Large-Scale Network Parameter Configuration Using An On-line Simulation Framework

Tao Ye, Hema Tahilramani Kaur, Shivkumar Kalyanaraman
Department of Electrical, Computer and System Engineering
Rensselaer Polytechnic Institute
Troy, New York 12180
{yet3,hema}@networks.ecse.rpi.edu, shivkuma@ecse.rpi.edu

***Abstract*— As the Internet infrastructure grows to support a variety of services (eg: VPNs), its legacy protocols (eg: OSPF, BGP) 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 online 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 online in a network operations center (NOC). Each simulation evaluates the network performance for a *particular setting* of protocol parameters. A recursive random search (RRS) technique is proposed to explore the large-dimensional parameter state space, where each sample point results in a single simulation. In other words, the overall parameter configuration problem is modeled as a *"black-box" optimization* problem. The goal of RRS is efficiency, i.e., to find *"good"* network protocol configurations for the current traffic conditions *quickly*. An important feature of the framework is its flexibility: it allows arbitrary choices in terms of the simulation engines used (eg: ns-2, SSFnet, future scalable simulators etc), network protocols to be simulated (eg: OSPF, BGP, RED, MPLS etc), 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. The first application (RED) is prototyped and evaluated in a Linux-based testbed using SNMP as the configuration interface.**

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

I. INTRODUCTION

Today's network protocols like BGP and OSPF were designed for one primary service: "best effort reachability." 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 is 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. Though some tools are emerging to aid operators, a lot more needs to be done.

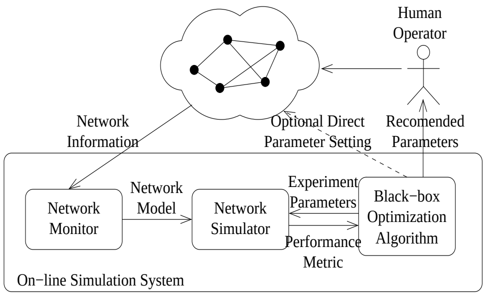

Fig. 1. On-line simulation framework for adaptive configuration of network protocols

In this paper, we propose a novel on-line simulation framework (OLS) to aid generic large-scale network protocol configuration. Realistically, the tool 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 shows the basic idea of this system. We formulate the network protocol configuration as a "black-box" optimization problem over the parameter state space. A sample point in the state space corresponds to a network simulation that evaluates the performance in terms of pre-determined metrics. The simulation also imports the current network topology and a digest of latest traffic patterns. We propose a Recursive Random Search (RRS) algorithm that intelligently searches the large-dimensional parameter state space to find "good" settings quickly. The "black-box" approach allows *flexibility* in terms of objectives of the desired optimization, and hence can be applied to a variety of protocols and configuration problems. The key assumptions of the framework is that the underlying network protocol performance is indeed sensitive to the settings of the parameter set chosen; and that the network topology, traffic and protocol can be reasonably modeled in simulation.

While these assumptions appear to be somewhat restrictive today, on the long-term, our framework can leverage improvements in modeling of topology [4], [5], [6], traffic [7], [8] and/or improvements in scalable network simulation [9], [10], [11] technology. To the best of our knowledge, our flexible approach is unique and the first of its kind as applied to IP network management. There have been proposed configuration-support or adaptive protocol techniques for specific problems (eg: BGP [12], OSPF [13],RED [14]). We discuss these related works in later sections.

A key component of the framework is an efficient parameter state space search algorithm. The requirements on such a "black-box" algorithm are as follows: **a)** it should scale to large-dimensional parameter state spaces; **b)** it should find "good" solutions quickly; **c)** it should be robust to noise (eg: minor inaccuracies in modeling) in the function evaluations; and **d)** it should be able to automatically reject negligible parameters (i.e. to which the protocol is insensitive). Traditional search algorithms (eg: genetic algorithms[15], multi-start hill-climbing, tabu search[16] and simulated annealing[17]) could not provide the above desired combination of properties as we will show in SectionII.

We propose a new search algorithm, Recursive Random Search (RRS), that completely relies on random sampling and is a good match to our search objectives. We demonstrate the flexibility and relevance of our overall 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 attributes. We use simulations to demonstrate these applications, i.e., perform the on-line simulation framework inside a "simulation". Furthermore, the first application (RED) is also prototyped and evaluated in a Linux-based testbed using SNMP as the configuration interface.

The rest of this paper is organized as follows: Section II describes the features of network parameter optimization problems and presents the overview of the Recursive Random Search(RRS) algorithm. 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

We model the network parameter configuration problem as solving 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. Given the desired performance metric p and the network scenario $\mathcal{N}$, Equation (1) calculates the required configuration $\mathcal{C}$. Due to the complexity of the Internet, the analytical derivation of Equation (1) is usually not realistic. However, with network simulation software, such as *ns*[18], *SSFNET*[10], 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 Equation (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 mathematically formulated as (assume minimization): given a real-valued objective function $f : \mathbb{R}^n \to \mathbb{R}$, 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 where the objective function is modeled as a black-box. Since little *a priori* knowledge is assumed, these black-box optimization problems are considered very hard to solve. In addition, since the objective functions are often non-linear and multi-modal, this type of optimization are also called *global optimization* in contrast to *local optimization* which has only one single extreme in $f(\mathbf{x})$ and is much easier to solve.

Most of black-box optimization problems are NP-hard and can only be solved for near-optimal solutions with heuristic search algorithms. Many heuristic search algorithms have been proposed and demonstrated to be very successful in practice, such as, multi-start hill-climbing[19], genetic algorithm[15] and simulated annealing[17]. However, there has been no consistent report on their performance. In fact, *No Free Lunch Theorem*[20], [21] has theoretically demonstrated that no matter what performance metric is used, no single optimization algorithm can consistently perform better in all problem classes than the others. The average performance of any algorithm is the same over all classes of problems. In other words, there exists no general all-purpose optimization algorithm. For one specific class of problems, its inherent properties have to be carefully investigated to perform efficient optimization.

A. Properties of Network Parameter Optimization Problems

The following features are usually present in network parameter optimization problems.

High efficiency is required for the desired search algorithm. More specifically, 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.

Furthermore, network parameter optimization is based on network simulation which might be very time-consuming. This also requires a highly efficient search.

