NIST Publication #904682
Measurement Science for Complex Information Systems

HOW TO MODEL A TCP/IP NETWORK USING ONLY 20 PARAMETERS

Kevin L. Mills
Information Technology Laboratory
National Inst. Of Stds. & Tech.
Gaithersburg, MD 20899, USA

Edward J. Schwartz
Dept. of Electrical & Computer Eng.
Carnegie Mellon University
Pittsburgh, PA 15213, USA

Jian Yuan
Dept. of Electronic Engineering
Tsinghua University
Beijing, 100084, P. R. CHINA

ABSTRACT

Most simulation models for data communication networks encompass hundreds of parameters that can each take on millions of values. Such models are difficult to understand, parameterize and investigate. This paper explains how to model a modern data communication network concisely, using only 20 parameters. Further, the paper demonstrates how this concise model supports efficient design of simulation experiments. The model has been implemented as a sequential simulation called MesoNet, which uses Simulation Language with Extensibility (SLX). The paper discusses model resource requirements and the performance of SLX. The model and principles delineated in this paper have been used to investigate parameter spaces for large (hundreds of thousands of simultaneously active flows), fast (hundreds of Gigabits/second) simulated networks under a variety of congestion control algorithms.

1 INTRODUCTION

Paxson and Floyd (1997) describe many difficult problems that impede simulation of large data communication networks, and recommend two main coping strategies: search for invariants and carefully explore the parameter space. Unfortunately, typical network simulators (e.g., Fall and Varadhan 2009, SSFNet 2009, Tyan et al. 2009) use hundreds of parameters that can each take on millions of values. Such simulations can be difficult to configure and usually require infeasible resources to explore the parameter space. Several researchers (Riley et al. 2004, Yaun et al. 2003, Zeng et al. 1998) investigate parallel techniques as a means to simulate larger, faster networks. Unfortunately, such techniques do not reduce the parameter space, which remains difficult to configure and continues to require significant resources when conducting careful exploration. As pointed out by Ammar (2005), the problems identified by Paxson and Floyd have gone largely unsolved. In this paper, we describe how to model a modern data communication network, including the transmission control protocol (TCP) and Internet protocol (IP), using only 20 parameters. We implemented the model using SLX[1] (Henriksen 2000) as a sequential simulation, called MesoNet. As we demonstrate, a concise parameter space can be searched efficiently and effectively using sequential simulations deployed in parallel, where each simulation explores a selected configuration of parameters. Elsewhere (Mills et al. 2010), we use MesoNet to study a variety of congestion control algorithms proposed for the Internet. In that study, we perform a sensitivity analysis of the model's parameter space, providing key insights that guide design of the experiments. Here, we discuss only two sample experiments to illustrate the utility and resource requirements of MesoNet.

The paper makes three contributions: (1) defines a concise TCP/IP network simulation model that can be configured using only 20 parameters; (2) shows how the model can be applied to design efficient experiments; and (3) discusses resource requirements for the model and selected performance properties of SLX, the underlying simulation platform. The ideas contained in this paper facilitate feasible exploration of the parameter space in large network simulations and should also stimulate other researchers to develop concise models for large distributed systems, such as computational grids and clouds.

The paper is organized in six main sections. In Sec. 2 we explain why the parameter space of simulation models can be difficult to explore and then discuss some theoretical techniques for reducing the search space. We also show the substantial reduction we were able to achieve in formulating our model. In Sec. 3 we introduce and define the 20 parameters of our model. Sec. 4 outlines two sample experiments, illustrating the utility of our reduced model. Sec. 5 discusses resource require-

[1] Any mention of commercial products within this paper is for information only; it does not imply recommendation or endorsement by NIST. Though MesoNet source code is in the public domain, the model requires SLX, a commercial simulation language and runtime environment developed and sold by Wolverine Software.

ments for our model and also related SLX performance characteristics. In Sec. 6 we discuss work by others who aim to enable simulation of large data networks. We conclude and suggest future work in Sec. 7.

2 SEARCH-SPACE REDUCTION: THEORY & PRACTICE

As illustrated in Fig. 1(a), a simulation model can be viewed as a function transforming a set of input parameters, x_1 to x_n, into a set of responses, y_1 to y_m. Each input parameter can take on a range of values, 1 to k in our example, defining a parameter space of size k^n, which can be very large. Fig. 1(c) shows the infeasible search space arising from a communication network model with n = 1000 parameters that can each take on $k = 2^{32}$ values.

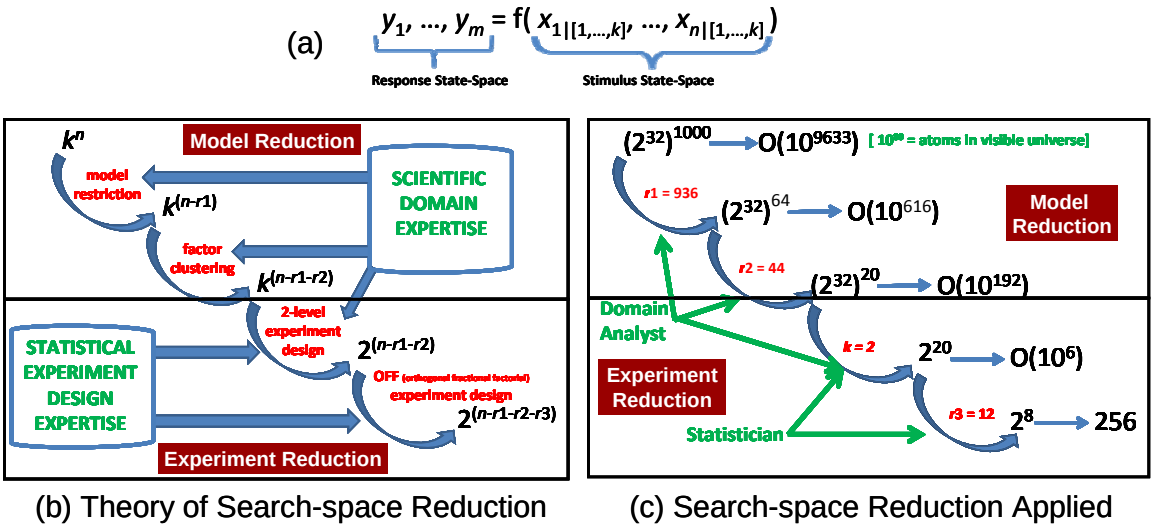

Figure 1: (a) Functional representation of a simulation model; (b) Theoretical explanation of search-space reduction; (c) Search-space reduction applied to MesoNet simulation model

