Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Networks 50 (2006) 1938–1951
Computer Networks
www.elsevier.com/locate/comnet

Measurement-based optimal routing on overlay architectures for unicast sessions ☆

Tuna Güven [a], Richard J. La [a,*], Mark A. Shayman [a], Bobby Bhattacharjee [b]

[a] *Department of Electrical and Computer Engineering, University of Maryland, A.V. Williams Building, College Park, MD 20742, United States*
[b] *Department of Computer Science, University of Maryland, College Park, MD, United States*

Available online 8 November 2005

Abstract

We propose a measurement-based routing algorithm to load-balance intradomain traffic along multiple paths for multiple unicast sources. Multiple paths are established using overlay nodes. The distributed algorithm is derived from simultaneous perturbation stochastic approximation (SPSA) and does not assume that the gradient of an analytical cost function is known. Instead, we rely on (potentially) noisy estimates from local measurements. Using the analytical model presented in the paper, we first show the almost sure convergence of the algorithm to the optimal solution under a decreasing step size policy (as with a standard SPSA model). Motivated by practical concerns, we next consider the constant step size case, for which we establish weak convergence. We provide simulation results to demonstrate the advantages of our proposed algorithm under various network scenarios, and also present a comparative study with MATE (an existing optimal routing algorithm).

Keywords: Mathematical programming/optimization; Load balancing; Simulations

1. Introduction

Rapid growth of the Internet and the emergence of new demanding services have sparked interest in Internet traffic engineering. As defined in [1], traffic engineering deals with the issue of performance evaluation and performance optimization of operational IP networks and encompasses the *measurement*, *characterization*, *modeling* and *control* of the Internet traffic.

Due to the evolution of the Internet from ARPANET, traditional routing algorithms for IP networks are mostly based on shortest-path routing. However, methods relying on a single path between a source–destination pair often do not utilize network resources efficiently and offer limited flexibility for traffic engineering [1]. Various solutions derived from shortest-path routing algorithms have been suggested, mainly by modifying link metrics in accordance with the network dynamics (e.g., [2,3]). However, these approaches have several

☆ A shorter version of this work appeared in the Proceedings of the IEEE Conference on Computer Communications (Infocom) 2004.

* Corresponding author. Tel.: +1 301 405 4914; fax: +1 301 314 9281.
E-mail address: hyongla@isr.umd.edu (R.J. La).

1389-1286/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2005.09.030

shortcomings that have not been addressed satisfactorily. First, they lead to network-wide effects and can result in undesirable and unanticipated traffic shifts [1]. Second, these schemes cannot distribute the load among the paths with different costs.

This paper focuses on the issue of *traffic mapping* (load balancing). More specifically, given some source–destination traffic matrix, we are interested in finding a traffic assignment onto pre-established paths so that the overall system performance is optimized with respect to a selected performance measure (e.g., sum of link costs). In this paper we adopt an overlay architecture proposed in [5] in order to establish multiple paths between a source–destination pair. However, our basic approach can be directly applied to other types of networks, such as MPLS-based networks.

We propose a distributed optimal routing algorithm based on stochastic approximation, using local network state information. Our model is similar to that of [4], with the following differences. In [4] the authors mention that the *cost derivatives* cannot be computed and hence, should be estimated by measurements. However, the mathematical analysis given in [4] implicitly assumes that the analytical gradient function is available. In addition, the details on how a cost gradient is estimated are not provided, and the method described in [6] appears to be a variant of a well-known *finite differences* method [7,8]. However, the issue of gradient estimate is not clearly or explicitly stated in the aforementioned references. We believe that the gradient estimation method plays a crucial role in the sense that the *convergence* of the optimal routing algorithm strongly depends on the conditions defining this estimation process as described in the stochastic approximation literature (see [8–10]).

In this study we consider the same problem (of mapping traffic onto multiple end-to-end paths) while relaxing the assumption that the analytical gradient function is available. We derive our proposed *measurement-based algorithm* from the idea of simultaneous perturbation stochastic approximation (SPSA). SPSA allows us to greatly reduce the number of measurements required for estimating the gradient, while at the same time it achieves approximately the same level of accuracy as the classical finite differences method at each iteration. By reducing the number of measurements, our algorithm achieves faster convergence. This is because each measurement requires a non-negligible amount of time in a networked environment. We will detail these issues of estimation, measurement, and convergence in the rest of the paper, and we show in Section 5, using simulations, that our SPSA based algorithm outperforms the algorithm proposed in [4].

From a broader perspective, a special case of the proposed algorithm (i.e., single source–destination (SD) case) provides an optimal solution to more general problems that have a *simplex constraint set*. Although applications of SPSA to the constrained optimization problems have generated a certain level of interest in the literature, the simplex constraint set problems have not been handled properly as we will discuss in the following section.

Note that the SPSA algorithm can achieve almost sure convergence, provided that the step size parameter diminishes with the number of iterations. However, such a policy may not be practical under dynamic network conditions as the algorithm may not be able to react to network changes in a timely manner after the step size parameter becomes small. As a result, in practice, we have to reset the step size value after some time interval to ensure that the algorithm is able to react to network dynamics appropriately. An obvious alternative is the use of a constant step size. Even though it is difficult to obtain similar almost-sure convergence in a constant step-size case, it has been shown in [11] that weak convergence (i.e., convergence in distribution) is possible under certain conditions. While the weak convergence is not as strong as almost-sure convergence, we demonstrate in Section 5 that it does not have a significant effect on the performance of the algorithm in a practical sense.

The rest of the paper is organized as follows: In Section 2 we define the optimization problem, and present a brief overview of stochastic approximation. In Section 3, we present an optimal routing algorithm, and prove its stability and optimality. We discuss implementation issues in Section 4. Section 5 describes the experimental setup and presents a set of simulation results. We conclude the paper and discuss possible topics of future work in Section 6.

2. The optimization problem

2.1. The routing model

In this section, we define the optimization problem of interest, describe the network model used for the analysis, and list basic assumptions we make.

We will closely follow the formulation in [4] due to the similarity of the problem.

The network is modeled by a set L of unidirectional links. Let $S = \{1, 2, \ldots, S\}$ denote the set of SD pairs. An SD pair s has a set $P_s \subseteq 2^L$ of paths available to it, and $N_s = |P_s|$, i.e., N_s is the number of paths available for SD pair s. Denote the set of links that belong to at least one path $p \in P_s$ by $L^s \subset L$. With a little abuse of notation we let $P_s = \{1, 2, \ldots, N_s\}$, and define the set of all paths $P = \cup_{s \in S} P_s = \{1, 2, \ldots, N\}$, where $N = \sum_{s \in S} N_s$. While by definition, none of the paths can be used by more than one SD pair, any two paths can share a link.

The total input traffic rate of an SD pair s is denoted by r_s, and the SD pair routes x_{sp} amount of traffic on path $p \in P_s$ such that

$$\sum_{p \in P_s} x_{sp} = r_s, \quad \text{for all } s. \tag{1}$$

Let $\boldsymbol{x}_s = (x_{sp}, p \in P_s)$ be the rate vector of SD pair s, and let $\boldsymbol{x} = (\boldsymbol{x}_s, s \in S)$ be the vector of all rates. The flow rate on a link $l \in L$ is given by

$$x^l = \sum_{s \in S} \sum_{p \in P_s:\, l \in p} x_{sp}. \tag{2}$$

For each link l, $C_l(x^l)$ represents the link cost as a function of the link flow rate x^l. We assume that, for all l, $C_l(\cdot)$ is convex and continuously differentiable. The objective is to minimize the total cost $C(\boldsymbol{x}) = \sum_l C_l(x^l)$ by mapping the traffic on paths in P:

$$\min_{\boldsymbol{x}} C(\boldsymbol{x}) = \min_{\boldsymbol{x}} \sum_l C_l(x^l) \tag{3}$$

$$\text{s.t.} \sum_{p \in P_s} x_{sp} = r_s, \quad \forall s \in S, \tag{4}$$

$$x_{sp} \geqslant \epsilon, \quad \forall p \in P_s,\ s \in S, \tag{5}$$

where ϵ is an arbitrarily small positive constant. For instance, some of the control packets may be routed along different paths available to an SD pair.