High dimensionality is another feature of these problems. For example, 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"[19].

Noise is often introduced into the evaluation of the objective function since network simulation may be used for function evaluations. Due to inaccuracies in network modeling, simulation, etc., the obtained empirical objective function may be distorted from the real one, in other words, affected by small random noises. Fig 2 shows an example of 2-dimensional empirical objective function obtained with network simulation. It can be seen that there exist many irregular small random fluctuations imposed on the overall structure.

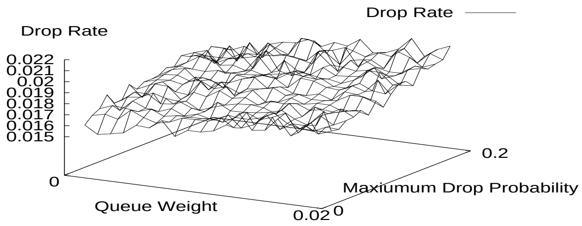

Fig. 2. An empirical objective function obtained with network simulation (RED buffer management)

Negligible parameters may also be included in the objective function. These parameters contribute little to the objective function and should be ruled out from the optimization process. However, in practice, they are normally very difficult to be identified and eliminated effectively. If the search algorithm is able to automatically excluded these parameters from the optimization process, the efficiency of the optimization will be significantly improved.

"Globally convex" or "big valley" structure[22], [23] 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 distribute far 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[24] has demonstrated the existence of this structure in complex Traveling Salesman Problem(TSP) and graph bisection problem, and presented an *intuitive* graph for this structure(Fig 3). The same structure has been found in circuit/graph partitioning and job-shop scheduling, etc.[25]. Leary[26] also confirmed that there exist similar "funnel" structures in molecular conformation problems where the potential energy from the forces between atoms is minimized.

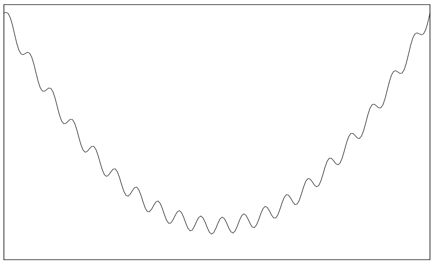

Fig. 3. Big valley structure

The issues described above are common in many practical optimization problems[27], [28]. For such class of problems, genetic algorithm[29] and simulated annealing[30], controlled random search[31], are the most common algorithms since they require little *a priori* information from the concerned problem and are generally applicable. However, these algorithms are mainly designed for full-optimization and often lacking in efficiency. In practice, they are often combined with local search techniques, such as, deepest descent and pattern search, to improve their efficiency. However, since these local search techniques use fixed local structures to guide the search process, they are usually susceptive to the effect of noises[32]. For example, in pattern search, the wrong pattern may easily be derived if the samples for pattern exploration are corrupted by noises. 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 many efforts on examining these low-quality local optima and essentially work like an inefficient random sampling.

B. *Recursive Random Search Algorithm*

Because of the disadvantages described above for traditional search algorithms, we have proposed the Recursive Random Search algorithm (RRS)[33] 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 re-align sample spaces to local optima. We have tested this algorithm on a suite of difficult benchmark functions and some network parameter optimization problems. The results have shown that in terms of quickly locating a good solution, RRS outperforms other search algorithms, such as multi-start 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 negligible parameters in the objective function has little effect on the efficiency of RRS. In the following we will first illustrate the initial high-efficiency feature of random sampling and then present a concise description of the algorithm. Readers can refer to [33] for more details and the test results.

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 2-dimensional space, volume in a 3-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 $\phi_D(y_r) = r, \quad r \in [0, 1]$, 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:

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

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 4 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 4, 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*. Basically, we perform random sampling for a number of times, then move or resize the sample space according to the previous samples and start another random sampling in the new sample space.

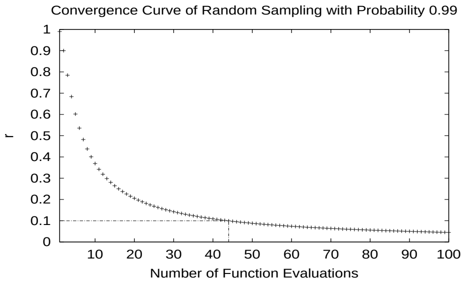

Fig. 4. $A_D(r)$ of $x^n_{(1)}$ in random sampling with probability 0.99

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 areas 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 areas and hence may waste much time in trivial areas. 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 5. The upper graph shows a contour plot of a 2-dimensional multi-modal 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 right 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

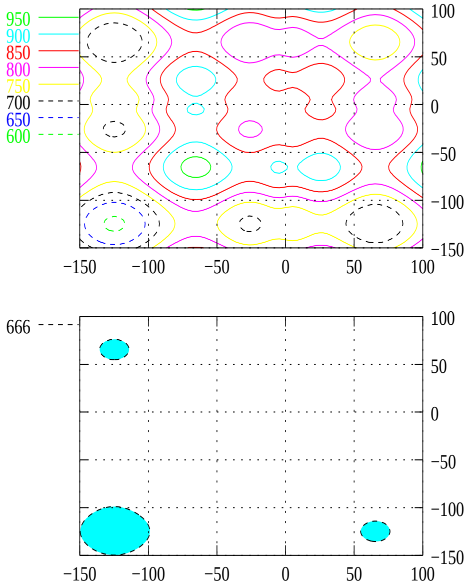

Fig. 5. Contour plot of an objective function(left) and its region of $A_D(0.05)$(right)

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 take a certain number of samples and use the best one to decide the location of $A_D(r)$. It then goes on into recursive random sampling process by shrinking or re-aligning 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-re-align 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. Interested readers can refer to [33] for the detail of the algorithm.

In contrast to most of the search algorithms, the RRS algorithm is mainly built upon random sampling. RRS performs the search process based on stochastic information on a certain sample area, therefore, its performance is less affected by noises. In addition, RRS is more efficient when dealing with the objective function with negligible 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 negligible 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 suite of benchmark functions[33].

