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

ROSS.NET: OPTIMISTIC PARALLEL SIMULATION FRAMEWORK FOR LARGE-SCALE INTERNET MODELS

David Bauer
Garrett Yaun
Christopher D. Carothers

Computer Science Department
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180, U.S.A.

Murat Yuksel
Shivkumar Kalyanaraman

Electrical and Computer Systems Engineering Department
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180, U.S.A.

ABSTRACT

ROSS.Net brings together the four major areas of networking research: network modeling, simulation, measurement and protocol design. ROSS.Net is a tool for computing large scale design of experiments through components such as a discrete-event simulation engine, default and extensible model designs, and a state of the art XML interface. ROSS.Net reads in predefined descriptions of network topologies and traffic scenarios which allows for in-depth analysis and insight into emerging feature interactions, cascading failures and protocol stability in a variety of situations. Developers will be able to design and implement their own protocol designs, network topologies and modeling scenarios, as well as implement existing platforms within the ROSS.Net platform. Also using ROSS.Net, designers are able to create experiments with varying levels of granularity, allowing for the highest-degree of scalability.

1 INTRODUCTION

Fundamental to the process of network protocol design and operations are a variety of performance analysis techniques (Jain 1991; Floyd and Paxson 2001). These techniques have been used by researchers in a variety of different contexts: analytic models (e.g., TCP models, Padhye et al. 2000; self-similar models, Leland et al. 1994; and topology models, Tangmunarunkit et al. 2002), simulation platforms (e.g., ns-2, Breslau et al. 2000; SSFNet `<www.ssfnet.org>`; and GloMoSim `<pcl.cs.ucla.edu/projects/glomosim>`), prototyping platforms (e.g., MIT Click Router toolkit, Kohler et al. 2000; XORP, Handley et al. 2003), tools for systematic design of experiments and exploring parameter state spaces (e.g., Recursive Random Search, Ye and Kalyanaraman 2003); STRESS, Helmy et al. 2000), experimental emulation platforms (e.g., Emulab, White et al. 2002), real-world overlay deployment platforms (e.g., Planetlab, Peterson et al. 2002), and real-world measurement and data-sets (`<www.cadia.org>`).

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: 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 (Jain 1991; Floyd and Paxson 2001; Floyd and Kohler 2003).

If we already have so many tools, it is natural to ask, why are more sophisticated tools required? Sally Floyd and Vern Paxson (2001) pinpoint a number of reasons why simulating the Internet (or a significant representative fraction of it) is difficult. They point to three reasons: scale, heterogeneity and rapid change.

To enable these capabilities requires a number of innovations across the fronts of modeling, simulation engine design and design of experiments (DOE). On the modeling and simulation front, ROSS.Net enables, (i) optimistic parallel simulation engine (called ROSS which stands for Rensselaer's Optimistic Simulation System) which 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 structures and algorithms, and (iii) measurement. These high-level objectives are translated into an integrated XML-based configuration and libraries platform that make it possible for a researcher to combine topology, parameter configuration information and to 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.

Scaling the simulation platform is just one dimension of our research. Recall that, beyond mere scaling of simulation platforms, we need capabilities to address the remaining two issues: heterogeneity and rapid change. Heterogeneity means more than just having support for multiple protocol models (e.g., ns-2) we urgently need large-scale experiment design capabilities that are integrated with the simulation and models to create an overarching test bed platform. This platform has the ability to help in the organization of multiple simulation experiments in the quest of the general invariant relationships between parameters and protocol response. Floyd (2001) sums the state-of affairs by saying: "...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..." To address this need, we have developed a large-scale experiment design platform called ROSS.Net that allows us to characterize 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. The result of this work includes a unified search and optimization framework with demonstrated ability to pose meaningful large-scale design questions and provide "good" characterizations rapidly.

There are many problems that only occur when attempting large scale simulation, including visualization, abstraction and analyzing the results. There is a difference between simulation and emulation. In emulation, we strive to recreate a real world situation. In simulation, we are allowed to make abstractions and generalizations. These abstractions are critical to the performance of the simulation engines. These abstractions can lead to high performance models, but at the same time yielding results that do not answer the networking questions being investigated. It is important that simulation systems provide a mechanism by which the appropriate level of abstraction can be achieved which maximizes simulator performance for the questions being asked. Large scale simulation systems must provide a way of collecting data and analyzing the results in a meaningful way. The experiment designer must be able to characterize each component of the system separately, and be able to analyze the growth patterns, see the effect of topology and protocols, and find and predict clusters. They need to be able to correlate topology and traffic characteristics to protocol, queuing and routing statistics. ROSS.Net allows experiment designers to do just that.

