Measurement Based Optimal Multi-path Routing

Tuna Güven, Chris Kommareddy, Richard J. La, Mark A. Shayman, Bobby Bhattacharjee
University of Maryland, College Park, MD 20742, USA
Email: {tguven@eng, kcr@cs, hyongla@eng, shayman@eng, bobby@cs}.umd.edu

***Abstract*— We propose a new architecture for efficient network monitoring and measurements in a traditional IP network. This new architecture enables establishment of multiple paths (tunnels) between source-destination pairs without having to modify the underlying routing protocol(s). Based on the proposed architecture we propose a measurement-based multi-path routing algorithm derived from simultaneous perturbation stochastic approximation. The proposed algorithm does not assume that the gradient of analytical cost function is known to the algorithm, but rather relies on noisy estimates from measurements. Using the analytical model presented in the paper we prove the convergence of the algorithm to the optimal solution. Simulation results are presented to demonstrate the advantages of the proposed algorithm under a variety of network scenarios. A comparative study with an existing optimal routing algorithm, MATE, is also provided.**

***Keywords* — Mathematical programming/optimization, Simulations**

I. Introduction

Rapid growth of the Internet and the emergence of new demanding services have sparked interests in the 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 cannot efficiently utilize network resources and offer limited control capabilities 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 (See [2], [3]). However, these approaches have several shortcomings that have not been addressed effectively. First, they tend to have network-wide effect and can result in undesirable and unanticipated traffic shifts [1]. Second, these schemes cannot distribute the load among the paths of different cost. Third, they do not consider the traffic/policy constraints, such as avoiding certain links for particular source-destination pairs [4].

MultiProtocol Label Switching (MPLS) technology has offered new traffic engineering capabilities that can help overcome these limitations [5], [6]. Many schemes have been proposed based on MPLS technology [4]. However, these methods require that the existing IP infrastructure be replaced with MPLS capable devices, and therefore raises a major investment question for the Internet Service Providers (ISPs).

In a recent study presented in [7] we have proposed a new architecture that provides traffic engineering capabilities within a domain without requiring major changes in the infrastructure of IP Networks, and addresses some of the limitations of basic shortest path schemes mentioned earlier. This new architecture does not need the traditional IP routers to be replaced or modified. Rather it requires simple devices (such as PCs or network processors) to be carefully placed inside the intra-domain network, creating overlay paths between source-destination (SD) pairs. Furthermore, the architecture allows *gradual deployment* of such devices, resulting in improved network performance with the addition of each new device. This provides ISPs with an alternative solution to achieve desired level of performance at potentially much lower costs. We will give a brief description of this architecture in Section IV. However, the details of this architecture are not the subject of this paper. For more details on the architecture refer to [7]. Here, we will assume that the overlay architecture provides the following traffic engineering capabilities required for optimal routing: establishment of *multiple paths* between SD pairs and efficient distribution of local network state information to the source nodes.

The focus of this paper is the *traffic mapping* (load balancing) problem; that is the assignment of traffic load onto pre-established paths to meet certain requirements [1]. In this paper, we propose an asynchronous distributed optimal routing algorithm based on stochastic approximation theory, using local network state information. The model is similar to that in [4], with the following differences. In [4], although the authors have mentioned that the *cost derivatives* cannot be computed and should be estimated by measurements, the mathematical analysis given in the paper does not consider this fact and implicitly assumes that the analytical gradient function is available to the algorithm. In addition, the details of the process of estimating the cost gradient are not given, and the method described in [8] appears to be a variant of well-known *finite differences* method ([9], [10]). However, this issue is not clearly or explicitly stated in the aforementioned references. This point is crucial 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 [10], [11], [12]).

In this study we consider the same problem while relaxing the assumption that the analytical gradient function is available. The proposed *measurement based algorithm* is derived from the idea of simultaneous perturbation stochastic approximation (SPSA). This allows us to greatly reduce the number of measurements required for estimating the gradient, while at the same time we have approximately the same level of

accuracy as the classical finite differences method at each iteration. By reducing the *number of measurements*, we obtain a better overall convergence rate due to the fact that each measurement requires a non-negligible amount of time in a networking environment. We will discuss these issues in more detail in the following sections. As presented in Section V, a simulation based study also demonstrates that the proposed algorithm outperforms the algorithm proposed in [4].

From a broader point of view, a special case of the proposed algorithm provides an optimal solution to more general problems that have a *simplex constraint set*. (Specifically, we are referring to the single SD pair scenarios as the special case.) 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.

The rest of the paper is organized as follows. In Section II we define the optimization problem, and give a brief overview on stochastic approximation for readers who are not familiar with the topic. Section III presents the optimal routing algorithm, and proves its stability and optimality. Section IV discusses the implementation issues. Section V describes the experimental setup used to study the performance of the proposed algorithm, and presents the simulation results. We conclude the paper and discuss possible topics of future work in Section VI.

II. The Optimization Problem

A. 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 $S = \{1, 2, \cdots, \mathrm{S}\}$ 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. With a little abuse of notation we let $P_s = \{1, 2, \cdots, N_s\}$, and define the set of all paths $P = \cup_{s \in S} P_s = \{1, 2, \cdots, N\}$, where $N = \sum_{s \in S} N_s$. While by definition, none of the paths can be used by more than one SD pair, the paths of two distinct SD pairs can share a link.

The total input traffic rate of an SD pair s is r_s and it 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 $x_s = (x_{sp},\ p \in P_s)$ be the rate vector of SD pair s, and let $x = (x_{sp},\ p \in P_s,\ s \in S)$ be the vector of all rates. Then, the flow on a link $l \in L$ has a rate that is the sum of source rates on all paths that traverse link l:

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

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

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

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

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

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

We 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:

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

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 iteration k ($[\nabla C(k)]_{sp} = \partial C/\partial x_{sp}$), $a(k) > 0$ is the step size, and $\Pi_\Theta[\vartheta]$ is the projection of a vector ϑ onto the feasible set 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 in an asynchronous fashion [13], [14]:

$$x_s(k+1) = \Pi_{\Theta_s}\big[x_s(k) - a_s(k)\nabla C_s(k)\big] \tag{7}$$

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

One problem with directly implementing (7) is that $\partial C/\partial x_{sp}$, the first derivative length of a path, may not be available in practice and can only be estimated empirically through noisy measurements of the cost function. This is mainly due to the fact that the link capacities typically fluctuate randomly [4] and the traffic patterns in the Internet are dynamic in nature. Therefore, it is necessary to use a gradient approximation method in the optimization problem. Clearly, stochastic approximation methods are reasonable solutions to such problems.