III. Adaptive Tuning of RED

Buffer management mechanisms can be used for congestion control by cooperating with TCP end-to-end congestion avoidance mechanism. Traditional DropTail could not effectively prevent the occurrence of serious congestion and often suffer from long queueing delays. Furthermore, the global synchronization may occur during the period of congestion, i.e., a large number of TCP connections experience packet drops and hence back off their sending rate at the same time, resulting in underutilization and large oscillation of queueing delay. Random Early Detection (RED) has been proposed [34] to address these problems. The basic idea of RED is to detect the inception of congestion and notify traffic sources early to avoid serious congestion. It has been demonstrated to be able to avoid global synchronization problem, maintain low average queueing delay and provide better utilization than DropTail[34]. Therefore, IETF has recommended RED as the single active buffer management for wide deployment in the Internet[35]. However, the setting of RED parameters has proved to be highly sensitive to network scenarios and the performance of misconfigured RED may suffer significantly [14], [36], [37]. Therefore, RED needs constant tuning to adapt to the prevailing network conditions. In view of this, it has been debated whether or not RED can achieve its claimed advantages[37], [38], [39].

Based on simplified models, some general guidelines for setting RED parameters have been proposed[34], [36], [40]. Intuitive modifications on RED have also been proposed to automate the tuning of RED under varying network conditions by adjusting one of the parameters[14], [41]. However, the effectiveness of these methods in complex network scenarios is still under investigation. Rather than relying on simplified models or intuition, here we employ the on-line simulation framework for the dynamical tuning of RED.

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 6, 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.

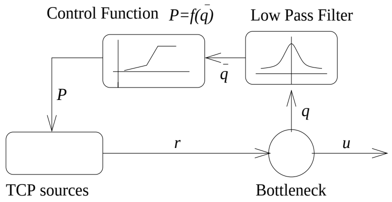

Fig. 6. RED working mechanism

A queue will build up and keep increasing if the offered load is larger than the bottleneck capacity; therefore, the objective of a buffer management algorithm is to stabilize the offered load around the bottleneck capacity. Basically, TCP sources increase their sending rate every round trip time; on the other hand, the packet drops cause TCP sources to lower their sending rates. In the equilibrium status, 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 status 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 rationale of RED is to search for an appropriate packet drop rate by varying the average queue size to counteract the increase of offered load.

There are four parameters in RED. Among them, 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 7. With this function, the drop probability can be calculated according to the average queue size. The equilibrium drop probability depends on two factors, the offered load increase rate and the granularity of congestion notification, i.e., the load decrement caused by one packet drop. With TCP fast recovery and fast retransmission mechanism, 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. When the average sending rate is large, for example, a small number of TCPs share a bottleneck, each packet drop will cause a large decrease in offered load, and *vice versa*. In different scenarios, the increase rate of offered load is also different. For example, the increase rate will be large when there are many TCP flows or the round trip time is short. As a result, the drop probability should be adjusted according to network scenarios to maintain a stable equilibrium point. 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 parameter which determines $f(\bar{q})$ should be dynamically tuned.

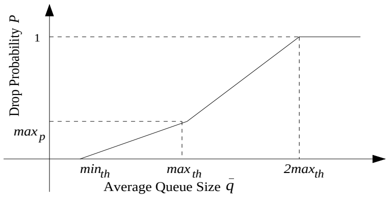

Fig. 7. RED control function $P = f(\bar{q})$

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 basically two 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*[34]. 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 multi-objective optimization problem and corresponding techniques should be employed to convert it into a tractable single objective problem.

One classic multi-objective 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 multi-objective 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 drifts beyond the control of RED, RED will become unstable, i.e., the queue status oscillates between full and empty[14], [36]. 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} as proposed in[41]. 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.

C. Simulation Results

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 8. We used ns[18] 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

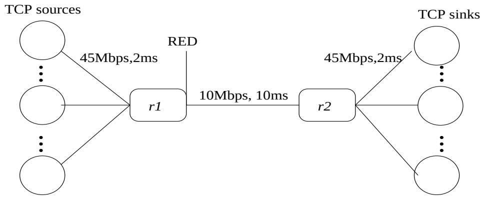

Fig. 8. Network topology for RED tuning simulation

changed twice during the simulation. We will compare the performance of standard RED and RED controlled with the on-line simulation framework under 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. 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. 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.

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 9. 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 not 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 to a certain level, the queue frequently becomes empty and this causes the underutilization of the link capacity. 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%.

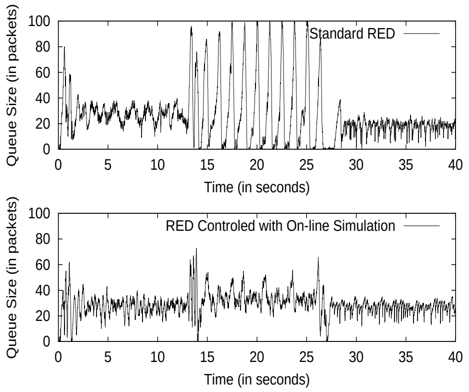

Fig. 9. Comparison of standard RED (upper graph) and RED controlled by on-line simulation (lower graph) under varying traffic load

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 18ms (not including queueing delay). After 13 seconds, the rtt of these flows is increased to 170ms. And after another 13 seconds, the rtt is reduced to around 2ms. The instantaneous queue sizes of standard RED and RED with on-line simulation control are shown in Fig 10. The upper graph shows that when rtt is increased to 170ms, 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 2ms, 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.

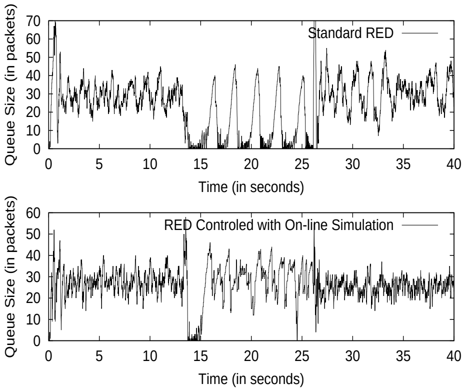

Fig. 10. Comparison of standard RED (upper graph) and RED controlled by on-line simulation (lower graph) under varying round trip time

D. Real Network Experiment for Optimization of Multiple RED Queues

To test the effectiveness of the OLS framework, real network experiments are also performed. This section presents one such experiment. A Linux-based testbed shown in Fig 11 is used 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 certain multi-objective technique has to be employed to combine the metrics

