Large-Scale Network Parameter Configuration Using an On-Line Simulation Framework

Tao Ye, Hema T. Kaur, Shivkumar Kalyanaraman, and Murat Yuksel

***Abstract*—As the Internet infrastructure grows to support a variety of services, its legacy protocols are being overloaded with new functions such as traffic engineering. Today, operators engineer such capabilities through clever, but *manual* parameter tuning. In this paper, we propose a back-end support tool for large-scale parameter configuration that is based on efficient parameter state space search techniques and on-line simulation. The framework is useful when the network protocol performance is sensitive to its parameter settings, and its performance can be reasonably modeled in simulation. In particular, our system imports the network topology, relevant protocol models and latest monitored traffic patterns into a simulation that runs on-line in a network operations center (NOC). Each simulation evaluates the network performance for a *particular setting* of protocol parameters. We propose an efficient large-dimensional parameter state space search technique called "recursive random search (RRS)." Each sample point chosen by RRS results in a single simulation. An important feature of this framework is its *flexibility*: it allows arbitrary choices in terms of the simulation engines used (e.g., ns-2, SSFnet), network protocols to be simulated (e.g., OSPF, BGP), and in the specification of the optimization objectives. We demonstrate the flexibility and relevance of this framework in three scenarios: joint tuning of the RED buffer management parameters at multiple bottlenecks, traffic engineering using OSPF link weight tuning, and outbound load-balancing of traffic at peering/transit points using BGP LOCAL_PREF parameter.**

***Index Terms*—Black-box optimization, network performance management, network protocol configuration, on-line simulation.**

I. Introduction

TODAY's network protocols like BGP and OSPF were designed for one primary service: "best effort performance." Increasingly, network operators want to use the IP infrastructure for complex functions like deploying Virtual Private Networks (VPN), managing traffic within ASes to meet Service Level Agreements (SLA), and between ASes (at peering points) to optimize on peering agreements. Such operational optimization is performed by using "parametric hooks" in protocols that can be "tweaked" appropriately. However, the parameter setting process today is manual and widely considered a black art. Recent studies [1] and common knowledge [2], [3] is that the configuration of many protocols, such as BGP, is tough, error prone and is likely to get harder as the protocol is overloaded to serve more functions. There is an urgent need for back-end support tools for network management. This paper presents one tool aimed at this need.

In this paper, we propose a novel *on-line* simulation framework (OLS) to aid generic large-scale network protocol and parameter configuration. In the past, simulations (e.g., using ns-2) have been used in the design of new protocols, but not in on-line network management. A number of large-scale simulation tools are emerging in the community [4] that allow us to explore the latter possibility. Our framework formulates network protocol configuration as a "black-box" optimization problem over a feasible parameter state space (Fig. 1 depicts a potential usage scenario). A sample point in the state space corresponds to a network simulation that *evaluates* the performance of the protocol in terms of predetermined metrics and for the given set of parameters. The simulation also imports the current network topology and a digest of latest traffic patterns. This "black-box" approach allows considerable *flexibility* in terms of objectives of the desired optimization, and hence can be applied to a variety of protocols and configuration problems. Realistically, the OLS framework may be used as a "recommendation service" to suggest a variety of "good" parameter settings and illustrate the resulting impacts of the settings so that operators are better informed than their current manual procedures.

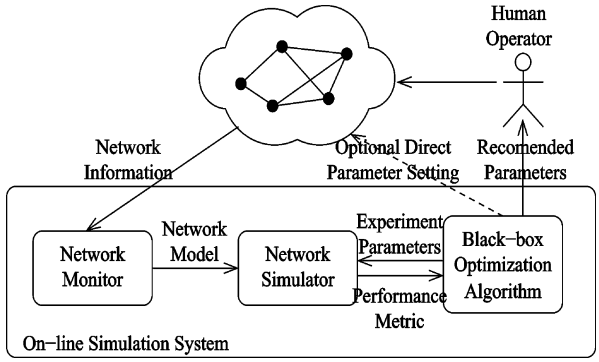

Fig. 1. On-line simulation framework for adaptive configuration of network protocols.

Manuscript received January 4, 2004; revised April 27, 2007, and June 2, 2008; published August 15, 2008 (projected); approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Towsley. This material is based upon work supported by DARPA Network Modeling and Simulation Program under Contract F30602-00-2-0537, the National Science Foundation under Grant NeTS-NR 0435259, and a Grant from AT&T Labs Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors. The work was done while authors were at Rensselaer Polytechnic Institute.

T. Ye is with One William Street Capital, New York, NY 10004 USA (e-mail: t_ye@yahoo.com).

H. T. Kaur is with Intel Corporation, Beaverton, OR 97006 USA (e-mail: hema.tahilramani@intel.com).

S. Kalyanaraman is with the IBM India Research Laboratory (IRL), Bangalore 560071, India (e-mail: shivkuma@gmail.com).

M. Yuksel is with the Department of Computer Science and Engineering, University of Nevada–Reno, Reno, NV 89557 USA (e-mail: yuksem@cse.unr.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2008.2001729

The key assumption of the framework is that the underlying network protocol performance is indeed sensitive to the chosen parameter set; and that the network topology, traffic and protocol can be reasonably modeled in simulation. On the long-term, our framework can leverage improvements in modeling of topology [5], [6], traffic [7], [8] and/or improvements in scalable network simulation [4] technology. To the best of our knowledge, our flexible black box approach is unique and the first of its kind as applied to IP network management. We discuss related work in later sections.

One of the crucial components of the OLS framework is an efficient parameter state space search algorithm, i.e., *a method to decide what simulations to perform*. The desired search algorithm is required to: **a)** be scalable to large-dimensional parameter state spaces; **b)** find "good" solutions quickly; **c)** be robust to noise (e.g., minor inaccuracies in modeling) in the function evaluations; and **d)** be able to automatically reject insignificant parameters (i.e., to which the protocol is insensitive). In order to provide the above desired combination of properties, we propose a new search algorithm, Recursive Random Search (RRS), which is completely based on random sampling and very efficient for such network optimization problems. In Section II, we will summarize RRS and why it can perform well on problems with the above-mentioned properties. Traditional search algorithms (e.g., genetic algorithms [9], multistart hill-climbing, tabu search [10] and simulated annealing [11]) can also be used as the parameter space search algorithm of the OLS framework; however, earlier work [12] comparatively showed that RRS performs significantly better than these traditional schemes on most of the optimization problems.

To demonstrate the effectiveness of this on-line simulation framework, we apply this framework to three scenarios: RED buffer management parameter turning (on a Linux testbed using SNMP interfaces), traffic engineering using OSPF link weight tuning, and outbound load-balancing by tuning BGP LOCAL_PREF attributes. Note that these examples illustrate only formulation process of network optimization problems and demonstrate the effectiveness and flexibility of this framework. The framework's application is not limited to these examples. For example, network operators may use other performance metrics or pose other black-box optimization questions to achieve different purposes.

The rest of this paper is organized as follows: Section II describes the features of network parameter optimization problems and presents a brief overview of the Recursive Random Search (RRS) algorithm. Then Sections II–V demonstrate how to formulate network parameter optimization problems and apply the on-line simulation to these problems. Section III investigates the application in adaptive tuning of RED. Section IV presents the application in traffic engineering by tuning OSPF link weights. Section V presents the application in outbound load balancing by tuning BGP. Finally, Section VI concludes this paper.

II. Network Parameter Optimization Problem

Network parameter configuration problems can be represented by the following equation:

$$\mathcal{C} = f(\mathcal{N}, p) \tag{1}$$

where $\mathcal{N}$ denotes network scenario, p the desired performance metric and $\mathcal{C}$ the parameter configuration of the concerned network protocol. Equation (1) calculates the required configuration $\mathcal{C}$ based on the desired performance metric p and the network scenario $\mathcal{N}$. Due to the complexity of the Internet, the analytical derivation of (1) is not realistic. However, with network simulation software, such as *ns* [13], *SSFNET* [14], it is possible to empirically examine network performance for a certain network configuration and scenario, i.e., establish the following empirical equation:

$$p = f^{-1}(\mathcal{N}, \mathcal{C}). \tag{2}$$

Based on this, for a certain network scenario $\mathcal{N}$ and a given parameter space of $\mathcal{C}$, an optimization algorithm can be employed to search for a good solution $\mathcal{C}_0$ which meets a certain performance objective p_0. With this black-box optimization approach, the problem defined in (1) can be empirically solved. This idea is the basis of the on-line simulation framework. Note that for network parameter optimization problems, traditional experiment design methods, such as, factorial design, are not applicable since they normally assume a relatively simple mathematical model and try to fit the problem into this model. Since little *a priori* knowledge is available, to formulate a proper model is very difficult.

Like optimization problems arising in many engineering areas, network parameter optimization can be formulated as (assume minimization): given a real-valued objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, find a global minimum $\mathbf{x}^*$,

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in D} f(\mathbf{x}) \tag{3}$$

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 and considered very hard to solve because of lack of *a priori* knowledge. In addition, since the objective functions are often nonlinear and multimodal, these problems 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. Many heuristic search algorithms have been proposed and used successfully in practice, such as, multistart hill-climbing [15], genetic algorithm [9] and simulated annealing [11]. The nature of heuristic algorithms is exactly that: "heuristic," i.e., time-bounded convergence to optima cannot be established for all classes of problems. However, heuristic algorithms are more general purpose compared to "approximation" algorithms that are used for specific classes of NP-hard problems. The *No Free Lunch Theorem* [16] has theoretically demonstrated that no matter what performance metric is used, no single optimization algorithm can consistently outperform the others in every class of problems. The inherent properties of the method have to be carefully investigated to perform efficient optimization. We will therefore examine the properties of our targeted network optimization problems below:

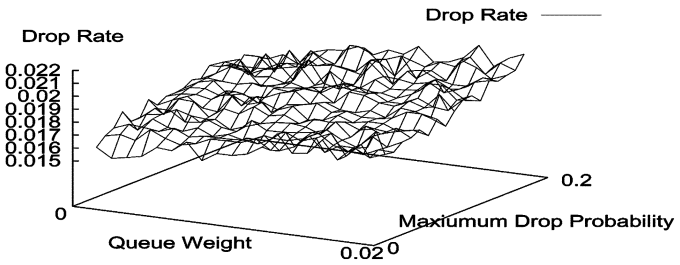

Fig. 2. An empirical objective function obtained with network simulation (RED buffer management).