2 ROSS.NET BIG PICTURE

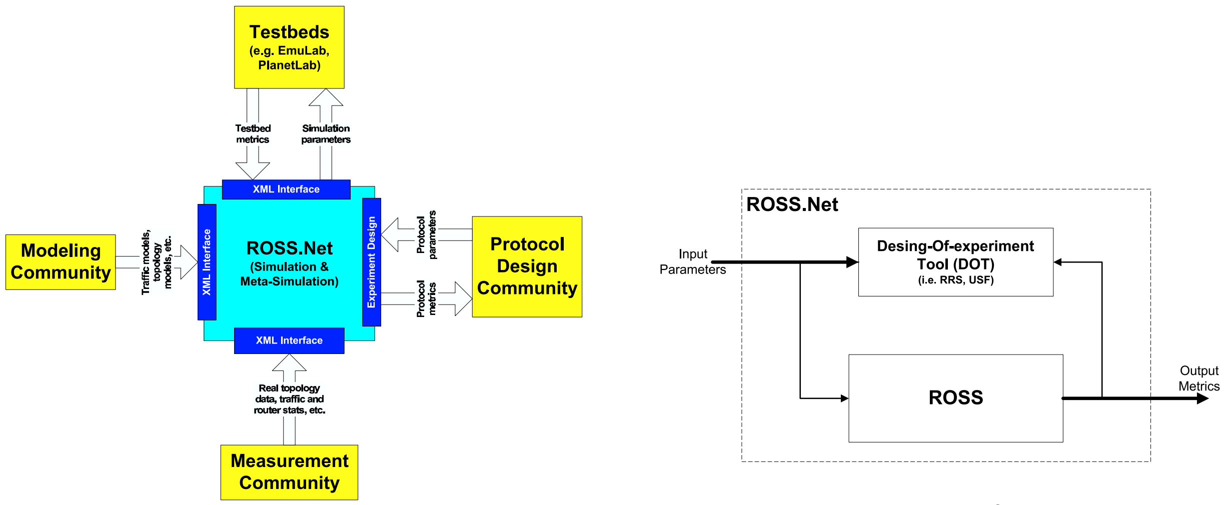
Unlike conventional network simulators where simple, flat models are simulated, ROSS.Net brings together the four major areas of networking research: network modeling, simulation, measurement and protocol design. The big picture is shown in Figure 1-a. Using a state of the art XML interface, ROSS.Net is able to read in predefined descriptions of network topologies and traffic scenarios which allows for the first time in-depth analysis and insight into emerging feature interactions, cascading failures and protocol stability in a variety of situations.

Developers are be able to design and implement their own protocol designs, network topologies and modeling scenarios, as well as implement existing platforms within the ROSS.Net platform. And using ROSS.Net, designers are be able to create experiments with varying levels of abstraction, allowing for the highest-degree of scalability. As a practical example, a designer could put together a ROSS.Net simulation which would combine real network topology data, such as an autonomous system with several hierarchical levels, including areas, subnets, gateways, stand-alone systems or endpoints, in addition to lower level elements such as bridges, switches and repeaters. On top of the topology, a traffic generating scenario could also be specified for each of the machines as appropriate. ROSS.Net would then be able to run the design in a multitude of varying conditions, computing statistical measurements, highlighting scalability issues, as well as pin-pointing network trends expected or otherwise. Figures 2-a and 2-b shows the structure of modeling and simulation of ROSS.Net. ROSS.Net basically constructs a shell on top of the ROSS simulation engine.

3 ROSS.NET FRAMEWORK

The ROSS.Net framework consists of several elements, including a discrete-event, optimistic parallel simulation engine ROSS (Carothers, Bauer, and Pearce 2002), a collection of protocol libraries and the overarching ROSS.Net simulation model, XML schema for describing models, and design of experiments tool called DOT. Where possible, ROSS.Net borrows from best use practices seen in other simulation systems. ROSS.Net adds to those features new design elements which have been missing and allows for the first time experimental design and analysis at a high level, incorporating techniques from the areas of simulation, modeling, and measurement.