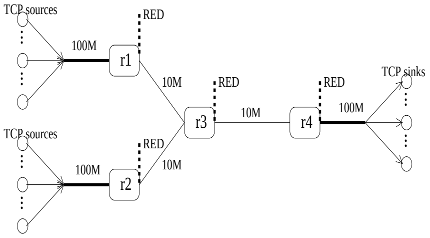

Fig. 11. Linux-based testbed topology with multiple RED queues

from every RED router. Instead, we have selected an end user performance metric, i.e., the Coefficient of Variation ($\frac{\sigma}{\mu}$) of goodputs for TCP connections, which measures the variation of TCP goodputs. This choice is somewhat arbitrary, only to demonstrate the effectiveness of our approach. In addition, choosing such a metric is also to demonstrate the flexibility of the approach, i.e., rather than being restricted to a few metrics like utilization and delay, RED can be tuned according to any performance metric defined by network operators though the mechanism of how RED affects this performance metric may be completely unknown.

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 12. 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 325 second, the on-line simulator starts and detects the misconfiguration of REDs. Soon the good configuration with a performance better than a predefined threshold is found 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.

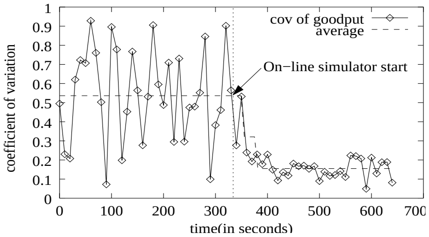

Fig. 12. Tuning multiple RED queues for optimizing coefficient of variation of goodputs

IV. Traffic Engineering by Tuning OSPF Link Weights

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 current Internet, IP traffic is mapped onto the network by standard routing protocols, such as, Open Shortest Path First (OSPF) protocol. OSPF is mainly used for intra-domain traffic routing. It routes traffic on shortest paths based on the advertised link weights. As a result, the link along the shortest path between the two nodes may become congested while the links on longer paths may remain idle. Many traffic measurements[42], [43] have observed large variations in link utilization across the network. OSPF allows for Equal Cost Multi Path(ECMP) where the traffic is distributed equally among various next hops of the equal cost paths between a source and a destination [44]. This is useful in distributing the load to several shortest paths. However, the problem of uneven mapping still remains.

Two main approaches have been taken to solve the intra-domain traffic engineering problem. One approach is to deploy the emerging MPLS technology which is not constrained by the shortest path nature of routing. Constraint-based routing can be used to compute routes in an MPLS network subject to QoS and policy constraints. 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, One earlier approach was to adapt link weights to reflect the local traffic conditions on a link or to avoid congestion ([45], [46], [47]). This is called adaptive routing or traffic-sensitive routing. However, adapting link weights to local traffic conditions leads to frequent route changes and is unstable (see [48], [49] for stability analysis). Additionally, adaptive routing is based on the local information and therefore cannot optimize traffic allocation from the viewpoint of the overall network. These drawbacks are alleviated in [13] where the configuration of OSPF link weights is modeled as a black-box optimization problem. The authors have chosen a heuristic cost function which is piecewise linear with offered load and applied a multi-start hillclimbing algorithm to find good solutions.

In this section, we will use the on-line simulation framework for the adaptive configuration of OSPF link weights. Instead of the heuristic metric used in [13], we have chosen the total packet drop rate in the network as the performance metric since it is a more accurate to indicate the congestion in the network and it also has significant impacts on the performance of some underlying protocols, such as TCP. The packet drop rate for one set of link weights could be estimated using packet-level or flow level simulation. However, the OLS framework allows flexibility in the type of method used for function evaluation (wherever appropriate). Here we use a GI/M/1/K queuing mode instead of a full-fledged simulation to calculate the packet drop rate, which is considerably faster. When calculating the drop rate, the mean and variance of the offered load should both be considered. This is a more complete representation of traffic conditions than the average offered load used in [13].

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], [50].

In the following, we will first show how to derive the drop probability for one link based on the offered load. Then we will formulate the optimal general routing problem 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) Link Drop Probability: Let P denote the packet drop probability on a link, λ, σ^2 denote the mean, variance of the offered load to this link in packets per second, and B, K denote its bandwidth and buffer space respectively. In order to find a closed-form expression for the packet drop probability P, let us assume an exponentially distributed packet size with mean $\bar{X}$. However, we consider a general arrival process. We compute the packet drop probability at the link using a GI/M/1/K queuing model. The drop probability of a finite GI/M/1/K has been approximated by an infinite buffer GI/M/1 queue [51] using the following equation.

$$P(N_K = K) = \frac{P(N_\infty = K)}{P(N_\infty \leq K)} \tag{9}$$

N_K denotes the number of packets in the finite buffered queue, whereas, N_∞ denotes number of packets in the infinite buffer GI/M/1 queue. The queue length distribution of GI/M/1 queue is given by [52]:

$$P(N_\infty = j) = A\omega^{j-1} \qquad (j \geq 0) \tag{10}$$

where A is the normalization constant and ω is a constant depending on the arrival process and service rate. ω can be obtained by solving the following equation:

$$\omega = \gamma\left((1-\omega)\mu\right) \tag{11}$$

where $\gamma(s)$ is the Laplace transform of the arrival process and μ is the service rate which is given by $\frac{B}{\bar{X}}$. In order to solve (11) for ω, we need to assume a inter-arrival time distribution for the arrival process. Let us consider the Generalized Exponential (GE) distribution for modeling the arrival process to first two moments. We discuss below the reason for choice of GE distribution.

