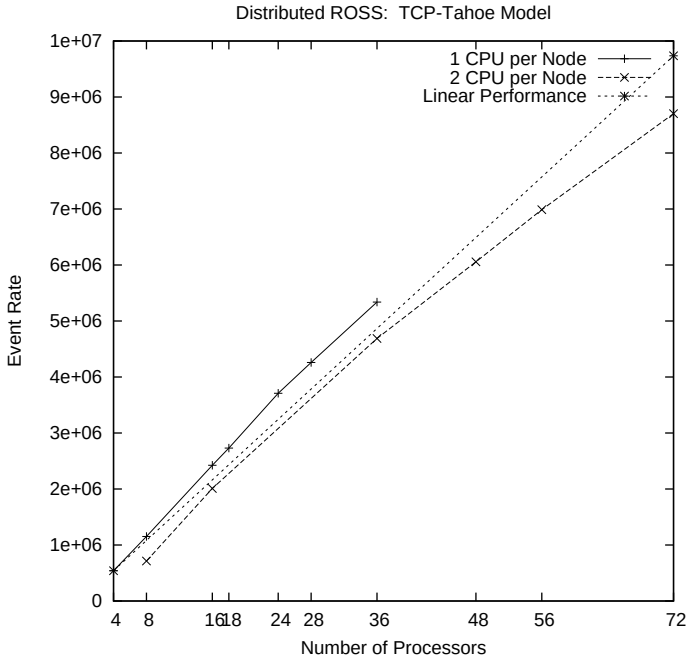


Figure 5.13: TCP model performance improves linearly, exhibiting a super-linear speedup. Comparing to the optimal performance leads to a higher quality analysis of simulator performance.

5.4.3 TCP Performance Data

5.4.3.1 TCP Model Comparison

Figure 5.11 illustrates the campus network, widely used to benchmark many network simulators [15, 17, 19, 20, 18] and is an interesting topology for network experimentation [21]. The campus network is comprised of 4 TCP servers, 30 routers in 4 LANs, and 504 TCP clients for a total of 538 nodes [14]. Limitations in our TCP model caused us to have 504 HTTP servers, one to serve each client. In addition, in order to maintain proper queue statistics at the client links, it was necessary to add an aggregation router per 10 or 12 client nodes. Our equivalent campus resulted in 1082 node LPs. Because the networks are equivalent, we refer to our campus network model as having 538 nodes for ease of comparison.

The campus network is comprised of 4 different networks. Network 0 consists of 3 routers, where node zero is the gateway router for the campus network. Network 1 is composed of 2 routers and the servers. Network 2 comprised of 7 routers, 7

Nodes Used	ROSS	PDNS
2	341853	n/a
4	730720	n/a
8	1493702	n/a
16	2817954	in swap
32	5508404	1069905

Table 5.4: Performance results measured for ROSS and PDNS for a ring of 256 campus networks. Only one processor was used on each computing node. Rates are packets executed per second.

LAN routers, and 294 clients. Network 3 contains 4 routers, 5 LAN routers, and 210 clients [15]. All router links have a bandwidth of 2Gb/s and have a propagation delay of 5ms with the exception of the network 0 to network 1 links, which have a delay of 1ms. The clients are connected to their LAN router with links of bandwidth 100Mb/s and 1ms delay.

For our experiments we connected multiple campus networks together at their gateway routers to form a ring. The links connecting the campuses together were 2Gb/s with delays of 200ms. Figure 5.12 shows a ring of 10 of these campuses connected. The traffic was comprised of TCP clients in one domain connecting to the server in the next domain in the ring. The server would transfer 500,000 bytes back to the client application. This approach replicates the network created for PDNS by Kalyan Permulla for [15]. We connected 1,008 campus networks together to create a network of size 542,304.

Figure 5.13 shows a super-linear speedup on the one processor case per node. This can be attributed to the decrease in context switching from the two processor per node case. This is a similar result to what was observed in the PHOLD results presented in [23].

We attempted to reproduce a model of similar size in PDNS on our Netfinity cluster but the memory requirements where too high for only 40 machines. The largest PDNS model that we could get to run was 137,728 LPs using 32 cluster nodes. We were able to execute this model on one node in ROSS and the results and performance numbers can be seem in Table 5.4. ROSS was able to achieve 5.5 million packets per second whereas PDNS processed 1.07 million packets per second. This