The design of experiments tool (DOT) allows for investigators to specify the type and duration of the experiment being designed. Once described, the DOT configures and executes the ROSS.Net core with the specified XML input data descriptors, simulator engine performance parameters, and protocol dependent parameters and executes a search algorithm such as random recursive search (Ye and Kalya-



(a) ROSS.Net and Other Major Experimentation Areas (b) Layered Architecture of ROSS.Net

Figure 1: ROSS.Net Big Picture

naraman 2003). Many executions might be needed in order to complete the DOT picture before complex analysis and validation can be completed. ROSS.Net can then generate reports and graphs based on the semantic content of the experiment as well as validate the experiment results.

At the heart of ROSS.Net is Rensselaer's Optimistic Simulation System or ROSS. ROSS demonstrated that stable, highly efficient execution using only a small constant more memory than required by the sequential simulation is possible. Running on top of ROSS is the ROSS.Net simulation. The ROSS.Net simulation pulls together all of the model modules into one cohesive structure and gives them access to ROSS.Net network simulation components such as global data structures represent the network topology, and a connection library for traffic scenarios as well as provides certain functionality such as a per node default IP layer for support of routing and queuing in cases where this is not provided by the sub-model directly. The ROSS.Net model also contains the XML interfaces for the input and output streams associated with the DOT.

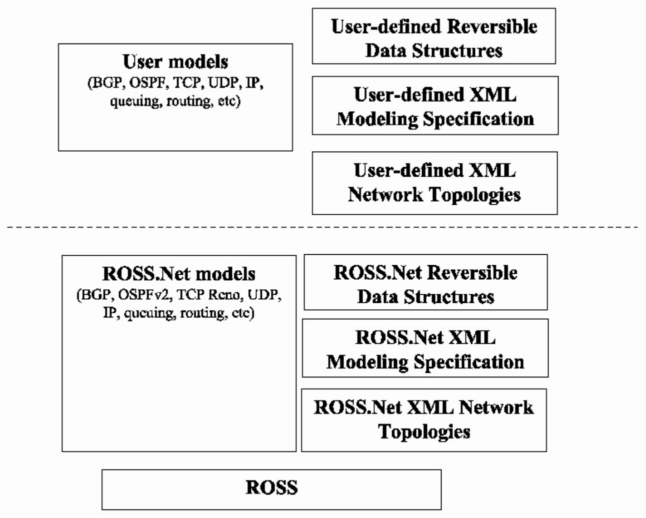
3.1 ROSS

ROSS is an optimistic parallel simulation engine that is targeted for low event granularity models. ROSS' API is based on a message-passing interface. LPs communicate strictly by exchanging timestamped event messages as direct reading and writing of LP state is prohibited. In addition, an efficient timer interface is provided for the purpose of network protocol modeling, as well as a reversible abstract data structure and memory management library. ROSS has already been used to develop models of IP-multi-casting, TCP, UDP, BGP4 and OSPFv2 as well as modeling of wireless and sensor networks. It's ability to model millions of network nodes has already been demonstrated conclusively (Yaun, Carothers, and Kalyanaraman 2003).

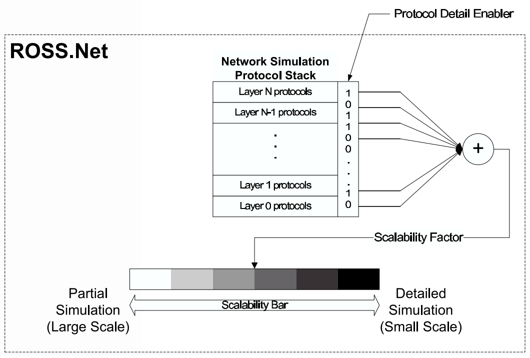
Reverse computation is the process of being able to undo operations that are speculatively executed out of order. When an out of order event is found in the system, events are "rolled back", or reverse computed. The key advantage of an optimistic approach is that it operates independent of the underlying network topology. Thus, it continues to exploit all the available parallelism even during dynamic changes in topology. 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 (Carothers, Perumalla and Fujimoto 1999). To address this problem, a new approach called reverse computation for realizing the undo operation of the state space is being used. With 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.