B. Stochastic Approximation

Stochastic Approximation (SA) is a recursive procedure for finding the root(s) of equations in the presence of noisy measurements, and is particularly useful for finding extrema of functions [11] (*e.g.,* [15] and [16]).

The general constrained SA has the same form as (6) with the gradient vector $\nabla C(k)$ replaced by its approximation $\hat{g}(k)$. The approximation is typically obtained through measurements of $C(x)$ around $x(k)$. Under appropriate conditions, one can show that $x(k)$ converges to the solution of (3) denoted by 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 $x(k)$ is perturbed one at a time and corresponding measurements $y(.)$ are obtained. Typically, the i-th component of $\hat{g}(k)$ ($i = 1, 2, ..., m$) for FD approximation is given by

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

where $c(k)$ is some positive number, e_i denotes a unit vector with one in the i-th position and zeros elsewhere, and $y(\cdot)$ denotes the measured cost function with measurement noise.

An alternative method to estimate the gradient is called the *Simultaneous Perturbation* (SP). In this method, all elements of $x(k)$ are randomly perturbed together to obtain two measurements $y(.)$. The i-th component of $\hat{g}(k)$ is computed by

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

where $\Delta(k) = (\Delta_1(k), \Delta_2(k), ..., \Delta_m(k))$, the vector of the random perturbations for SP, needs to satisfy certain conditions as will be discussed in the following section.

Both of the above approximations have a "two-sided" form in the sense that they use the measurements $y(x(k) \pm perturbation)$. On the other hand, one-sided gradient approximations involve measurements of $y\left(x(k)\right)$ and $y\left(x(k) + perturbation\right)$. 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 [9]. In this paper we assume that the one-sided form is utilized for the approximation process for both methods unless stated otherwise.

SA algorithms using one of the gradient approximations above are referred to as FDSA or SPSA. One should note that, in an SPSA algorithm the gradient approximation uses only two cost-function measurements, independent of the number of parameters being optimized. Standard (two-sided) finite-difference approximation requires $2m$ measurements to estimate the gradient. In [11] 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 [9]. This is certainly an important property especially if the measurements are costly and/or time consuming. Clearly, this is the case for the optimal routing problem at hand as measurements require resources and must be collected and reported in a timely manner. In other words, SPSA suggests a potential for better statistical accuracy under the same period of "time" due to a much shorter required measurement period, even though the two methods have the same statistical accuracy with the same number of "iterations". This result can be promising in the sense that the algorithm based on SPSA will be able to track and respond to changes in the network much faster than another algorithm based on FDSA and improve the overall network performance.

In [11], Spall gives a formal proof of convergence of SPSA algorithm for the "unconstrained" case. Convergence of SPSA algorithm under inequality constraints are presented in [17] as well as [12]. However, these results do not consider the case where $x(k) \pm c(k)\Delta(k) \notin \Theta$, which may be the case in the optimal routing problem. Particularly, in [17] Sadegh suggests to project $x(k)$ to a point $x^{'}(k) \in \Theta$ such that $x^{'}(k) \pm c(k)\Delta(k) \in \Theta$. If $x^{'}(k) - x(k) \rightarrow 0$ as $k \rightarrow \infty$, convergence can still be established. However, when Θ is a simplex, if $c(k)\sum_j \Delta_j(k) \neq 0$ then $x^{'}(k) \pm c(k)\Delta(k) \notin \Theta$ for all $x^{'}(k)$. Under these conditions, there is no existing proof on the convergence of an SPSA algorithm that we can directly apply to our problem. (In [12], although authors claim that they have proved the convergence for the case of a network of queues with similar constraints, they do not consider the issue mentioned above 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.

III. Optimal Routing using SPSA

A. The Optimal Routing Algorithm

In this section we propose an optimal routing algorithm and prove its stability and optimality. We know from [13] 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

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

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

$$\begin{aligned}\hat{g}_{s,i}(k) &:= \frac{N_s}{N_s - 1}\frac{y_s(x(k) + c(k)\Delta(k)) - y_s(x(k))}{c_s(k)\Delta_{s,i}(k)} \\ &= \frac{N_s}{N_s - 1}\frac{(C^+(k) + \mu_s^+(k)) - (C^-(k) - \mu_s^-(k))}{c_s(k)\Delta_{s,i}(k)}\ ,\end{aligned} \tag{9}$$

where $C^-(k) = C(x(k))$, $C^+(k)$ is the cost with $x(k)$ plus perturbation terms and $\mu_s^+(k)$ and $\mu_s^-(k)$ are measurement noise terms. Note that the noise terms observed by each SD pair is allowed to be different. In addition, while $c_s(k)$ is a positive scalar as in standard SA, we redefine $c(k)$ as a $N \times N$ diagonal matrix whose j-th diagonal entry is equal to c_{s_j} (s_j being the SD pair associated with the j-th component of $\Delta(k)$). This definition allows the possibility to have different $c_s(k)$ values for different SD pairs. In addition, 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 $x_s(k) + c_s(k)\Delta_s(k)$ to Θ_s for all $s \in S$ using L_2 projection while calculating $\hat{g}_s(k)$. This is explained in the Appendix in details. Finally, if $\Pi_{\Theta_s}[x_s(k) + c_s(k)\Delta_s(k)] = x_s(k)$, the SD pair draws a new $\Delta_s(k)$ until $x_s(k) \neq \Pi_{\Theta_s}[x_s(k) + c_s(k)\Delta_s(k)]$.

[1]Here asynchronism refers to the fact that the updates by different SD pairs do not need to take place at the same time.