The pdf of GE distribution is given by

$$g(x) = (1-p)\delta(x) + pae^{-ax} \tag{12}$$

where $\delta(x)$ is the delta function, p and a two constant parameters. As can be seen from (12), a GE process is characterized by two parameters, p and a. GE distribution is a special case of H_2 distribution and can be used to model general inter-arrival processes that are more bursty than Poisson process. For a Poisson process the variance is equal to the square of mean. Hence, GE distribution may be used to model the first two moments of processes with variance greater than the square of mean. If the arrival process is represented by a GE distribution, then, with probability p the inter-arrival time is exponentially distributed with mean a and with probability $1-p$, the inter-arrival time is zero. Hence, this distribution represents a batch arrival process with geometrically distributed batch size and exponentially distributed inter-batch arrival times. For a link with λ, σ as its mean and variance of the offered load, we can have the parameters of the GE distribution representing the arrival process:

$$p = \frac{2\lambda^2}{\sigma^2 + \lambda^2} \text{ and } a = p\lambda \tag{13}$$

The merging of N independent GE(p_i,a_i) processes is a bulk-arrival Poisson process with mean arrival rate a equal to $\sum_{i=1}^{N} a_i$ and p equal to $a/\sum \frac{a_i}{p_i}$. Similarly, splitting of a GE(p,a) process into N streams according to a Bernoulli filter $r_1, r_2, ...r_N$, the parameters of the i^{th} process are

$$p_i = \frac{p}{p(1-r_i)+r_i} \text{ and } a_i = r_i a. \tag{14}$$

Reader may refer to [53], Section 1.4 for more details.

The packet arrival process of a single TCP flow is bursty in nature with a "bulk" of packets arriving every round-trip time. The model that we have considered implies that we have "bulk" arrivals (in form of bursts of packets from competing TCP sources) of varying sizes arriving into a queue. Our model does not capture the feedback effect of packet drops on TCP flows because we have considered the aggregate traffic arriving at an OSPF router as our demand estimate.

Taking the Laplace transform of (12), we get,

$$G(s) = 1 - p + \frac{pa}{s+a} \tag{15}$$

Then substitute it into (11) and solve it for ω for the GE arrival process gives

$$\omega = \rho + (1-p) \tag{16}$$

where,

$$\rho = \frac{a}{\mu} = \frac{a\bar{X}}{B}. \tag{17}$$

Finally, using (9), (10), (11) and (15), we get the packet drop probability

$$P = \frac{(p-\rho)(\rho+1-p)^K}{1-(\rho+1-p)^{K+1}} \tag{18}$$

In summary, Equation (18) represents the closed form expression of packet drop probability, P, on a single link as a function of mean, variance λ, σ^2 of the arrival process, mean packet size $\bar{X}$, link bandwidth B and buffer space K. Figure 13 shows the drop probability as a function of the offered load for difference values of variance of the inter-arrival time for a buffer size of 20 packets. As expected, higher drop probability is observed when the arrival process has a high variance, i.e., when the incoming traffic is more bursty.

2) 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[13]. It is the best that can be achieved by carefully setting up multiple Label Switched Paths (LSPs) in MPLS. Using link packet drop probabilities obtained from (18), we can formulate the optimal general routing problem as:

$$\Phi = \sum_{l \in \mathcal{L}} \lambda_l P_l \tag{19}$$

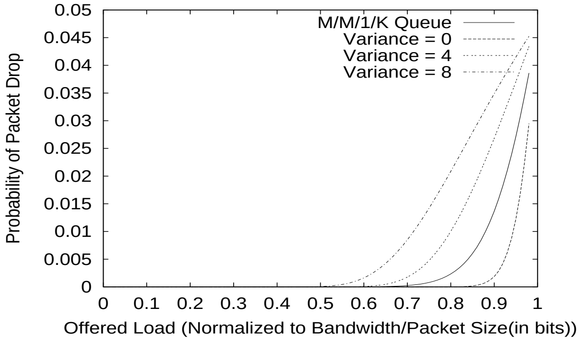

Fig. 13. Packet drop probability as a function of offered load for a GE/M/1/20 queue for different values of variance

where λ_l is the arrival rate for link l and P_l is its drop rate calculated by (18). 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

$$\sum_{i:(i,j)\in\mathcal{L}} f_{(i,j)}^{(s,t)} - \sum_{i:(j,i)\in\mathcal{L}} f_{(j,i)}^{(s,t)} = \begin{cases} -\mathcal{D}(s,t) & \text{if } j=s \\ \mathcal{D}(s,t) & \text{if } j=t \\ 0 & \text{Otherwise} \end{cases} \tag{20}$$

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

$$\lambda_l = \sum_{(s,t)\in\mathcal{N}\times\mathcal{N}} f_l^{(s,t)} \tag{21}$$

The parameter $p^{(s,t)}$ for the GE process used to fit the demand $\mathcal{D}(s,t)$ is given according to (13):

$$p^{(s,t)} = \frac{2\mathcal{D}(s,t)^2}{\mathcal{D}(s,t)^2 + \mathcal{V}(s,t)} \tag{22}$$

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

$$r_l^{(s,t)} = \frac{f_l^{(s,t)}}{\mathcal{D}(s,t)} \tag{23}$$

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 according to (14):

$$p_l^{(s,t)} = \frac{p^{(s,t)}}{p^{(s,t)}(1-r_l^{(s,t)}) + r_l^{(s,t)}} \tag{24}$$

The total offered load on link l is given by λ_l (21), 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

$$p_l = \lambda_l(\sum_{(s,t)\in\mathcal{N}\times\mathcal{N}} f_l^{(s,t)} p_l^{(s,t)})^{-1} \tag{25}$$

If ρ_l is equal to $\frac{\lambda_l p_l \bar{X}}{B_l}$, then, using (18), the probability of packet dropped at link l is given by

$$P_l = \frac{(p_l-\rho_l)(\rho_l+1-p_l)^{K_l}}{1-(\rho_l+1-p_l)^{K_l+1}} \tag{26}$$

The optimal general routing problem is given by (19), subject to the constraints given by (21), (22), (23), (24), (25), (26). 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 (19).

B. Optimization of OSPF Weights Using On-line Simulation

The general optimal routing problem, where the objective function is completely defined by (19)-(26), may possibly be solved for $f_l^{(s,t)} \forall l \in \mathcal{L}$ by using some non-linear programming techniques. However, under constraints of OSPF routing, the relation between the link weights and optimization metric can no longer be analytically defined. In [54], 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, i.e., minimize the packet drop rate given by (19), is also NP-hard. For such NP-hard problems, heuristic optimization algorithms are usually used to search for approximate solutions. Instead of choosing a different heuristic for each NP-hard problem, we can apply the RRS technique to perform efficient search in most of such problems.