2.1 Theory of Search-space Reduction

Fig. 1(b) illustrates two processes that can help reduce the search space: reduce the number of parameters in the model and reduce the number of parameter configurations through judicious experiment design. The process of model reduction involves two main steps. First, restrict model parameters to only that set of $(n - r1)$ factors relevant to the questions under investigation. Second, identify parameters that can be clustered together as facets of a single factor, leaving a reduced set of factors numbering $(n - r1 - r2)$. These two steps require expertise within the domain of investigation. In many cases, a reduced model parameter space remains infeasible to search, requiring two additional reduction steps to limit the number of experiments. The first step involves selecting only two levels to assign for each parameter – reducing k to 2. Choosing appropriate levels requires domain knowledge. If the reduced search space of $2^{(n-r1-r2)}$ remains too expensive, then one can adopt an orthogonal fractional factorial (OFF) experiment design (Box, Hunter and Hunter 2005) to further reduce the space to $2^{(n-r1-r2-r3)}$, providing the most information possible for the available resources.

2.2 Search-space Reduction in Practice

Fig. 1(c) illustrates the practical reduction we achieved in constructing a communication network model intended to compare proposed congestion control algorithms for the Internet. Assuming a detailed network model requires 1000 parameters, we identified 64 parameters germane to our investigation, achieving an initial reduction of $r1 = 936$. Subsequently, we grouped some of the 64 parameters together to create a reduction of $r2 = 44$, leaving the 20-parameter model that we describe below in Sec. 3. In Sec. 4 we give examples using a two-level, OFF experiment design to further reduce the search space.

3 THE MODEL

Table 1 identifies the 20 parameters composing our model of a TCP/IP network. We organize the parameters into five categories: (1) network configuration, (2) sources and receivers, (3) user behavior, (4) protocols and (5) simulation and measurement control. We discuss each category in turn, defining every parameter in detail.

Table 1: Model Parameters

Category	Identifier	Name
Network Configuration	X1	Topology
	X2	Propagation Delay
	X3	Network Speed
	X4	Buffer Provisioning
Sources & Receivers	X5	Number of Sources & Receivers
	X6	Distribution of Sources
	X7	Distribution of Receivers
	X8	Source & Receiver Interface Speeds
User Behavior	X9	Think Time
	X10	Patience
	X11	Web Object Size for Browsing
	X12	Proportion & Size of Larger File Downloads
	X13	Selected Spatiotemporal Congestion
	X14	Long-lived Flows
Protocols	X15	Congestion Control Algorithms
	X16	Initial Congestion Window Size
	X17	Initial Slow Start Threshold
Simulation & Measurement Control	X18	Measurement Interval Size
	X19	Simulation Duration
	X20	Startup Pattern

3.1 Network Configuration

A network configuration requires a topology (parameter X1) of routers and links, as shown for example in Fig. 2, adapted from the topology of a modern Internet service provider. MesoNet supports topologies with up to three hierarchical router tiers: backbone routers (A-P in Fig. 2), point of presence (PoP) routers (A1-P2) and access routers (A1a-P2g). To model heterogeneity in network access, MesoNet allows three different types of access routers: **D**-class (e.g., eight red nodes in Fig. 2, which connect directly to backbone routers), **F**-class (e.g., 40 green nodes) and **N**-class (e.g., 122 small gray nodes). Classifying access routers enables different speeds to be assigned to each class. As discussed later, sources and receivers compose a fourth tier distributed below access routers. Packets flowing between a source-receiver pair follow a single ingress/egress path between an access router and a top-tier backbone router. In MesoNet ingress/egress paths are not subject to propagation delays. Propagation delays on backbone links are an intrinsic property of the topology. Table 2 shows propagation delays associated with each of the 24 backbone links in Fig. 2. A topology also specifies the paths taken by packets flowing among backbone routers. Given a cost metric for each link (Table 2 col. 3), one can use Dijkstra's shortest-path first (or equivalent) algorithm to generate least-cost paths. The topology in Fig. 2 and cost metrics in Table 2 generate 240 backbone paths with an average length of 3.63 router hops. Adding in the hops for sources and receivers to reach the backbone routers increases the average path length to 9.53 hops. As illustrated in Table 2 (cols. 5 and 6), parameter X2 can scale down (e.g., X2 = 0.5) or up (e.g., X2 = 2) propagation delays on all backbone links.

Unlike real networks, where links have transmission speeds and associated buffers, MesoNet assigns transmission speeds to routers. Each router multiplexes packet forwarding from a single buffer shared among all attached links. Because MesoNet packets have no size, router speeds are assigned in units of packets/millisecond. Six parameters, shown in Table 3 col. 1, are needed to define the speed of all router classes (col. 3), using relationships shown in col. 4. Note that every defined relationship includes parameter *s*1. By assigning values to the remaining parameters, e.g., as in col. 2, one can establish reasonable engineering relationships among the speeds of the various router classes. Then, by equating *s*1 with model parameter X3, the speeds of all routers in a topology can be scaled appropriately by changing the value of X3, as shown in cols. 5 and 6, which indicate the speed of each router class in packets/millisecond.

To provision router buffers, MesoNet allows buffer size (in packets) to be selected using any of four algorithms: (1) $RTT \times C$, recommended practice (Bush and Meyer 2003), where *RTT* is the average round-trip time among all backbone routes and *C* is the capacity, derived from X3, for the router class; (2) ($RTT \times C/\sqrt{n}$), recommended by some researchers (Appenzeller et al. 2004), where *n* is the expected number of flows transiting a router; (3) the average of (1) and (2); or (4) a designated value. In addition, a variable, *Qfactor*, can scale the buffer sizes computed by the chosen algorithm. Thus, parameter X4 may be a value pair (buffer sizing algorithm, *Qfactor*) or one may fix the buffer sizing algorithm and equate X4 to *Qfactor* or fix *Qfactor* and equate X4 to a buffer sizing algorithm.