3.2 EXAMPLE EXPERIMENT

As an example of the types of experiments that can be performed with ROSS.Net, we summarize our recent performance study using the AT&T topology as part of an overall large-scale TCP model simulation (Yaun, Carothers, and Kalyanaraman 2003). This network topology obtained from the Rocketfuel website (Spring, Mahajan and Wetherall 2002). As shown in Figure 3, the core US AT&T network topology contains 13,173 router nodes and 38,164 links. What makes Internet topologies like the AT&T network both interesting and challenging from a modeling prospective is the spareness and power-law structure (Spring, Mahajan


User models
(BGP, OSPF, TCP, UDP, IP, queuing, routing, etc)
User-defined Reversible Data Structures
User-defined XML Modeling Specification
User-defined XML Network Topologies
ROSS.Net models
(BGP, OSPFv2, TCP Reno, UDP, IP, queuing, routing, etc)
ROSS.Net Reversible Data Structures
ROSS.Net XML Modeling Specification
ROSS.Net XML Network Topologies
ROSS

(a) Subscription Structure of ROSS.Net Models



(b) Multi-Abstraction Paradigm of ROSS.Net

Figure 2: ROSS.Net Modeling and Simulation Concepts

and Wetherall 2002). In the case of AT&T, there are less than 3 links on average. However, at the super core there is a high-degree connectivity. Typically, an Internet service provider's super core will be configured as a fully connected mesh. Consequently, backbone routers will have up to 67 connections to other routers, some of which are other backbone or super core routers and other links to region core routers. Once at the region core level, the number of links per router reduces and thus the connectivity between other region cores is spare. Most of the connectivity is dedicated to connecting local points of presence (PoPs).

In performing a breath-first-search of the AT&T topology, there are distinct eight levels. At the backbone, there are 414 routers. At each successive level yields the following router count : 4,861, 5,021, 1,117, 118, 58, 6 and at the final level there are 5 nodes. There were a number of routers not directly reachable from within this network. Those routers are most likely transit routers going strictly between autonomous systems (AS). With the transit routers removed, our AT&T network scenario has 11,670 routers. Link weights are derived based on the relative bandwidth of the link in comparison to other available links. In this configuration, routing is keep static, however we do have dynamic routing currently working on a light-weight OSPF model in which we plan to integrate with our TCP model in the very near future.

The bandwidth ranged from almost 10 Gb/sec at the top-level down to 70 Kb/sec at the lower levels. The buffer size and link delay ranged from 12 MB and between 10ms to 30ms delay at the top-level to 5 KB buffer size and 5ms of delay at the lower layers of the network. This configuration was scaled to both a medium case with 96,500 LPs (end hosts plus routers) and a large case with 266,160 LPs. In each configuration, half of the end hosts establish a TCP session to a *randomly selected* receiving host. *We observe this configuration is almost pathological for a parallel network simulation because the amount of remote network traffic will be much greater than is typical in practice*. Our goal is to demonstrate simulator efficiency under high-stress workloads for realistic topologies.

We observed over 99% efficiency for our parallel runs. However, despite the efficiency, the speedup results were marginal but encouraging. Our best case speedup as 1.25 for the medium configuration and 1.29 for the large configuration. The platform used in this performance study was a dual Hyper-threaded 2.8 GHz Pentium 4 Xeon processors, which multiplexes two instruction streams or threads per processor. We attribute the disparity between efficiency and end speedup to the enormous amount of remote messages sent between instruction streams/processors. The AT&T network topology for a round-robin LP to processor mapping results in almost 80% of the all processed events being remotely schedule. We hypothesize that behavior on the part of the model reduces memory locality and results in much higher cache miss rates. Consequently, all instruction streams are spending more time stalled waiting for memory requests to be satisfied. With a better load distribution, we believe the speedup results will be much higher and potentially scale to larger processor configurations. The memory requirements for the AT&T scenario were 269 MB for the medium size network and 328 MB for the large size network, yielding a per TCP connection overhead of 2.8 KB and 1.3 KB respectively.