A. Properties of Network Parameter Optimization Problems

We impose the following requirements on our search method:

High efficiency: The emphasis of the search algorithm should be on finding a better operating point within the limited time frame instead of seeking the strictly global optimum. Network conditions vary with time and the search algorithm should *quickly find better network parameters* before significant changes in the network occur.

High dimensionality of the parameter state space: AT&T's network has thousands of routers and links [6]. If all OSPF link weights of this network are to be configured, there will be thousands of parameters present in the optimization. High-dimensional optimization problems are usually much more difficult to solve than low-dimensional problems because of "curse of dimensionality" [15].

Noise is often introduced into the evaluation of the objective function since network simulations are used for function evaluations. Due to inaccuracies in network modeling and simulation, the resulting empirical objective function may be distorted by small random noises. Fig. 2 shows an example of two-dimensional empirical objective function obtained with network simulation (for RED parameter response). It can be seen that there exist many irregular small random fluctuations imposed on the overall structure.

Insignificant parameters: Not all parameters are important, but it is unknown a priori which parameters are important. If the search algorithm can automatically exclude these parameters from the optimization process, the efficiency of the optimization will be significantly improved.

"Globally convex" or "big valley" structure [17] may be present in the objective functions. That is, high-quality local optima tend to center around the global one and be close to each other, whereas low-quality local optima tend to be distributed further away from the global one. "Globally convex" structure appears in many practical optimization problems, especially in the situations when the objective function is affected by random noises. Boese [18] has demonstrated the existence of this structure in complex Traveling Salesman Problem (TSP) and graph bisection problem. The same structure has been found in circuit/graph partitioning and job-shop scheduling, etc. [19]. Leary [20] also confirmed that there exist similar "funnel" structures in molecular conformation problems where the potential energy from the forces between atoms is minimized.

The issues described above are common in many practical optimization problems [21]. For such class of problems, genetic algorithm [22] and simulated annealing [23], controlled random search [24], are the most common algorithms since they require little *a priori* information from the concerned problem and are generally applicable. However, these algorithms often lack efficiency (i.e., the property of giving "good results quickly"). In practice, they are often combined with *local search* techniques, such as, deepest descent and pattern search, to improve their efficiency. Since these local search techniques use fixed local structures to guide the search process, they are usually susceptible to the effect of noise [25]. For example, in pattern search, the wrong pattern may easily be derived if the samples for pattern exploration are corrupted by noise. Furthermore, for the objective function with "globally convex" structures, local methods also perform inefficiently since there exist a large number of low-quality local optima. For example, multistart local search algorithms may waste much effort on examining these low-quality local optima and essentially work like an inefficient random sampling.

B. Recursive Random Search Algorithm

Because of the disadvantages of traditional search algorithms, we have recently proposed the Recursive Random Search algorithm (RRS) [12] to meet the requirements of network parameter optimization. RRS is based on the high-efficiency feature of random sampling at initial steps. The idea is to use initial high-efficiency random samples to identify promising areas and then start recursive random sampling processes in these areas which shrink and realign sample spaces to local optima. RRS is an example of a multiscale random sampling technique. We have tested this algorithm on a suite of well-known and difficult benchmark functions [12]. The results have shown that in terms of *quickly* locating a "good" solution, RRS outperforms other search algorithms, such as multistart pattern search and controlled random search. The test results have also demonstrated that RRS is much more robust to noise than those local-search-based method. Furthermore, the inclusion of insignificant parameters in the objective function has little effect on the efficiency of RRS. To allow this paper to be self-contained, we present a description of the RRS algorithm. The detailed benchmark test results are provided in [12].

1) Initial Efficiency of Random Sampling: Given an measurable objective function $f(\mathbf{x})$ on the parameter space D with a range of $[y_{\min}, y_{\max}]$, we can define the *distribution function* of objective function values as

$$\phi_D(y) = \frac{m(\{\mathbf{x} \in D \mid f(\mathbf{x}) \leq y\})}{m(D)} \tag{4}$$

where $y \in [y_{\min}, y_{\max}]$ and $m(\cdot)$ denotes *Lebesgue measure*, a measure of the size of a set. For example, *Lebesgue measure* is area in a two-dimensional space, volume in a three-dimensional space, and so on. Basically, the above equation represents the portion of the points in the parameter space whose function values are smaller than a certain level y. $\phi_D(y)$ is a monotonously increasing function of y in $[y_{\min}, y_{\max}]$, its maximum value is 1 when $y = y_{\max}$ and its minimum value is $m(\mathbf{x}^*)/m(D)$ where $\mathbf{x}^*$ is the set of global optima. Without loss of generality, we assume that $f(\mathbf{x})$ is a continuous function and $m(\{\mathbf{x} \in D | f(\mathbf{x}) = y\}) = 0, \forall\, y \in [y_{\min}, y_{\max}]$, then $\phi(y)$ will be a monotonously increasing continuous function with a range of $[0, 1]$. Assuming a $y_r \in [y_{\min}, y_{\max}]$ such that $\phi_D(y_r) = r$,

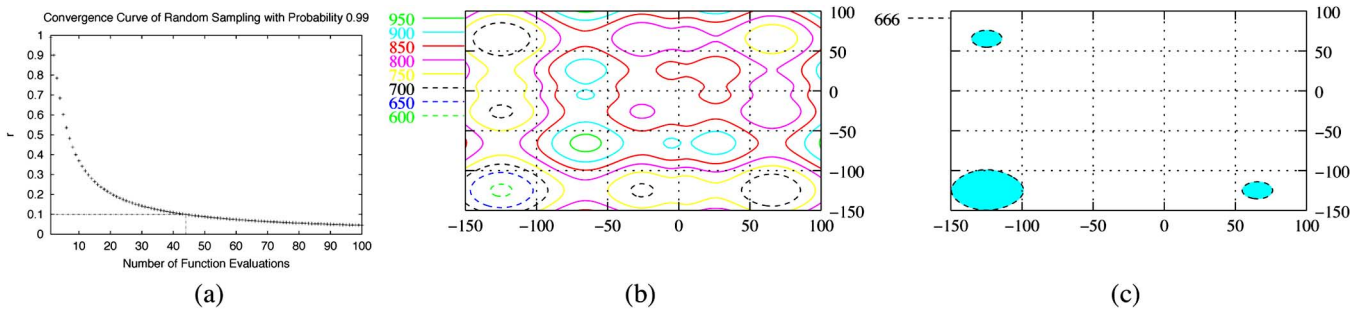

Fig. 3. (a) $A_D(r)$ of $x^n_{(1)}$ in random sampling with probability 0.99. (b) Contour plot of an objective function and (c) its region of $A_D(0.05)$.

$r \in [0,1]$, a *r-percentile* set in the parameter space D can be defined:

$$A_D(r) = \{\mathbf{x} \in D \mid f(\mathbf{x}) \leq y_r\}. \tag{5}$$

Note that $A_D(1)$ is just the whole parameter space D and $\lim_{\epsilon \to 0} A_D(\epsilon)$ will converge to the global optima. Suppose the sample sequence generated by n steps of random sampling is $\mathbf{x}_i, i = 1 \ldots n$ and $\mathbf{x}^n_{(1)}$ is the one with the minimum function value, then the probability of $\mathbf{x}^n_{(1)}$ in $A_D(r)$ is:

$$P\left(\mathbf{x}^n_{(1)} \in A_D(r)\right) = 1 - (1-r)^n = p. \tag{6}$$

Alternatively, the r value of the *r-percentile* set that $\mathbf{x}^n_{(1)}$ will reach with probability p can be represented as:

$$r = 1 - (1-p)^{1/n}. \tag{7}$$

For any probability $p < 1$, r will tend to 0 with increasing n, that means, random sampling will converge to the global optima with increasing number of samples. Fig. 3(a) shows the *r-percentile* set that n steps of random sampling can reach with a probability of 99%. We can see that *random sampling is highly efficient at initial steps since r decreases exponentially with increasing n, and its inefficiency is from later samples*. As shown in Fig. 3(a), it takes only 44 samples to reach a point in $A_D(0.1)$ area, whereas all future samples can only improve r value of $\mathbf{x}^n_{(1)}$ at most by 0.1.

2) Overview of Recursive Random Search: The basic idea of RRS is to maintain the initial efficiency of random sampling by "restarting" it before its efficiency becomes low. However, unlike the other methods, such as hillclimbing, random sampling cannot be restarted by simply selecting a new starting point. Instead we accomplish the "restart" of random sampling by *changing its sample space*, i.e., rescaling the search.

A stochastic search algorithm usually comprises two elements: *exploration* and *exploitation*. Exploration examines the macroscopic features of the objective function and aims to identify promising regions in the parameter space, while exploitation focuses on the microscopic features and attempts to exploit local information to improve the solution quickly. Many search algorithms, such as multistart type algorithms, *do not* differentiate regions using any statistical basis and hence may waste much time in trivial areas. In contrast, RRS attempts to identify a certain *r-percentile* set $A_D(r)$ and only start exploitation from this set. In this way, most of trivial areas will be excluded from exploitation and thus the overall efficiency of the search process can be improved. This can be illustrated by the example shown in Fig. 3.