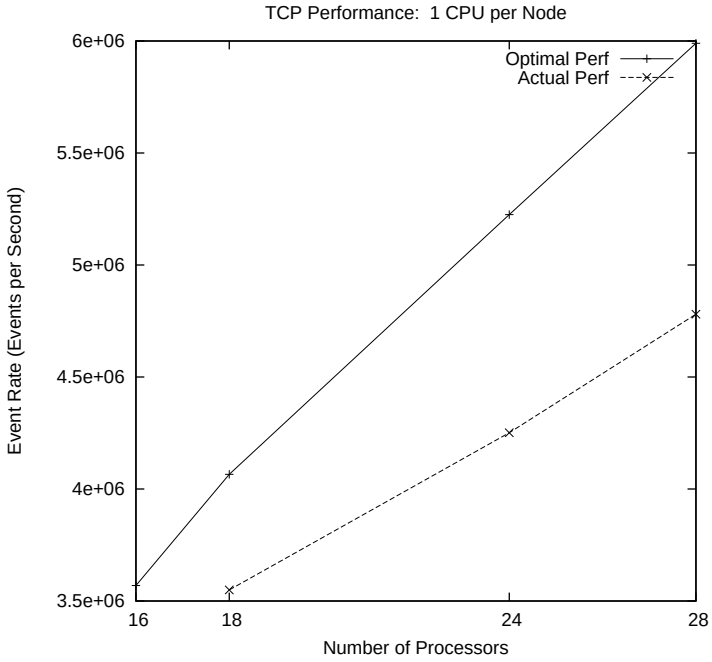


Figure 5.14: TCP model performance improves linearly. Optimal performance achieves a speedup of about 1.6, and the actual performance about 1.35.

shows that our ROSS implementation has a significantly smaller memory footprint and achieves 5.14 times in speedup over PDNS.

5.4.3.2 Optimal Performance

In order to measure the optimal performance of the TCP model, we connected each TCP client to a server within the local campus, rather than a remote campus within the ring. This allowed us to run the TCP model in the expected configuration without changing the execution or mapping of the model.

We measured the TCP model's optimal performance using the Sith Itanium cluster at Georgia Tech. Comparing TCP performance to optimal performance indicates that ROSS performance is not in fact super-linear when the hardware and model mapping is taken into account. Figure 5.14 shows that the actual TCP model performance achieves a speedup of approximately 1.35, compared to a possibly 1.6 in optimal performance. Similarly, Figure 5.15 shows that for two CPUs utilized per node, the TCP model achieves a speedup of 2.5 compared to a possible 3.3 optimal.

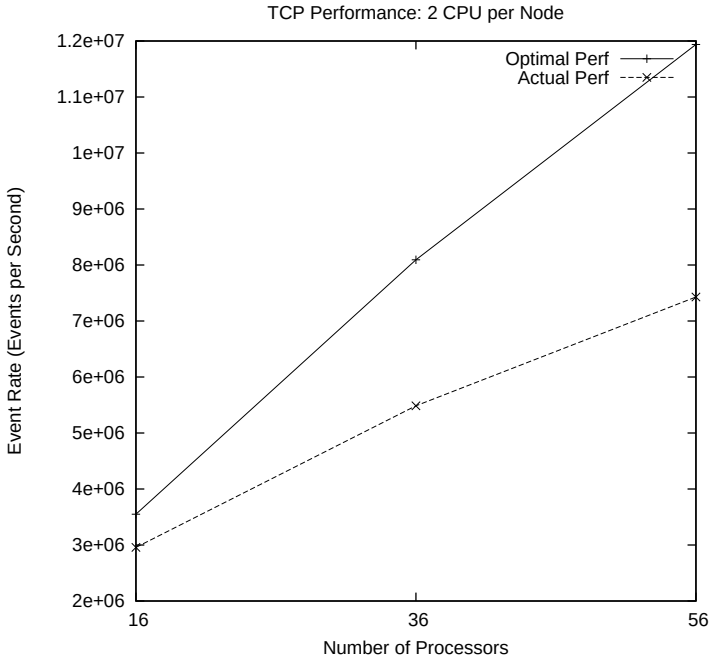


Figure 5.15: TCP model performance improves linearly. Optimal performance achieves a speedup of about 3.3, and the actual performance about 2.5.

5.5 Related Work

Samadi's algorithm [63] solves the simultaneous reporting problem through the use of message acknowledgments. Here, processors will tag any "ack" message that it sends in the period from when the processor reports its LVT until it receives the new GVT value. This process prevents any messages from "slipping through the cracks" [73].

In addition to Fujimoto's and Mattern's GVT algorithms, there have been a number of predecessors. Preiss [64] introduced a scheme which places the PEs into a ring. The first round completes when a token has been passed around the ring and returns to the initiating PE. When the initial token is received, a PE begins accounting for remotely sent messages which may or may not complete prior to the GVT computation. On receiving the second token, a local GVT value is computed as the minimum between the value stored in the token and the PEs local minimum. Improvements to this algorithm have been proposed by Bellenot [65] which reduce the complexity of message passing by organizing PEs into trees rather than a ring.

Lin and Lazowska [72] propose a new data structure that reduces the frequency of acknowledgment messages. The work done by Tomlinson and Garg [66] uses counters to detect transient messages. However, this scheme does not employ the use of a "cut" as Mattern's algorithm does. Pancerella [67] propose a hardware based scheme whereby host systems using custom network interface cards are interconnected to form an efficient reduction network to rapidly compute GVT.

In follow-up research, Lin [68] uses control messages to efficiently find/flush out transient messages. At about the same time, D'Souza et. al [69], proposes a statistical approach to estimating GVT.