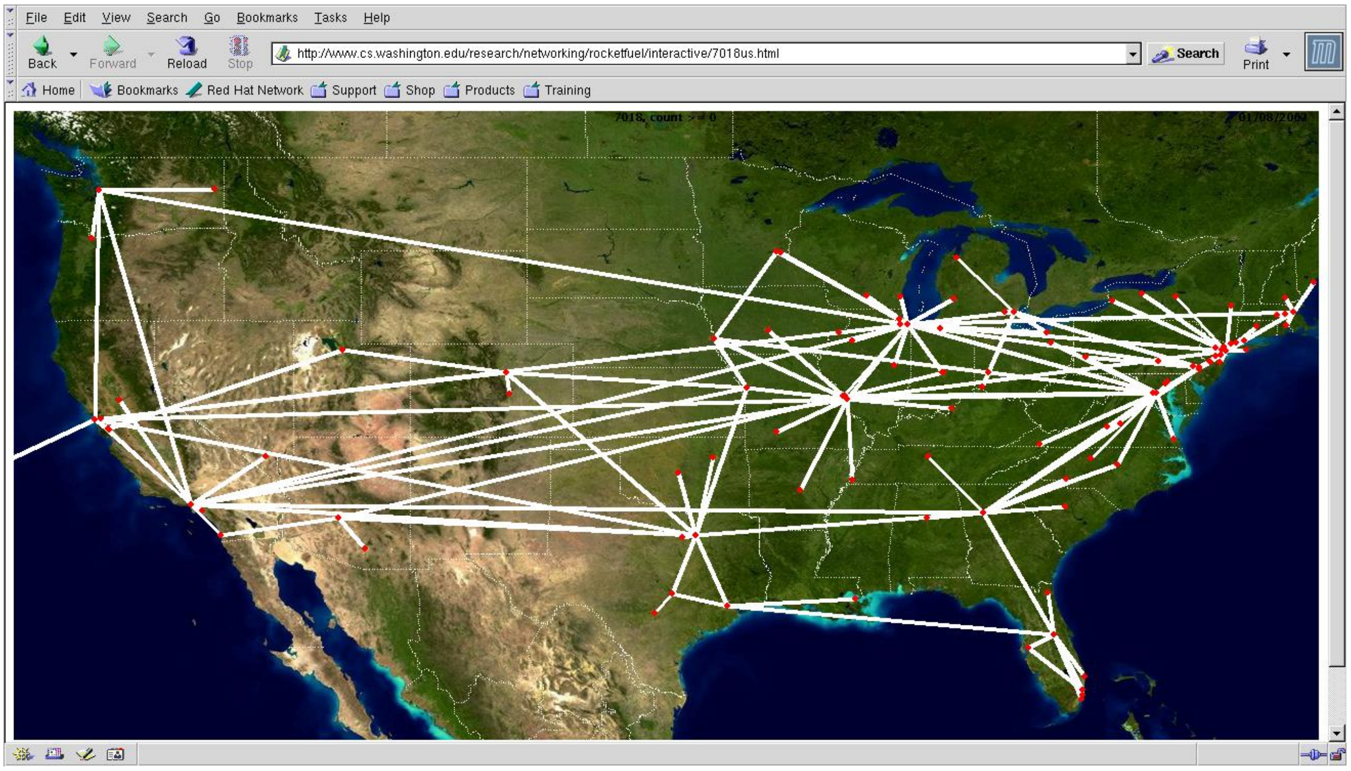

File Edit View Search Go Bookmarks Tasks Help
Back
Forward
Reload
Stop
http://www.cs.washington.edu/research/networking/rocketfuel/interactive/7018us.html
Search
Print
Home
Bookmarks
Red Hat Network
Support
Shop
Products
Training
7018, count >= 0

Figure 3: AT&T Network Topology (AS 7118) from the Rocketfuel Data Bank for the Continental U.S.A.

4 LARGE-SCALE EXPERIMENT DESIGN AND ANALYSIS

Scaling the simulation platform is just one dimension of our research. Recall that, beyond mere scaling of simulation platforms, we need capabilities to address the issues of heterogeneity and rapid change in simulating large networks to extract and interpret meaningful performance data. "Heterogeneity" means more than just having support for multiple protocol models (e.g., ns-2): we urgently need experiment design or "meta-simulation" capabilities that are integrated with the simulation and models to create an overarching test bed platform (see Figure 1-b). The purpose of the large-scale experiment design piece 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.

Design of Experiments or "experiment design" is a well known branch of performance analysis, specifically, a sub branch of statistics (Jain 1991; Montgomery 2001). It has been used extensively in areas like agriculture, industrial process design and quality control (Montgomery 2001), and has been introduced to the area of practical computer and network systems design by Raj Jain (Jain 1991). 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 (e.g., a simulation run, an Emulab or Planetlab test run) 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 (Jain 1991; Montgomery 2001). The usual end-goal of formulating regression models is to observe the effects of both individual parameters and parameter interactions (Jain 1991; Montgomery 2001). 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 (Jain 1991; Montgomery 2001).

The next step beyond characterization (i.e., developing input-output regression models) is to determine the region in the important factors that leads to best-possible response. The output or response in general will have an unknown surface topology, also known as “response surface”. The approach 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. Well-known small-polynomial order response-surface methods include central composite design (CCD) (efficient fitting of second order models), Box-Behnken designs (Montgomery 2001). 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 (Taguchi 1986), that have been credited for triggering a quality-control revolution in the 1980s-90s (Montgomery 2001).