Note that SD pairs may have different step sizes $a_s(k)$ for a given iteration. This brings about a level of asynchronism between SD pairs in the sense that SD pairs can independently respond to the dynamics of the network. [2] However, we assume that SD pairs update their rates once every iteration after they start running the algorithm. This assumption makes sense since at each iteration SD pairs should make use of the monitoring information that is already available. This is, however, not to say that the updates take place simultaneously. The error due to this asychronism is assumed to be absorbed into the error terms $\mu_s^+(k)$ in (9).

For the optimality of the new algorithm, we need to show (8) converges to the same point x_s^* as (7) for all SD pairs. For this, we use the following result of [18] for the standard SA algorithm:

Proposition 3.1: Suppose $\sum_{k=1}^{\infty} a(k) = \infty$. If

- $C(x(k))$ is differentiable for each $x(k) \in \Theta$, and either convex or unimodal,
- $b(k) \to 0$ w. p. 1, and
- $\sum_{k=1}^{\infty} E[\xi(k)^T \xi(k)] a^2(k) < \infty$ w. p. 1;

then $x(k) \to x^*$ with probability 1, where $b(k)$ and $\xi(k)$ are defined as

$$b(k)=E[\hat{g}(k)|x(k)] - \nabla C(x(k)) \tag{10}$$

$$\xi(k)=\hat{g}(k) - E[\hat{g}(k)|x(k)] \ . \tag{11}$$

For the convergence of the algorithm we assume that the following conditions are true:

A1. $C\left(x\left(k\right)\right)$ is differentiable for each $x(k) \in \Theta$, and either convex or unimodal.

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 $(x(l), l = 0, 1, \cdots, k)$. $E[(\Delta_{s,i}(k))^{-2}]$ are bounded for all k.

A3. $E[\mu_s^{(\pm)^2}(k)]$ are bounded and $E[\mu_s^+(k) - \mu_s^-(k)|\Delta(k), \mathcal{F}_k] = 0$ a. s. for all k, where $\mathcal{F}_k \equiv \{x(0), x(1), \cdots, x(k)\}$ or the σ-field generated by $\{x(0), \cdots, x(k)\}$.

A4. $\sum_{k=1}^{\infty} \frac{a_s^2(k)}{c_s^2(k)} < \infty$ and $\left(\frac{c_s(k)}{c_{s'}(k)}\right)^2 = O(1)$ for all $s, s' \in S$.

A5. There exists a positive constant M such that

$$\frac{1}{M} \leq \frac{a_s(k)}{a_{s'}(k)} \leq M$$

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 $s \in S$

$$\sum_{k=1}^{\infty} (\hat{a}(k) - a_s(k)) < \infty \ ,$$

and

$$\lim_{k \to \infty} \frac{a_s(k)}{\hat{a}(k)} = 1 \quad \text{for all } s \in S \ .$$

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

Proposition 3.2: Under Assumptions A1 - A6, the sequence $x(k) = (x_s(k), s \in S)$ generated by the algorithm defined by (8) converges to x^* with probability 1, regardless of the initial vector $(x_s(0), s \in S)$.

Proof: The proof of the Proposition 3.2 is given in Appendix. ■

Note that in our model each SD pair runs the algorithm independently in a distributed fashion.

B. Measurement process

In this section, we provide the details of the measurement process and its effect on the overall performance of the proposed algorithm. We will also point out benefits of SPSA based algorithms over the FDSA alternatives.

As we mentioned earlier, the Simultaneous Perturbation idea allows us to estimate a $m \times 1$ gradient vector by only two measurements while the Finite Differences method requires $m+1$ for one-sided and $2m$ for two-sided measurements. When we consider the routing problem, this result suggests that an SD pair can simultaneously perturb all of its paths if an SPSA based algorithm is employed. However, by definition, an FDSA based optimal algorithm requires an SD pair to perturb these paths one at a time.

For the same reason, FDSA based algorithms necessitate that each SD pair should start doing measurements (*i.e.*, perturb its paths) at different times. As mentioned in [4], this requires a special coordination protocol and limits the independence of actions made by SD pairs. Besides, it creates an additional traffic load (*i.e.*, overhead) to the network. On the other hand, once again the theory of SPSA enables simultaneous operation of SD pairs due to the following reason. Since the perturbations $(c_s(k)\Delta_s(k))$ made by SD pairs are all zero mean, the effect of SD pairs to each other can effectively be modeled as a zero mean noise. In other words, when different SD pairs that are sharing common links do measurements simultaneously, they will create an additional noise term to each other. However, from Proposition 3.1 and Proposition 3.2, we know that the convergence of the overall algorithm is valid under these conditions. Due to this reason, we have the important flexibility to allow SD pairs to operate in a totally independent fashion so that each SD pair can freely perturb its paths. As a consequence, a potential overhead that would be caused by the coordination protocols is eliminated. Furthermore, we can significantly reduce the time spent on the measurement process by simply overlapping these measurements. So, we can achieve a much faster convergence with respect to an FDSA alternative, since we effectively reduce the time between iterations by overlapping measurements while the accuracy of each iteration remains approximately the same as discussed earlier.

Here we would like to note that even though the simultaneous operation of SD pairs is beneficial to the convergence process, on a given sample path that the algorithm follows it may increase the magnitude of the overall error term observed during the measurements. In that case, it may actually slow down the convergence temporarily especially when the sign of one or more component of the gradient is inverted due to high amount

of noise. However, since the additional noise term due to this simultaneous operation is zero mean, on the average there is no effect on the convergence process.