Theoretically if the exact gradient values are known, one can use the well known gradient projection algorithm to solve this constrained optimization problem, where the constraint set Θ is defined by (4) and (5). Each iteration of the algorithm takes the form:

$$\boldsymbol{x}(k+1) = \Pi_{\Theta}[\boldsymbol{x}(k) - a(k)\nabla C(k)], \tag{6}$$

where $\nabla C(k)$ is the gradient vector whose (s,p)th element is the first derivative length of path $p \in P_s$ at the kth iteration $([\nabla C(k)]_{sp} = \partial C(\boldsymbol{x}(k))/\partial x_{sp})$, $a(k) > 0$ is the step size, and $\Pi_{\Theta}[\vartheta]$ denotes the projection of a vector ϑ onto the feasible set Θ with respect to the Euclidean norm.

The above iteration can be carried out in a distributed manner by each pair s without the need to coordinate with other pairs [12,13]. In other words, the source of each SD pair s updates its rates $\boldsymbol{x}_s$ independently of other SD pairs:

$$\boldsymbol{x}_s(k+1) = \Pi_{\Theta_s}[\boldsymbol{x}_s(k) - a_s(k)\nabla C_s(k)], \tag{7}$$

where $\nabla C_s(k) = (\partial C(\boldsymbol{x}(k))/\partial x_{sp}, p \in P_s)$ is the vector of first derivative lengths of paths in P_s, i.e., a subvector of $\nabla C(k)$ with elements correponding to the paths in P_s, and Π_{Θ_s} denotes a projection onto the feasible set Θ_s of SD pair s.

One problem with directly implementing (7) is that the first derivative length of a path, $\partial C/\partial x_{sp}$, may not be available in practice and can only be estimated empirically through *noisy* measurements of the cost function. This is due to the fact that the packet arrival processes are both stochastic and time varying. Therefore, one must resort to a gradient approximation method to obtain an estimate to be used in (7). Stochastic approximation methods are natural candidates for such problems.

2.2. Stochastic approximation

Stochastic approximation (SA) is a recursive procedure for finding the root(s) of equations using noisy measurements, and is particularly useful for finding extrema of functions [9] (e.g., [14,15]).

General constrained SA has the same form as (6) with the gradient vector $\nabla C(k)$ replaced by its approximation $\hat{\boldsymbol{g}}(k)$. The approximation is obtained through measurements of $C(\boldsymbol{x})$ around $\boldsymbol{x}(k)$. Under appropriate conditions, one can show that $\boldsymbol{x}(k)$ converges to the solution set of (3), which we denote by $\mathbf{x}^*$.

A critical issue in SA is the approximation of gradient vector. The standard approach motivated from the definition of gradient is the *finite differences* (FD) method, in which each component of $\boldsymbol{x}(k)$ is perturbed one at a time and corresponding measurements $y(\cdot)$ are obtained. Typically, the ith component of $\hat{\boldsymbol{g}}(k)$ $(i = 1, 2, \ldots, m)$ under the FD method is given by

$$\hat{g}_i(k) = \frac{y(\boldsymbol{x}(k) + c(k)\boldsymbol{e}_i) - y(\boldsymbol{x}(k) - c(k)\boldsymbol{e}_i)}{2c(k)},$$

where $c(k)$ is some positive number, $\boldsymbol{e}_i$ denotes a unit vector with one in the ith position and zeros

elsewhere, and $y(\cdot)$ denotes the measured cost function with measurement noise.

An alternative method for estimating the gradient is called the *simultaneous perturbation* (SP). In this method, all elements of $\boldsymbol{x}(k)$ are randomly perturbed together to obtain two measurements. The ith component of $\hat{\boldsymbol{g}}(k)$ is computed by

$$\hat{g}_i(k) = \frac{y(\boldsymbol{x}(k) + c(k)\boldsymbol{\Delta}(k)) - y(\boldsymbol{x}(k) - c(k)\boldsymbol{\Delta}(k))}{2c(k)\Delta_i(k)},$$

where the vector of the random perturbations for SP, $\boldsymbol{\Delta}(k) = (\Delta_1(k), \Delta_2(k), \ldots, \Delta_m(k))$, needs to satisfy certain conditions that will be discussed in the following section. Here m denotes the dimension of the vector $\boldsymbol{x}$.

Both of the above approximations have a "two-sided" form in the sense that they use the measurements $y(\boldsymbol{x}(k) \pm \textit{perturbation})$. On the other hand, one-sided gradient approximations require measurements of $y(\boldsymbol{x}(k))$ and $y(\boldsymbol{x}(k) + \textit{perturbation})$. Although it is known that the standard two-sided form gives more accurate estimates compared to one-sided forms, for real-time applications one-sided gradient approximation may be preferred when the underlying system dynamics change too rapidly to get an accurate gradient estimate with two successive measurements [7]. In this paper we assume that the one-sided form is utilized for the gradient approximation under both methods unless stated otherwise.

An SA algorithm using the FD (resp. SP) gradient approximation method is referred to as a FDSA (resp. SPSA) algorithm. One should note that, in an SPSA algorithm the gradient approximation requires only two cost function measurements, regardless of the value of m. Standard (two-sided) FD approximation requires $2m$ measurements to estimate the gradient. In [9] it is shown that under reasonably general conditions, SPSA and FDSA achieve the same level of statistical accuracy for a given number of iterations even though SPSA uses m times fewer function evaluations than FDSA. This theoretical result has been confirmed in many numerical studies, even in cases where m is on the order of several hundreds or thousands [7]. This is certainly an important property especially if the measurements are costly and/or time consuming. Clearly, this is the case for the routing problem at hand as measurements require resources and must be collected and reported in a timely manner. In other words, SPSA promises a potential for better statistical accuracy over the same period of "time" due to a much shorter measurement period required at each iteration, even though the two methods have the same statistical accuracy with the same number of "iterations". This suggests that the algorithm based on SPSA will be able to track and respond to changes in a network much faster than another algorithm based on FDSA and thus improve the overall network performance.

In [9], Spall provides a formal proof of convergence of SPSA algorithm for the "unconstrained" case. The convergence of an SPSA algorithm under inequality constraints are presented in [10,16]. However, these results do not consider the case where $\boldsymbol{x}(k) \pm c(k)\boldsymbol{\Delta}(k) \notin \Theta$, which may occur in our routing problem. In [16] Sadegh suggests projecting $\boldsymbol{x}(k)$ to a point $\boldsymbol{x}'(k) \in \Theta$ such that $\boldsymbol{x}'(k) \pm c(k)\boldsymbol{\Delta}(k) \in \Theta$. If $\boldsymbol{x}'(k) - \boldsymbol{x}(k) \to 0$ as $k \to \infty$, convergence can still be established. However, when Θ is a simplex, if $c(k)\sum_j \Delta_j(k) \neq 0$ then $\boldsymbol{x}'(k) \pm c(k)\boldsymbol{\Delta}(k) \notin \Theta$ for all $\boldsymbol{x}'(k)$ (as the demand constraint in (5) is violated). Under these conditions, there is no established convergence result of an SPSA algorithm that we can directly apply to our problem. (In [10], although authors claim that they have shown the convergence for the case of a network of queues with similar constraints, they do not consider the aforementioned issue in the proofs.)