Taguchi’s RPDs use highly fractionated factorial designs and other fractional designs obtained from orthogonal arrays.

5 XML MODELING

There are typically two main data inputs for any given model: topology and a traffic scenario. Several systems have designed excellent approaches to describing some or all of these inputs, but what is lacking is an abstract way in which network models are described so that they can be used across multiple simulation systems. Additionally, little or no work has been done on abstractly validating these inputs or, describing the results of the simulations. Even when work is done on describing the inputs and outputs to a simulation, it is invariably tied to a particular simulation system and cannot be re-used or verified in another system without a great amount of effort.

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere (<www.w3c.org/XML/>). XML is one of the fastest growing modern technologies which was designed primarily to make data application independent much as the Java programming language attempts to make applications platform independent. Our goal for using XML is to make the modeling data transparent to the simulation engine. In addition to using XML for modeling, in the future we intend to use XML interfaces to online components of the system (e.g., PlanetLab, Emulab in both the control and data plane; as well as interactive user interfaces, e.g., SNMP, RMON).

We use model data from real networks and are still able to capture minute trends resulting from protocol changes, policy changes and/or parameter changes. XML allows us to find and clarify these results by designing experiments which can be tightly controlled in the large-scale. XML allows us to view scenarios and topologies as simply variables in the parameter space of a design of experiments and to treat those parameters as though they were a black box.

6 ROSS.NET MODEL

The ROSS.Net models which come with the distribution include default models for protocols such as TCP, UDP, OSPFv2, BGP4 etc. Each protocol defines an XML descriptor for the protocol for configuration and validation. These models were generated by the ROSS.Net team at RPI and have been optimized using complex abstraction techniques where appropriate, and conform to the RFC specifications. New models may be designed and included by the user for protocol development and testing purposes by extending the existing models or by providing complete model implementations. Further, models may be generated and tested in the isolated environment of ROSS, and then ported simply and quickly to the ROSS.Net API. The ROSS.Net API is based on the message passing API available in ROSS. ROSS.Net is a fully modular and extensible system making it simple for users to generate models in most languages with a C language interface and incorporate them into the overall ROSS.Net structure. Users simply need to setup their module within the ROSS.Net system and provide an XML parser for any user-defined XML descriptors used by their model.

ROSS.Net also provides a simple API to model developers for the use of ROSS timer events, and memory buffers. Models should use these functions as they have been designed to be as efficient as possible within the ROSS memory usage paradigm. They also aid in the quick development of models because they provide common functionality through an abstract API.

ROSS.Net parses the XML modeling specification and generates the necessary global data structure representing the network topology and connection database for use by all models through the use of access functions public in the ROSS.Net API. The global data structure and connection database, with their associated API functions provide a high level approach to accessing the input data and allow models to be able to efficiently compute necessary constructs such as routing tables and connection streams.

All discrete-event engines define some form of entities or logical processes (LPs) which describe the state of the processes in the system. LPs must be mapped to the model in some way that minimizes remote message passing between processing elements which require mutual exclusion devices. ROSS.Net uses the XML model descriptor to complete this mapping for the user and conforms to the OSI/ISO model for networking layers. A typical model description includes a node on the network, and the protocol layers that it is expected to simulate. By analyzing the links between the network nodes, ROSS.Net is able to determine a mapping of network nodes to LPs automatically for the user, which reduces the number of remote messages being passed. An assumption we make is that network nodes with high degrees of connectivity will have a higher number of events being passed to them. ROSS.Net also maps the protocol layers onto the LPs in such a way that muxing/demuxing of packet streams is handled automatically by a stream port number.

To further maximize the capabilities of discrete-event message passing, ROSS.Net handles internally the mapping of protocol layers to the OSI/ISO layering model within each LP. This allows each modeled layer to act completely independently of each other. ROSS.Net handles the details of event allocation and message passing between layers. Events passed between layers are not directly sent through the ROSS core, but rather handed directly through the following layer for processing. By not passing the event through the ROSS core, ROSS.Net is able to simulate much higher levels of packet passing than in a conventional discrete-event simulator. The only control given to each model when sending events is whether the event should be propagated up or down through the protocol stack. Because the models act completely independently of each other, protocol designers are free to design their models in the maximum number of configurations. For example, running OSPF over IP would be a typical model representing current Internet behavior, but OSPF could be tested over TCP without any consideration in the OSPF model. Each protocol becomes a building block where there are no limits on how they are placed together.