Prior to Fujimoto's GVT algorithm, Xiao et al. [70], proposes an asynchronous GVT algorithm that exploits shared-memory multiprocessor architectures. Here the concept of "cages" is used, where each processor own some set of "cages" that it places or lays down in order to track the LVT of that processor. It is interesting to note, that here a global *Cage Flag* much like the `gvt_flag` in Fujimoto's Algorithm, is used to "kick off" the GVT computation.

Related to both "consistent cuts" and sequentially consistent memory is the issue of causality. Most recently, Zhou et al [71], propose a "relaxed" causal receive ordering algorithm called *critical causality* for distributed virtual environments.

5.6 Summary

We present a new idea, Network Atomic Operations, and apply it to solving the global virtual time problem. We use an NAO to generate a zero-cost cut in a distributed system which is both scalable and efficient. Our experimental results indicate a super-linear speedup over sequential. We also suggest that there are two forms of speedup, one due to parallelization and one due to distribution. The parallel scheduler results were already super-linear, however there is more speedup inherent to distribution because we eliminate parallelism bottlenecks such as the memory bus and context switching. In fact, our results indicate super-linear speedup is possible beyond the amount of speedup due to parallelism.

NAOs lead us to viewing the behavior of Time Warp in the frequency domain because they define the relationship between virtual time and wall-clock time. In the

future, it would be interesting to investigate maximum rollback lengths using NAOs. Computing a spectral analysis of the Time Warp system should allow analysis of the system which would detect when the forward processing of events is optimal. When we quantify the system in this manner, it becomes possible to derive a proof suggesting (assuming the data supports our theory) that optimistic simulation can consistently perform as well as, if not better than, conservative simulation.

This work is important because it allows us to extend the simulation platform to the cluster computing environment. The Seven O'Clock algorithm is the first example of a scalable GVT algorithm which allows for the first time super-linear results without a loss of model clarity. This is important because it allows us to create large-scale network simulations without the loss of individual packet granularity.

CHAPTER 6
Discussions and Conclusions

In this thesis a new parallel and distributed discrete event simulator is introduced. Innovative ideas such as kernel processes, network atomic operations and a new GVT algorithm, Seven O'Clock, are combined into a single system to provide a highly efficient, high performance, low memory system. To date, ROSS continues to be a state of the art Time Warp system with unparalleled performance.

Also introduced in this thesis is a new tool for the meta-simulation of internet scale networks: ROSS.Net. ROSS.Net aims to bring together the four major areas of network research: simulation, protocol design, network modeling and measurement, and experiment design and analysis. ROSS.Net employs such search algorithms as RRS to "provide good results fast".

ROSS introduces kernel processes as a new construct for aggregating the high cost of fossil collecting memory in a Time Warp simulator. The trade-off is potentially longer rollbacks. The performance analysis for kernel processes shows that overhead for rollbacks was dramatically reduced which allowed the simulation engine to scale properly. Also, overhead due to longer rollbacks was not observed primarily due to better use of the memory subsystem which resulted in a more stable system. In addition, for the first time ROSS showed that an optimistic simulator could run efficiently in a small amount of memory beyond the needs of sequential execution.

Using network atomic operations we were able to create a zero-cost consistent cut and the first truly scalable GVT algorithm for the Time Warp protocol. This advance allows ROSS to scale to millions of logical processes across large cluster computing platforms. In our performance analysis we presented super-linear results for the benchmark model rPHOLD, and linear results for our TCP model. We presented results on a variety of cluster environments ranging from 16 to 80 processors.

With ROSS.Net we take the first steps towards a scalable integration framework capable of simulating traffic from multiple network protocol models. We integrated protocol models with a measurement topology data, namely, the Rocketfuel

description of the internet. For the first time realistic million node topologies were presented and simulated in high detail.

ROSS.Net incorporates meta-simulation components of the Unified Search Framework and Random Recursive Search algorithm. We presented a design of experiments of the OSPFv2 protocol for analysis of the performance of the system. This design investigated the OSPFv2 convergence response plane for tens of thousands of routers configured within a single OSPF area. We compared a full factorial design generating 16,807 experiment runs, 20 times more than the RRS design. The full factorial design yielded less information in the areas that we were interested in studying. RRS also generated a "good" value for a convergence time of 0.11 seconds, which was within 7% of the Full Factorial design best value. In particular, we identified the main parameters affecting convergence to be the HELLO protocol parameters and created a model of these parameters and their effect on the response.

Most recently, we have completed integration of a BGPv4 model into ROSS.Net which allows us to apply our meta-simulation techniques to even larger and more complex network scenarios. The BGP network protocol allows us to expand our investigation into multiple autonomous systems and OSPF areas. We presented a design of experiments to investigate feature interactions between OSPFv2 and BGP4. Our response plane focused on the effects of OSPF-caused updates on BGP, and BGP-caused updates on the OSPF protocol in the control plane. To date, work in this area has primarily been conducted using measurement data. Using ROSS.Net, we are able to investigate a much larger parameter space and expand the scope of the model produced.