In the next section, we will resolve this technical issue by a simple method and present a formal proof of the SPSA algorithm under these constraints.

3. Optimal routing using SPSA

3.1. An optimal routing algorithm—decreasing step size

In this section we propose an optimal routing algorithm and prove its stability and optimality. We know from [12] that if each SD pair runs (7) independently and asynchronously,[1] the overall algorithm converges. Let us now consider the use of SPSA in place of (7).

At time k, SD pair s updates its rate according to

$$\boldsymbol{x}_s(k+1) = \Pi_{\Theta_s}[\boldsymbol{x}_s(k) - a_s(k)\hat{\boldsymbol{g}}_s(k)], \tag{8}$$

where $\hat{\boldsymbol{g}}_s(k)$ is the approximation to $\nabla C_s(k)$ obtained by the SPSA algorithm and is given by

[1] Here asynchrony refers to the fact there might be a time lag between SD pairs in continuous time operation in practice.

$$\hat{g}_{s,i}(k) := \frac{N_s}{N_s-1}\frac{y_s(\boldsymbol{x}(k)+\mathbf{C}(k)\boldsymbol{\Delta}(k))-y_s(\boldsymbol{x}(k))}{c_s(k)\Delta_{s,i}(k)} = \frac{N_s}{N_s-1}\frac{(C_s^+(k)+\mu_s^+(k))-(C_s^-(k)+\mu_s^-(k))}{c_s(k)\Delta_{s,i}(k)}, \quad \text{for } i=1,\ldots,N_s. \tag{9}$$

Here $\boldsymbol{\Delta}(k)=(\boldsymbol{\Delta}_s(k), s\in S)$, $\boldsymbol{\Delta}_s(k)$ is the random perturbation vector for source s at iteration k, $\mathbf{C}(k)$ is an $N\times N$ diagonal matrix whose jth diagonal entry is equal to c_{s_j} (s_j being the SD pair associated with the jth component of $\boldsymbol{\Delta}(k)$), and $y_s(\cdot)$ are the noisy measurements of the *partial* cost information, which is the summation of the link costs over the set L^s. Specifically, $C_s^-(k)=\Lambda_s(\boldsymbol{x}(k)) := \sum_{l\in L^s} C_l(x^l(k))$ and $C_s^+(k)=\Lambda_s(\boldsymbol{x}(k)+\mathbf{C}(k)\boldsymbol{\Delta}(k))$ with $\mu_s^+(k)$ and $\mu_s^-(k)$ being the measurement noise terms.

Note that (9) differs from the standard SA in the following ways. First, each SD pair uses only *partial* cost information (i.e., summation of the costs of the links in L^s) as opposed to the total network cost, which is the summation of the costs of all the links in the network. In addition, the noise terms observed by each SD pair are allowed to be different. Second, while $c(k)$ is a positive scalar in standard SA, in our case $\mathbf{C}(k)$ is an $N\times N$ diagonal matrix. This allows the possibility of having different $c_s(k)$ values for different SD pairs. Also, note that we have an extra multiplicative factor $\frac{N_s}{N_s-1}$ in (9) compared to the standard SA. This is due to the projection of $\boldsymbol{x}_s(k)+c_s(k)\boldsymbol{\Delta}_s(k)$ to Θ_s for all $s\in S$ using L_2 projection while calculating $\hat{\boldsymbol{g}}_s(k)$. This is explained in Appendix II in detail. Finally, if $\Pi_{\Theta_s}[\boldsymbol{x}_s(k)+c_s(k)\boldsymbol{\Delta}_s(k)]=\boldsymbol{x}_s(k)$, the SD pair draws a new perturbation vector $\boldsymbol{\Delta}_s(k)$ until $\boldsymbol{x}_s(k)\neq\Pi_{\Theta_s}[\boldsymbol{x}_s(k)+c_s(k)\boldsymbol{\Delta}_s(k)]$.

Note from (8) that SD pairs may have different step sizes $a_s(k)$. This allows a certain level of asynchrony between SD pairs in the sense that SD pairs can respond to the network changes independently of each other to some extent.[2] However, we assume that SD pairs update their rates once every iteration after they start the algorithm. This assumption is reasonable in our case, for at each iteration SD pairs should make use of the collected information that is already available. This is, however, not to say that the updates take place simultaneously. The errors due to this asynchrony are assumed to be absorbed into the error terms $\mu_s^{\pm}(k)$ in (9).

[2] For instance, this formulation covers the case where SD pairs start the algorithm at different times.

For the optimality of the new algorithm, we need to show that (8) converges to the same point $\boldsymbol{x}_s^*$ as (7) for all SD pairs. For the convergence of the algorithm we assume that the following conditions are true:

A1. $C(\boldsymbol{x}(k))$ is differentiable for each $\boldsymbol{x}(k)\in\Theta$, and convex.

A2. $\Delta_{s,i}(k)$ are (i) mutually independent with zero mean for all $s\in S$ and $i\in P_s$, (ii) uniformly bounded by some finite constant α, and (iii) independent of $(\boldsymbol{x}(n),\ n=0,1,\ldots,k)$. $\mathbf{E}[(\Delta_{s,i}(k))^{-1}]$ and $\mathbf{E}[(\Delta_{s,i}(k))^{-2}]$ are bounded for all k.

A3. $\mathbf{E}[\mu_s^{(\pm)^2}(k)]$ are bounded and $\mathbf{E}[\mu_s^+(k)-\mu_s^-(k)|\boldsymbol{\Delta}(k),\mathscr{F}_k]=0$ a.s. for all k, where $\mathscr{F}_k$ is the σ-field generated by $\{\boldsymbol{x}(0),\ldots,\boldsymbol{x}(k)\}$.

A4. (i) $\sum_{k=0}^{\infty}a_s(k)=\infty$, (ii) $a_s(k)\to 0$ as $k\to\infty$, (iii) $\sum_{k=0}^{\infty}\frac{a_s^2(k)}{c_s^2(k)}<\infty$, (iv) $c_s(k)\to 0$ as $k\to\infty$, and (v) $\frac{c_s(k)}{c_{s'}(k)}=\mathrm{O}(1)$ for all $s,s'\in S$.

A5. There exists a finite positive constant M such that
$$\frac{1}{M}\leqslant\frac{a_s(k)}{a_{s'}(k)}\leqslant M \tag{10}$$
for all s, $s'\in S$ and for all k.

A6. Let $\hat{a}(k)=\max_{s\in S}a_s(k)$. Then, for all $s\in S$
$$\sum_{k=0}^{\infty}(\hat{a}(k)-a_s(k))<\infty, \quad \text{and}$$
$$\lim_{k\to\infty}\frac{a_s(k)}{\hat{a}(k)}=1 \qquad \text{for all } s\in S.$$

Proposition 1. *Under Assumptions* A1–A6, *the sequence* $x(k)=(x_s(k),\ s\in S)$ *generated by the algorithm defined by* (8) *converges to* $\mathbf{x}^*$ *with probability* 1, *where* $\mathbf{x}^*$ *is the solution set of the optimization problem in* (3), *regardless of the initial vector* $(x_s(0), s\in S)\in\Theta$.