ROSS.Net also provides a default IP layer for models which do not specify an IP layer. This facilitates constructs such as multi-homed TCP hosts, where the TCP modeler is not concerned with the routing decision making. The ROSS.Net default IP layer has several configurable options, including queue usage, routing and demuxing of datagram streams. The experiment designer can choose from within the DOT interface what types of queuing mechanisms should be used for each class of network node, and then the IP layer for those nodes are automatically configured within ROSS.Net. Conversely, the experiment design chosen in the DOT may prescribe which types of queuing mechanism, if any, are needed in order to collect the relevant data for the experiment. In this way, the modeler no longer needs to choose each and every configuration option, and ensure their correctness, but can instead rely on high level experiment design to choose appropriate values and configurations for them. Decision making at the DOT level also helps to select the appropriate level of detail to collect during the simulation execution(s), thereby maximizing the simulation performance.

ROSS.Net provides a generic, compressed packet for event transmissions. The ROSS.Net packet is intended to simplify some of the packet details commonly found in different protocols. The ROSS.Net packet header contains information accessible to all model layers such as the source and destination address and the stream port number as well as some layer information such as direction the packet is currently taking through the layer stack. ROSS.Net models use this packet as the final envelope for message passing within the ROSS core. All protocol messages are packed into the ROSS.Net packet and transmitted to the appropriate destination. The ROSS.Net LP then provides each layer with both the ROSS.Net packet, as well as the encapsulated model header and data information. This not only reduces the event size being sent through the system, but enables the different model layers to act independently of each other. In the downstream transmission, the FTP protocol model only receives and understands how to handle FTP protocol packets. An underlying TCP protocol model only receives the application layer packet size and a pointer to the meta-data. The TCP layer decides how to appropriately send the meta-data, and generates several smaller TCP packets which ROSS.Net hands to the underlying IP layer for transmission to the next LP in the routing table. The FTP layer does not explicitly call a TCP provided send function, nor does the TCP layer call an explicit IP layer send function. ROSS.Net simply hands the data either up or down between the layers. The combination of the layers determine the LPs complexity of work. Several components may be connected together within an LP to form the overall logical process, and ROSS.Net manages the interactions between them through the use of the abstract ROSS.Net packet.

7 CONCLUSIONS

ROSS.Net brings together the four major areas of networking research: network modeling, simulation, measurement and protocol design. ROSS.Net is a tool for computing large scale design of experiments through components such as a discrete-event simulation engine, default and extensible model designs, and a state of the art XML interface. Developers will be able to design and implement their own protocol designs, network topologies and modeling scenarios, as well as implement existing platforms within the ROSS.Net platform.

In the future we will develop online interfaces between ROSS.Net and live SNMP and RMON. Work has already been completed on SNMP XML interfaces, so connecting to these live sources should be relatively simple. We also intend to develop interfaces to emulation platforms such as PlanetLab and Emulab.

ACKNOWLEDGMENTS

This research is supported by an NSF CAREER Award CCR-0133488, the DARPA Network Modeling and Simulation program, contract #F30602-00-2-0537 and a AT&T University Relations Program Grant.

REFERENCES

Breslau, L., D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. 2000. Advances in network simulation. *IEEE Computer* 3(10): 59–67.

Carothers, C. D., Bauer, D., and S. Pearce, 2002. Ross: a high-performance, low memory, modular time warp system. *Journal of Parallel and Distributed Computing* 62: 1648–1669.

Carothers, C. D., K. S. Perumalla, and R. M. Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulation* 9(3): 224–253.

Floyd, S. 2001. Simulation is crucial. *IEEE Spectrum* 38(1): 76, sidebar article.

Floyd, S., and E. Kohler. 2003. Internet research needs better models. In *First Workshop on Hot Topics in Networks (HotNets-I)*, Special Issue of *ACM SIGCOMM Computer Communication Review* 33(1): 29–34.

Floyd, S., and V. Paxson. 2001. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking* 9(4): 392–403.

Handley, M., O. Hodson, and E. Kohler. 2003. XORP: open platforms for network research. In *First Workshop on Hot Topics in Networks (HotNets-I)*, Special Issue of *ACM SIGCOMM Computer Communication Review* 33(1): 53–58.

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

Jain, R. 1991. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley - Interscience.