The upper graph shows a contour plot of a two-dimensional multimodal objective function and the lower graph shows the set of $A_D(0.05)$. As shown in the figure, the function has many local optima; however, only three regions remain in $A_D(0.05)$ (shaded areas in the lower plot). Each of these regions encloses a local optimum and the one with the biggest size happens to contain the global optimum. It is desirable that the size of $A_D(r)$ region identified by exploration is as small as possible such that most of trivial areas are filtered out. On the other hand, its smallest size is limited by the efficiency of random sampling, i.e., it should be within the reach of initial high-efficiency steps of random sampling so that identifying a point in it will not take too long to lower the overall efficiency.

To identify a $A_D(r)$ area, RRS first takes a certain number of samples and uses the best one to decide the location of $A_D(r)$. It then goes on into recursive random sampling process by shrinking or realigning the sample space. In recursive random sampling, random sampling is performed for a number of times, if it fails to find a better point, the sample space is shrunk by a certain ratio. Otherwise, the sample space keeps its size unchanged, but moves its center to the new improved sample. This shrink-and-realign procedure is repeated until the size of the sample space decreases below a threshold. Then we identify another $A_D(r)$ and restart the above search process.

In contrast to most of the search algorithms, the RRS algorithm is built on random sampling. On the long run it performs like random search (the explore part of RRS), but biases better results early due to the multiscale (i.e., recursive) nature of the exploit phase in the algorithm. Since RRS performs the search process based on stochastic information on a certain sample area, therefore, its performance is less affected by noise. In addition, RRS is more efficient when dealing with the objective function with insignificant parameters. This is because that random samples will still maintain its uniform distribution in the subspace composed of only those important parameters, and hence effectively removes insignificant parameters from the optimization process. In this way, the efficiency of the search can be improved significantly. For the objective function with "globally convex" feature, RRS is able to detect the overall structure by its initial extensive sampling and then approach global optima with recursive sampling very quickly. These features have been empirically validated by the tests on a large suite of benchmark functions [12].

Uniform and simultaneous scaling of all parameters of a large-scale system might lead to unpredictable settings, if the response surface (i.e., system behavior as a function of parameters) is highly bumpy with large bumps in narrow regions of the hyperspace. RRS within the OLS framework could then be used for guidance rather than any automated settings. Further, the amount of incremental change in the system response relative to the current baseline could be incorporated as a penalty in the optimization formulation. Also, the results of global heuristic optimization may be similar to local optimization if there are dominant parameters, and probably useful in the cases where the operator does not know which subset of parameters matter or are less relevant.

3) Exploration: In the exploration phase of RRS, random sampling is used to identify a point in $A_D(r)$ for exploitation. The value of r should be first chosen. Based on this value and a predefined confidence probability p, the number of samples required to make $Pr\left(\mathbf{x}^n_{(1)} \in A_D(r)\right) = p$ can be calculated as (according to (6)): $n = \frac{\ln(1-p)}{\ln(1-r)}$. The algorithm uses the value of $f\left(\mathbf{x}^n_{(1)}\right)$ in the first n samples as the threshold value y_r and any future sample with a smaller function value than y_r is considered to belong to $A_D(r)$. In later exploration, a new $\mathbf{x}^n_{(1)}$ is obtained every n samples and y_r is updated with the average of these $\mathbf{x}^n_{(1)}$. Note that this calculation of y_r is not intended to be an accurate estimation of the threshold for $A_D(r)$, instead it only functions as the adjustment for the balance between exploration and exploitation. In other words, it is to ensure that on the average the exploration process will not continue for n samples and hence enter its low-efficiency phase.

In this exploration method, the confidence probability p should choose a value close to 1, for example, 0.99. The value of r decides the balance between exploration and exploitation and should be chosen carefully as discussed before. According to the current experience, we have used $r = 0.1$ and $p = 0.99$ in the algorithm, and with such values it only takes 44 samples to find a point for the estimation of y_r.

4) Exploitation: As soon as exploration finds a promising point $\mathbf{x_0}$ whose function value is smaller than y_r, RRS starts a recursive random sampling procedure in the neighborhood $N(\mathbf{x}_0)$ of $\mathbf{x_0}$. The initial size of $N(\mathbf{x}_0)$ is taken as the size of $A(r)$, i.e., $r \cdot m(D)$, where D is the original parameter space since $\mathbf{x}_0$ belongs to $A(r)$ with a high probability. Currently a simple method is used to construct $N(\mathbf{x}_0)$: assume the parameter space D is defined by the upper and lower limits for its ith element, $[l_i, u_i]$, the neighborhood of $\mathbf{x}_0$ with a size of $r \cdot m(D)$ is the original parameter space scaled down by r, i.e., $N_{S,r}(\mathbf{x_0}) = \{z \in S \mid |z_i - x_{0,i}| < r^{1/n} \cdot (u_i - l_i)\}$, where $x_{0,i}$ is ith element of $\mathbf{x}_0$ and z_i ith element of $\mathbf{z}$. With this new sample space $N_{S,r}(\mathbf{x_0})$, random sampling is continued. And then based on the obtained samples, the sample space is realigned or shrunk as exemplified in Fig. 4 until its size falls below a predefined level s_l, which decides the resolution of the optimization.

III. Adaptive Tuning of RED

The on-line simulation framework can be applied to a wide range of network protocols. This section will present one of these applications, i.e., the tuning of Random Early Detection (RED) active queue management (AQM) algorithm [26]. The basic idea of RED is to detect the inception of congestion and notify traffic sources early to avoid serious congestion. RED avoids the problem of global synchronization, maintains low average queueing delay and provides better link utilization than DropTail [26]. However, it is well known that the performance of RED is highly sensitive to its parameter settings and the there is a significant performance penalty when RED is misconfigured [27]–[29]. Some general guidelines for setting RED parameters have been proposed [26], [30]. Adaptive versions of RED have also been proposed focussing on one of RED's parameters [27]. However, the effectiveness of these methods in complex network scenarios is still under investigation. Rather than relying on simplified models or intuition, the OLS framework exploits the advantage of network simulation technique and formulates the optimal configuration of RED as a black-box optimization problem. Also, the OLS framework gives the significant additional capability of tuning multiple REDs (see Section III-D) at the same time for a common goal, which cannot be attained by techniques local to one of the RED queues in the system.

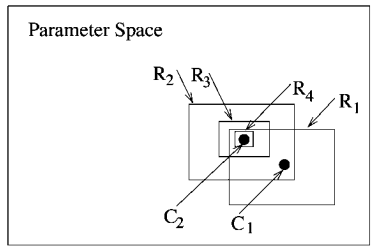

Fig. 4. Shrink and realign process.

An alternate approach to combat this issue is to periodically reconfigure any subset of RED parameters (aided by our online simulation framework and RRS search technique) to automatically adapt to prevailing network conditions. This technique operates completely in the management plane (through SNMP) and does not require modification of router hardware/software (e.g., new AQM schemes require router upgrades at all key routers).

A. Problem Formulation

RED uses the average queue size $\bar{q}$ as an indicator of the congestion extent and determines the packet drop rate accordingly.

As shown in Fig. 5(a), the instantaneous queue size q is sampled at every packet arrival and then passed through a low-pass filter to remove transient noises. Based on the smoothed average queue size $\bar{q}$, the drop probability P is calculated with a control function $P = f(\bar{q})$. The arriving packets are randomly dropped (or marked) according to this probability P. Traffic sources react to these drops and adjust offered load r accordingly. Therefore, RED is mainly designed to work with TCP traffic sources which are responsive to packet drops and it will not work well in the cases like UDP traffic or short-life HTTP traffic.

In the equilibrium state, the increase rate of TCP traffic should be approximately equal to its decrease rate caused by packet drops and thus the offered load will stabilize around a certain level. If this equilibrium state is achieved while maintaining a certain queue size, the link utilization will be close to 1, i.e., the offered load will stabilize around the bottleneck capacity. The

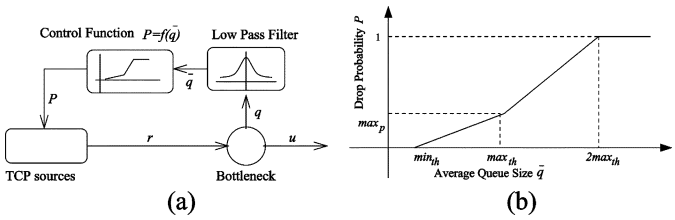

Fig. 5. Random Early Drop (RED) Queue Management Scheme: (a) major components of the scheme; (b) RED control function $P = f(\bar{q})$.

goal of RED is to search for an appropriate packet drop rate to counteract the increase of offered load.

In RED configuration, there are four parameters which we vary in the optimization process. Among these parameters, the moving average weight w_q determines the cut-off frequency of the low-pass filter, and the other three parameters, i.e., minimum threshold $\min_{th}$, maximum threshold $\max_{th}$ and maximum drop probability $\max_p$, determine the control function $P = f(\bar{q})$. In the standard version of RED, the control function is determined by the parameters as illustrated in Fig. 5(b). This function indicates that the drop probability can be calculated according to the average queue size. Also, the equilibrium drop probability depends on two factors: (i) the offered load increase rate and (ii) the granularity of congestion notification (i.e., the load decrement caused by one packet drop). Due to the fast recovery and fast retransmission mechanisms, each drop will cause a TCP source to decrease its sending rate by half. Therefore, the granularity of the congestion notification is determined by the average TCP sending rate. For example, when the average sending rate is *large*, each packet drop will cause a large decrease in offered load since a *small* number of TCPs share a bottleneck, and *vice versa*. Further, the increase rate of offered load is different in different scenarios, e.g., the increase rate will be large when there are many TCP flows or the round-trip time is short. As a result, to maintain a stable equilibrium operation point, the drop probability should be adjusted according to the varying network scenarios. If the control function remains unchanged, the average queue size has to be varied to obtain the new equilibrium drop probability. Therefore, to keep the average queue size stable around a certain level in varying conditions, the control function has to be adjusted accordingly, i.e., the three parameters determining $f(\bar{q})$ should be dynamically tuned.

w_q controls the cut-off frequency of the low-pass filter. The cut-off frequency should be high enough to detect manageable traffic variations, while low enough to filter out transient traffic oscillations which can not be effectively controlled by RED. For example, the oscillation within one round-trip time rtt should be removed. Therefore, the optimal w_q is usually related to rtt. In addition, since the average queue size is calculated at every packet arrival instead of a constant interval, different link speeds will result in different packet arrival intervals and hence affect the cut-off frequency of the low-pass filter. Consequently, the optimal w_q is also dependent on the link speed.