Proof. The proof of the Proposition 1 is given in Appendix I. □

3.2. The optimal routing algorithm—constant step size

In the previous subsection we have employed decreasing step sizes $a_s(k)$, $k=0,1,\ldots$, as defined in Assumption A4. Although it is possible to use decreasing step sizes, it is of practical importance to consider the case with constant step size $a>0$ given in the following form:

$$\boldsymbol{x}_s(k+1) = \Pi_{\Theta_s}[\boldsymbol{x}_s(k) - a \cdot \hat{\boldsymbol{g}}_s(k)]. \quad (11)$$

This is because in the case of decreasing step sizes, one has to make sure that each source node resets the step size after a certain period of time when the step size becomes too small to effectively react to the dynamics of the network. Consequently, such a requirement introduces additional complexity which can be avoided by using a constant step size. The main difficulty with constant step size algorithms is that it is difficult to establish almost sure convergence. However, as shown in [11], under certain conditions constant step size SA algorithms can achieve weak convergence (i.e., convergence in distribution), which can be interpreted as convergence to a neighborhood of the optimal operating point(s). Since the performance of the system near the optimal operating point(s) may be comparable to that of the optimal solution(s) in a network problem, the performance degradation, if there is any, due to a constant step size may not be significant. This is supported by simulation results in Section 5.

The following proposition establishes the (weak) convergence of the algorithm with a constant step size.

Proposition 2. *Under Assumptions* A1–A3, *for any* $\delta > 0$, *the fraction of time the sequence* $x(k) = (x_s(k), s \in S)$ *generated by the algorithm defined by* (11) *spends in* δ*-neighborhood of* $\mathbf{x}^*$ *on* $[0,k]$, *goes to one* (*in probability*) *as* $a \to 0$ *and* $k \to \infty$ *regardless of the initial vector* $(x_s(0), s \in S) \in \Theta$.

Proof. The proof of Proposition 2 is given in Appendix III. □

3.3. Measurement process

In this section, we discuss some of the issues regarding the measurement process and their effect on the overall performance of the proposed algorithm. We will also point out the benefits of SPSA-based algorithms over the FDSA alternatives.

By definition, an FDSA-based algorithm requires SD pair s to perturb its paths *one at a time*, requiring $N_s + 1$ measurements for the one-sided form and $2N_s$ measurements for the two-sided form in each iteration for the estimation of an $N_s \times 1$ gradient vector. For this reason, FDSA-based algorithms need each SD pair to collect measurements (i.e., perturb its paths) at different times. As mentioned in [4], this requires a coordination protocol that determines the order in which paths are perturbed. Moreover, it increases network traffic load from the overhead.

SPSA, on the other hand, allows us to estimate the gradient vector using only two measurements. In the context of our routing problem, this implies that an SD pair can perturb all of its paths simultaneously if an SPSA-based algorithm is employed. In addition, it also suggests that all SD pairs can execute the perturbation in parallel without the need for strict coordination needed in FDSA (Propositions 1 and 2). This enables SD pairs to operate independently of each other to some extent. As a consequence, a potential overhead that would be incurred by a coordination protocol under an FDSA algorithm is avoided. Furthermore, we can significantly reduce the duration of measurement periods by concurrently executing the measurement process at the SD pairs. Since the statistical accuracy of SPSA does not degrade from that of FDSA, we can achieve much faster convergence due to the fact that we significantly reduce the time required for each iteration.

Another issue regarding the measurement process is the effect of asynchronous operation of SD pairs in practice due to the lack of perfect synchronization. It is proved in [4] that, with increasing level of asynchrony, the convergence becomes slower. For each $s \in S$, let t_0^s be the time lag between the time traffic measurements are collected and the time SD pair s carries out a rate update. In other words, t_0^s is the delay between the collection of measurements and execution of an SPSA algorithm by SD pair s. Define $t_0 = \max_{s \in S} t_0^s$. Then, the larger the t_0 is, the slower the convergence will be due to the use of delayed information. On the other hand, in the SPSA case as the level of asynchrony between the SD pairs increases, on the average, the magnitude of the error term in measurements tends to become smaller since the duration of the interval over which the measurements overlap with each other gets shorter, and this may cause a marginal increase in the overall system performance. As we will see in Section 5, these two effects mainly cancel each other and the performance of the algorithm does not degrade significantly if the time lag is not large.

4. Implementation issues

In this section, we give a brief overview of an overlay architecture used to enable traffic engineering

capabilities. The details of the overlay architecture can be found in [5].

4.1. Path establishment

Alternative paths between SD pairs are created using overlay nodes. The overlay nodes are located at all source–destination nodes as well as at some core nodes. The basic idea is similar to the ones presented in [17,18], with the difference being that the overlay is implemented within a domain (i.e., intra-domain) as opposed to inter-domain. When a packet is sent along the default path (e.g., the shortest path), it is forwarded in the same way as in traditional IP networks. However, when a packet is to be routed through an alternative path, it is processed at the source overlay node and an additional IP header is attached to the packet. This way the packet is first tunneled to a specific overlay node that lies in the selected alternative path, using the underlying routing protocol in use. When the overlay node receives the packet, it removes the outer IP header and forwards the packet to the final destination (or possibly to another overlay node). It is plain to see that one can utilize as many alternative paths as needed by placing overlay nodes at appropriate locations. Note that using this architecture, we can still employ the simple shortest path routing inside the network, without having to modify the existing traditional routers. The overlay capabilities can be realized by attaching a simple device (e.g., a host with network processor) to the existing routers. This device simply processes the packets, and adds or removes IP headers before the basic forwarding operation is performed at the routers.

As a final remark, we would like to emphasize the point that the basic mechanism of the proposed routing algorithm can be employed in different types of networks. For instance, it can also be deployed in an MPLS based network, where the overlay paths are replaced with LSPs (label switched paths). The proposed use of overlay architecture described in this paper enables us to adopt the algorithm in the traditional IP-based networks.

4.2. Traffic monitoring

Traffic monitoring is also handled by the overlay architecture. Each link in the network is mapped to the closest overlay node with a tie-breaking rule that gives a unique mapping [5]. Overlay nodes periodically poll the links that they are responsible for, process the data and forward necessary local state information to the SD pairs utilizing the corresponding links. This eliminates the need for *each* SD pair to probe the links used by it. Note that, before forwarding the link-cost information to source nodes of SD pairs, the overlay nodes can aggregate the information gathered from different links. For example, due to the additive cost structure (according to the definition given in (3)), if the overlay nodes are aware of L^s, an overlay node can first compute the sum of the link costs over the links in L^s it is responsible for and then report only the total cost to the source node of SD pair s.

As a consequence, the overhead caused by the distribution of the link-state information is minimized.

5. Experimental setup and simulation results

In this section we evaluate the performance of the proposed algorithms through simulations under various network conditions, and study their convergence rates. To this end, we developed a packet-level discrete-event simulator to carry out the simulations. Each plot presented in this section is the average of 10 independent simulation runs with different random seeds.

For the simulations we select the cost function of the form

$$C_l(\boldsymbol{x}) = d_l(\boldsymbol{x}) + u_l(\boldsymbol{x})^2, \tag{12}$$

where $d_l(\boldsymbol{x})$ is the number of packets dropped on link l,[3] and $u_l(\boldsymbol{x})$ is the link utilization. In all simulations, the measurement period is selected to be 1 s. As a consequence, SD pairs can update their rates at best approximately every 2 s since they require two measurements for estimating the gradient vector using the SPSA algorithm.