Our first design created a high level characterization of the effects of feature interactions across multiple (5) ASes running BGP between each AS, and OSPF within each AS. Parameters were classified according the the protocols with which they were associated and a complete investigation of each classes effects on the response was studied. This design quantified the effects of each parameter class on the response.

The second design investigated directly the effects of varying network management policies. Each policy was optimized to determine the overall effects on the

response. Each policy was compared to the "global" policy to ascertain the effectiveness of the optimization. It was determined that optimizing the BGP domain had little quantifiable effect on reducing the number of interaction update messages in the system. However, optimizing the OSPF domain, and more specifically, to reduce OSPF caused BGP updates yielded much better results in reducing the number of interaction updates in the system.

Now that we have a validation that the OSPF domain adversely impacts the BGP domain, we can begin to focus our experiments on the hypothesized cause of the interruptions. Design 3 investigated the effects of cold versus hot potato routing. In this design we perform a simple full-factorial of RRS optimizations, turning Hot and Cold Potato routing on and off within the BGP decision algorithm. Our work shows that while these two policies do have an impact on the response, they did not account for a large proportion of the response. Our quantification suggested that effort would be better placed in reducing the effects of LOCAL-PREF and AS Path Padding.

Our fourth and final experiment design investigated the effects of network robustness on the response plane. We examined varying degrees of link status failures and OSPF link weight changes. From the results we see that the response is independent of the link weight changes. This was surprising since aggressive link weight policies are known to produce routing loops among other problems. While aggressive changes impacted the OSPF domain internally, those updates did not appear to generate subsequent updates in the BGP domain. Link status changes had a much greater impact on the OSPF caused BGP updates however, because they directly impacted the iBGP connections. Our results showed that the response could more than double when link status changes ranged from 1 to 15%. We also were able to quantify the amount of improvement generated by the heuristic search to be generally between 10 and 35% over the average configurations.

LITERATURE CITED

[1] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: A High-performance, Low Memory, Modular Time Warp System", *Journal of Parallel and Distributed Computing*, 2002.

[2] K. Fall and S. Floyd, "Simulation-based comparison of Tahoe, Reno, and Sack TCP", *Computer Communication Review*, vol. 26, pp. 5–21, 1996.

[3] K. S. Permalla C. D. Carothers and R. M. Fujimoto, "The Effect of State-saving on a Cache Coherent, Non-uniform Memory Access Architecture", *Proceedings of the 1999 Winter Simulation Conference (WSC'99)*, December 1999.

[4] R. Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling", Wiley - Interscience, 1991.

[5] S. Floyd and V. Paxson, "Difficulties in simulating the internet", *IEEE/ACM Transactions on Networking*, February 2001.

[6] S. Floyd, "Simulation is crucial", *IEEE Spectrum*, January 2001.

[7] "SSFnet",`http://www.ssfnet.org`.

[8] "Glomosim",`http://pcl.cs.ucla.edu/projects/glomosim`.

[9] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router", *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[10] Mark Handley, Orion Hodson, and Eddie Kohler, "XORP: Open platforms for network research", *First Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.

[11] D. C. Montgomery, *Design and Analysis of Experiments*, John Wiley and Sons, 2001.

[12] T. Ye and S. Kalyanaraman, "A recursive random search algorithm for large-scale network parameter configuration", *Proceedings of ACM SIGMETRICS (part of FCRC)*, 2003.

[13] A. Helmy, D. Estrin, and S. Gupta, "Systematic testing of multicast routing protocols: Analysis of forward and backward search techniques", *International Conference on Computer Communications and Networks (ICCCN)*, 2000.

[14] J. Liu, NMS (Network Modeling and Simulation DARPA Program) baseline model. See web site at `http://www.crhc.uiuc.edu/~jasonliu/projects/ssfnet/dmlintro/baseline-dml.htm`

[15] R. M. Fujimoto, K. S. Perumalla, A. Park, H. Wu, M. H. Ammar, G. F. Riley. "Large-Scale Network Simulation: How Big? How Fast?", MASCOTS 2003.

[16] T. Ye and S. Kalyanaraman, "A Unified Search Framework for Large-scale Black-box Optimization", Rensselaer Polytechnic Institute, ECSE Department, Networks Lab, 2003.

[17] Y. Liu, B. K. Szymansk, "Distributed Packet-Level Simulation for BGP Networks under Genesis", *Proc. Summer Computer Simulation Conference*, SCS Press, San Diego, CA, July 2004, pp. 271-278.

[18] B. Szymanski, Y. Liu and R. Gupta. "Parallel network simulation under distributed Genesis", *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, June 2003.

[19] G Riley, Mostafa Ammar, Richard Fujimoto, Alfred Park, Kalyan Perumalla and Donghua Xu, "Federated Approach to Distributed Network Simulation.", *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 14, No. 2, April 2004.

[20] G. F. Riley. "Large-scale network simulations with GTNetS". *Proceedings of the 2003 Winter Simulation Conference*, pages 676-684, 2003.

[21] J. J. Farris, D. M. Nicol, "Evaluation of secure peer-to-peer overlay routing for survivable scada system", *Proceedings of the 2004 Winter Simulation Conference*, 2004.