B. Optimization Objective

For a buffer management algorithm, there are two main performance metrics, i.e., link utilization and average queue size. The main objective of RED is to *maintain a high utilization while keeping a low average queue size* [26]. However, optimizing one of the performance metrics may compromise the other. For example, a high link utilization can always be obtained by increasing $\min_{th}$ or decreasing $\max_p$, hence virtually increasing the average queue size. On the other hand, a low average queue size can be obtained by decreasing $\max_{th}$ or increasing $\max_p$. However, this obviously will cause underutilization of the link. Therefore, an appropriate tradeoff has to be made to reflect the requirement of network operators. This is essentially a multiobjective optimization problem and corresponding techniques should be employed to convert it into a tractable single objective problem.

One classic multiobjective optimization technique is to optimize the weighted average of the performance metrics. The weights for different metrics reflect the quantitative tradeoff among them and are critical to the effectiveness of optimization results. However, the weights are normally difficult to determine. Another common technique is to define the lower limits for less significant metrics, and only optimize the most important one with the restriction that the other metrics are not below their limits. In this paper, instead of using traditional multiobjective optimization techniques to directly work on link utilization and queueing delay, we have proposed a performance metric whose optimization will cause RED to settle in a equilibrium status and hence achieve high utilization and low queueing delay.

As mentioned above, in the equilibrium status, the average queue size of RED stabilizes around a certain level. When traffic pattern changes, the equilibrium point may also shift which makes the average queue size move around. When the average queue size varies beyond the range what RED can control, RED will become unstable, i.e., the queue status oscillates between full and empty [27]. This not only causes end users to experience significant delay jitters, but also results in link underutilization. Therefore, it is important to keep the average queue size of RED stable at a target level, such as the middle between $\min_{th}$ and $\max_{th}$. In consideration of this, we define the performance metric to be optimized as:

$$m = \frac{\sum_{i=1}^{N} (\bar{q}_i - q_0)^2}{N} \tag{8}$$

where q_0 is the expected average queue size predefined by network operators, $\bar{q}_i$ is the periodic sample of the average queue size and N is the number of samples. This metric essentially calculates the variance of the average queue size relative to q_0 over a certain period of time. When the equilibrium level of RED is far from the expected level, m will be large. Or when RED is misconfigured and hence the equilibrium cannot be reached, the queue size will oscillate substantially, also resulting in a large m. Therefore, minimizing m will cause RED to avoid both situations and always maintain an equilibrium around q_0. Thus, high link utilization and stable queueing delay can both be achieved.

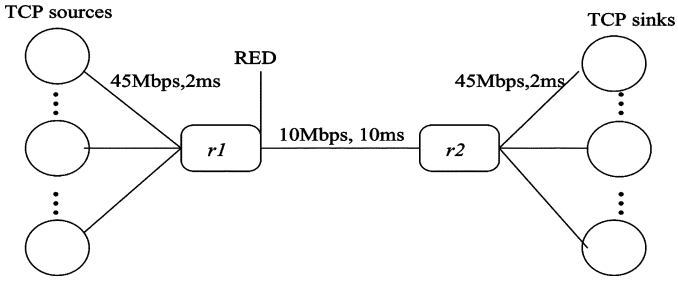

Fig. 6. Network topology for RED tuning simulation.

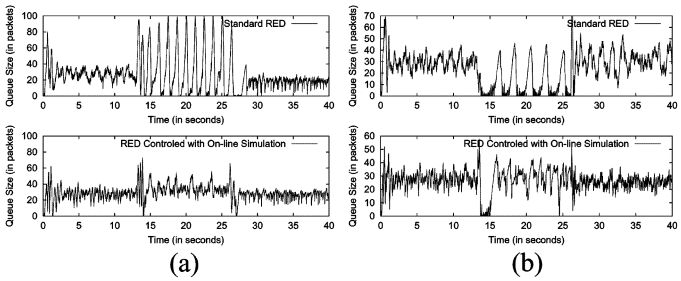

Fig. 7. Comparison of standard RED (upper graph) and RED controlled by on-line simulation (lower graph): (a) varying traffic load; (b) varying round-trip time.

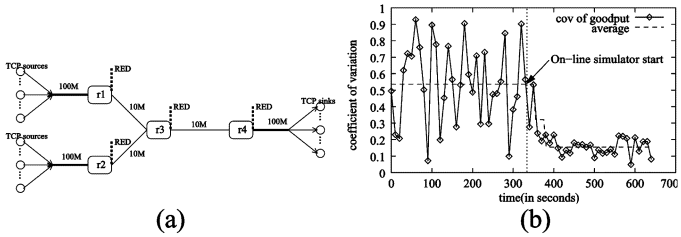

Fig. 8. Tuning multiple RED queues for optimizing coefficient of variation (CoV) of goodputs in a Linux-based testbed topology. (a) Testbed topology. (b) CoV of goodputs.

C. Simulation Results: RED Parameter Tuning

The simulations of on-line RED tuning are performed for varying traffic load and round-trip time, two major factors affecting RED performance. The network topology used in the simulations is shown in Fig. 6. We used ns [13] as the simulation tool. Infinite FTP traffic between TCP sources and sinks is generated to build up a queue at router $r1$. RED is configured on $r1$ to manage a 100-packet buffer. Each simulation runs for 40 seconds and network conditions are changed twice during the simulation. We will compare the performance of standard RED and RED controlled with the OLS framework under changing network conditions. According to the common guideline of RED parameter setting, we use $\min_{th} = 15$, $\max_{th} = 45$, $\max_p = 0.1$, $w_q = 0.002$ for standard RED. In the case of RED supported by the OLS framework, we vary these four parameters by RRS according to the changing network conditions.

We define an expected average queue size of 30 packets and the objective is to maintain the equilibrium status of RED around this level. We also assume that the on-line simulation system can promptly detect the change in network conditions and trigger the optimization process of RED parameters. In reality, this can be achieved by monitoring the change in performance metrics or analyzing traffic statistics directly. Our traffic model is not fully representative of reality, where several self-similar sources along with various application sources should exist. However, this is expected to have a minimal impact on the overall end-result. Thus, for simplicity we continue with this FTP-based traffic model.

First we test the tuning of RED to varying traffic load. The number of TCP flows in the simulation starts with 16, then increases to 64 after around 13 seconds, and finally decreases to 4 after another 13 seconds. The instantaneous queue sizes of standard RED and RED with on-line simulation control are shown in Fig. 7(a). The upper graph shows that for the standard RED, when the traffic load increases beyond the control of current RED parameter setting, the equilibrium status may be broken and the queue remains in a very unstable status where large oscillations between full and empty queue persist. On the other hand, when the traffic load decreases (e.g., after 26th second in Fig. 7(a)), the level of the queue becomes lower than the target level. Such lower queue levels might be undesirable by the operators if the target utilization is 100%. The lower graph shows that when dynamically tuned, RED always maintains an equilibrium status where the queue size remains very stable and the utilization is close to 100%. It is remarkable that even though the traffic load changes, OLS framework finds the appropriate RED parameters such that the target queueing level is achieved. Such capability can be leveraged for achieving not only a target link utilization and queue level but also a target per-hop queuing delay, which might be of interest for further studies.

Then we test the tuning of RED to varying round-trip time. The simulation starts with 16 TCP flows and each with a round-trip time of 18 ms (not including queueing delay). After 13 seconds, the rtt of these flows is increased to 170 ms. And after another 13 seconds, the rtt is reduced to around 2 ms. The instantaneous queue sizes of standard RED and RED with on-line simulation control are shown in Fig. 7(b). The upper graph shows that when rtt is increased to 170 ms, the equilibrium of standard RED queue is again broken and the queue keep oscillating between full and empty status. And when rtt is reduced to 2 ms, although the queue does reach an equilibrium status, there still exist big variations in queue size. As shown in the lower graph, the dynamically tuned RED eliminated these problems.

D. Real Network Experiment for Optimization of Multiple RED Queues

As mentioned before, we have implemented a prototype of the on-line simulation system in Linux. This section presents a real network experiment which applies this prototype to RED parameter configuration in a Linux testbed. The testbed topology is shown in Fig. 8(a) and ns is adopted for network simulation in the on-line simulation system.