The requirements on the cost function are stated in Assumption A1, and one can argue that the selected cost function satisfies these requirements as follows. Since we deal with backbone networks, the packet arrival process at a source node is an aggregate of many individual flows. We assume that each individual flow generates packets according to an equilibrium renewal process, i.e., interarrival times of packets from a flow have a fixed distribu-

[3] In the simulation we have used the observed number of packet drops, while in the model this term in the cost function can be interpreted as the expected number of packet losses as a function of the link load.

tion, and these equilibrium renewal processes are mutually independent. Then, by the Palm–Khintchine theorem [19], the superposition of these independent renewal processes can be approximated by a Poisson process, where interarrival times of packets are exponentially distributed.

In addition, according to the work presented in [20], the packet size distribution of Internet traffic has two peaks at 500 and 1500 bytes. Using this observation, we can approximate the packet size as a Bernoulli random variable with values at 500 and 1500 bytes.

Under the above conditions, we can approximate the links in the network as $M/G/1/K$ queues, where K is the buffer size. Following this assumption we can justify the assumption on convexity of the cost function as follows. One can check that in the regime of interest (e.g., with utilization level being less than 150%), the link cost function is convex in the case of $M/M/1/K$ queue. In the case of $M/G/1/K$ queue one can show that the approximation functions for blocking probability of an $M/G/1/K$ queue (e.g., Gelenbe's formula [21] and two-moment approximation in [22]), are indeed convex in the regime of interest under various parameter settings.

Simulations are carried out under two different network topologies. The first topology, which is used in [4], is given in Fig. 1. Due to its simplicity this topology allows us to obtain insights into the fundamental behavior of the proposed algorithm. In addition, it serves us as a base setup so that we can make a comparison with the MATE algorithm presented in [4]. We have three SD pairs (S1–D1, S2–D2 and S3–D3) and each pair has two distinct paths. Note that this creates a considerable amount of interaction between these SD pairs.

The setup we use is similar to the one adopted in [4] for comparison purposes. (See [23] for the details of this setup.) The network consists of identical

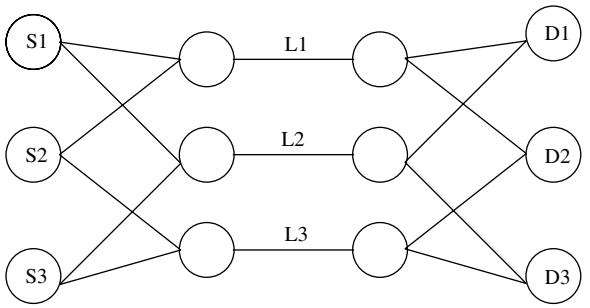

Fig. 1. Network topology 1.

Table 1
The cross-traffic dynamics

Link	Load distribution in time (s)		
	[0–1000)	[1000–2500)	[2500–3600)
L_1	0.77	0.44	0.44
L_2	0.33	0.33	0.67
L_3	0.33	0.33	0.33

links with a bandwidth of 45 Mbps. The average packet size is 257 bytes. Each SD pair initially uses only the default shortest (minimum hop distance) path. Since all paths have equal length, the default min-hop paths are selected such that L2 is along the default shortest path of S1–D1, while the default shortest paths of S2–D2 and S3–D3 both traverse L3. Each SD pair generates traffic according to a Poisson process with an average rate of 19.8 Mbps (corresponding to 0.44 link utilization). In addition, links L1, L2 and L3 carry uncontrolled cross-traffic generated by Poisson processes. The average rate of cross-traffic normalized by the link capacity is given in Table 1. A random delay is introduced before each SD pair starts running the optimal routing algorithm to ensure that the SD pairs are not synchronized. (The maximum value of this random delay is defined as offset.) In this simulation a decreasing step size policy is adopted that satisfies the conditions in Assumptions A3–A6. Specifically, the step sizes are given by $a_s(k) = 15/(k + 100)^{0.602}$ and $c_s(k) = 75/(k^{0.101})$.[4]

As shown in Figs. 2 and 3, the algorithm quickly eliminates the congestion and successfully balances the traffic. Moreover, these results show that the proposed algorithm clearly outperforms the MATE algorithm [4]; while the MATE algorithm requires around 400–500 s to converge,[5] our algorithm takes around 200 s. Furthermore, our algorithm quickly (around 50 s) eliminates packet drops unlike MATE. (See Figs. 10 and 11 presented in [4].)

Figs. 4 and 5 illustrate the effect of increased asynchrony between SD pairs. We increase the asynchrony between SD pairs by simply increasing the offset values. From these plots we can conclude that the algorithm is still able to converge in a short

[4] $a_s(k)$ and $c_s(k)$ are reset to their initial values at simulation times 1001 and 2501 s for faster convergence.

[5] Since simulation code and packet size distributions for the MATE algorithm are proprietary, it was not possible to simulate MATE. Therefore, we base our comparison on the results presented in [4].

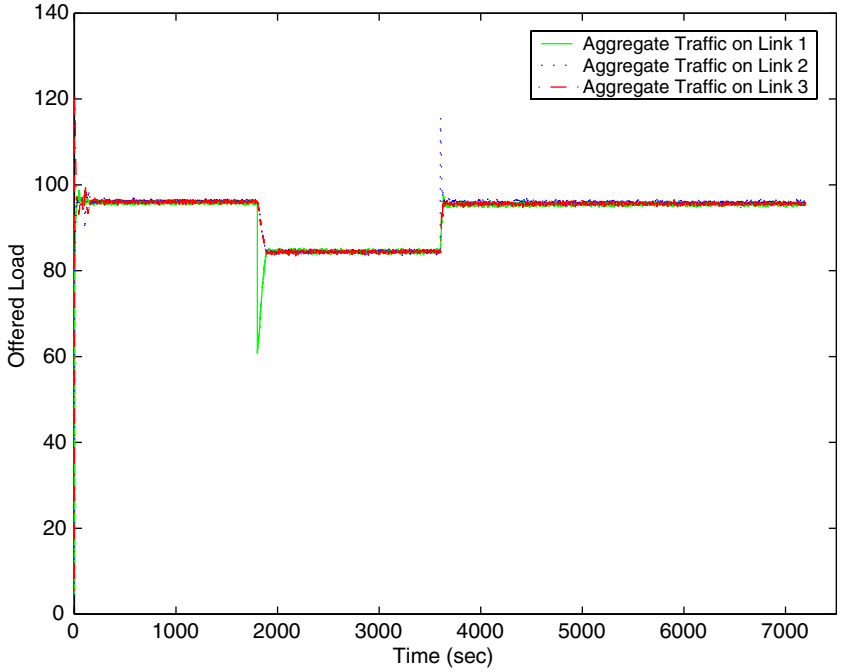

Fig. 2. Network topology 1 with an offset of 50 ms.