[22] D. M. Nicol and G. Yan, "Simulation of Network Traffic at Coarse Timescales", *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 141–150, 2005.

[23] D. Bauer, G. Yaun, C.D. Carothers, M. Yuksel, and S. Kalyanaraman, "Seven-O'Clock: A New Distributed GVT Algorithm Using Network Atomic Operations", *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS '05)*, 2005.

[24] Brian White, Jay Lepreau, et. al., "An integrated experimental environment for distributed systems and networks", *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, Dec. 2002, USENIX Association, pp. 255–270.

[25] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe, "A blueprint for introducing disruptive technology into the internet", *First*

Workshop on Hot Topics in Networks (HotNets-I), October 2002, `http://www.planet-lab.org`.

[26] "Caida – the cooperative association for internet data analysis", `http://www.caida.org`.

[27] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel", *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2002.

[28] "Internet research needs better models", *First Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.

[29] G. Taguchi, *Introduction to Quality Engineering*, Asian Productivity Organization, UNIPUB, White Plains, NY, 1986.

[30] Aimo Törn and Antanas Žilinskas, *Global Optimization*, vol. 350 of *Lecture Notes in Computer Science*, Springer-Verlag, 1989.

[31] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, Springer, 1999.

[32] R. Ostrovsky and B. Patt-Shamir, "Optimal and efficient clock synchronization under drifting clocks",In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*, pp. 3 - 12, 1999.

[33] Aimo Törn and Antanas Žilinskas, *Global Optimization*, vol. 350 of *Lecture Notes in Computer Science*, Springer-Verlag, 1989.

[34] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, MA: Addison Wesley, 1989.

[35] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, 1989.

[36] Nicholas J. Radcliffe and Patrick D. Surry, "Fundamental limitations on search algorithms: Evolutionary computing in perspective", *Computer Science Today*, pp. 275–291. 1995.

[37] D. h. Wolpert and W. G. Macready, "No free lunch theorems for optimization", *IEEE Transaction on Evolutionary Computing*, vol. 1, pp. 67–82, 1997.

[38] D. M. Nicol and J. Liu, "Composite synchronization in parallel discrete-event simulation", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 5, 2002.

[39] "Javasim",http://javasim.cs.uiuc.edu, 1999.

[40] "UCB/LBLN/VINT network simulator - ns (version 2)", http://www-mash.cs.berkeley.edu/ns, 1997.

[41] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, "A generic framework for parallelization of network simulations", *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 1999, pp. 128–135.

[42] B. K. Szymanski, A. Saifee, A. Sastry, Y. Liu, and K. Madnani, "Genesis: A system for large-scale parallel network simulation", *Proceedings of Workshop on Parallel and Distributed Simulation (PADS '02)*, 2002, pp. 89–96.

[43] "Staged network simulator - sns", http://www.cs.cornell.edu/People/egs/sns, 2003.

[44] D. Nicol, "Scalability of network simulators revisited", *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS) part of Western Multi-Conference (WMC)*, 2003.

[45] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation", *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, 1999.

[46] C. D. Carothers, K. Perumalla, and R. M. Fujimoto, "Efficient parallel simulation using reverse computation", *ACM Transactions on Modeling and Computer Simulation*, vol. 9, no. 3, pp. 224–253, July 1999.

[47] G. Yaun, C. Carothers, and S. Kalyanaraman, "Large-scale tcp models using optimistic parallel simulation", *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS) (part of FCRC)*, 2003.

[48] Intel, "Intel Itanium-II Reference Manuals", Available via the web at: http://www.intel.com/design/itanium/documentation.htm?iid=ipp_srvr_proc_itanium2+techdocs&

[49] Intel, "Intel Pentium 4 and Xeon Processor Optimization Reference Manual",Available via the web at: http://developer.intel.com/design/pentium4/manuals/248966.htm

[50] D. Nagle, R. Uhlig, T. Stanley, S. Sechrest, T. N. Mudge, and R. B. Brown, "Design tradeoffs for software-managed TLBs",in *ISCA*, pp. 27–38, 1993.

[51] G. R. Yaun, H. L. Bhutada, D. Bauer, C. D. Carothers, M. Yuksel, and S. Kalyanaraman, "Large-scale network simulation techniques: Examples of tcp and ospf models", *ACM SIGCOMM Computer Communication Review*, 2003.

[52] "Rocketfuel internet topology database", http://www.cs.washington.edu/research/networking/rocketfuel.

[53] C. Alaettinoglu, V. Jacobson, and H. Yu, "Towards millisecond igp convergence", *NANOG*, October 2000.

[54] G. R. Yaun, D. Bauer, H.L. Bhutada, C. D. Carothers, M.Yuksel, and S. Kalyanaraman, "Large-scale network simulation techniques: Examples of tcp and ospf models", *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 33, no. 3, 2003.

[55] Jiang Li and Shivkumar Kalyanaraman, "Ormcc : A simple and effective single-rate multicast congestion control scheme", *Submitted*, `http://www.cs.rpi.edu/\~~lij6/Research/index.html`, 2002.