There are 4 Linux routers in the network and each of them is configured with a RED queue which is monitored and controlled by the on-line simulation system through SNMP. Again, infinite FTP sources are used to generate network traffic. Note that in this test we will try to tune the parameters for all four RED concurrently. Since optimizing each RED individually may compromise the performance of the others, we have taken

all RED queues as a single black-box system with a total of 16 parameters. Consequently, a global performance metric has to be defined based on the objective of network operators. If using ISP-based metrics, such as utilization and queueing delay, a multiobjective technique has to be employed to combine the metrics from every RED router. There are several possible ways of defining a combined single metric for these RED queues such as (i) average total queueing delay experienced by all flows traversing these queues, (ii) average total goodput of all traversing flows, and (iii) fairness among all traversing flows. We have selected the last one as our metric and quantified fairness of the network to the traversing user flows. Specifically, we use the Coefficient of Variation $\frac{\sigma}{\mu}$ of goodputs for all TCP connections traversing the RED queues. This CoV metric measures the variation of TCP goodputs, and quantifies how different transport performance is being experienced by each user in comparison to the other users' TCP flows. Intuitively, lower CoV means that users are experiencing similar TCP goodput performance which means a fairer service to the users. One important point to note here is that the CoV is useful only when there is high traffic load, i.e., high link utilizations. When utilizations are very low, the optimization process might lead to RED settings where CoV is zero and the utilization is almost zero, as the CoV would be zero when there is no traffic. Our assumption here is that there is always high traffic loads. A network operator has the option of choosing different metrics for different real cases, e.g., goodput when utilization is low and CoV when the utilization is high.

During the experiment, a number of TCP flows are generated from one side to the other. The goodputs of these TCP flows are collected periodically from TCP sinks. The Coefficient of Variation (CoV) of the goodputs is calculated and plotted as a function of time as shown in Fig. 8(b). In the beginning, the parameters of these RED queues are set to *random* values to represent a misconfigured system, which results in a large unfairness between TCP flows, i.e., a high average CoV value and large oscillations. At the 325th second, the on-line simulator starts and detects the misconfiguration of REDs. The good configuration with a performance better than a predefined threshold is quickly found within seconds and the network is reconfigured. This results in an immediate performance improvement as shown in the plot: the average of CoV drops to a very low value and the instantaneous CoV curve becomes stable over time.

IV. Traffic Engineering by Tuning OSPF Link Weights

In this section, we will present another application of the OLS system, i.e., tuning OSPF routing protocol for traffic engineering. The term "traffic engineering" refers to a broad set of capabilities where traffic flows are mapped onto a network topology to meet a variety of performance objectives specified by operators. In the current Internet, IP traffic is mapped onto the network by shortest-path routing protocols, such as, OSPF. In such protocols, the common links along the shortest path between multiple nodes may become congested while the links on longer paths may remain idle. Many traffic measurements [31], [32] confirm this behavior.

Two main approaches have been taken to solve the intra-domain traffic engineering problem. One approach is to deploy the MPLS (multiprotocol label switching) technology which is not constrained by the shortest path nature of routing. Another approach is to adjust the link weights of the existing network (running OSPF) such that the OSPF routing with these link weights leads to desired routes. For example, traffic-sensitive routing methods adapt link weights to reflect the local traffic conditions on a link or to avoid congestion [33], [34]. However this method is unstable and leads to frequent route changes (see [35]). These drawbacks are alleviated in [36] where the OSPF configuration is modeled as a black-box optimization problem and a local search algorithm is used. The authors have chosen a heuristic cost function which is piecewise linear with offered load.

In our example, we have chosen the total packet drop rate in the network as the performance metric since it is a more accurate indicator of congestion and affects TCP performance directly. To indicate the flexibility of our framework, we replace the simulation of queues with a GI/M/1/K queuing model to calculate the packet drop rate (much faster computationally). The subsequent sections present a case for the GI/M/1/K model. When calculating the drop rate, the mean and variance of the offered load are both be considered (compared to the average offered load used in [36]).

A. The Objective Function

Our goal for OSPF configuration is to minimize the packet drop rate in the network for a given mean and variance of the aggregate demands between each source and destination routers. Let us consider a network represented by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ and $\mathcal{L}$ represent respectively the set of routers and links in the network. Each link $l \in \mathcal{L}$ has bandwidth denoted by B_l and a buffer space of K_l packets. We assume that packets arriving when the buffer space at a link is full are dropped and there is no other active queue management algorithm running at the routers. In addition to the knowledge of bandwidth and buffers at all the links, we assume that an estimate of the mean and variance of the aggregate demand from each source s to destination t is known. Let $\mathcal{D}$, $\mathcal{V}$ denote the mean and variance matrix of the estimated aggregate demand. In practice, all such information can be obtained using the tools described in [8], [37].

As a first step towards formulating the OSPF optimization problem, we first derive the drop probability for one link based on the offered load. We use a GI/M/1/K queuing model to formulate the link drop probability [38]. By using this link-based drop probability model, we, then, formulate the optimal general routing problem in Section IV-B, which aims to optimize the overall packet drop rate for the network. Note that the OSPF optimization problem is just the optimal general routing subject to the shortest path constraint.

1) The Optimal General Routing: The optimal general routing represents routing where there is no limitation on the way a flow is split among multiple paths available between a source and destination [36]. It is the best that can be achieved by carefully setting up multiple Label Switched Paths (LSPs) in

MPLS. Using link packet drop probabilities, we can formulate the optimal general routing problem as:

$$\Phi = \sum_{l \in \mathcal{L}} \lambda_l P_l \tag{9}$$

where λ_l is the arrival rate for link l and P_l is its drop probability. This is a constrained optimization problem with the flow constraints at each router j for each demand $\mathcal{D}(s,t)$ between source s and destination t. If $f_l^{(s,t)}$ denotes the fraction of the demand $\mathcal{D}(s,t)$ on link l, then the flow balance constraints are given by

$$\sum_{i:(i,j)\in\mathcal{L}} f_{(i,j)}^{(s,t)} - \sum_{i:(j,i)\in\mathcal{L}} f_{(j,i)}^{(s,t)} = \begin{cases} -\mathcal{D}(s,t), & \text{if } j = s \\ \mathcal{D}(s,t), & \text{if } j = t \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

The mean packet arrival rate to a link l, λ_l, is given by

$$\lambda_l = \sum_{(s,t)\in\mathcal{N}\times\mathcal{N}} f_l^{(s,t)}. \tag{11}$$

We use Generalized Exponential (GE) to model the inter-arrival times of the traffic on the links. The parameter $p^{(s,t)}$ for the GE process used to fit the demand $\mathcal{D}(s,t)$ is given according to

$$p^{(s,t)} = \frac{2\mathcal{D}(s,t)^2}{\mathcal{D}(s,t)^2 + \mathcal{V}(s,t)}. \tag{12}$$

Let $r_l^{(s,t)}$ denote the probability with which the demand $\mathcal{D}(s,t)$ is sent on link l. Then $r_l^{(s,t)}$ is given by

$$r_l^{(s,t)} = \frac{f_l^{(s,t)}}{\mathcal{D}(s,t)}. \tag{13}$$

Let $p_l^{(s,t)}$ denote the parameter p of the GE process after splitting the demand $\mathcal{D}(s,t)$ with probability $r_l^{(s,t)}$. Then $p_l^{(s,t)}$ denotes the parameter p of the GE process representing the flow $f_l^{(s,t)}$. The parameter $p_l^{(s,t)}$ is given by (see [39, Sec. 1.4], for more details):

$$p_l^{(s,t)} = \frac{p^{(s,t)}}{p^{(s,t)}\left(1 - r_l^{(s,t)}\right) + r_l^{(s,t)}}. \tag{14}$$

The total offered load on link l is given by λ_l (11), the parameter p of the associated GE distribution may be obtained by merging the flows $f_l^{(s,t)}$ going through l. If p_l denotes the parameter p of the GE process associated with the aggregate traffic on link l, then p_l is given by

$$p_l = \lambda_l \left(\sum_{(s,t)\in\mathcal{N}\times\mathcal{N}} f_l^{(s,t)} p_l^{(s,t)} \right)^{-1}. \tag{15}$$

If ρ_l is equal to $\frac{\lambda_l p_l \bar{X}}{B_l}$, then, using GI/M/1/K link drop probability formulation [38], the probability of packet dropped at link l is given by

$$P_l = \frac{(p_l - \rho_l)(\rho_l + 1 - p_l)^{K_l}}{1 - (\rho_l + 1 - p_l)^{K_l+1}}. \tag{16}$$

The optimal general routing problem is given by (9), subject to the constraints given by (11), (12), (13), (14), (15) and (16). It may be noted that we are casting the traffic according to the routing in order to obtain the mean and variance of the total offered traffic to each $l \in \mathcal{L}$. However, we are not iterating to obtain the equilibrium traffic parameters. Essentially, we are using the upper bound on the packet drop probability in (9).

2) Comparison of Search Schemes: In this section, we present the results of comparison of the RRS with the local search scheme proposed in [36]. In optimization literature, the comparison between algorithms is usually done in terms of the number of function evaluations instead of the absolute time taken to find a "good" parameter setting. This is done because the computation time is considerably dependent on many other factors, such as, implementation efficiency, testing platform, and compiler. Assuming that the main computation time is spent for function evaluations, the number of function evaluations is a more appropriate performance metric under the assumption that the computation time per function evaluation is approximately the same for both schemes. Note this assumption is not exactly true in the context of our problem, where one function evaluation represents one optimization metric computation for a specific set of link weights. In [36], authors have used incremental shortest path computations to improve the speed of search as very few link weights change from one iteration to the next which is reported to have 15% improvements on an average. In spite of this, we use the number of function evaluations as our algorithm performance metric.

It should be noted that even after taking the 15% improvement for the local search scheme of [40] into consideration, our algorithm is significantly faster. Loosely, we refer to the number of function evaluations required to obtain a "good" parameter setting as the speed of convergence. A "good" parameter setting has been defined as the OSPF link weight setting that give metric value lower than that by setting all link weights equal to one (called unit OSPF weights). This definition is a simple setting used for the sake of comparison. A "good" parameter setting may have been defined alternatively as the link weight setting to achieve performance metric equal to, say, 80% of the unit OSPF.

By taking the packet drop rate metric as defined in (9), Fig. 9 shows the comparison results of the optimization convergence speed. The results clearly show that RRS significantly outperforms the local search algorithm proposed in [36]. Table I shows that for the packet drop rate metric, RRS took 70% or fewer function evaluations to obtain a "good" OSPF link weight setting.