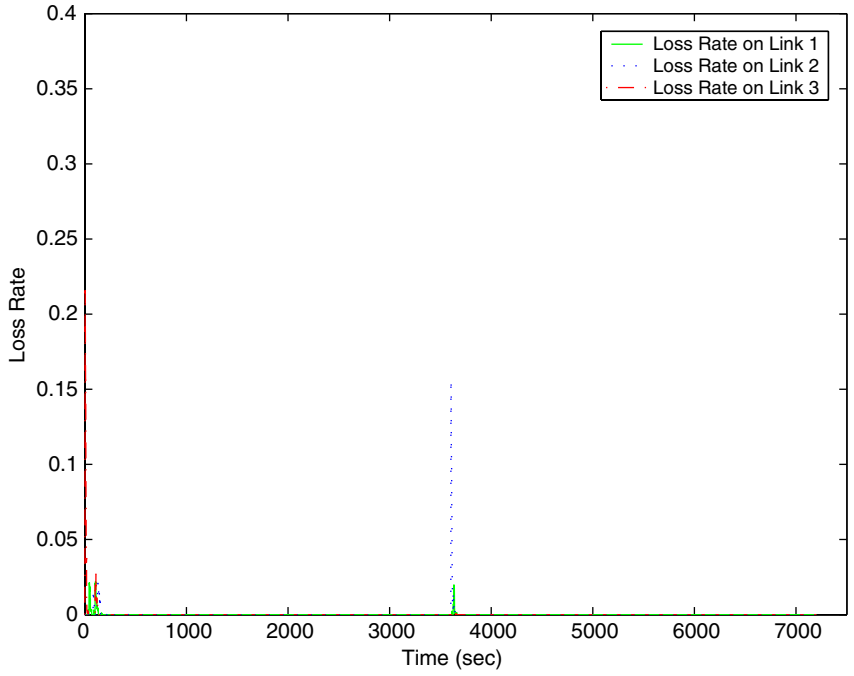

Fig. 3. Network topology 1 with an offset of 50 ms.

time. As we see from Figs. 2 and 4, the performance is almost the same for offset values of 50 ms and 200 ms. However, when we increase the offset to 500 ms, we see that the convergence of our algorithm becomes slightly slower. Thus, these results validate the earlier discussion made in Section 3.3.

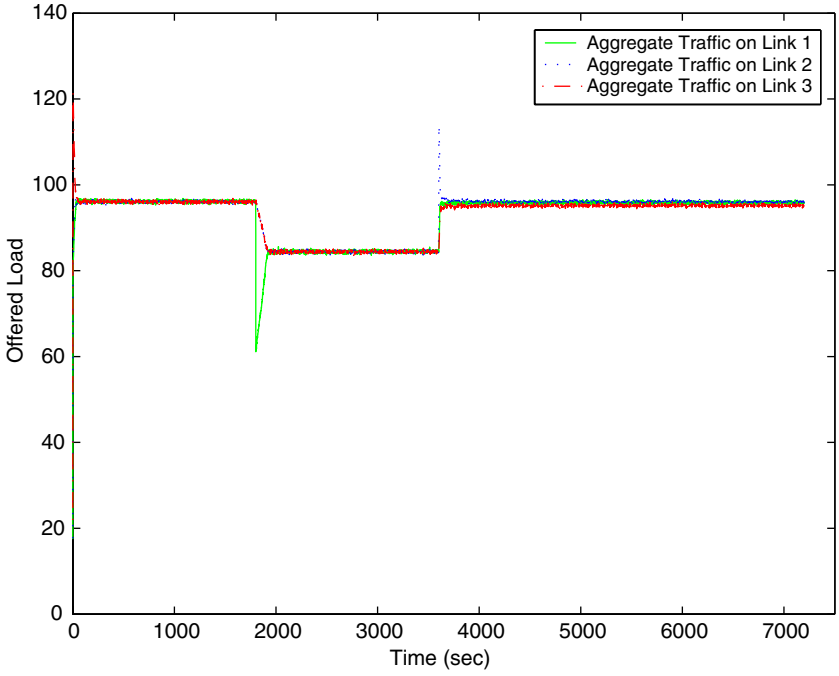

Fig. 4. Network topology 1 with an offset of 200 ms.

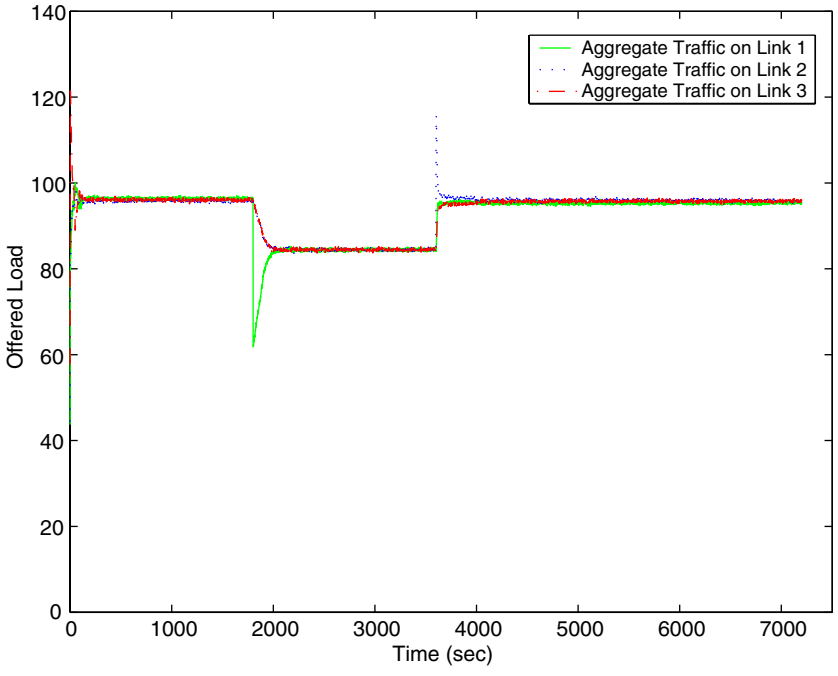

Fig. 5. Network topology 1 with an offset of 500 ms.

Fig. 6 represents the second topology we consider in this paper. This topology is also used in [24,25] and closely resembles the MCI Internet topology presented in [26]. Using this topology, we analyze the performance of the proposed algorithm under more realistic network conditions.

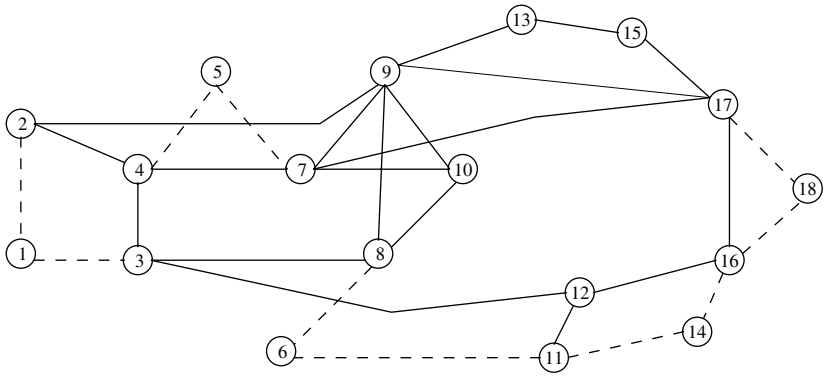

Fig. 6. Network topology 2.