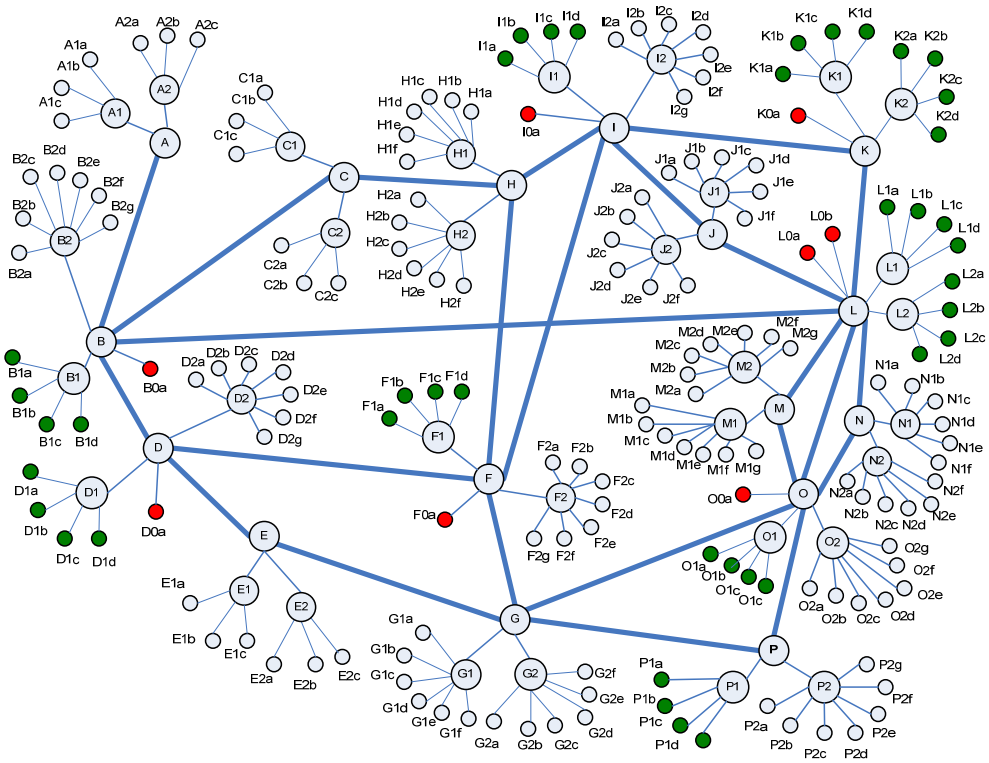

Figure 2: Three Tier Topology with 16 Backbone Routers (A-P), 32 Point of Presence Routers (A1-P2) and 170 Access Routers (A1a-P2g) – 8 red and 40 green Access Routers may operate at different speeds from the 122 others

Table 2: Topology Link Characteristics and Scaling Propagation Delay with Parameter X2

Link#	Endpoints	Cost Metric	Prop. Delay (ms)	X2 = 0.5	X2 = 2
1	A-B	50	21	10.5	42
2	B-C	10	25	12.5	50
3	B-D	50	8	4	16
4	B-L	223	75	37.5	150
5	C-H	100	12	6	24
6	D-E	10	10	5	20
7	D-F	108	33	16.5	66
8	E-G	100	33	16.5	66
9	F-G	10	7	3.5	14
10	F-H	50	12	6	24
11	F-I	55	22	11	44
12	G-O	104	23	11.5	46
13	G-P	110	19	9.5	38
14	I-H	10	14	7	28
15	I-J	50	8	4	16
16	I-K	147	22	11	44
17	J-L	60	20	10	40
18	K-L	50	7	3.5	14
19	L-M	50	12	6	24
20	L-N	39	6	3	12
21	L-O	10	14	7	28
22	M-O	10	6	3	12
23	N-O	10	8	4	16
24	O-P	10	14	7	28

Table 3: Defined Speed Relationships among Router Classes used to Scale Router Speeds with Parameter X3

Parameter	**Value**	**Speed Relationships**		**Speed Scaling with X3**	
*s*1	X3	**Router Class**	**Speed**	**X3 = 800**	**X3 = 1600**
*s*2	4	Backbone	*s*1 x *BBspeedup*	1600	3200
*s*3	10	PoP	*s*1/ *s*2	400	800
BBspeedup	2	**N**-Class	*s*1/ *s*2/ *s*3	40	80
Bfast	2	**F**-Class	*s*1/ *s*2/ *s*3 x *Bfast*	80	160
Bdirect	10	**D**-Class	*s*1/ *s*2/ *s*3 x *Bdirect*	400	800

3.2 Sources and Receivers

Given a three-tier topology of routers and links, a fourth tier of sources and receivers must be created and then distributed under (and attached to) access routers. Sources equate to computers that have information that receivers wish to download. The model fixes the number of receivers to be four times the number of sources. The number of sources in a topology should be suited to the network speed, otherwise the network will experience an inappropriate traffic load. To accommodate this engineering relationship, the model includes a variable, *baseSources*, which should be set to a value appropriate for the network speed. Model parameter X5 serves as a multiplier to scale the number of sources and receivers. For example, given that *baseSources* = 100 and X5 = 3, then about 300 sources and 1200 receivers would be attached to each access router – so the topology in Fig. 2, which has 170 access routers, would contain about 51,000 sources and 204,000 receivers. These numbers are only approximate because, as discussed next, changing the distribution of sources and receivers causes adjustments.

Table 4: Sample Computation of Number and Distribution of Sources and Receivers
(given Fig. 2 and *baseSources* = 100, X5 = 3, *probNs* = 0.1, *probNsf* = 0.6, *probNr* = 0.8, *probNrf* = 0.1)

Class	**#routers**	**srcs/router**	**#srcs**	**%srcs**	**rcvrs/router**	**#rcvrs**	**%rcvrs**	**Flow class**	**%flows**
N-class	122	90	10,980	31.6	960	117,120	95.3	**NN**-flows	30.1
								FN-flows	60.5
F-class	40	540	21,600	62.2	120	4,800	3.9	**FF**-flows	2.4
								DN-flows	6.1
D-class	8	270	2,160	6.2	120	960	0.8	**DF**-flows	0.74
								DD-flows	0.05