B. Optimization of OSPF Weights Using On-Line Simulation

The general optimal routing problem, where the objective function is completely defined by (9)–(16), may possibly be solved for $f_l^{(s,t)} \forall\, l \in \mathcal{L}$ by using some nonlinear programming techniques. However, under constraints of OSPF routing, the relation between the link weights and optimization metric can no longer be analytically defined. In [40], authors have proved that it is NP-hard to find OSPF link weight settings for an optimization metric piecewise linear in offered load. It is straightforward to show, by proceeding along the same lines, that our problem,

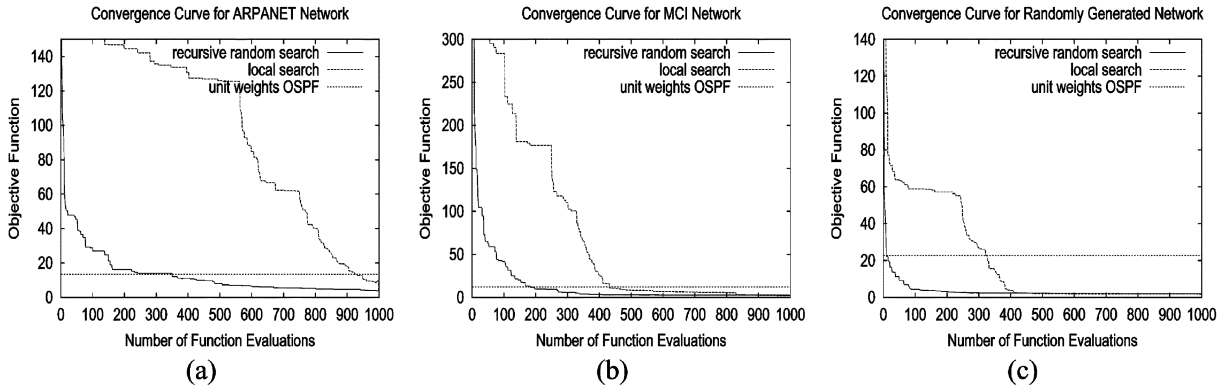

Fig. 9. The convergence curve of the total packet drop rate for (a) ARPANET; (b) MCI; (c) randomly generated network topology.

TABLE I
NUMBER OF FUNCTION EVALUATIONS NEEDED TO OBTAIN A "GOOD" PARAMETER SETTING FOR THE PACKET DROP RATE METRIC GIVEN IN (9)

Scheme	ARPANET	MCI	EXODUS
Local Search	882	469	372
RRS	210	125	54
Improvement	76.1%	73.3%	85.5%

i.e., minimize the packet drop rate given by (9), is also NP-hard. For such NP-hard problems, heuristic optimization algorithms (like our proposed RRS algorithm) are an attractive method to search for approximate solutions.

The optimal routing in OSPF can be formulated as the following "black box" optimization problem:

$$\min \Phi(\mathbf{w}) \tag{17}$$

where $\mathbf{w}$ is the vector of network link weights and $\Phi(\cdot)$ the objective function, which is unknown prior to the calculations of (9). The traffic load λ_l on a link l requires knowledge of shortest-path routing for a given traffic matrix, which defines edge-to-edge traffic rates of the ISP's network. Basically, in order to obtain the value of Φ for a given OSPF weight setting, we run modified Floyd Warshall's algorithm (modified to obtain equal cost paths also) to obtain the routing. Then, through the calculated shortest-path routes, the traffic is mapped on individual links to obtain parameters of the aggregate packet arrival process and drop probability for every link $l \in \mathcal{L}$ using (11), (12), (13), (14), (15) and (16). Finally the value of Φ may be calculated by (9).

C. Simulation Results: OSPF Parameter Tuning

We have considered three network topologies to demonstrate our results. In these topologies, each link is assumed to consist of two simplex link whose weights may be set independently. Two are well-known ARPANET topology and MCI topology. The ARPANET topology consists of 48 routers and 140 simplex links, and the MCI topology 19 routers and 62 simplex links. We also performed the simulation on a real large-scale ISP network topology, i.e., EXODUS network, obtained from Rocketfuel project [6]. This topology includes 244 core routers from EXODUS network and 1040 simplex links.

In the simulations, random amount of traffic was sent from every node to every other node in the network. This random traffic was generated using the method outlined in [36]. For each node u, two random numbers are generated $O_u, D_u \in [0,1]$. For each pair of nodes (u,v) another random number $C_{(u,v)} \in [0,1]$ was generated. If Δ denotes the largest Eucledian distance between any pair of nodes and if α denotes a constant, the average demand between u and v is given by

$$\mathcal{D}(u,v) = \alpha O_u D_v C_{(u,v)} e^{\frac{-\delta(u,v)}{2\Delta}}$$

where, $\delta(u,v)$ denotes the Eucledian distance between the nodes u and v. This method of generating random traffic (the term $e^{\frac{-\delta(u,v)}{2\Delta}}$) ensures more traffic for source destination pairs that are closer to each other. Since a product of three random variables is taken to generate the demands, there is actually a large variation in the traffic demands. The ratio of square of mean to the variance was assumed to be a uniformly distributed random variable in $[0,1]$. The mean and variance of the traffic demands are generated using the above procedure. All the links in the network have 1 Mb/s bandwidth with a buffer size of 50 packets. The packet size was chosen to be exponentially distributed with mean packet size of 200 bytes.

We used ns [13] to simulate the real network running OSPF. The traffic in the network was generated with the method described above. Every 200 seconds the traffic pattern (the mean and variance of demand matrix) was changed to introduce a dynamic scenario. The traffic generator is implemented over UDP to generate bursty traffic with Generalized Exponential (GE) inter-arrival distribution. In the simulation, we assume the online simulation (OLS) has a complete knowledge of necessary network information, such as, traffic demands, network topology, etc. Whenever a change of traffic pattern happens, OLS performs the optimization procedure for a certain time to obtain a good OSPF link weight setting. If the optimized setting is better than the original, it will be deployed at 100 seconds after the traffic change. The 100-seconds time difference is used because we want to observe the performance difference between before optimization and after optimization. Note that here we assume the running time of the optimization process is less than the traffic change period, i.e., the optimization has been finished at 100 seconds after the traffic change. In our simulation, the optimization procedure typically finds a better solution with a few hundred to a few thousands of function evaluations (depending on the size of the network), which can be interpreted to a computation time of minutes to hours when using a single Pentium III

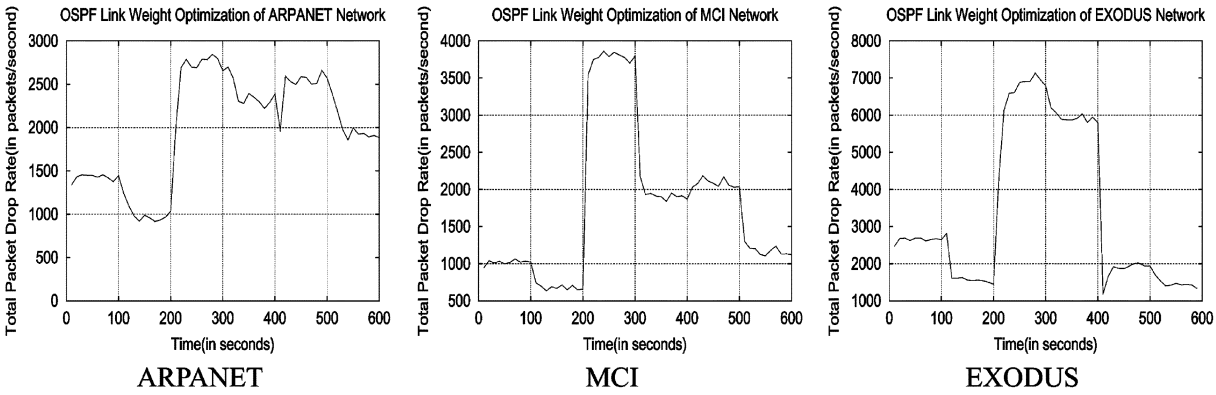

Fig. 10. Simulations for adaptive configuration of OSPF link weights: Traffic pattern changes (the mean and variance of the demand/traffic matrix) at 0th, 200th, and 400th seconds. After the traffic pattern change, the OLS framework calculates new OSPF links optimized according to the new traffic pattern. The OLS-optimized OSPF weights are deployed 100 seconds after the traffic pattern changes, i.e., they are deployed at the 100th, 300th, and 500th seconds.

class PC. This time can be further reduced by performing the optimization on a more powerful computer or multiple computers.

The actual packet drop rates are collected during the simulation for all the traffic sinks in the network and then summed together to get the total packet drop rate. Fig. 10 shows total packet drop rate in the network as a function of time. The maximum improvement in packet drop rates are 31.8%, 60.2%, and 35.7% for ARPANET, MCI, and EXODUS topologies respectively. Note that more or less improvements may result depending on the topology and traffic conditions. We observe that OLS can demonstrate improvements of the order of 30–60% in the total drop rate.

V. Outbound Load Balancing in BGP Environment

This section describes another application of the online simulation (OLS) system i.e., tuning of Border Gateway Protocol (BGP) routing for load balancing of outbound traffic, to show the breadth of our approach's applicability. Again, the application described here is only one of many possible ways of formulating the BGP outbound load balancing problem.

Inter-domain traffic engineering (TE) has mainly focused on the scenarios with multihomed Autonomous Systems (AS), where load-balancing of the in-bound/out-bound traffic between adjacent ASes using BGP attributes (e.g., MED, LOCAL_PREF, AS_PATH) is possible [41]. BGP routing decisions are made by a series of policy filters. Usually an AS may use the shortest AS path but this may lead to unbalanced load distribution among the multiple outbound interfaces.