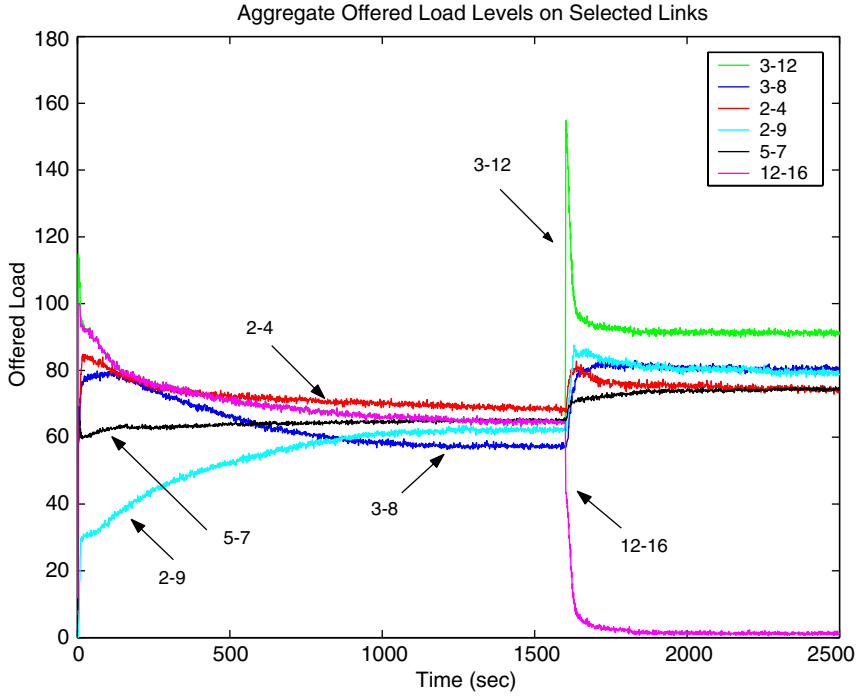

Fig. 7. Offered load on network topology 2.

Nodes 1, 5, 6, 14 and 18 serve as both source and destination nodes. This gives us a total of 20 SD pairs. Each pair has at least two paths to reach the destination. A total of 78 paths are created between these 20 SD pairs, using overlay architecture. Overlay capability is available at all source/destination nodes as well as nodes 2, 10 and 13. In this experiment, the offset is set to 0.1 s. The links shown as dashed lines have a capacity of 50 Mbps, while the links represented by solid lines have a capacity of 20 Mbps. The packet size for this scenario is selected to be 500 bytes. All SD pairs initially use only the shortest paths. Each SD pair generates traffic with a rate of 11.5 Mbps. In addition, the cross-traffic traverses link (3–12), starting at simulation time 1600 s. The cross-traffic rate is 18 Mbps and cannot be shifted to any alternative paths as before.

In Fig. 7, we illustrate how the load is distributed by our algorithm.[6] The links for which we plotted the load are selected in such a way that each of them

[6] $a_s(k)$ is reset to its initial value (0.15) at simulation time 1601 s for faster convergence.

is located on a distinct alternative path that can be used to divert the traffic sent on link (3–12). The only exception is link (12–16), which is a downstream link of link (3–12) for some of the paths

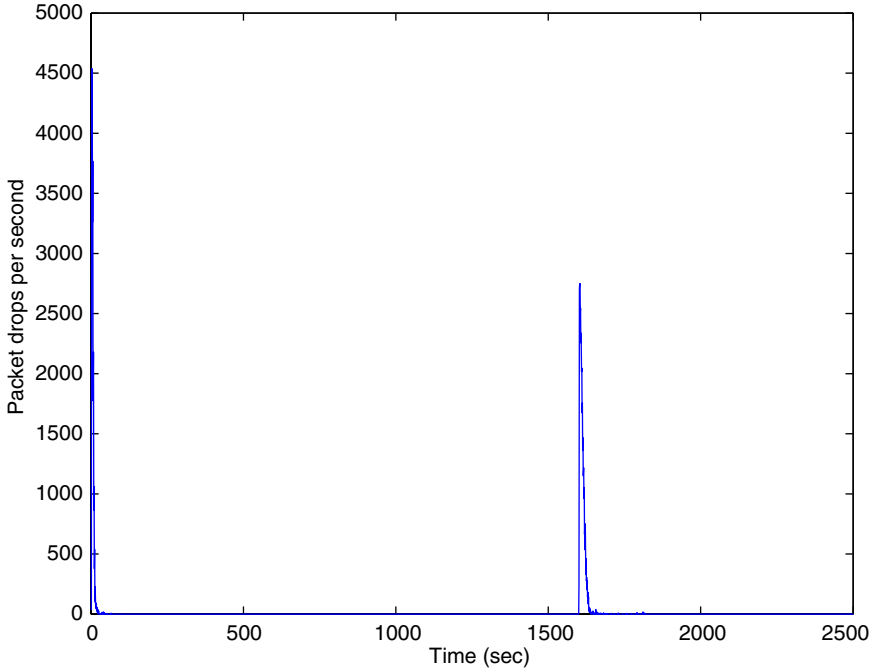

Fig. 8. Total packet drops on network topology 2.

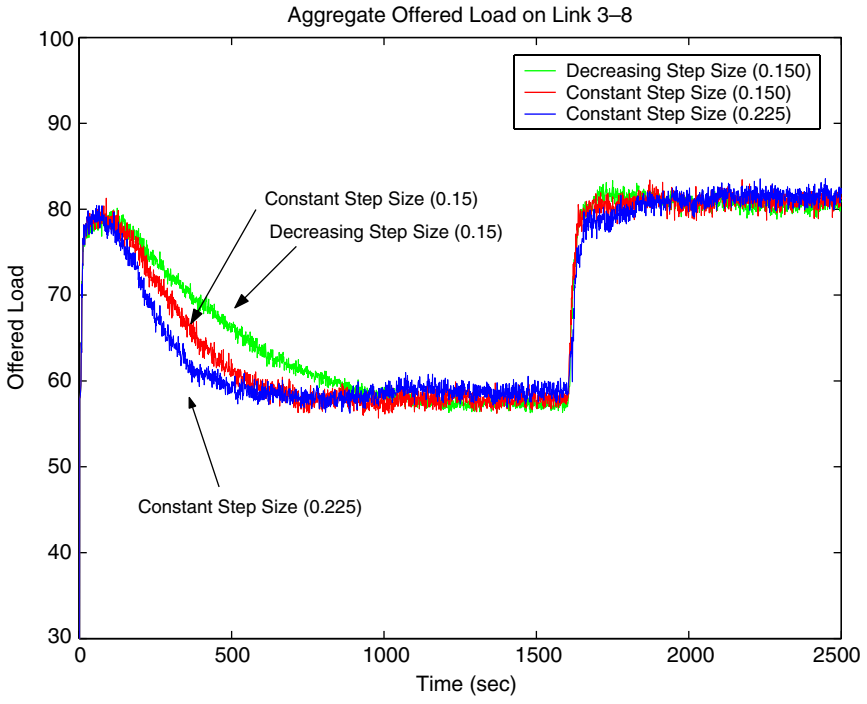

Fig. 9. Offered load on link 3–8.

utilized by SD pairs. The plot of traffic on link (12–16) shows how the traffic load is migrated away from the paths that traverse link (3–12). In addition, Fig. 8 shows the total number of packets dropped in the entire network. We observe from both figures that the algorithm quickly removes congestion as the packet drops are eliminated quickly and distributes the load among the multiple paths between the SD pairs in a reasonable amount of time. This result indicates that the proposed algorithm successfully converges under the scenarios where many SD pairs operate in an independent and asynchronous fashion.

Fig. 9 demonstrates the effect of using a constant step size. We plot the variation of traffic load on link (3–8) using two different values of fixed step size $a_s(k) = a = 0.15$ and $a_s(k) = a = 0.225$, and compare their performance with the decreasing step size case where the step size for each SD pair is given by $a_s(k) = 15/(k + 100)^{0.602}$. Even though we do not have almost-sure convergence with a constant step size, our results suggest that the convergence in distribution is sufficient for practical purposes. Moreover, the initial convergence rate improves slightly as the algorithm is able to reach a small neighborhood of the optimal operating point faster compared to the decreasing step size case. Since this neighborhood appears to be small and a constant step size policy gives us the robustness to track dynamical changes in the network, this result supports the use of a constant step size algorithm in a practical implementation.