Recall that access routers come in three classes, as show in Table 4 col. 1. The precise number of sources under access routers of each type can be adjusted by assigning the probability, *probNs*, a source is under an **N**-class router and the probability, *probNsf*, a source is under an **F**-class router. The probability a source is under a **D**-class router is then 1 – (*probNS* + *probNsf*). For example, if each router class has a target of 300 sources, then the total number of sources under three routers, one of each class will be (3 x 300 =) 900. Assigning *probNs* = 0.1 and *probNsf* = 0.6 would reapportion the 900 routers as shown in Table 4 col. 3. Given the quantity of routers in each class (Table 4 col. 2), the total number of sources under each router class would be as shown in Table 4 col. 4, and so the aggregate number of sources in the topology would be 34,740 instead of 51,000. Table 4 col. 5 gives the proportion of sources in the topology located under each router class. Similar computations can be made by assigning *probNr* and *probNrf* to reapportion receivers, as shown in cols. 6 and 7, where the total number of receivers in the topology is reduced to 122,880. Table 4 col. 8 gives the proportion of receivers in the topology located under each router class. As discussed below, each source will periodically transfer a flow of packets, after randomly selecting a receiver from under a parent backbone router that differs from the source's parent backbone router. Since access routers of different classes have differing speeds, the locations of a source-receiver pair influence the characteristics of the path for each flow of packets. Table 4 col. 9 lists six possible flow classes, as determined by the location of the source and receiver for a flow. For the parameters given in the caption of Table 4, col. 10 shows the proportion of flows in each class. One can view **NN**-flows as a form of peer-to-peer (P2P) traffic, while the remaining flow classes can be viewed as Web-centric traffic. Table 4 represents a network with about 30% P2P flows and 70% Web-centric flows. Model parameter X6 specifies the distribution of sources with a pair of probabilities (*probNs*, *probNsf*) and parameter X7 specifies the distribution of receivers with another pair of probabilities (*probNr*, *probNrf*).

The final property of sources and receivers concerns the maximum speed at which they can transfer packets to the network. The model includes two settings: *Hbase* and *Hfast*, which specify a number of packets/millisecond. For example setting *Hbase* = 8 corresponds to 8,000 packets/second, which equates to 96 Mbps, assuming 1500-byte packets. Similarly, set-

ting *Hfast* = 80 corresponds to 80,000 packets/second, which equates to 960 Mbps. Parameter X8 specifies the probability that a source or receiver connects at a speed of *Hfast*.

3.3 User Behavior

User behavior is modeled through periodic activity by sources. As shown in Fig. 3, sources cycle between thinking and sending[2]. As explained later, a source may begin in either state. Prior to entering the **Thinking** state, a source selects a random residence time from an exponential distribution with a mean given by parameter X9. Upon expiration of residence, the source enters the **Sending** state, where a flow of packets is transmitted to a randomly selected receiver. Once all packets in a flow are acknowledged, the source follows the **Finished** transition, reentering the **Thinking** state. Flows may be associated with human users that have finite patience or with programs that have infinite patience. Human users expect short flows to be completed within a reasonable time and long flows to progress at a reasonable rate. Violation of these expectations cause a source to enter the thinking state, following the appropriate failure transition: **Too Slow** or **Too Long**. Parameter X10 specifies the probability that a source has finite patience.

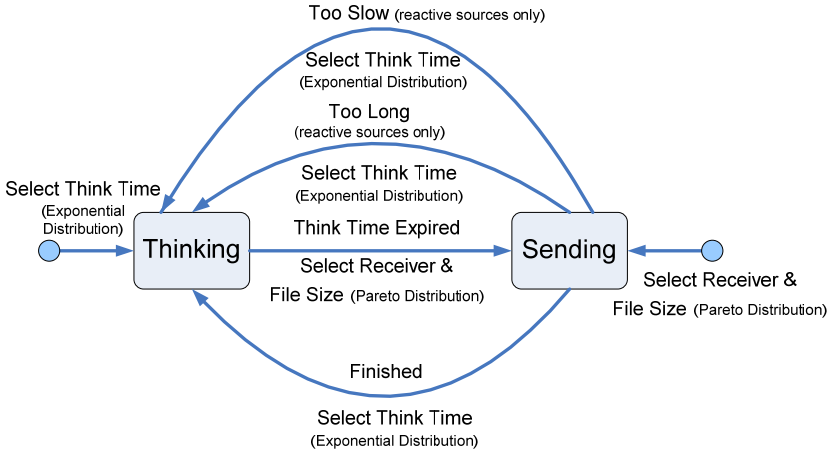

Figure 3: User Behavior Represented through Periodic Activity by Sources

Prior to entering the **Sending** state, a source selects a Web object size (in packets) from a Pareto distribution with a shape parameter α and a mean λ, which define parameter X11. Alternatively, one can fix α and use X11 as the mean size. We adopt a Pareto distribution to mimic long-tailed file sizes observed in Internet traffic (Crovella, Taqqu and Bestavros 1998). The model also allows sources to transmit larger files in three categories: documents, software updates and movies, with corresponding multipliers (*Fx*, *Sx* and *Mx*) that scale the selected Web object to a larger size. The model includes variables to specify a corresponding probability of transmission (*Fp*, *Sp* and *Mp*) for each category. Parameter X12 can be a set of multiplier and probability pairs {(*Fx*, *Fp*), (*Sx*, *Sp*), (*Mx*, *Mp*)} or the multipliers may be fixed, allowing X12 to specify a set of probabilities for transferring larger files. The probability of a Web object is 1 – (*Fp* + *Sp* + *Mp*). The model also allows simulation of spatiotemporal congestion by specifying a time period during which jumbo files will be transferred on every **DD** flow. Parameter X13 specifies this with a set of three variables: proportion (*Jon*) of simulated time before jumbo files commence, proportion (*Joff*) after which jumbo transfers cease and a multiplier (*Jx*) applied to convert Web object sizes into jumbo files.