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

$$\min \Phi(\mathbf{w}) \tag{27}$$

where $\mathbf{w}$ is the vector of network link weights and $\Phi(\cdot)$ the objective function, which is unknown. 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 the traffic is cast to obtain parameters of the aggregate packet arrival process and drop probability for every link $l \in \mathcal{L}$ using (21), (22), (23), (24), (25) and (26). Finally the value of Φ may be calculated by (19).

C. Simulation Results

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 perform the simulation on a large-scale real ISP network topology, i.e., EXODUS network, obtained from Rocketfuel project[55]. This topology includes 244 core routers from EXODUS network and 1040 simplex links. Fig 14 shows the figures of these topologies generated by NAM[56].

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 [13]. 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 1Mbps 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[18] 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 the GE inter-arrival distribution described in (12). In the simulation, we assume 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 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. Figure 15 shows total packet drop rate in the network as a function of time. Table I summarizes the maximum improvement in packet drop rates for different topologies. 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.

	ARPANET	MCI	EXODUS
Improvement	31.8%	60.2%	35.7%

TABLE I
TABLE SUMMARIZING THE MAXIMUM PERCENTAGE IMPROVEMENT IN THE PACKET DROP RATES OBTAINED FOR DIFFERENT TOPOLOGIES FOR THE RESULTS SHOWN IN FIGURE 15

V. OUTBOUND LOAD BALANCING IN BGP ENVIRONMENT

For inter-domain Traffic Engineering, the traffic demand statistics are usually kept private and the control over routers outside the local domain is normally not available. The global TE approach like those for intra-domain TE is not practical. Therefore, inter-domain TE has mainly focused on multi-homed Autonomous Systems (AS), in-bound/out-bound load-balancing between adjacent ASes using BGP attributes (e.g. MED, LOCAL_PREF, AS_PATH, etc.) [57].

The ASes are increasingly becoming multi-homed [57]. The outbound traffic of an AS may be routed on one of several outbound links, depending on the decision made by the inter-AS routing algorithm, usually Border Gateway Protocol(BGP). BGP routing decisions are made by a series of policy filters. Usually an AS may use the shortest AS path for most destinations. This may lead to unbalanced load distribution among the multiple outbound interfaces. In this section, we consider the problem of load-balancing outbound traffic in BGP from the perspective of a single AS. We show that this is an NP-hard problem and use the OLS framework to solve this problem.

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 to a particular address prefix. Usually it is used by the service providers on the request of their multi-homed customers. Lately, it is also being used between the service providers. The AS_PATH attribute has also been used to achieve TE objectives. AS_PATH is "stuffed" or "padded" with additional instances of the same AS number to increase its length and expect lower amount of inbound traffic from the neighbor AS to whom it is announced. However, this may lead to a large overhead if done too often. Another way used to achieve some TE is to subvert the BGP-CIDR address aggregation process. In particular an AS may extract more-specifics, or de-aggregate it and re-advertise the more-specifics to other ASes. The longest-prefix match rule in IP forwarding will lead to a different route for the more specific address. However, this is achieved at the expense of larger number of entries in forwarding tables. This is an indirect and undesirable way to achieve inbound load-balancing. One way to avoid subverting CIDR aggregation (shown in our recent work [58]), in the case of multi-homed *stub AS*, is by mapping the inbound load-balancing problem to an address management problem. Alternatively, AS neighbors may agree on BGP community attributes [59] (that are not re-advertised) to specify traffic engineering. We notice that inbound load-balancing is considerably complex and requires re-advertisements or support from neighboring ASes. How-

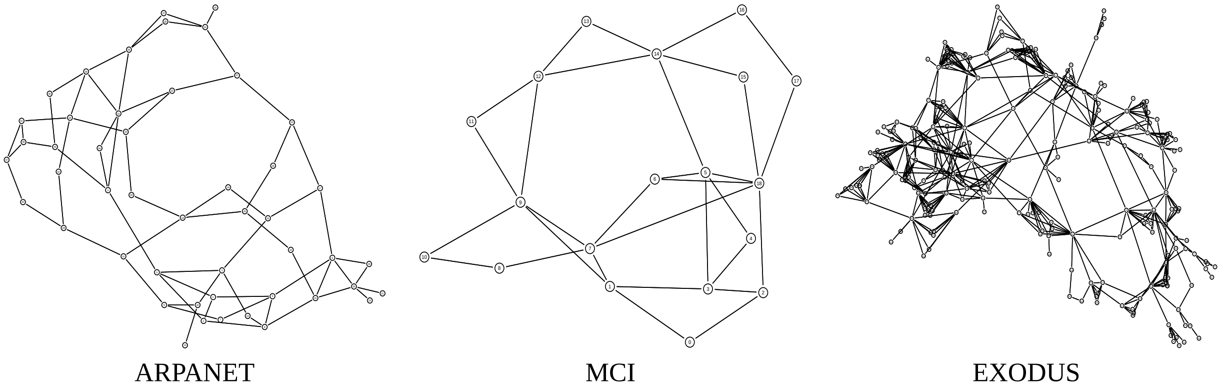

Fig. 14. Network topologies for simulations of OSPF link weight configuration

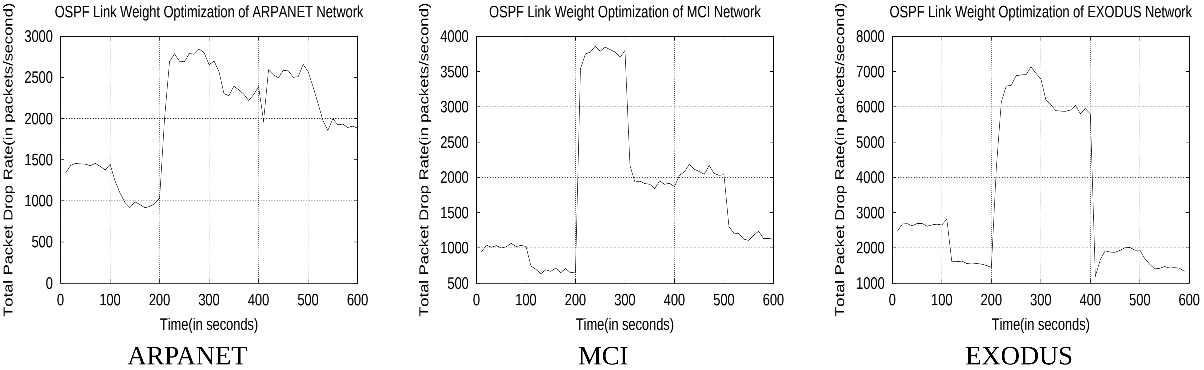

Fig. 15. OSPF Link weights adaptive configuration simulations: Traffic pattern was changed at times 0, 200, 400..., the optimized OSPF weights were deployed at times 100, 300,...

ever, outbound load-balancing is simpler, and can be achieved by impacting local policy changes.