6. Conclusion

In this paper, we have considered optimal multipath routing in environments where the link cost derivatives can be estimated (but for which an analytic expression may not exist). We have mathematically proven the optimality and stability of our proposed scheme, which is based on simultaneous perturbations. We have demonstrated that an SPSA algorithm provides significant improvements over an algorithm based on traditional finite-difference methods. Specifically, we have shown that our scheme results in much shorter measurement periods during the gradient estimation phase, and as a result, converges faster. Our simulation results show that our scheme can quickly alleviate network congestion and distribute load efficiently under dynamic network conditions.

Our work provides a basis for an architecture for effective traffic engineering in IP Networks. A promising extension to this work would be the adaption of our scheme to a Differentiated Services (DiffServ) capable network.

Appendix A. Supplementary data

Proofs of Propositions 1 and 2 [27,28] can be found, in the online version, at doi:10.1016/j.comnet.2005.09.030.

References

[1] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, X. Xiao, Overview and principles of internet traffic engineering, RFC 3272, 2002.
[2] B. Fortz, M. Thorup, Internet traffic engineering by optimizing OSPF weights, in: Proceedings of the Conference on Computer Communications (IEEE Infocom), Tel-Aviv, Israel, 2000.
[3] M.A. Rodrigues, K.G. Ramakrishnan, Optimal routing in shortest-path networks, in: International Telecommunications Symposium (IEEE ITS), Rio de Janerio, Brazil, 1994.
[4] A. Elwalid, C. Jin, S. Low, I. Widjaja, MATE: MPLS adaptive traffic engineering, in: Proceedings of the Conference on Computer Communications (IEEE Infocom), Anchorage, Alaska, 2001.
[5] C. Kommareddy, T. Güven, B. Bhattacharjee, R.J. La, M.A. Shayman, Overlay routing for path multiplicity, Tech. Rep. UMIACS-TR# 2003-70, Available from: <http://www.cs.umd.edu/Library/TRs/CS-TR-4500/CS-TR-4501.pdf>.
[6] A. Elwalid, C. Jin, S. Low, I. Widjaja, MATE: Multipath adaptive traffic engineering, Comput. Networks—Int. J. Comput. Telecommun. Network. 40 (2002) 695–709.
[7] J.C. Spall, Stochastic optimization, stochastic approximation and simulated annealing, in: J.G. Webster (Ed.), Encyclopedia of Electrical and Electronics Engineering, vol. 20, Wiley, New York, 1999, pp. 529–542.
[8] J.C. Spall, Stochastic optimization and the simultaneous perturbation method, in: Proceedings of the Winter Simulation Conference, Phoenix, AZ, 1999.
[9] J.C. Spall, Multivariate stochastic approximation using a simultaneous perturbation gradient approximation, IEEE Trans. Automat. Contr. 37 (1992) 332–347.
[10] M. Fu, S.D. Hill, Optimization of discrete event systems via simultaneous perturbation stochastic approximation, Trans. Inst. Indust. Eng. 29 (1997) 223–243.
[11] H.J. Kushner, G.G. Yin, Stochastic Approximation Algorithms and Applications, Springer-Verlag, 1997.
[12] J.N. Tsitsiklis, D.P. Bertsekas, Distributed asynchronous optimal routing in data networks, IEEE Trans. Automat. Contr. 31 (1986) 325–332.
[13] D. Bertsekas, R. Gallager, Data Networks, second ed., Prentice-Hall Inc., 1992.
[14] J. Kiefer, J. Wolfowitz, Stochastic estimation of a regression function, Ann. Math. Stat. 23 (1952) 462–466.
[15] J.R. Blum, Multidimensional stochastic approximation methods, Ann. Math. Stat. 25 (1954) 737–744.
[16] P. Sadegh, Constraint optimization via stochastic approximation with a simultaneous perturbation gradient approximation, Automatica 33 (1997) 889–892.

[17] A. Collins, The detour framework for packet rerouting, Ph.D. thesis, University of Washington, 1998.
[18] D. Andersen, H. Balakrishnan, F. Kaashoek, R. Morris, Resilient overlay networks, in: Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP), Banff, Canada, 2001.
[19] D.P. Heyman, M.J. Sobel, Stochastic Models in Operations Research, McGraw-Hill, 1982.
[20] K. Claffy, G. Miller, K. Thompson, The nature of the beast: recent traffic measurements from an internet backbone, in: Internet Society's Networking Conference (INET), Geneva, Switzerland, 1998.
[21] E. Gelenbe, On approximate computer system models, J. ACM 22 (1975) 261–269.
[22] J.M. Smith, F.R.B. Cruz, The buffer allocation problem for general finite buffer queuing networks, unpublished, Available from: <http://www.ecs.umass.edu/mie/faculty/smith/>.
[23] K. Sinha, S. Patek, OpIATE: optimization integrated adaptive traffic engineering, Tech. Rep., 2001, Available from: <http://www.sys.virginia.edu/techreps/2002/sie-020001.pdf>.
[24] S. Nelakuditi, Z.L. Zhang, A localized adaptive proportioning approach to QoS routing, IEEE Commun. Mag. 40 (2002) 66–71.
[25] G. Apostolopoulos, R. Guerin, S. Kamat, S. Tripathi, Quality of service based routing: a performance perspective, ACM SIGCOMM, Vancouver, 1998.
[26] Q. Ma, P. Steenkiste, On path selection for traffic with bandwidth guarantees, in: IEEE International Conference on Network Protocols, 1997.
[27] H.J. Kushner, D.S. Clark, Stochastic Approximation Methods for Constrained and Unconstrained Systems, Springer-Verlag, 1978.
[28] G.R. Grimmett, D.R. Stirzaker, Probability and Random Processes, second ed., Oxford Science Publications, 1998.

Tuna Güven received the B.S. degree in Electrical and Electronics Engineering from Middle East Technical University, Ankara, Turkey in 2000, and M.S. in Electrical and Computer Engineering from University of Maryland at College Park, M.D. in 2002. He is currently pursuing a Ph.D. degree in Electrical and Computer Engineering at University of Maryland at College Park, MD. His research interests includes traffic engineering, unicast and multicast routing, as well as wireless networks.

Richard J. La received his B.S.E.E. from the University of Maryland, College Park in 1994 and M.S. and Ph.D. degrees in Electrical Engineering from the University of California, Berkeley in 1997 and 2000, respectively.

From 1999 to 2000 he was also with Alcatel USA. From 2000 to 2001 he was with the Mathematics of Communication Networks group at Motorola Inc. Since 2001 he has been on the faculty of the Department of Electrical and Computer Engineering at the University of Maryland. He was a recipient of an NSF CAREER Award in 2003.

Mark A. Shayman (M'81-SM'03) received the Ph.D. degree in Applied Mathematics from Harvard University, Cambridge, MA, in 1981. He served as the Faculty of Washington University, St. Louis, MO, and the University of Maryland, College Park, where he is currently Professor of Electrical and Computer Engineering. His research interests are in communication networks, including MPLS, optical and wireless networks.

He received the Donald P. Eckman Award from the American Automatic Control Council and the Presidential Young Investigator Award from the National Science Foundation. He has served as Associate Editor of the IEEE Transactions on Automatic Control.

Bobby Bhattacharjee is an assistant professor in the Computer Science Department at the University of Maryland, College Park. His research interests are in the design and implementation of scalable systems, protocol security, and peer-to-peer systems. He is a member of the ACM.