BGP provides only some simple capabilities for TE between AS neighbors. The MED attribute can be used by an AS to inform its neighbor of a preferred connection (among multiple physical connections) for inbound traffic associated to a particular address prefix. Similarly, the AS_PATH attribute has also been used to achieve TE objectives, i.e., ASes can "stuff" or "pad" AS_PATH with the same AS number to increase its announced length. Another way used to achieve some inter-domain TE is to subvert the BGP-CIDR address aggregation process, i.e., an AS may extract some specific class prefixes (also known as "more-specifics") or de-aggregate them from the classes prefixes, and readvertise these specific prefixes to other ASes instead of classless addresses. One way to avoid subverting CIDR aggregation (shown in our past work [42]), in the case of multihomed *stub AS*, is by mapping the inbound load-balancing problem to an address management problem. Alternatively, AS neighbors may agree on BGP community attributes [43] (that are not readvertised) to specify TE.

We focus on selecting an outbound direction by means of the LOCAL_PREF attribute, as it is an attribute set by network operators typically by hand. Network operators use the LOCAL_PREF attribute locally within the AS to prefer an outbound direction for a chosen destination prefix, AS, or exit router. LOCAL_PREF holds the highest priority in the policy filter hierarchy, and therefore has the highest impact on the inter-domain TE performance. Recent work [3] observed that it is possible to adjust traffic distribution over outbound links by changing LOCAL_PREF of some "hot-prefixes" and shifting them away from congested links. However, the problem of unbalanced traffic distribution still remains. That is, the problem of "shifting" these hot-prefixes to achieve load balancing is an NP-hard problem (see later sections). Our approach is to apply our OLS framework to heuristically tackle this NP-hard problem and perform automatic outbound load balancing.

A. Granularity of Traffic Demands

Given a certain outbound traffic demand, load balancing aims to split this traffic demand and distribute them evenly among outbound links. Usually, the traffic demand can be divided into a number of traffic flows. In the finest granularity, a traffic flow is determined by the source and destination IP addresses and the port number. In a coarse granularity, a traffic flow can be identified by the source and destination AS-pair. Internet measurements have shown that traffic aggregates based on destination prefixes in the routing table are more suitable for load balancing [31] since they are relatively stable through the day and on per-hour time scales. We have used this granularity for defining a flow in our load balancing scheme. In other words, the traffic demand is split into flows at the level of per destination-prefix.

A typical BGP routing table consists of thousands of destination prefix entries. It will be very complex to work with such a large number of traffic flows. However, many traffic measurements [31], [32] have demonstrated the existence of so-called elephant and mice phenomenon. That is, a small number of traffic streams, known as *elephants*, generate a large portion of

total traffic whereas a large number of streams, *mice*, generate a small portion of total traffic. For example, it has been found that the top 9% of flows between ASes account for 86.7% of the packets or 90.7% of the bytes transmitted [32]. Furthermore, these elephant traffic flows are usually very stable over time and hence are suitable to be rerouted for load-balancing purpose. Based on these observations, our load balancing scheme only attempt to adjust the routing of the top 10% destination prefixes in the routing table based on their traffic demands. However, this fraction of optimized destination prefixes can be kept fixed or increased in the event of increase in routing tables. In future, a smaller fraction of destination prefixes may be used if 10% gives a very large number.

B. Optimal Routing Calculation for Load Balancing

Given the knowledge of traffic demand and outbound link information, the optimal routing for load balancing can be calculated. Let m be the number of outbound links in the concerned AS. Also let l_i and c_i denote the ith outbound link and its capacity (or bandwidth) respectively, where $i = 1 \dots m$. We assume that all the outbound traffic of this AS will be routed on these links and there is no other exit point for the outbound traffic. If s_i denotes the total outbound traffic carried by the ith link, then the utilization of the link l_i is given by s_i/c_i. Based on this notation, we define the objective of load balancing as to *minimize the maximum link utilization among all the outbound links*, i.e.,

$$\text{minimize} \max_{i=1\dots m} \frac{s_i}{c_i}. \tag{18}$$

As we discussed in the previous section, we select only 10% of all prefixes existing in the AS routing tables. Let n denote the number of selected destination prefixes and $d_j, j = 1 \dots n$, denote the average offered load for these destinations. Our load balancing scheme attempts to adjust the routing of these n prefixes in order to minimize the objective function in (18). Let $\mathcal{D}_i$ denote the set of the n prefixes that are routed on link l_i by adjusted routing. If f_i denotes the load on link l_i generated by the other 90% of the traffic flows, which are routed to this link by the default BGP routing, then (18) becomes

$$\text{minimize } \Phi = \max_{i=1\dots m} \left(\sum_{j \in \mathcal{D}_i} \frac{d_j}{c_i} \right) + \frac{f_i}{c_i} \tag{19}$$

where, the first term represents the percentage load due to the selected 10% flows and the second term represents the percentage load generated by the other 90% flows on link l_i. This problem can also be written as the following integer programming problem:

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & \sum_{j=1}^{n} x_{ij} \frac{d_j}{c_i} + \frac{f_i}{c_i} \le t, \quad i = 1 \dots m \\ & \sum_{i=1}^{m} x_{ij} = 1, \quad j = 1 \dots n \\ & x_{ij} \in \{0,1\}, \quad i = 1 \dots m, \ \ j = 1 \dots n \end{aligned} \tag{20}$$

where x_{ij} is a binary number. When $x_{ij} = 1$, the flow d_j is exiting the AS through link l_i. Otherwise, the flow d_j is exiting the AS through a link other than l_i. Note that traffic flow d_j may not have all outbound links as its alternative paths. One way to include this in the optimization process is to assume an arbitrarily large d_j/c_i for such links over which the flow d_j can never pass.

The problem represented by (20) is actually a classical task scheduling problem with unrelated parallel machines [44], where a number of tasks with different sizes are assigned to a set of parallel machines. The processing time of each task is different on different machines and the objective there is to minimize the completion time of all tasks by carefully distributing these tasks onto the parallel machines. This problem is NP-hard and approximation algorithms can be used to obtain near-optimal solutions. For example, in [45] a linear programming technique is first used to obtain a basic solution where there are at most $m - 1$ nonintegral x_{ij} values. Then for these nonintegral x_{ij} values, an exhaustive enumeration is performed to find the optimal scheduling. Combining the solutions of these two steps can produce an approximate solution with an upper bound of $2t^*$, where t^* denotes the value of t produced by the optimal solution. The time complexity of this method is exponential with respect to the value of m.

Instead of the integer programming approach, we have applied the OLS framework to this load balancing problem. With the flexibility of OLS, it is possible to optimize for various performance objectives besides load balancing. For example, in addition to load-balancing, the network operator may also prefer to use the shortest paths. It is possible to formulate a multiobjective optimization problem and obtain a solution, using OLS, that meets both load-balancing and shortest path criteria. However, we focus only on the load-balancing problem in our study.

The complete optimization procedure performed by the OLS framework can be summarized as follows:

- Step 1) Extract top 10% destination prefixes[1], with traffic demands $d_j, j = 1 \dots n$, from the routing table;
- Step 2) Calculate d_j/c_i and $f_i/c_i, i = 1 \dots m, j = 1 \dots n$ according to the traffic demand for each prefix and the capacity of each outbound link.
- Step 3) Each destination prefix may be reachable by all or some of the outbound links. This information can be obtained from Adj-RIBs-In at a BGP router. Assign a very large value of d_j/c_i for the infeasible routes, so the solution (minimization) will not result in an infeasible solution.
- Step 4) Measure or compute the value of Φ for default routing using (19) denoted by Φ^0.
- Step 5) Run RRS till a stopping criteria is reached. A stopping criteria can be a limit on time, number of iterations etc. To find a smaller value for Φ in (19), RRS tries different values for the x_{ij} binary values. This means that there are nm parameters each having a 0 or 1 value, which composes a total search space

[1]It is possible to use more or less than 10% of the destination prefixes in the optimization. The 10% value is just for illustration purposes. In reality, the network operators can correlate this value to the routing table size.

TABLE II
UTILIZATION (IN %) OF OUTBOUND LINKS BEFORE AND AFTER OPTIMIZATION

	Outbound Link Capacity							
	100	**100**	**100**	**100**	**45**	**45**	**45**	**12**
Before	7	25	20	33	60	45	48	**91**
After	25	33	27	30	**35**	33	34	23

size of 2^{nm}. Let Φ^*, $\overline{r}^*$ denote the value of objective function and corresponding routing at the end of optimization.

Step 6) If $\left|\frac{\Phi^0-\Phi^*}{\Phi^0}\right| \geq \Delta$, where Δ is the predefined threshold, deploy $\overline{r}^*$ by setting a high LOCAL_PREF of desired links for appropriate destination prefixes.

C. Simulation Results: BGP Parameter Tuning

The simulations presented in this section demonstrate the load balancing for an AS with $m = 8$ outbound links whose normalized capacities are 100, 100, 100, 100, 45, 45, 45, and 12, respectively. We assume the number of top 10% destination prefixes generating most traffic is $n = 148$. Note that this number is chosen somewhat arbitrarily only for the illustration purpose. This means that the total search space size RRS has to explore is $2^{nm} = 2^{1184}$, which is not possible to exhaustively search in practice where real-time adjustment of outbound BGP routes is required. In the simulation, we generate only 148 traffic flows instead of all the traffic flows since the actual effect of the other 90% flows on the simulation is only to reduce the capacity of the links by a certain amount. Therefore, ignoring these flows will not compromise the validity of the simulation results in any way. We assign each destination prefix a certain load such that the total offered load is the 30% of the total capacity of all the links.