Moreover, one can still improve the performance observed on a given sample path by making simple modifications to the base algorithm as we explain below. Let us first give an example to illustrate how the sign of the gradient can be inverted by simultaneous operation of SD pairs. Suppose an SD pair s has a path passing through a bottleneck link [3], which is also shared by some two other SD pairs. Suppose also that s increases the amount of load it is sending on this path as a result of a random perturbation made by the gradient estimation process. At the same time, it is possible that the other two SD pairs decrease their corresponding path rates and ultimately the overall effect may be a decrease in the cost of the bottleneck link. Under these conditions, SD pair s will possibly observe a decrease in the overall cost although it increases its rate over the bottleneck link. This may result in an erroneous decision in the next iteration and slows the convergence process as a result. However, with simple modifications using problem specific information that is already available at the source nodes, the adverse effects of this noise term can be eliminated. Specifically, by taking the current state of the paths into consideration, a source node can double check the decisions made at the current iteration using the information it already has and avoid taking erroneous actions like the one given in the example above. Particularly, the existence of the following conditions are checked by the source nodes at each iteration:

- An SD pair s tries to increase the load of a path that is already realizing drops.
- An SD pair s tries to increase/decrease the utilization of a path, which is already the highest/lowest utilization path.
- An SD pair s tries to increase/decrease the load of a path, whose utilization level is closer to highest/lowest utilization path than to the lowest/highest utilization path.

Whenever such a situation is detected[4], the algorithm simply ignores the calculated iterate values and continues to use old rates (*i.e.,* $x_{s,i}(k+1) = x_{s,i}(k)$). As a result, we limit the possible adverse effects of the simultaneous perturbation where the sign of an entry in the gradient vector is estimated wrongly. On the other hand, when the sign of the entries of the estimated gradient vector does not change, the projection algorithm will still be working in the negative derivative direction. Consequently, we still get closer to the neighborhood of the optimal operating point though it may be with a slower rate under certain cases compared to the noiseless case.

Considering these facts, we can intuitively say that the performance of the algorithm improves with this modification. Although a formal treatment of the convergence rate of the proposed algorithm is required before drawing any definitive conclusions about the behavior of the algorithm, simulation results presented in Section V show that the optimal routing scheme clearly outperforms the algorithm suggested in [4].

Another issue regarding the measurement process is the effects of asynchronous operation of SD pairs. It is proved in [4] that, with increasing asynchronism, the convergence process gets slower. In other words, this result suggests that the larger the value of t_0 gets, the slower will be the convergence, where t_0 is defined to be maximum time lag between the iterate point ($x(t)$) and time when the measurements are taken ($x(t-t_0)$). On the other hand, in the SPSA case as asynchronism between the SD pairs increases, the magnitude of the error term in measurements gets smaller since the time that the measurements overlap with each other gets shorter and this may cause a marginal performance increase on the overall system with increased asynchronism. Considering these two effects of asynchronism, we can say that there exists a trade-off between the benefits gained by overlapping the measurements and benefits of having relatively less noisy measurements. As we will see in Section V, up to a certain level of asynchronism both effects mainly cancel each other and the performance of the algorithm does not change. When the asynchronism increases further, it turns out that t_0 is dominant over the benefits of less noisy measurements and the convergence starts to get slower.

When we look to the FDSA case, it is hard to discuss asynchronism since we need a certain level of coordination between SD pairs so that each SD pair does measurements (*i.e.,* perturb its paths) at different times. However, the time lag between the iterate point ($x(t)$) and time when the earliest measurement is taken ($x(t-t_0)$) can be assumed as a source of asynchronism according to the definition of t_0 given in [4]. This is because a large t_0 can force source nodes to use outdated measurement information considering the dynamic nature of networks. Consequently, this means the convergence should be slower in the FDSA case when compared to SPSA not only because the time between iterate points is longer than SPSA, but also it forces the system to use more outdated information. (A critical issue is that there is no formal guarantee of convergence when the measurements made in FDSA overlap in time and therefore it is not possible to minimize the size of t_0 by partially overlapping measurements in time.)

C. Cost Function

The requirements on the cost function are stated in Proposition 3.1. Selecting the link cost function in the following form is sufficient to satisfy these conditions:

$$C_l(t) = d_l(t) + u_l(t)^2 \tag{12}$$

where, $d_l(t)$ is the number of packets dropped on link l during the (t, t+1) period and $u_l(t)$ is the link utilization level at the same time period.

The arrival process at a source node is an aggregate process of many individual flows. We assume that each individual flow generates packets according to an equilibrium renewal process,

[3]We assume that a bottleneck link has an arrival rate that tends to be greater than its departure rate.

[4]Some of the conditions given above are valid specifically for networks having links with equal capacities and paths with equal path lengths. However, similar conditions can easily be defined for more general network settings.

i.e., interarrival times of packets from a flow have a fixed distribution, 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], there exists two peaks at 500 and 1500 bytes in the packet size distribution of Internet traffic. Using this result, we can roughly model the packet size distribution as a Bernoulli random variable with values at 500 and 1500 bytes.

Under the conditions above, we can approximate the links in the network as $M/G/1/K$ queues. 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 percent), 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.

IV. Implementation Issues

In this section, we present a new overlay architecture to provide traffic engineering capabilities. Here, we will give a brief overview of the architecture. The details of the overlay architecture can be found in [7].

A. Path Establishment

Alternative paths between SD pairs are created using overlay nodes. The overlay nodes are located at all the source-destination nodes as well as at some core nodes. The idea is similar to the ones presented in [23] and [24], with the difference that the overlaying is done intra-domain as opposed to inter-domain. When a packet is sent along the shortest path, it will be forwarded in the same way as the traditional IP networks. On the other hand, if the packet is to be sent through an alternative path, it will be processed at the source overlay node and an additional IP header will be attached to the packet. This way the packet can be forwarded to a carefully placed overlay node that is lying along the particular alternative path. As soon as this overlay node gets the packet, it removes the outer IP header and forwards the packet to the final destination (or possibly to another overlay node). By this methodology, one can utilize as many alternative paths as needed. Note that using this architecture, we can still employ the simple shortest path routing inside the network. This allows us to use the existing traditional routers without any modification. The overlaying capabilities can be realized by attaching a simple device (*e.g.,* a PC or a network processor) to the existing routers. This device simply processes the packets, adds or removes IP headers before the basic forwarding operation is made at the routers.

As a final remark, we would like to emphasize the point that the proposed optimal routing algorithm does not necessitate the use of the overlaying architecture. For instance, it can also be employed in an MPLS based network, where the overlay paths are replaced with LSPs (Label Switched Paths). The use of overlaying architecture actually gives us the additional opportunity to use the proposed algorithm in the traditional IP-based networks.