The LOCAL_PREF attribute is used 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, i.e. the BGP will choose the path with highest LOCAL_PREF over other policy attributes. Therefore, if we know the desired routing to meet the traffic engineering objective, we can use the LOCAL_PREF to override the default routing. Recent work [3] observes 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. In fact, how to "shift" these hot-prefixes to achieve load balancing is a NP-hard problem as we will show later. Here we use the on-line simulation framework to tackle this 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 [42] 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 [42], [43] 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 [43]. Furthermore, these elephant traffic flows are usually very stable over time and hence are suitable to be re-routed 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. [1].

[1] The 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. Let l_i and c_i, $i = 1 \ldots m$, denote the i^{th} outbound link and its capacity (or bandwidth), respectively. All the outbound traffic of this AS will be routed on these links. If s_i, $i = 1 \ldots m$, denotes the total outbound traffic carried by the i^{th} link, then the utilization of link l_i is given by s_i/c_i. The objective of load balancing is to minimize the maximum link utilization among all the outbound links, i.e.,

$$\text{minimize} \max_{i=1\ldots m} \frac{s_i}{c_i} \tag{28}$$

Let n denote the number of selected destination prefixes and d_j, $j = 1 \ldots n$, 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 Equation (28). Let $\mathcal{D}_i$ denote the subset 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% traffic flows, which are routed to this link by the default BGP routing, then Equation (28) becomes

$$\text{minimize } \Phi \;=\; \max_{i=1\ldots m} (\sum_{j \in \mathcal{D}_i} \frac{d_j}{c_i}) + \frac{f_i}{c_i} \tag{29}$$

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.

$$\begin{aligned} \text{minimize} \quad & t \qquad\qquad\qquad\qquad\qquad\qquad (30) \\ \text{subject to} \quad & \sum_{j=1}^{n} x_{ij} \frac{d_j}{c_i} + \frac{f_i}{c_i} \leq t, \; i = 1 \ldots m \\ & \sum_{i=1}^{m} x_{ij} = 1, \; j = 1 \ldots n \\ & x_{ij} \in \{0, 1\}, \; i = 1 \ldots m, \; j = 1 \ldots n \end{aligned}$$

where x_{ij} is a binary number and $x_{ij} = 1$ means flow d_j is output on link l_i, otherwise $x_{ij} = 0$. Note that traffic flow d_j may not have all outbound links as its alternative paths. One can assume an arbitrarily large d_j/c_i for those links. The problem represented by Equation (30) is actually a classical task scheduling problem with unrelated parallel machines [61], 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 [62] a linear programming technique is first used to obtain a basic solution where there are at most $m - 1$ non-integral x_{ij}. Then for these non-integral x_{ij}, an exhaustive enumeration is performed to find the optimal scheduling. Combining the solutions of these two steps can produce an approximate solution with a upper bound of $2t^*$, where t^* denotes the value of t produced by the optimal solution. The time complexity of this method is exponential in 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 also prefers to use the shortest paths. It is possible to formulate a multi-objective optimization problem and obtain a solution, using OLS, that meets both load-balancing and shortest path criteria.

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

Step 1 Extract top 10% destination prefixes, with traffic demands d_j, $j = 1 \ldots n$, from the routing table;

Step 2 Calculate d_j/c_i and f_i/c_i, $i = 1 \ldots m$, $j = 1 \ldots n$ 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 Equation (29) 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.. Let Φ^*, $\overline{r}^*$ denote the value of objective function and corresponding routing at the end of optimization.

Step 6 If $|\frac{\Phi^0 - \Phi^*}{\Phi^0}| \geq \Delta$, 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

The simulations presented in this section demonstrated the load balancing for an AS including 8 outbound links with normalized capacities of 100, 100, 100, 100, 45, 45, 45, 12, respectively. We assume the number of top 10% destination prefixes which generate most of the traffic is 148. Note this number is chosen somewhat arbitrarily only for the description purpose. 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

Outbound Link Capacity	100	100	100	100	45	45	45	12
Before Optimization	0.07	0.25	0.20	0.33	0.60	0.45	0.48	**0.91**
After Optimization	0.25	0.33	0.27	0.30	**0.35**	0.33	0.34	0.23

TABLE II

LINK UTILIZATION OF OUTBOUND LINKS BEFORE AND AFTER OPTIMIZATION

distribution become 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. The on-line simulation framework tackles the parameter configuration problem with 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. The essence of the OLS framework is to formulate network parameter configuration as a back-box optimization problem. The major features of these optimization problems are examined and an efficient search algorithm, Recursive Random Search algorithm, is designed to address these problems. RRS emphasizes on finding a "good" solution within the limited time frame instead of full optimization, which is very important since the optimization is performed under "quasi-stationary" network conditions. The RRS algorithm performs very efficiently in the concerned context and is especially advantageous when handling objective functions affected by noises and those with negligible 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. In addition to these applications, many other network protocols can also be configured with this framework. Exploring other possible applications is one of our future research directions.

The overall approach proposed in this paper is limited by the extent we can model, measure and simulate networks. We can leverage more accurate and rapidly evaluated models/simulations in the future within this framework.

REFERENCES

[1] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. In *Proceedings of ACM SIGCOMM*, 2002.

[2] Dave Katz. Why are we scared of spf? igp scaling and stability. NANOG, June 2002.

[3] Nick Feamster, Jennifer Rexford, and Jay Borkenhagen. Controlling the impact of bgp policy changes on ip traffic. NANOG, June 2002.

[4] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and Willinger. Network topology generators: Degree-based vs. structural. In *Proceedings of ACM SIGCOMM 2002*, August 2002.

[5] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Proceedings of MASCOTS'01*, August 2001.

[6] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. In *Proceedings of ACM SIGCOMM 2002*, August 2002.

[7] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1), February 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 Transaction on Networking*, pages 265–278, June 2001.