The model accommodates simulation of long-lived flows that, once activated, send as many packets as possible in the course of a simulation. Each long-lived flow is specified by four parameters: proportion (*Lon*) of simulated time before the flow starts, access routers under which the source (*sLocation*) and receiver (*rLocation*) are located and identifier (*sType*) for the congestion control algorithm used by the source. If no *sType* is selected from Table 5 col. 2, then the congestion control algorithm will be assigned using the probabilities given in col. 3. Each parameter set, LLF_x = {*Lon*, *sLocation*, *rLocation*, *sType*}, describes a single long-lived flow *x*. Parameter X14 consists of a set of sets {LFF_1, …, LFF_n} describing all long-lived flows in an experiment. MesoNet measures long-lived flows in detail, which permits observation of intricate behaviors in individual flows. In addition, long-lived flows can be positioned deterministically within a topology to investigate the effects of spatiotemporal congestion on individual flows. Finally, because empirical results are available for long-lived flows

[2] For simplicity, Fig. 3 omits a flow connection phase that occurs prior to sending, and also the potential for connection failure after which a source reenters the thinking state.

sharing a bottleneck link, the behavior of MesoNet congestion control algorithms can be verified against empirical measurements (Li, Leith and Shorten 2007).

3.4 Protocols

MesoNet was created to compare congestion control algorithms, which regulate the rate at which sources send packets on individual network flows. A congestion control algorithm allows a source to estimate the transmission rate available to a flow, to attempt to increase the rate, and to reduce the rate in response to packet losses, which are assumed to result from network congestion on a path. Each source requires a congestion control algorithm, such as TCP, which is implemented in most computers connected to the Internet. In outline, standard TCP probes (during a process known as initial slow start) for available transmission capacity on a flow by first sending a few packets and then increasing the rate exponentially as acknowledgments arrive. When a packet is lost, TCP switches to a process known as congestion avoidance, reducing transmission rate by 50% and then increasing the rate linearly on subsequent acknowledgments. When repeated, the resulting behavior exhibits a sawtooth pattern of increasing and decreasing transmission rate on a flow. As we explain elsewhere (Mills et al. 2010),various researchers have critiqued the efficacy of this behavior in future networks with increased capacities, leading to several proposals for alternate congestion control algorithms intended to coexist with (or replace) standard TCP. MesoNet includes models for seven congestion control algorithms, as shown in Table 5 (consult Mills et al. 2010 for the details of each algorithm). The variables listed in col. 3 can be set to specify the probability that a source implements the related congestion control algorithm. Model parameter X15 comprises this list of probabilities (which must sum to 1). We validated our model for each congestion control algorithm against empirical results (Li, Leith and Shorten 2007 and Leith et al. 2008) from measured behavior of long-lived flows in a small topology of Linux nodes. The empirical results plotted congestion window (*cwnd*) vs. time for competing flows transiting a bottleneck link under various combinations of transmission capacity, buffer size and propagation delay. We simulated the same parameter combinations and generated plots matching the empirical plots.

Table 5: MesoNet Congestion Control Algorithms, Identifiers and Probabilities of Source Implementation

Congestion Control Algorithm	Identifier	Probability of Source Implementation
Transmission Control Protocol (TCP)	1	*prTCP*
High Speed TCP (HSTCP)	2	*prHSTCP*
Compound TCP (CTCP)	3	*prTCP*
Scalable TCP (STCP)	4	*prSTCP*
FAST AQM Scalable TCP (FAST)	5	*prFAST*
Hamilton TCP (HTCP)	6	*prHTCP*
Binary Increase Congestion (BIC)	7	*prBIC*

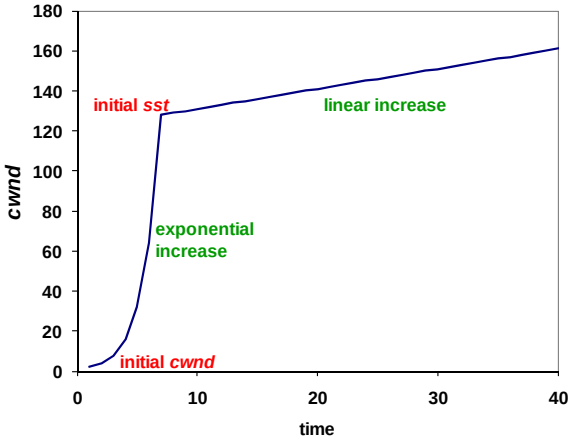

Figure 4: Illustration of Initial Slow Start and Switch to Congestion Avoidance Absent Packet Loss

The six alternate congestion control algorithms listed in Table 5 only alter the TCP congestion avoidance process, for initial rate probing they all use the standard TCP initial slow start process. Upon connecting to a receiver, a source first sends a specified number of packets, known as the initial congestion window. As acknowledgments arrive from the receiver, the source increases the *cwnd* exponentially. Upon first lost packet, the source switches to congestion avoidance and adopts the

procedures associated with the congestion control algorithm implemented by the source. Absent any losses, a source switches to congestion avoidance once the congestion window reaches an initial slow start threshold (*sst*). Fig. 4 illustrates this process for TCP. Model parameter X16 specifies the initial *cwnd* and X17 defines the initial *sst*.

3.5 Simulation Measurement and Control

MesoNet measures numerous aspects of each simulation run. Most measurements are made as time series, such as illustrated in Fig. 5, which sample system state at periodic intervals (parameter X18) of size *M*. Model parameter X19 is the number (*MI*) of measurement intervals to be recorded, so simulation duration is *M* x *MI*.

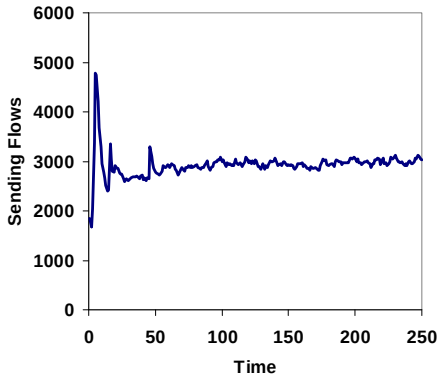

Figure 5: Count of Flows in the Sending State Measured every *M* = 200 ms for *MI* = 250 intervals – Simulation Duration (.2 s x 250 =) 50 s – *prON* = 0.25, *prONsecond* = 0.08, *prONthird* = 0.17

Model parameter X20 comprises a set of three variables: (1) probability a source starts in the **Sending** state (*prON*), (2) probability (*prONsecond*) a source exits from an initial **Thinking** state after a random time with mean 33% of X9 and (2) probability (*prONthird*) a source exits from initial **Thinking** after a random time with mean 66% of X9. Sources exit initial **Thinking** after a random time with mean X9 with probability 1 – (*prON* + *prONsecond* + *prONthird*). Accelerating source startup helps a system to reach equilibrium more quickly than would otherwise occur. Fig. 5 illustrates use of parameter X20, which leads to three spikes in the count of sending flows, as each wave of sources enters the **Sending** state.