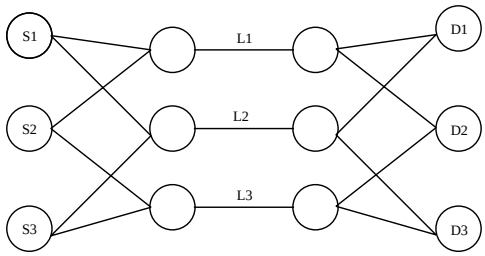

Fig. 1. Network Topology 1

B. 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 certain tie-breaking rule that gives a unique mapping [7]. 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 in a coordinated way. (Note that this way the links are not required to be probed by each SD pair.) While sending the information to a source node of a specific SD pair, the overlay nodes also aggregate the information gathered from different links as much as possible. For instance, the cost information obtained from the links that are on a particular path of an SD pair s are aggregated by the overlay nodes, using the fact that the cost structure is additive according to the definition given in (3). As a consequence, the overhead caused by the distribution of the link state information is minimized.

C. Traffic Filtering

For QoS purposes, special care should be given while splitting the traffic at the source nodes. Specifically, one should avoid the well-known reordering problems especially for the TCP traffic. The optimal routing algorithm proposed in this paper does not require and specify how a particular packet should be routed along the network. Instead, it calculates the rates at which the traffic should be distributed along the alternative paths between SD pairs. Therefore, any existing filtering scheme that minimizes the reordering problem can be used for this purpose. A possible solution is presented in [4] that depends on the use of hash-functions.

V. Experimental Setup and Simulation Results

The purpose of this section is to identify the characteristics of the proposed routing algorithm and evaluate its performance under various networking conditions. Using simulations, we would like to verify that the algorithm is stable and robust in such a way that it minimizes congestion and quickly balances the load among multiple-paths between SD pairs in a reasonable period of time.

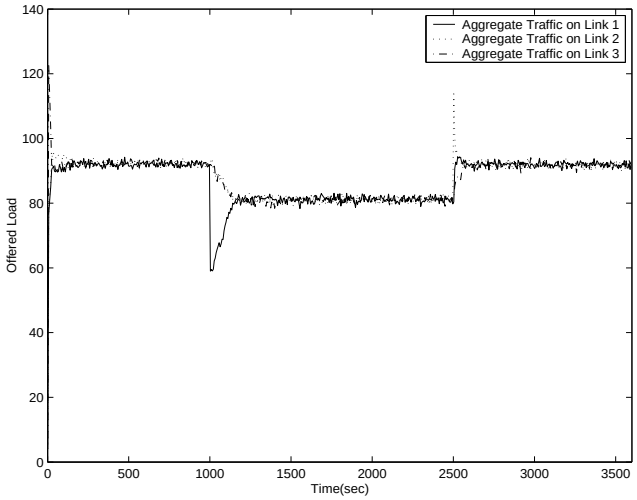

Fig. 2. Network topology 1 with an offset of 50 ms

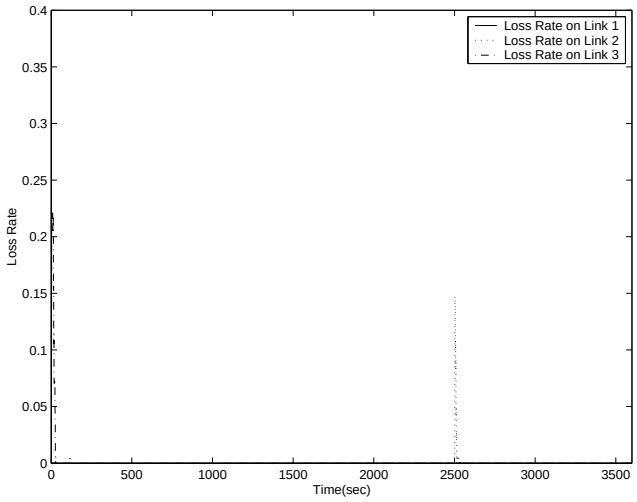

Fig. 3. Network topology 1 with an offset of 50 ms

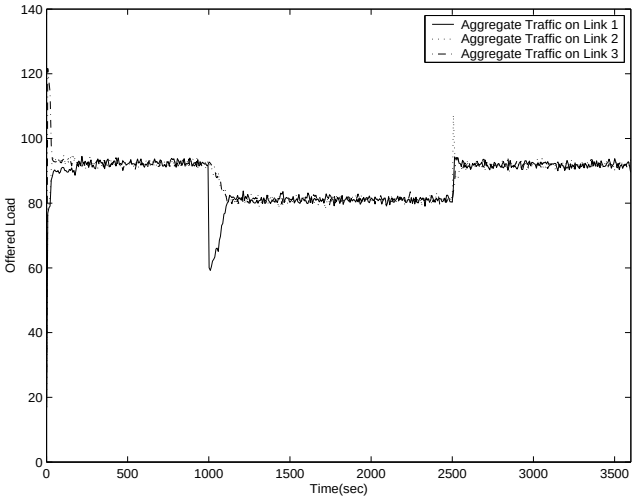

Fig. 4. Network topology 1 with an offset of 200 ms

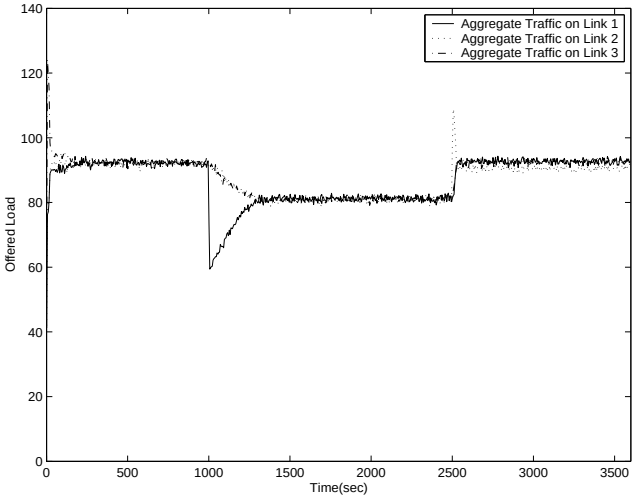

Fig. 5. Network topology 1 with an offset of 500 ms

In all simulations, the period of link state measurements is selected as one second. As a consequence, SD pairs can update their rates at best approximately every two seconds since we require two measurements for estimating the gradient vector according to the SPSA.