[9] C. Carothers, D. Bauer, and S. Pearce. Ross: A high-performance, low memory, modular time warp system. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*, pages 53–60, May 2000.

[10] SSFNET. network simulator. http://www.ssfnet.org.

[11] OPFNET. network simulator. http://www.opnet.com.

[12] Nick Feamster and Jennifer Rexford. Network-wide bgp route prediction for traffic engineering. August 2002.

[13] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing ospf weights. In *Proceedings of the INFOCOM 2000*, pages 519–528, 2000.

[14] Wu chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shi. A self-configuring RED gateway. *Proceedings of IEEE Infocom 1999*, March 1999.

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

[16] Fred Glover. Tabu search– part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

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

[18] NS. network simulator. http://www.isi.edu/nsnam/ns/.

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

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

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

[22] T. C. Hu, V. Klee, and D. Larman. Optimization of globally convex functions. *SIAM Journal on Control and Optimization*, 27(5):1026–1047, 1989.

[23] K. D. Boese, A. B. Kahng, and S. Muddu. On the big valley and adaptive multi-start for discrete global optimizations. Technical Report TR-930015, UCLA CS Department, 1993.

[24] K. D. Boese, A. B. Kahng, and S. Muddu. a new adaptive multi-start technique for combinatorial global optimizations. *Operation Research Letters*, 16:101–113, 1994.

[25] Justin A. Boyan and Andrew W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1(2000):77–112, 2000.

[26] Robert H. Leary. Global optimization on funneling landscapes. *Journal of Global Optimization*, 18(4):367–383, December 2000.

[27] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo, and S. Lucidi. A new version of the price's algorithm for global optimization. *Journal of Global Optimization*, 10:165–184, 1997.

[28] Zelda B. Zabinsky. Stochastic methods for practical global optimization. *Journal of Global Optimization*, 13:433–444, 1998.

[29] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[30] S. Kirkpatrick, D.C. Gelatt, and M.P. Vechhi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[31] W. L. Price. Global optimization by controlled random search. *Journal of Optimization Theory and Applications*, 40:333–348, 1978.

[32] Soraya Rana, L. Darrell Whitley, and Ronald Cogswell. Searching in the presence of noise. In H. Voigt, W. Ebeling, I. Rechenberg, and

Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV (Berlin, 1996) (Lecture Notes in Computer Science 1141)*, pages 198–207, Berlin, 1996. Springer.

[33] Tao Ye and Shivkumar Kalyanaraman. A recursive random search for optimizing network protocol parameters. Technical report, ECSE Department, Rensslaer Polytechnique Institute, Dec 2001.

[34] S. Floyd and V. Jacobson. Random early detection gateways for congetsion avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413, August 1993.

[35] B. Braden and et al. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, 1998.

[36] Victor Firoiu and Marty Borden. A study of active queue management for congestion control. In *INFOCOM (3)*, pages 1435–1444, 2000.

[37] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. Technical report, INRIA Sophia-Antipolis, France, 1999.

[38] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith. Tuning RED for web traffic. In *Proceeding of ACM SIGCOMM*, 2000.

[39] Thomas Bonald, Martin May, and Jean-Chrysostome Bolot. Analytic evaluation of RED performance. *IEEE Infocom 2000*, pages 1415–1444, 2000.

[40] C.V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong. A control theoretic analysis of red. In *Proceedings of IEEE Infocom 2001*, 2001.

[41] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. unpubished, 2001.

[42] 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*, November 2001.

[43] Wenjia Fang and Larry Peterson. Inter-as traffic patterns and their implications. In *Proceedings of Global Internet 99*, Rio, Brazil, 1999.

[44] J. Moy. Ospf version 2. RFC 2178, April 1998.

[45] Atul Khanna and John Zinky. The revised arpanet routing metric. In *Proceedings of the ACM SIGCOMM*, pages 45–56, 1989.

[46] David W. Glazer and Carl Tropper. A new metric for dynamic routing algorithms. *IEEE Transactions on Communications*, 38(3), March 1990.

[47] A. Sakamoto D. S. Lee, G. Ramamurthy and B. Sengupta. Performance analysis of a threshold-based dynamic routing algorithm. In *Proceedings of the Fourteenth International Teletraffic Congress*, 1994.

[48] Dimitri P. Bertsekas. Dynamic models of shortest path routing algorithms for communication networks with multiple destinations. In *Proceedings of 1979 IEEE Conference on Decision and Control*, pages 127–133, Ft. Lauderdale, FL, Dec 1979.

[49] Jon Crowcroft Zheng Wang. Analysis of shortest-path routing algorithms in a dynamic network environment. *Computer Communication Review*, 22(2):63–71, 1992.

[50] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Netscope: traffic engineering for ip networks. *IEEE Network Magazine: special issue on Internet traffic engineeringh*, pages 11–19, March/April 2000.

[51] Ramesh Nagarajan, James F. Kurose, and Don Towsley. Approximation techniques for computing packet loss in finite-buffered voice multiplexers. *IEEE J.Select.Areas Commun*, 9(3):368–337, April 1991.

[52] Robert B. Cooper. *Introduction to Queueing Theory*. New York : North Holland, second edition, 1981.

[53] Harry G. Perros. *Queueing Networks With Blocking, Exact and Approximate Solutions*. Oxford University Press, 1994.

[54] Bernard Fortz and Mikkel Thorup. Increasing internet capacity using local search. *IEEE Transaction on Networking*, 2000.

[55] RocketFuel. Isp topology mapping engine. http://www.cs.washington.edu/research/networking/rocketfuel/.

[56] NAM. network animator. http://www.isi.edu/nsnam/nam/.

[57] G. Huston. Commentary on inter-domain routing in the internet. RFC 3221, December 2001.

[58] 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. ISMA Workshop by CAIDA, December 2001.

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

[60] Jay Borkenhagen, Nick Feamster, and Jennifer Rexford. Controlling the impact of BGP policy changes on IP traffic. NANOG, June 2002.

[61] L. A. Hall. *Approximation Algorithms for NP-hard Problems*, chapter Approximation Algorithms for Scheduling. PWS Publishing Company, 1995.

[62] C. N. Potts. Analyssis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Appl. Math.*, 10:155–164, 1985.