In the beginning of the simulation, the offered load is randomly distributed over the outbound links. Then we apply the proposed load balancing scheme to the network. The link utilization of outbound links are compared in Table II. As shown in the table, before optimization, the load distribution across the outbound links is rather uneven. For example, one link is greatly under-utilized with a utilization of 7% while another link is heavily used with a utilization of 0.91%. After applying the load balancing scheme, the load distribution becomes much more even and the utilization of each link is very close to the ideal value, i.e., the average utilization 30%. The maximum link utilization drops from 91% to 35%.

VI. CONCLUSION

In this paper, we presented an on-line simulation framework for adaptive large-scale network parameter configuration that uses a black-box optimization approach. As a result, it allows great flexibility in the choice of performance objectives to be achieved and is generally applicable to a variety of network protocols and configuration problems. An efficient search algorithm, Recursive Random Search (RRS) algorithm, is designed (i.e., biased) to find "good" solutions within a limited time frame instead of full optimization. The RRS algorithm is especially advantageous when handling objective functions affected by noises and those with insignificant parameters because of its basis on random sampling.

The application of the OLS framework to three network protocols, RED, OSPF and BGP, has been investigated. Simulations and experiments have demonstrated that OLS is very successful to adapt the protocol configuration to the prevailing network conditions and achieve various network performance objectives. These applications are given as examples of the *breadth* and *flexibility* of our approach's applicability. With the flexibility of the black-box approach, our system can always incorporate alternate problem formulations, for example, an alternate performance metric or simulation engine, and give *good results fast*. In this paper, we have shown the use of OLS in an implementation context (RED), an analytic black-box evaluation context (OSPF) and a simulation context (BGP). These contexts illustrate the generality of application scenarios and variety of optimization formulations that can fit into our framework.

REFERENCES

[1] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. ACM SIGCOMM*, 2002, pp. 3–16.

[2] D. Katz, "Why are we scared of SPF? IGP scaling and stability," in *NANOG 25*, Toronto, Canada, Jun. 2002 [Online]. Available: http://www.nanog.org/mtg-0206/katz.html

[3] N. Feamster, J. Rexford, and J. Borkenhagen, "Controlling the impact of BGP policy changes on IP traffic," in *NANOG 25*, Toronto, Canada, Jun. 2002 [Online]. Available: http://www.nanog.org/mtg-0206/feamster.html

[4] C. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low memory, modular time warp system," in *Proc. Workshop on Parallel and Distributed Simulation*, May 2000, pp. 53–60.

[5] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and Willinger, "Network topology generators: Degree-based vs. structural," in *Proc. ACM SIGCOMM*, 2002, pp. 147–159.

[6] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, 2002, pp. 133–145.

[7] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. Networking*, vol. 2, no. 1, pp. 1–15, Feb. 1994.

[8] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: Methodology and experience," *IEEE/ACM Trans. Networking*, vol. 9, no. 3, pp. 265–278, Jun. 2001.

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

[10] F. Glover, "Tabu search–part I," *ORSA J. Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[11] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*. New York: Wiley, 1989.

[12] T. Ye and S. Kalyanaraman, "A recursive random search algorithm for large-scale network parameter configuration," in *Proc. ACM SIGMETRICS 2003*, San Diego, CA, Jun. 2003, pp. 196–205.

[13] NS Network Simulator. [Online]. Available: http://www.isi.edu/nsnam/ns/

[14] SSFNET Network Simulator. [Online]. Available: http://www.ssfnet.org

[15] A. Törn and A. Žilinskas, *Global Optimization*. New York: Springer-Verlag, 1989, vol. LNCS 350.

[16] D. H. Wolpert and W. G. Macready, "No Free Lunch theorems for optimization," *IEEE Trans. Evolutionary Computing*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[17] T. C. Hu, V. Klee, and D. Larman, "Optimization of globally convex functions," *SIAM J. Control and Optimization*, vol. 27, no. 5, pp. 1026–1047, 1989.

[18] K. D. Boese, A. B. Kahng, and S. Muddu, "A new adaptive multistart technique for combinatorial global optimizations," *Oper. Res. Lett.*, vol. 16, pp. 101–113, 1994.

[19] J. A. Boyan and A. W. Moore, "Learning evaluation functions to improve optimization by local search," *J. Machine Learning Res.*, vol. 1, no. 2000, pp. 77–112, 2000.

[20] R. H. Leary, "Global optimization on funneling landscapes," *J. Global Optimization*, vol. 18, no. 4, pp. 367–383, Dec. 2000.
[21] Z. B. Zabinsky, "Stochastic methods for practical global optimization," *J. Global Optimization*, vol. 13, pp. 433–444, 1998.
[22] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: The MIT Press, 1996.
[23] S. Kirkpatrick, D. Gelatt, and M. Vechhi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.
[24] W. L. Price, "Global optimization by controlled random search," *J. Optimization Theory and Applications*, vol. 40, pp. 333–348, 1978.
[25] S. Rana, L. D. Whitley, and R. Cogswell, "Searching in the presence of noise," in *Parallel Problem Solving from Nature—PPSN IV*. Berlin: Springer, 1996, vol. LNCS 1141, pp. 198–207.
[26] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.
[27] W.-C. Feng, D. D. Kandlur, D. Saha, and K. G. Shi, "A self-configuring RED gateway," in *Proc. IEEE INFOCOM*, 1999, vol. 3, pp. 1320–1328.
[28] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for web traffic," in *Proc. ACM SIGCOMM*, 2000, pp. 139–150.
[29] T. Bonald, M. May, and J.-C. Bolot, "Analytic evaluation of RED performance," in *Proc. IEEE INFOCOM*, 2000, pp. 1415–1444.
[30] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "A control theoretic analysis of RED," in *Proc. IEEE INFOCOM*, 2001, vol. 3, pp. 1510–1519.
[31] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft, "Pop-level and access-link-level traffic dynamics in a tier-1 pop," in *ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001, pp. 39–53.
[32] W. Fang and L. Peterson, "Inter-AS traffic patterns and their implications," in *Proc. Globecom'99*, Rio de Janeiro, Brazil, 1999, vol. 3, pp. 1859–1868.
[33] A. Khanna and J. Zinky, "The revised ARPANET routing metric," in *Proc. ACM SIGCOMM*, 1989, pp. 45–56.
[34] D. W. Glazer and C. Tropper, "A new metric for dynamic routing algorithms," *IEEE Trans. Commun.*, vol. 38, no. 3, pp. 360–367, Mar. 1990.
[35] Z. Wang and J. Crowcroft, "Analysis of shortest-path routing algorithms in a dynamic network environment," *ACM Comput. Commun. Rev.*, vol. 22, no. 2, pp. 63–71, 1992.
[36] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, 2000, pp. 519–528.
[37] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, "NetScope: traffic engineering for IP networks," *IEEE Network*, vol. 14, no. 2, pp. 11–19, Mar./Apr. 2000.
[38] R. Nagarajan, J. F. Kurose, and D. Towsley, "Approximation techniques for computing packet loss in finite-buffered voice multiplexers," *IEEE J. Select. Areas Commun.*, vol. 9, no. 4, pp. 368–337, Apr. 1991.
[39] H. G. Perros, *Queueing Networks With Blocking, Exact and Approximate Solutions*. Oxford, U.K.: Oxford Univ. Press, 1994.
[40] B. Fortz and M. Thorup, "Increasing Internet capacity using local search," *Computation Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.
[41] G. Huston, "Commentary on inter-domain routing in the Internet," RFC 3221, Dec. 2001.
[42] T. Ye, S. Yadav, M. Doshi, A. Gandhi, S. Kalyanaraman, and H. T. Kaur, "Load balancing in BGP environment using online simulation and dynamic NAT," presented at the ISMA Workshop by CAIDA, San Diego, CA, Dec. 2001.
[43] J. Stewart, III, *BGP-4 Inter-Domain Routing in the Internet*. Reading, MA: Addison-Wesley, 1999.
[44] L. A. Hall, *Approximation Algorithms for NP-Hard Problems*. Boston, MA: PWS Publishing, 1995, ch. Approximation Algorithms for Scheduling.
[45] C. N. Potts, "Analysis of a linear programming heuristic for scheduling unrelated parallel machines," *Discrete Appl. Math.*, vol. 10, pp. 155–164, 1985.

Tao Ye received the M.S. degree in electrical engineering from Xi'an Jiaotong University, Xi'an, China, in 1997, and the Ph.D. degree in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2003.

He is a Vice President with One William Street Capital, New York, NY.

Hema T. Kaur received the B.E. degree in electronics engineering from Bhilai Institute of Technology, India, the M.T. degree in electrical engineering from IIT, Kanpur, India. She received the Ph.D. degree in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2002.

She is a Senior Software Engineer with Intel Corporation, Beaverton, OR. Her research interests include system power and performance analysis, wireless communications and networking protocols. She is currently working on manageability and security architecture for enterprise notebook systems.

Shivkumar Kalyanaraman (M'03–SM'07) received the B.Tech. degree in computer science from IIT, Madras, India, and the M.S. and Ph.D. degrees from Ohio State University, Columbus.

He is with IBM India Research Laboratory, Bangalore, India. He is a Professor (on leave) in the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY. His research interests include various traffic management topics and protocols for emerging tetherless networks. He is a senior member of the ACM.

Murat Yuksel (S'99–M'03) received the B.S. degree in computer engineering from Ege University, Izmir, Turkey. He received the M.S. and Ph.D. degrees in computer science from Rensselaer Polytechnic Institute, Troy, NY, in 1999 and 2002, respectively.

He is an Assistant Professor at the University of Nevada, Reno. His research is on various networking issues including wireless routing, free-space-optical mobile ad hoc networks (FSO-MANET), network modeling and economics, protocol design, peer-to-peer, and performance analysis. He is a member of the ACM and Sigma Xi.