Experiments are simulated under two network topologies. The first topology, which is borrowed from [4] is given in Figure 1. This topology allows us to obtain insights about the fundamental behavior of the proposed algorithm due its simplicity. 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 network consists of identical links with a bandwidth of 45 Mbps. Packet size is given as 257 bytes. Each 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 a 19.8 Mbps (corresponding to 0.44 link utilization) Poisson traffic on the average. In addition, L1, L2 and L3 carry uncontrolled cross traffic. The cross traffic dynamics is given in Table I. This setup is effectively the same as the one given in [4]. (See [25] for the details of this setup.) A random delay is introduced before each SD pair starts running the optimal routing algorithm to guarantee that the SD pairs are not synchronized. (The maximum value of this random delay is defined as offset.) As shown in Figures 2 and 3, the algorithm quickly eliminates the congestion and successfully balances the traffic in a short time. Moreover, these results show that the proposed algorithm clearly outperforms the MATE algorithm. While MATE requires around 400-500 [5]

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

TABLE I

THE CROSS TRAFFIC DYNAMICS

Link	Load Distribution in time (sec)		
	$[0-1000)$	$[1000-2500)$	$[2500-3600)$
$L1$	0.77	0.44	0.44
$L2$	0.33	0.33	0.67
$L3$	0.33	0.33	0.33

seconds to converge, it takes around 200 seconds[6] in the case of the proposed algorithm. Besides, the proposed algorithm quickly (around 50 seconds) clears out the packet drops unlike MATE. (See Figures 10 and 11 presented in [4].)

Figures 4 and 5 illustrates the effect of increased asynchronism between SD pairs. We increase the asynchronism between SD pairs by simply increasing the offset value. From both graphs we can conclude that the algorithm is still able to converge in a short time. As we see from Figures 2 and 4, the performance is almost the same for offset values 50 ms and 200 ms. However, when we increase the offset to 500 ms, we see that the convergence of the proposed algorithm gets slightly slower. Thus, these results validate the earlier discussion made in Section III-B.

Figure 6 represents the second topology we consider in this paper. This topology is also used in [26], [27] and considered to be typical of a large ISP's network. (This topology closely resembles the MCI Internet topology [28].) Using this topology, we intend to analyze the performance of the proposed algorithm under more realistic networking conditions.

Nodes 1, 5, 6, 14 and 18 are both source and destination nodes. This gives us a total of 20 SD pairs. Each pair has at least two paths to reach to destination. A total of 78 paths are created between these 20 SD pairs using overlaying architecture. Overlay capability is available at all source/destination nodes as well as the nodes 2, 10 and 13. In this experiment, the offset is set to 0.1 sec. The dashed links have a capacity of 50 Mbps, while solid links have 20 Mbps. The packet size for this scenario is selected as 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 the network on link (3-12) starting at simulation time 1600 sec. The cross traffic rate is 18 Mbps and cannot be shifted to any alternative paths as before.

In Figure 7, we illustrate how the load is distributed after the algorithm starts. The links that we have plotted are selected in such a way that each of them is located on a different alternative path that can divert the traffic sent through link (3-12). The only exception is link (12-16), which demonstrates how the traffic load is migrated away from the paths that were traversing link (3-12). In addition, Figure 8 shows the total number of

[6]This performance result is verified under several sample paths created by different random seeds.

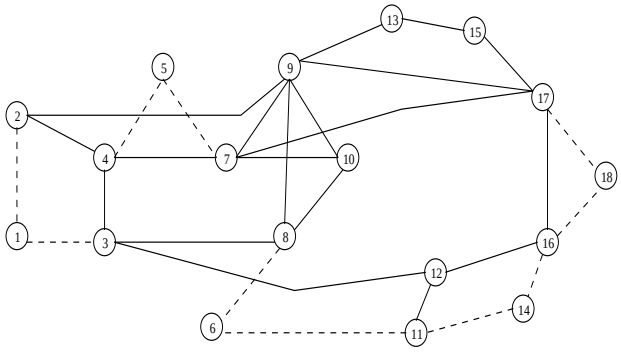

Fig. 6. Network Topology 2

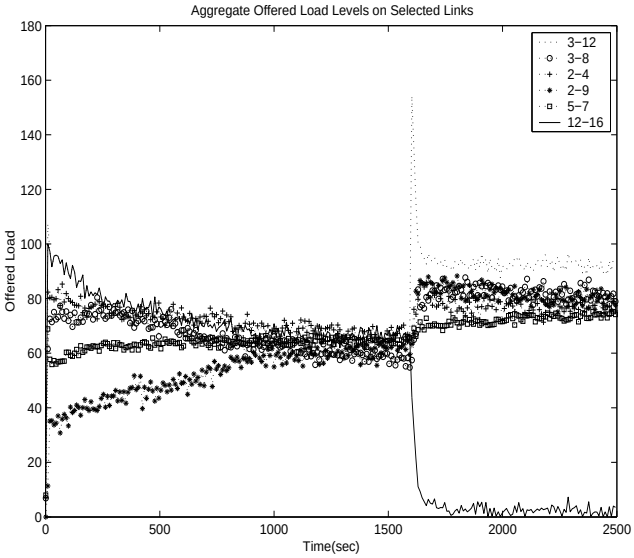

Fig. 7. Offered Load on Network Topology 2

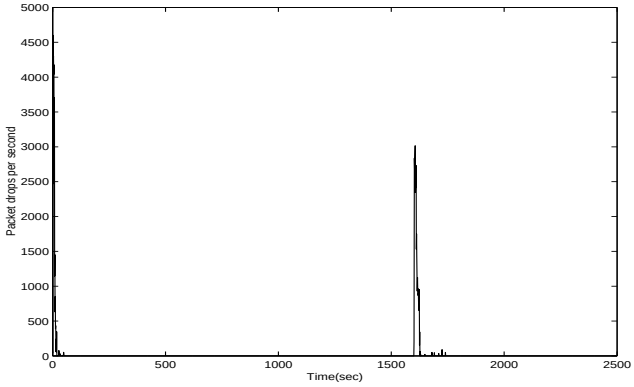

Fig. 8. Total packet drops on Network Topology 2

packets dropped in the entire network. We observe from both figures that the algorithm can rapidly eliminate congestion and distribute the load among the multiple paths between the SD pairs. This result is encouraging in the sense that the proposed algorithm converges in reasonable time scales even under the cases where many SD pairs have independent and asynchronous operation.