[56] Tao Ye, *Large-scale Network Parameter Configuration Using On-line Simulation Framework*, Ph.D. thesis, Rensslaer Polytechnique Institute, May 2003.

[57] Y. Rekhter and T. Li. "A Border Gateway Protocol 4 (BGP-4)", *IETF RFC 1771*, March 1995.

[58] J. Moy. "OSPF version 2", *IETF RFC 2328*, April 1998.

[59] J. Stewart III, *BGP-4 Inter-domain routing in the Internet*, Addison-Wesley, 1999.

[60] "The r project for statistical computing",http://www.r-project.org.

[61] O. Berry and D. R. Jefferson, "Critical path analysis of distributed simulation", *Proc. 1985 SCS Multiconference on Distributed Simulation*, January 1985, pp. 57–60.

[62] M. Goyal, K. K. Ramakrishnan, and W. Feng, "Achieving faster failure detection in ospf networks", *Proceedings of the Internation Conference on Communications (ICC)*, 2003.

[63] B. Samadi. "Distributed Simulation, Algorithms and Performance Analysis", Ph. D. Thesis, University of California, Los Angeles (UCLA), 1985.

[64] B. R. Preiss, "The Yaddes Distributed Discrete Event Simulation Specification Language and Execution Environments", In *Proceedings of the SCS Multiconference on Distributed Simulation*, vol. 21, March, pp. 139–144, 1989.

[65] S. Bellenot, "Global Virtual Time Algorithms", In *Proceedings of theSCS Multiconference on Distributed Simulation*, Vol. 22, pp. 122–127, 1990.

[66] A. Tomlinson and V. K. Gang, "An Algorithm for Minimally Latent Global Virtual Time", In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93)*, pp. 35–42, 1993.

[67] C. M. Pancerella, and P. F. Reynolds, "Disseminating Critical Target-Specific Synchronization Information in Parallel Discrete-Event Simulation", In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93)*, pp. 52–59, 1993.

[68] Y-B. Lin, "Determining the Global Progress of Parallel Simulation with FIFO Communication Property", *Information Processing Letters*, 50, pp. 13–17, 1994.

[69] L. M. D'Souza, X. Fan, and P. A. Wilsey, "pGVT: An Algorithm for Accurate GVT Estimation", In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, pp. 102–109, 1994.

[70] Z. Xiao, J. Cleary, F. Gomes, and B. Unger. "A Fast Asynchronous Continuous GVT Algorithm for Shared Memory Multiprocessors Architectures", In *9th Workshop on Parallel and Distributed Simulation (PADS '95)*, June, 1995.

[71] S. Zhou, W. Cai, S. J. Turner and F. Lee, "Critical Causality in Distributed Virtual Environments", In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS '02)*, pp. 53–59, 2002.

[72] Y-B. Lin and E. D. Lazowska, "Determining the Global Virtual Time in a Distributed Simulation", In *Proceedings of the 1990 International Conference on Parallel Processing*, vol. 3, August, pp. 201–209, 1990.

[73] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, "John Wiley and Sons, Inc.", 2000.

[74] F. Mattern, "Efficient distributed snapshots and global virtual time algorithms for non-fifo systems", *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423–434, August 1993.

[75] R. M. Fujimoto and M. Hybinette, "Computing global virtual time in shared-memory multiprocessors", *ACM Trans. Model. Comput. Simul.*, vol. 7, no. 4, pp. 425–446, 1997.

[76] L. Lamport, "How to make a multiprocessor compute that correctly executes multiprocess programs", *IEEE Transactions on Computers*, vol. 28, no. 9, pp. 690–691, September 1979.

[77] *Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors*, 1988.

[78] J. Laudon and D. Lenoski, "The sgi origin: a ccnuma highly scalable server",1997.

[79] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "The effect of state-saving in optimistic simulation on a cache-coherent non-uniform memory access architecture", *Proceedings of the 1999 Winter Simulation Conference*, 1999.

[80] H. Avril, and C. Tropper. "Clustered Time Warp and Logic Simulation". *Proceedings of the* 9^{th} *Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 112–119, June 1995.

[81] S. Bellenot. "State Skipping Performance with the Time Warp Operating System". *In Proceedings of the* 6^{th} *Workshop on Parallel and Distributed Simulation (PADS '92)*, pages 53–64. January 1992.

[82] R. Brown. "Calendar Queues: A Fast *O(1)* Priority Queue Implementation for the Simulation Event Set Problem". *Communications of the ACM (CACM)*, volume 31, number 10, pages 1220–1227, October 1988.

[83] C. D. Carothers and R. M. Fujimoto. "Background Execution of Time Warp Programs". *Proceedings of the* 10^{th} *Workshop on Parallel and Distributed Simulation (PADS'96)*, pages 12–19, May 1996.

[84] C. D. Carothers, K. S. Permalla, and R. M. Fujimoto. "Efficient Optimistic Parallel Simulations using Reverse Computation", *Proceedings of the* 13^{th} *Workshop on Parallel and Distributed Simulation (PADS'99)*, pages 126–135, May 1999.