4 TWO SAMPLE EXPERIMENTS

To demonstrate the utility of our model, we describe two related experiments. We simulated a fixed topology based on the Abilene network, explained elsewhere (Mills et al. 2010), operating for 60 minutes with sources transferring a mix of Web objects, documents, software downloads and movies. Sources use either standard TCP or one of seven alternate congestion control algorithms[3]. In one experiment, we simulated a modest sized network with up to 26,085 sources running at moderate speed (up to 1600 packets/ms). In the second experiment, we increased network size and speed by a factor of 10.

Based on earlier sensitivity analyses (Mills et al. 2010), we identified nine of the 20 model parameters to vary and selected two values for each, which created (2^9 =) 512 parameter configurations. Comparing seven congestion control algorithms under all configurations would require 3584 simulations, giving a total of 7168 for both experiments. Our facilities and available time allowed us to execute $\leq$ 256 runs per experiment. For this reason, we constructed a 2^{9-4} OFF design, yielding 32 configurations against which to run each of the congestion control algorithms. This required (32 x 7 =) 224 simulations per experiment. Fig. 6 shows the 32 configurations used for experiment #1, where *baseSources* = 100. We adopted the router speed relationships and values shown in Table 3. We fixed the buffer-sizing algorithm to *RTT* x *C*, equating parameter X4 with *Qfactor*, which we varied. Since X11 leads to variation in file sizes, we fixed multipliers for larger files (*Fx* = 10, *Sx* = 1000 and *Mx* =10000) and equated factor X12 with the probabilities of transferring files of each size. We set *Hbase* = 8 and *Hfast* = 80. We used parameter X15 to specify the probability a source implements the designated alternate congestion control algorithm in place of TCP. For the second experiment, we increased *baseSources* to 1000 and multiplied settings for network speed (X3) by 10 to yield 32 additional conditions, requiring another 224 simulations. Table 6 reports the values we fixed across all simulations for the remaining 11 model parameters.

[3] We added FAST-AT, a version of FAST that dynamically adjusts one of the algorithm's parameters.

5 MODEL IMPLEMENTATION & RESOURCE REQUIREMENTS

For each sample experiment, Table 7 reports the aggregate number of flows completed and data packets sent, as well as the per run average, minimum and maximum. Table 8 recounts the processing (CPU) hours required for both experiments, as well as the average memory usage. Table 7 shows that an order of magnitude increase in network size and speed leads to a tenfold increase in flows completed and packets sent. Table 8 shows a twelvefold increase in memory requirements, while processing time increased about 16 times. These resource increases must be understood in the context of SLX, which comes in two versions, requiring either 32-bit or 64-bit address space. Simulating the larger, faster network required SLX-64, while the smaller, slower network could be simulated with SLX-32. In SLX-64, address references require more memory. This accounts for the extra increase in memory usage. In addition, 64-bit operations run more slowly than 32-bit operations and the large, fast model has larger event lists, which require SLX additional time to manage.

Factor->	X2	X3	X4	X5	X7	X9	X11	X12	X15
Condition	--	--	--	--	--	--	--	--	--
1	1	800	0.5	3	0.7	5000	100	0.04/0.004/0.0004	0.7
2	1	1600	0.5	2	0.3	5000	100	0.04/0.004/0.0004	0.3
3	2	800	0.5	2	0.7	5000	100	0.02/0.002/0.0002	0.3
4	2	1600	0.5	3	0.3	5000	100	0.02/0.002/0.0002	0.7
5	1	800	1	2	0.3	5000	100	0.02/0.002/0.0002	0.7
6	1	1600	1	3	0.7	5000	100	0.02/0.002/0.0002	0.3
7	2	800	1	3	0.3	5000	100	0.04/0.004/0.0004	0.3
8	2	1600	1	2	0.7	5000	100	0.04/0.004/0.0004	0.7
9	1	800	0.5	3	0.3	7500	100	0.02/0.002/0.0002	0.3
10	1	1600	0.5	2	0.7	7500	100	0.02/0.002/0.0002	0.7
11	2	800	0.5	2	0.3	7500	100	0.04/0.004/0.0004	0.7
12	2	1600	0.5	3	0.7	7500	100	0.04/0.004/0.0004	0.3
13	1	800	1	2	0.7	7500	100	0.04/0.004/0.0004	0.3
14	1	1600	1	3	0.3	7500	100	0.04/0.004/0.0004	0.7
15	2	800	1	3	0.7	7500	100	0.02/0.002/0.0002	0.7
16	2	1600	1	2	0.3	7500	100	0.02/0.002/0.0002	0.3
17	1	800	0.5	2	0.3	5000	150	0.02/0.002/0.0002	0.3
18	1	1600	0.5	3	0.7	5000	150	0.02/0.002/0.0002	0.7
19	2	800	0.5	3	0.3	5000	150	0.04/0.004/0.0004	0.7
20	2	1600	0.5	2	0.7	5000	150	0.04/0.004/0.0004	0.3
21	1	800	1	3	0.7	5000	150	0.04/0.004/0.0004	0.3
22	1	1600	1	2	0.3	5000	150	0.04/0.004/0.0004	0.7
23	2	800	1	2	0.7	5000	150	0.02/0.002/0.0002	0.7
24	2	1600	1	3	0.3	5000	150	0.02/0.002/0.0002	0.3
25	1	800	0.5	2	0.7	7500	150	0.04/0.004/0.0004	0.7
26	1	1600	0.5	3	0.3	7500	150	0.04/0.004/0.0004	0.3
27	2	800	0.5	3	0.7	7500	150	0.02/0.002/0.0002	0.3
28	2	1600	0.5	2	0.3	7500	150	0.02/0.002/0.0002	0.7
29	1	800	1	3	0.3	7500	150	0.02/0.002/0.0002	0.7
30	1	1600	1	2	0.7	7500	150	0.02/0.002/0.0002	0.3
31	2	800	1	2	0.3	7500	150	0.04/0.004/0.0004	0.3
32	2	1600	1	3	0.7	7500	150	0.04/0.004/0.0004	0.7

Figure 6: Definition of the 32 Parameter Configurations used to Simulate a Modest Size, Moderate Speed Network (red values for X3 were multiplied by 10 and *baseSources* increased to 1000 to Simulate a Larger, Faster Network)