VI. Conclusion

In this paper, we have focused on the optimal multi-path routing problem where the link cost derivatives can only be estimated but cannot be calculated analytically. We mathematically proved the optimality and stability of the proposed algorithm. We have applied the technique of SPSA, which offers significant benefits over traditional finite-difference methods. This way we obtained much shorter measurement times while estimating the gradient and as a result achieved a faster convergence. Simulation results show that the proposed algorithm can swiftly and effectively minimize the congestion and distribute traffic load efficiently under dynamic network conditions. Finally, we have presented a new architecture to effectively apply traffic engineering in IP Networks. A possible future work is the integration of the proposed algorithm with the Differentiated Services environment where there exist several traffic classes with different Quality of Service (QoS) requirements.

Acknowledgments

We would like to thank Professor Michael C. Fu for his valuable comments.

References

[1] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Overview and principles of internet traffic engineering," RFC 3272, May 2002.

[2] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Tel-Aviv, Israel, Mar. 2000.

[3] M. A. Rodrigues and K. G. Ramakrishnan, "Optimal routing in shortest-path networks," in *ITS'94*, Rio de Janeiro, Brazil.

[4] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Anchorage, Alaska, Apr. 2001.

[5] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. MacManus, "Requirements for traffic engineering over MPLS," RFC 2702, Sept. 1999.

[6] E. Rosen, A. Vishwanathan, and R. Callon, "Multiprotocol label switching architecture," RFC 3031, Jan. 2001.

[7] C. Kommareddy, T. Güven, B. Bhattacharjee, R. J. La, and M. A. Shayman. Overlay routing for path multiplicity. Tech. Rep. UMIACS-TR# 2003-70. [Online]. Available: http://www.cs.umd.edu/Library/TRs/CS-TR-4500/CS-TR-4501.pdf

[8] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," *Computer Networks-The International Journal of Computer and Telecommunications Networking*, vol. 40, no. 6, pp. 695–709, Dec. 2002.

[9] J. C. Spall, "Stochastic optimization, stochastic approximation and simulated annealing," *Encyclopedia of Electrical and Electronics Engineering (J. G. Webster, ed.), Wiley, New York*, vol. 20, pp. 529–542, 1999.

[10] ——, "Stochastic optimization and the simultaneous perturbation method," in *Proceedings of the Winter Simulation Conference*, 1999.

[11] ——, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 332–341, 1992.

[12] M. Fu and S. D. Hill, "Optimization of discrete event systems via simultaneous perturbation stochastic approximation," *Transactions of the Institute of Industrial Engineers*, vol. 29, no. 3, pp. 223–243, 1997.

[13] J. N. Tsitsiklis and D. P. Bertsekas, "Distributed asynchronous optimal routing in data networks," *IEEE Trans. Automat. Contr.*, vol. AC-31, no. 4, pp. 325–332, Apr. 1986.

[14] D. Bertsekas and R. Gallager, *Data Networks*. Prentice-Hall Inc., 2nd edition, 1992.

[15] J. Kiefer and J. Wolfowitz, "Stochastic estimation of a regression function," *Ann. Math. Stat.*, vol. 23, pp. 462–466, 1952.

[16] J. R. Blum, "Multidimensional stochastic approximation methods," *Ann. Math. Stat.*, vol. 25, pp. 737–744, 1954.

[17] P. Sadegh, "Constraint optimization via stochastic approximation with a simultaneous perturbation gradient approximation," *Automatica*, vol. 33, no. 5, pp. 889–892, 1997.

[18] P. L'Ecuyer and P. W. Glynn, "Stochastic optimization by simulation: convergence proofs of the GI/G/1 queue in steady-state," *Management Science*, vol. 40, no. 11, pp. 1562–1578, 1994.

[19] D. P. Heyman and M. J. Sobel, *Stochastic Models in Operations Research*. McGraw-Hill, 1982.

[20] K. Claffy, G. Miller, and K. Thompson, "The nature of the beast: Recent traffic measurements from an internet backbone," in *INET 1998*.

[21] E. Gelenbe, "On approximate computer system models," *JSAM*, vol. 22, no. 2, pp. 261–269, 1975.

[22] J. Smith and F. Cruz. The buffer allocation problem for general finite buffer queueing networks. Unpublished. [Online]. Available: http://www.ecs.umass.edu/mie/faculty/smith/

[23] A. Collins, "The detour framework for packet rerouting," Ph.D. dissertation, Univ. of Washington, 1998. [Online]. Available: http://www.cs.washington.edu/homes/acollins/quals/quals.ps

[24] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. 18th ACM Symp. on Operating Systems Principles (SOSP)*, Banff, Canada, 2001.

[25] K. Sinha and S. Patek. Opiate: Optimization integrated adaptive traffic engineering. Tech. Rep. [Online]. Available: http://www.sys.virginia.edu/techreps/2002/sie-020001.pdf

[26] S. Nelakuditi and Z. L. Zhang, "A localized adaptive proportioning approach to QoS routing," *IEEE Commun. Mag.*, vol. 40, no. 6, pp. 66–71, 2002.

[27] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Quality of service based routing: A performance perspective," in *ACM SIGCOMM*, 1998.

[28] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *IEEE International Conference on Network Protocols*, 1997.

[29] T. Güven, C. Kommareddy, R. J. La, M. A. Shayman, and B. Bhattacharjee. Measurement based optimal multi-routing. Tech. Rep. UMIACS-TR# 2003-69. [Online]. Available: http://www.cs.umd.edu/Library/TRs/CS-TR-4500/CS-TR-4500.ps

[30] H. Kushner and D. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.