[85] C. D. Carothers, R. M. Fujimoto, and Y-B. Lin. "A Case Study in Simulating PCS Networks Using Time Warp.", *Proceedings of the* 9^{th} *Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 87–94, June 1995.

[86] S. Das and R. M. Fujimoto. "A Performance Study of the Cancelback Protocol for Time Warp". *In Proceedings of the* 7^{th} *Workshop on Parallel and Distributed Simulation (PADS '93)*, pages 135–142. May 1993.

[87] S. Das, and R. M. Fujimoto. "An Adaptive Memory Management Protocol for Time Warp Parallel Simulator". *In Proceedings of the ACM Sigmetrics Conferences on Measurement and Modeling of Computer Systems (SIGMETRICS '94)*, pages 201–210, May 1994.

[88] S. Das, R. M. Fujimoto, K. Panesar, D. Allison and M. Hybinette. "GTW: A Time Warp System for Shared Memory Multiprocessors.", *In Proceedings of the 1994 Winter Simulation Conference*, pages 1332–1339, December 1994.

[89] E. Deelman and B. K. Szymanski. "Breadth-First Rollback in Spatially Explicit Simulations", *In Proceedings of the* 11^{th} *Workshop on Parallel and Distributed Simulation (PADS'97)*, pages 124–131, June 1997.

[90] R. M. Fujimoto and M. Hybinette. "Computing Global Virtual Time in Shared Memory Multiprocessors", *ACM Transactions on Modeling and Computer Simulation*, volume 7, number 4, pages 425–446, October 1997.

[91] R. M. Fujimoto and K. S. Panesar. "Buffer Management in Shared-Memory Time Warp Systems". *In Proceedings of the* 9^{th} *Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 149–156, June 1995.

[92] R. M. Fujimoto. "Time Warp on a shared memory multiprocessor.", *Proceedings of the 1989 International Conference on Parallel Processing*, volume 3, pages 242–249, August 1989.

[93] R. M. Fujimoto. Time Warp on a shared memory multiprocessor. *Transactions of the Society for Computer Simulation*, 6(3):211–239, July 1989.

[94] F. Gomes. "Optimizing Incremental State-Saving and Restoration.",Ph.D. thesis, Dept. of Computer Science, University of Calgary, 1996.

[95] Personal correspondence with Intel engineers regarding the Intel NX450 PCI chipset. See `www.intel.com` for specifications on on this chipset.

[96] D. R. Jefferson. "Virtual Time II: The Cancelback Protocol for Storage Management in Distributed Simulation". *In Proceedings of the* 9^{th} *ACM Symposium on Principles of Distributed Computing*, pages 75–90, August 1990.

[97] J. Laudon and D. Lenoski. The SGI Origin: a ccNUMA highly scalable server *Proceedings of the 24th International Symposium on Computer Architecture*, pages 241–251, June 1997.

[98] P. L'Ecuyer and T. H. Andres. "A Random Number Generator Based on the Combination of Four LCGs.",*Mathematics and Computers in Simulation*, volume 44, pages 99–107, 1997.

[99] Y-B. Lin and B. R. Preiss. "Optimal Memory Management for Time Warp Parallel Simulation", *ACM Transactions on Modeling and Computer Simulation*, volume 1, number 4, pages 283–307, October 1991.

[100] A. Markopoulou, G.Iannaccone, S.Bhattacharyya, C.N.Chuah, and C.Diot. "Characterization of Failures in an IP Backbone Network", *Proceedings of IEEE INFOCOM 2004*, March 2004.

[101] G.M. Amdahl, "Validity of single-processor approach to achieving large-scale computing capability", Proceedings of AFIPS Conference, Reston, VA. 1967. pp. 483-485.

[102] D. Bailey, "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers", RNR Technical Report, RNR-90-020, NASA Ames Research Center, 1991.

[103] T,H, Corman, C.E. Leiserson and R.L. Rivest, "Introduction to Algorithms", ISBN: 0-262-03141-8, MIT Press, 1990. pp. 926-932.

[104] J.L. Gustafson, "Reevaluating Amdahl's Law", CACM, 31(5), 1988. pp. 532-533.

[105] T.G. Lewis, and H. El-Rewini, "Introduction to Parallel Computing", Prentice Hall, ISBN: 0-13-498924-4,1992. pp. 32-33.

[106] Y. Shi, "Reevaluating Amdahl's Law and Gustafson's Law", `http://www.cis.temple.edu/~shi/docs/amdahl/amdahl.html`, Oct, 1996.

[107] D. Nicol. "Performance Bounds on Parallel Self-Initiating Discrete-Event Simulations". *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, volume 1, number 1, pages 24–50, January 1991.

[108] L. Perchelt. "Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences". *Communications of the ACM (CACM)*, volume 42, number 10, pages 109–111, October, 1999.