Table 6: Fixed Values Assigned to 11 Model Parameters for All Simulation Runs Reported Here

Parameter	**Assigned Value**
X1	Abilene Topology (Backbone: 11 routers and 14 links; 22 PoP routers; 139 Access routers)
X6	*probNs* = 0.1, *probNsf* = 0.6
X7	*probNr* = 0.6, *probNrf* = 0.2
X10	0 (all users have infinite patience)
X13	*Jon* = 1; *Joff* = 1; *Jx* = 1 (no explicit spatiotemporal congestion)
X14	no long-lived flows
X16	initial *cwnd* = 2 (default Microsoft Windows™ value)
X17	initial *sst* = 2^{31}/2 (arbitrary large value)
X18	*M* = 200 ms
X19	*MI* = 18,000 (x .2 *M* =) 3600 s
X20	*prON* = 0.25, *prONsecond* = 0.08, *prONthird* = 0.17

Experiment #1 required 35 weeks of processor time, which we completed in only one week by running the 224 sequential simulations in parallel on 48 processors. Had we had 224 processors available, we could have completed the simulations in just under two days (i.e., the maximum run required 44 processor hours). Similarly, experiment #2 required 131 months (about 11 years) of processor time, which we completed in three months, again by running the 224 sequential simulations in parallel. Full parallelization would have enabled us to complete these simulations in about 31 days (i.e., 739 hours).

To compute the event rate at which SLX processes the simulations, we need to estimate the number of events per packet. First, since every data packet receives an acknowledgment, the number of packets must be doubled. Average source-to-receiver path length is 9.43 hops for the Abilene topology. Using these figures, we estimate the number of events processed in all simulation runs for experiment #1 as 1.5E+13, which divided by the number of CPU seconds used becomes about 725,359 events/second. Taking a similar approach for experiment #2 yields an estimate of 439,864 events/second, where the cost of 64-bit processing and larger event lists causes MesoNet to run about 40% slower. These event rates can be compared with those reported (Yaun et al. 2003) for RossNet, a parallel simulator for large networks. Simulating synthetic topologies of 4 to 32 nodes on one, two or four instruction streams, RossNet averaged 256,244 events/second. Simulating a larger network, based on an AT&T topology, RossNet averaged 150,720 events/second. From this, we surmise that MesoNet, when implemented as a sequential SLX simulation and used to run parameter configurations in parallel, can provide event rates competitive with (perhaps[4] superior to) approaches that execute individual network simulations using parallel processing. For example, given 48 processors, a sequential simulator can execute 48 configurations in parallel, while a parallel simulator using 4 processors per simulation could run only 12 configurations in parallel. This implies that the parallel simulator would need a speedup of 4 to obtain the same throughput as the sequential simulator. The RossNet results report average speedup just under 1.7 (maximum 3.2) for synthetic topologies when using 4 instruction streams. For the larger topology, RossNet achieved a speedup just under 1.3. On the other hand, if a sequential simulator has sufficient processors to simulate all configurations in parallel, the run requiring maximum processing time will determine the simulation latency. Under similar assumptions, the speedup achieved by a 4-processor parallel simulator would reduce the simulation latency, but at the cost of requiring four times more processors than sequential simulations.

Table 7: Flows Completed and Data Packets Sent in Simulation Runs Reported Here

	Experiment #1 – Slow, Small Network		**Experiment #2 – Large, Fast Network**	
Statistic	**Flows Completed**	**Data Packets Sent**	**Flows Completed**	**Data Packets Sent**
Avg./Run	11,466,429	3,414,017,482	116,317,093	33,351,040,358
Min./Run	7,258,056	2,138,998,764	72,944,797	21,069,357,409
Max./Run	17,390,781	5,048,119,166	175,947,632	50,932,067,100
Total All Runs	2,568,480,122	764,739,915,978	26,055,028,851	7,470,633,040,199

Table 8: Resource Requirements for Simulation Runs Reported Here

	Experiment #1	**Experiment #2**
CPU hours (224 runs)	5,857.18	94,355.28
Avg. CPU hours/Run	26.15	421.23
Min. CPU hours/Run	12.58	203.04
Max. CPU hours/Run	43.97	739.04
Avg. Memory Usage (Mbytes)	196.56	2,392.41

6 RELATED WORK

Providing feasible simulation of large, fast communication networks remains an active area of research. Several researchers (e.g., Riley et al. 2004, Yaun et al. 2003, Zeng et al. 1998) investigate the use of parallel processing to simulate TCP/IP networks. For example, RossNet (Yaun et al. 2003) can simulate large, fast networks with hundreds of thousands of simultaneous flows. Unfortunately, parallel simulation alone does not reduce the parameter space required to explore a wide range of conditions. We demonstrate an approach to reduce the parameter search space. In addition, as discussed above, the sequential nature of data communication inhibits the ability of parallel simulators to achieve speedup matching the number of processors employed. We describe an alternate approach using sequential simulations to explore multiple parameter configurations in parallel, where careful scheduling of runs achieves efficient speedup. Even so, parallel simulators deserve continued re-

[4] The MesoNet simulations had the advantage of executing on 3 GHz dual-core Opteron processors, while most of the RossNet simulations ran on dual hyper-threaded 2.8 GHz Pentium-4 Xeon processors.

search because multi-core, multichip technology promises substantial increase in the availability of cost-effective processors, which could significantly reduce simulation latencies.

As an alternative to parallel simulators, several researchers (e.g., Towsley, Misra and Gong 2000, Yi and Shakkottai 2007) propose to model communication networks as topologies where router behavior is described approximately as fluid-flows, using differential equations. Such models may be solved efficiently using numerical methods. While promising, fluid approximation currently exhibits two main shortcomings: (1) inaccuracy (Geurts, Khayat and Leduc 2006) arising from an inability to satisfactorily describe packet loss processes (Genin and Marbukh 2009) and (2) limited dynamics, capturing only steady-state behaviors averaged over long time intervals (Lee et al. 2007). Genin and Marbukh (2009) suggest one means to address these limitations. As another alternative to parallel simulators, Lee and colleagues (2007) propose a hybrid modeling framework that continuously approximates discrete variables by averaging over short intervals of time. Constraining the averaging interval allows generation of significant events, such as packet drops and related adjustments in congestion windows. The approach yields accurate results that can be produced with only about 20% of the processing resources required by sequential discrete-event simulators. Further work remains to extend Lee's hybrid model to have all the features necessary to conduct experiments such as those we describe in Sec. 4. Both fluid approximations and hybrid discrete/continuous-time models appear to be promising alternatives to parallel simulators. Regardless of the underlying modeling approach adopted, the parameter and experiment reduction techniques we describe in this paper should enable researchers to produce models that are easier to understand, parameterize and investigate.