Appendix

In this section we provide a sketch of the proof of Proposition 3.2. The complete proof of this can be found in [29].

Proposition 3.1 (i.e. Proposition 1 of [18]) relies on Theorem 5.3.1 of [30]. We will prove Proposition 3.2 by adapting the same theorem. First, note that given any $N_s \times 1$ vector ϑ, the solution of the minimization problem

$$\min_{\phi} \ ||\vartheta - \phi||^2 \qquad \text{s. t. } \phi^T u = r_s \tag{13}$$

is given by

$$\phi_i = \vartheta_i + \frac{r_s - \sum_{j=1}^{N_s} \vartheta_j}{N_s} \ , \tag{14}$$

where $u = [1, 1, \cdots, 1]^T$. Obviously, if $\phi_i \geq 0$ for all i, this solution is equivalent to the L_2 projection. Here for the purpose of temporary perturbation we replace (5) with a non-negativity constraint. Thus, the projection of $x_s(k) + c_s(k)\Delta_s(k)$ can be calculated using (14) if

$$x_{s,i}(k) + c_s(k)\left(\Delta_{s,i}(k) - \frac{\sum_{j=1}^{N_s}(\Delta_{s,j}(k))}{N_s}\right) \geq 0 \ . \quad (15)$$

Recall that $\Delta_{s,i}(k)$ is bounded by α from Assumption A2. Hence, (15) holds if

$$c_s(k) \leq \frac{\min_j\{x_{s,j}(k)\}}{2\alpha} \ . \quad (16)$$

From (5) we know $\frac{\min_j\{x_{s,j}(k)\}}{2\alpha} \geq \frac{\epsilon}{2\alpha}$. Since, $c_s(k) \to 0$, there exists finite K_1 such that $c_s(k) \leq \frac{\epsilon}{2\alpha}$ for all $k > K_1$. Therefore, (14) can be used to compute the projection of $x_s(k) + c_s(k)\Delta_s(k)$ for sufficiently large $k > K_1$.

Let us first define the notation to be used in the proof. Let $\bar{\Delta}_s(k)$ be an $N \times 1$ vector, where values of entries corresponding to those of SD pair s are $\Delta_{s,i}(k)$ and zero otherwise. Hence, $\sum_{s\in S}\bar{\Delta}_s(k) = (\Delta_{s,i}, s \in S, i \in P_s)$. Similarly, u_s is an $N \times 1$ vector, where the values of entries corresponding to those of SD pair s are one and zero otherwise. Following the proof in [12] and using Taylor's theorem, for $k > K_1$ and $s \in S$ we have

$$\begin{aligned}
&E\left[\,\hat{g}_{s,i}(k)|x(k)\,\right] \\
=&\frac{N_s}{N_s-1}E\left[\frac{C^+(k) - C^-(k) + \mu_s^+(k) - \mu_s^-(k)}{c_s(k)\Delta_{s,i}(k)}\middle| x(k)\right] \\
=&\frac{N_s}{N_s-1}E\left[\frac{E\left[C^+(k) - C^-(k)|\Delta(k)\right]}{c_s(k)\Delta_{s,i}(k)}\middle| x(k)\right] \\
=&\frac{N_s}{N_s-1}\left(E\left[\frac{\nabla C^T(x(k))\sum_{s'\in S}c_{s'}(k)\bar{\Delta}_{s'}(k)}{c_s(k)\Delta_{s,i}(k)}\middle| x(k)\right]\right. \\
&-E\left[\frac{\nabla C^T(x(k))\sum_{s'\in S}\frac{c_{s'}(k)\sum_{j=1}^{N_{s'}}\Delta_{s,j}(k)}{N_{s'}}u_{s'}}{c_s(k)\Delta_{s,i}(k)}\middle| x(k)\right] \\
&\left.+E\left[\sum_{s'\in S}\frac{O\left(c_{s'}^2(k)\Delta_{s'}^2(k)\right)}{c_s(k)\Delta_{s,i}(k)}\right]\right) \\
=&\frac{N_s}{N_s-1}\left(\frac{N_s-1}{N_s}\nabla C_{s,i}(x(k)) + O(c_s(k))\right) \\
=&\nabla C_{s,i}(x(k)) + O(c_s(k))
\end{aligned}$$

where $C^-(k) = C\left(x(k)\right)$,

$$C^+(k) = C\left(x(k) + \sum_{s'\in S}c_{s'}(k)\tilde{\Delta}_{s'}(k)\right) \ ,$$

and

$$\tilde{\Delta}_{s'}(k) = \bar{\Delta}_{s'}(k) - \frac{\sum_{j=1}^{N_{s'}}\Delta_{s',j}(k)}{N_{s'}}u_{s'} \ .$$

Therefore, one can see that $b(k) \to 0$ with probability one. From the assumption that $E[\mu_s^+(k) - \mu_s^-(k)|\mathcal{F}_k] = 0$ and using the independence of $\mu_s^{\pm}(k)$ and $\Delta_s(k)$, we can bound the second moment of $\hat{g}_s(k)$ as follows:

$$\begin{aligned}
&E[(\hat{g}_{s,i}(k))^2] \quad (17)\\
=&E\left[\left(\frac{C^+(k) - C^-(k) + \mu_s^+(k) - \mu_s^-(k)}{c_s(k)\Delta_{s,i}(k)}\right)^2\right] \\
=&E\left[\left(\frac{C^+(k) - C^-(k)}{c_s(k)\Delta_{s,i}(k)}\right)^2 + \left(\frac{\mu_s^+(k) - \mu_s^-(k)}{c_s(k)\Delta_{s,i}(k)}\right)^2\right]
\end{aligned}$$

Following a similar argument used above one can show that the first term in (17) is $O(1)$ and the second term is $O(c_s(k)^{-2})$, using the bounds on $E[(\Delta_s(k))^2]$, $E[(\Delta_s(k))^{-2}]$, and $E[(\mu_s^{\pm}(k))^2]$.

Now as shown in [29], the convergence of the algorithm can be proved by adapting the proof of Theorem 5.3.1 of [30] under the assumptions A1-A6.