Kohler, E., R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. 2000. The click modular router. *ACM Transactions on Computer Systems* 18(3): 263–297.

Leland, W., M. Taqqu, W. Willinger, and D. Wilson. 1994. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking* 2(1): 1–15.

Montgomery, D. C. 2001. *Design and analysis of experiments*. John Wiley and Sons.

Padhye, J., V. Firoiu, D. Towsley, and J. Kurose. 2000. Modeling tcp reno performance: A simple model and its empirical validation. *IEEE/ACM Transactions on Networking* 8(2): 133–145.

Peterson, L., T. Anderson, D. Culler, and T. Roscoe. 2002. A blueprint for introducing disruptive technology into the internet. In *First Workshop on Hot Topics in Networks (HotNets-I)*. Special Issue of *ACM SIGCOMM Computer Communication Review* 33(1): 59–64.

Sikdar, B., S. Kalyanaraman, and K. S. Vastola. 2001. An integrated model for the latency and steady state throughput of tcp connections. *Performance Evaluation* 46: 139–154.

Spring, N., R. Mahajan, and D. Wetherall. 2002. Measuring isp topologies with rocketfuel. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 133–145.

Taguchi, G. 1986. *Introduction to quality engineering*. Asian Productivity Organization, UNIPUB, White Plains, NY.

Tangmunarunkit, H., R. Govindan, S. Jamin, S. Shenker, and W. Willinger. 2002. Network topology generators – structural vs. degree-based. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 147–159.

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

Yaun, G., C. D. Carothers, and S. Kalyanaraman. 2003. Large-scale tcp models using optimistic parallel simulation. To appear in *the Proceedings of the* 17th *Workshop on Parallel and Distributed Simulation*.

Ye, T., and S. Kalyanaraman. 2003. A recursive random search algorithm for large-scale network parameter configuration. To appear in *the Proceedings of the 2003 ACM SIGMETRICS Conference*.

AUTHOR BIOGRAPHIES

DAVID BAUER is currently a completing his PhD in computer science at Rensselaer Polytechnic Institute. While pursuing his PhD at RPI, he has designed and developed ROSS and ROSS.Net in addition to other commercial distributed systems. At GE/CRD he developed a design of experiments tool for use in the research and development of mercury consumption in fluorescent bulbs. While At the MapInfo Corp. Mr. Bauer developed software for the coordination and configuration of multiple XML and Java servers. His research interests include parallel and distributed systems and network simulation with a focus on performance optimizations. His email address is `<bauerd@cs.rpi.edu>`.

GARRETT YAUN is a Ph.D. student in the Department of Computer Science at Rensselaer Polytechnic Institute. His recent research in efficient TCP models for optimistic parallel simulation won best paper at PADS 2003. Garrett's research interests include parallel and distributed systems, networking and modeling and simulation. His email address is `<yaung@cs.rpi.edu>`

CHRISTOPHER CAROTHERS is an assistant professor in the Computer Science Department at Rensselaer Polytechnic Institute. He received the PhD, MS, and BS from Georgia Institute of Technology in 1997, 1996, and 1991, respectively. Prior to joining RPI, he was a research scientist at the Georgia Institute of Technology. As a PhD student, he interned twice with Bellcore, where he worked on wireless network models. In 1996, he interned at MITRE Corporation, where he was part of the DoD High Level Architecture development team. His research interests include parallel and distributed systems, simulation, and networking. His email address is `<chrisc@cs.rpi.edu>`.

MURAT YUKSEL is currently a Post-Doctoral Research Associate at ECSE Department of Rensselaer Polytechnic Institute (RPI), Troy, NY. He received a BS degree from Computer Engineering Department of Ege University, Izmir, Turkey in 1996. He received MS and PhD degrees from Computer Science Department of RPI in 1999 and 2002 respectively. His research interests are as network pricing, routing in wireless networks, large-scale network simulation, networking with free-space optics and performance analysis. His email address is `<yuksem@ecse.rpi.edu>`.

SHIVKUMAR KALYANARAMAN is an Associate Professor at the Department of Electrical, Computer and Systems Engineering at Rensselaer Polytechnic Institute in Troy, NY. He received a B.Tech degree from the Indian Institute of Technology, Madras, India in July 1993, followed by M.S. and Ph.D. degrees in Computer and Information Sciences at the Ohio State University in 1994 and 1997 respectively. His research is in topics such as congestion control architectures, quality of service, and free-space optical networking. His email address is `<shivkumar@ecse.rpi.edu>`.