7 CONCLUSIONS AND FUTURE WORK

We defined a concise TCP/IP network model that can be configured using only 20 parameters. Further, we showed how the model can be combined with two-level orthogonal fractional factorial techniques to design efficient experiments to investigate behavior under a wide range of conditions. Using SLX, we implemented the model as a sequential process capable of simulating large (hundreds of thousands of simultaneously active flows), fast (hundreds of Gigabits/second) networks under a variety of congestion control algorithms. We demonstrated how to carefully explore a parameter space using parallel instances of a sequential simulator. We discussed resource requirements for the model and related performance properties of SLX. We found our approach competitive in throughput with a parallel simulator, but showed that parallel simulators should achieve superior simulation latency at the cost of extra processors. Future work remains to generalize our ideas in two directions. First, we need to demonstrate that the approach can be applied to other large distributed systems, such as computation clouds or grids. Second, we need to establish that the approach can produce parallel simulators, fluid approximations and hybrid models that are easier to understand, parameterize and investigate.

ACKNOWLEDGMENTS

The work reported in this paper was funded under the Complex Systems Program within the Information Technology Laboratory at the National Institute for Standards and Technology.

REFERENCES

Ammar, M. 2005. Why we still don't know how to simulate networks. In *Proceedings of the 38th Simulation Symposium*, 3.

Appenzeller G., I. Keslassy and N. McKeown. 2004. Sizing Router Buffers. In *Proceedings of ACM SIGCOMM*, 34:4, 281-292.

Box, G., Hunter, J. and Hunter, W. 2005. *Statistics for Experimenters*. 2nd ed. Hoboken, New Jersey: Wiley.

Bush R. and D. Meyer. 2003. Some Internet Architectural Guidelines and Philosophy. *RFC 3439*.

Crovella M., M. Taqqu and A Bestavros. 1998. Heavy-tailed probability distributions in the World Wide Web. Chapter 1 in *A Practical Guide to Heavy Tails*, Chapman & Hall, 3-26.

Fall, K. and K. Varadhan, eds. 2009. The *ns* Manual. Available via <http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf> [accessed December 2, 2009].

Henriksen, J.O. 2000. SLX: the X is for extensibility. In *Proceedings of the 32nd Winter Simulation Conference*,183-190.

Genin, D. and V. Marbukh. 2009. Bursty fluid approximation of TCP for modeling Internet congestion at the flow level. In *Proceedings of the 47th Annual Allerton Conference on Communication, Control and Computing*, Paper ThD4.3.

Geurts, P., I. El Khayat and G. Leduc. 2006 On the accuracy of analytical models of TCP throughput. volume 3976 Springer.

Lee, J., S. Bohacek, J. Hespanha and K. Obraczka. 2007. Modeling Communication Networks with Hybrid Systems. In *IEEE/ACM Transactions on Networking*, 15:3, 630-643.

Leith, D., L. Andrew, T. Quetchenbach, R. Shorten and K. Lavi. 2008. Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms. In *Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks*, 6 pages.

Li, Y.-T., D. Leith and R. Shorten. 2007. Experimental Evaluation of TCP Protocols for High-Speed Networks. In *IEEE/ACM Transactions on Networking*, 15:5, 1109-1122.

Mills, K., J. Filliben, D. Cho, E. Schwartz and D. Genin. 2010. *Study of Proposed Internet Congestion-Control Mechanisms*. NIST Special Publication 500-TBD.

Paxson, V. and S. Floyd. 1997. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, ed. S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, 1037-1044.

Riley, G., M. Ammar, F. Fujimoto, A. Park, K. Perumalla and D. Xu. 2004. A Federated Approach to Distributed Network Simulation. In *ACM Transactions on Modeling and Computer Simulation*, 14:2, 116-148.

SSFNet. 2009. How to use SSFNet. Available via <http://www.ssfnet.org/internetPage.html> [accessed December 2, 2009]

Towsley, D., V. Misra and W. Gong. 2000. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of SIGCOMM*, 30:4, 151,160.

Tyan, H-Y., A. Sobeih and J. Hou. 2009. Design, Realization and Evaluation of a Component-based, Compositional Network Simulation Environment. In *Simulation*, 85:3, 159-181.

Yaun, G., D. Bauer, H. Bhutada, C. Carothers, M. Yukel and S. Kalyanaraman. 2003. Large-Scale Network Simulation Techniques: Examples of TCP and OSFP Models. In *SIGCOM Computer Communications Review*, 33:3, 27-41.

Yi, Y. and S. Shakkottai. 2007. FluNet: A hybrid internet simulator for fast queue regimes, In *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51:18, 4919-4937.

Zeng. X., R. Bagrodia and M. Gerla. 1998. GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulations*, 154-161.

AUTHOR BIOGRAPHIES

KEVIN L. MILLS is a Senior Research Scientist at the U.S. National Institute of Standards and Technology (NIST). He received his Ph.D. in information technology from George Mason University (GMU). From 1996 to 2006 he served on the adjunct faculty of the department of computer science at GMU. From 1996 to 1999 he was a program manager at the Defense Advanced Research Projects Agency (DARPA). His research interests include complex distributed systems. His email is <kmills@nist.gov>.

EDWARD J. SCHWARTZ is a Ph.D. student in the department of electrical and computer engineering at Carnegie Mellon University. He contributed to this work at NIST as a summer university research fellow of the National Science Foundation. His B.S. in computer science is from Millersville University in 2007. His research interests include cyber security. His email is <edmcman@gmail.com>.

JIAN YUAN is an Associate Professor in the department of electronic engineering at Tsinghua University. He received his Ph.D. in communication and electronic systems from the University of Electronic Science and Technology of China. Formerly, he contributed to this work at NIST as a guest scientist between 2000 and 2004. His research interests include complex distributed systems. His email address is <jyuan@tsinghua.edu.cn>.