[109] G. D. Sharma *et al.* "Time Warp Simulation on Clumps". *Proceedings of the* 13^{th} *Workshop on Parallel and Distributed Simulation (PADS '99)*, 1999, pages 174–181.

[110] R. Smith, R. Andress and G. M. Parsons. "Experience in Retrofitting a Large Sequential Ada Simulator to Two Versions of Time Warp". *Proceedings of the* 13^{th} *Workshop on Parallel and Distributed Simulation (PADS'99)*, pages 74–81, May 1999.

[111] J. S. Steinman. "SPEEDES: Synchronous Parallel Environment for Emulation and Discrete-event Simulation". *In Advances in Parallel and Distributed Simulation*, volume 23, pages 95–103, SCS Simulation Series, January 1991.

[112] J. S. Steinman. "Breathing Time Warp". *In Proceedings of the* 7^{th} *Workshop on Parallel and Distributed Simulation (PADS '93)*, pages 109–118, May 1993.

[113] J. S. Steinman. "Incremental state-saving in SPEEDES using C++.", *In Proceedings of the 1993 Winter Simulation Conference*, December 1993, pages 687–696.

[114] D. Bauer, G. Yaun, C. D. Carothers, M. Yuksel and S. Kalyanmaraman. "Large-Scale Network Protocol Meta-Simulation Design and Performance Analysis.", *In Proceedings of the 2004 Winter Simulation Conference*, December 2004.

[115] F. Wieland, E. Blair and T. Zukas. "Parallel Discrete-Event Simulation (PDES): A Case Study in Design, Development and Performance Using SPEEDES". *In Proceedings of the* 9^{th} *Workshop on Parallel and Distributed Simulation (PADS '95)*, pages 103–110, June 1995.

[116] C. H. Young, R. Radhakrishnan, and P. A. Wilsey. "Optimism: Not Just for Event Execution Anymore", *In Proceedings of the* 13^{th} *Workshop on Parallel and Distributed Simulation (PADS '99)*, pages 136–143, May 1999.

[117] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. "In Search of Path Diversity in ISP Networks", *Proceedings of IMC*, 2003.

[118] R. Teixeira and A. Shaikh and T. Griffin and J. Rexford. "Dynamics of Hot-Potato Routing in IP Networks", *Proceedings of SIGMETRICS*, 2004.

[119] R. Teixeira and A. Shaikh and T. Griffin and G. M. Voelker. "Network Sensitivity to Hot-Potato Disruptions", *Proceedings of SIGCOMM*, 2004.

[120] R. Agarwal and C.-N. Chuah and S. Bhattacharyya and C. Diot. "The Impact of BGP Dynamics on Intra-Domain Traffic", *Proceedings of SIGMETRICS*, 2004.

[121] A. Shaikh and C. Isset and A. Greenberg and M. Roughan and J. Gottlieb. "A Case Study of OSPF Behavior in a Large Enterprise Network", *Proceedings of SIGCOMM Internet Measurement Workshop*, 2002.

[122] A. Papachristodoulou and L. Li and J. C. Doyle. "Methodological Frameworks for Large-Scale Network Analysis and Design", *ACM SIGCOMM Computer Communication Review*, October 2004.

[123] J. Rexford and J. Wang and Z. Xiao and Y. Zhang. "BGP Routing Stability of Popular Destinations", *Proceedings of Internet Measurement Workshop (IMW)*, 2002.

[124] A. Feldmann and O. Maennel and Z. M. Mao and A. Berger and B. Maggs. "Locating Internet Routing Instabilities", *Proceedings of SIGCOMM*, 2004.

[125] D. F. Chang and R. Govindan and J. Heidemann. "The Temporal and Topological Characteristics of BGP Path Changes", *Proceedings of the Internation Conference Network Protocols*, 2003.

[126] D. Watson and F. Jahanian and C. Labovitz. "Experiences with Monitoring OSPF on a Regional Service Provider Network", *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[127] J. Cowie and A. Ogielski and B. J. Premore and Y. Yuan. "Global routing instabilities triggered by Code Red II and Nimda worm attacks", *Renesys Corporation Tech Report*, 2001.

[128] L. Wang and X. Zhao and D. Pei and R. Bush and D. Massey and A. Mankin and S. F. Wu and L. Zhang. "Observation and Analysis of BGP Behavior Under Stress", *Proceedings of ACM SIGCOMM Workshop on Internet Measurement*, 2002.

[129] T. G. Griffin and F. B. Shepherd and G. Wilfong. "The Stable Paths Problem and Interdomain Routing", *IEEE/ACM Transactions on Networking*, April 2002.

[130] T. G. Griffin and G. Wilfong. "A Safe Path Vector Protocol", *Proceedings of INFOCOM*, 2000.

[131] T. G. Griffin and G. Wilfong. "Analysis of the MED Oscillation Problem in BGP", *Proceedings of the Internation Conference Network Protocols*, 2002.

[132] Cisco Systems, Inc., "How the bgp deterministic-med Command Differs from the bgp always-compare-med Command ", *Document ID: 16046*, `http://www.cisco.com/warp/public/459/bgp-med.html`, March 2005.