Optimized RIO for DiffServ Networks

Rahul Vaidya and Shalabh Bhatnagar
[1] Department of Computer Science and Automation
Indian Institute of Science
Bangalore-560012, India
{rahulv,shalabh}@csa.iisc.ernet.in

Abstract

Even though Differentiated Services with Assured Forwarding provide bandwidth and other guarantees, the equilibrium queue size of the router depends on network conditions as well as network settings. The queuing delay guarantees are not possible with the present framework. Also the absence of priority flows in networks affects the utilization of the router. We propose the Optimized RIO algorithm as an alternative Adaptive Queuing Mechanism over the DiffServ router, which dynamically changes the parameter settings for efficient usage. The optimization algorithm used is presented. A proof of convergence of the algorithm is also given. Simulation results are presented for validation of our approach. Optimized RIO is found to show better performance than traditional RIO.

1 Introduction

Random Early Detection (RED) [1] is widely used Adaptive Queue Mechanism (AQM) used in the internet. It controls congestion by detecting the incipient persistent congestion while allowing the transient congestion in the network. It uses a randomized algorithm to notify the connection of incipient congestion. RED maintains an exponentially weighted average queue length (avg_q). The decision to drop or enqueue the arriving packet depends on the this parameter. The arriving packet is always enqueued if avg_q is below the minimum threshold (min_{th}) and is always dropped if it is above the maximum threshold (max_{th}). Between max_{th} and min_{th}, the packet is dropped with a probability that increases linearly between the minimum and maximum thresholds. By randomly dropping packets, the router utilization increases since global synchronization is avoided. Also, since weighted mean average avg_q and not the actual queue length is taken as a measure of congestion, the bias against bursty connection is reduced.

Previous studies have shown that the RED algorithm is highly sensitive to parameter settings. Also, the optimal parameter settings depend on the network conditions. Hence it is not possible to statically

[1]This work was supported in part by Grant Number SR/S3/EE/43/2002-SERC-Engg. from Department of Science and Technology, Government of India.

set the parameters such that the routers perform optimally under all network conditions [2, 3, 4]. Since the interaction between TCP and RED is not precisely known, obtaining analytical solution is not feasible. Further, since a variation of RED is used as AQM in DiffServ, with the latter having a large number of parameters, the problem of setting these is even more acute with multiple customer classes.

The DiffServ [5] architecture has been proposed by the Internet Engineering Task Force (IETF) to provide Quality of Service (QoS) guarantees over the present IP networks. Under this framework, a special field in the IP header is used to indicate priority of the packet that in turn determines the level of service, or Per Hop Behavior (PHB), to be applied to the packet.

A standard PHB is Assured Forwarding (AF) [6]. Under AF, the packets can be classified in four different classes. Packets from different classes are processed in different physical queues and managed by a scheduler mechanism. In a single class, packets can be further classified into a maximum of three categories signifying the drop levels or precedences. Packets with the highest precedence are dropped with the lowest probability. Such differential discarding can be achieved using any AQM mechanism. However, RED is the most preferred AQM mechanism to be used in conjunction with DiffServ. The algorithm of RED with DiffServ is described in [5] and is called RED with In and Out (RIO).

The performance of RIO (like RED) is dependent on parameter settings as well as the type of load on the network [7, 8]. Specifically, it has been observed that the queuing delay observed by the packets in the network depends on the number of priority flows in the network as well as the parameter settings. Hence, under heavy load conditions, the packets experience high queuing delays while under low loads, the router is underutilized [9, 10].

Some techniques have been proposed for setting parameters adaptively to improve performance [10, 9]. These algorithms intuitively vary one of the parameters depending on the performance of the router. This approach does not scale well because of multiple parameters involved in the algorithm. In this paper, we propose black box optimization of RED routers for optimal parameter settings. The parameter update in each iteration is directly done on the router depending on the loss function measured on it. Any optimization algorithm can be employed for black box optimization. Since the parameter updates directly affect the performance of connections over the routers, a slow and graceful convergence of the parameters is preferred. In this paper we propose a robust stochastic approximation based algorithm for optimizing RIO parameters. In a related recent work [11], a similar algorithm has been used for finding optimal parameters for RED routers, though not for the case of the DiffServ architecture with multiple customer priority classes as we consider here. The number of parameters for DiffServ, as we shall see, increases dramatically with the number of priority classes which is not so with [11]. We refer the reader to [11] for a proof of convergence of the above algorithm in a general setting.

In what follows, we discuss the DiffServ Architecture in § 2. We also explain the RIO congestion avoidance algorithm used in the DiffServ Architecture. We then present the robust stochastic approximation algorithm in a general setting in § 3. Its convergence analysis is briefly sketched in § 4. The adaptation of the algorithm for tuning RIO parameters is presented in § 5. Simulations using network simulator are then shown in § 6 to validate the proposed approach. The results demonstrate that RIO with dynamically optimized parameters is capable of performing in accordance with the defined objectives. Finally, § 7 provides the concluding remarks.

Top

2 Differentiated Services Architecture

The IETF has proposed Differentiated Services Framework to provide the 'Allocated Capacity' service over IP. Using this various Quality of Service (QoS) guarantees can be provided with minimum change in the present internet. The Type of Service (ToS) field in the IP header gives indication of the service preference of the packet. The packets with higher priority are given preferential treatment in the routers. The packets are marked at the source or edge routers to conform to the Service Level Agreements (SLA). Thus the complexity of the protocol is pushed to the edge routers.

In the core routers, the Code Point in the ToS field of IP header determines the level of service, or Per Hop Behavior (PHB), which the packet receives. One of the standard PHB is the Assured Forwarding PHB (AFPHB), where only statistical guarantees about the level of the service received by the connections are provided. The AFPHB guarantee that the packets with lower priority will be dropped with higher probability than the packets with higher priority. This ensures that the packets with higher priority receive better service. The packets of different priorities are kept in different virtual queues and each queue is managed by some AQM mechanism, usually RED. We consider the variant of RED as described in [5].

RIO (RED In/Out) expects the packets are marked with two Code Points, one for in-profile packets (marked as IN packets) and out-profile packets (marked as OUT). The IN packets are those which conform to the SLA and should be given better service than OUT packets which do not conform to SLA. The IN and OUT packets are considered to be in two different virtual queues and managed by RED mechanism with different parameters. The relation between the parameters of IN and OUT packets is as shown in Figure 1.

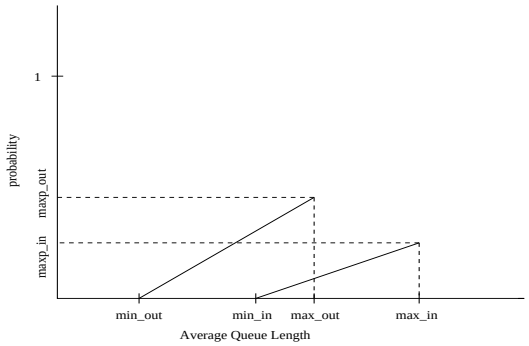


Figure 1: RIO Parameters

To provide a higher degree of service to IN packets, the thresholds of IN packets are set higher than those of OUT packets. Also the maximum drop probability (max_p) of IN packets is less than that of OUT packets. This means that for a certain queue length, the probability that an IN packet is dropped is less than the same for OUT packets. Hence RIO allows more number of IN packets than OUT packets.

Since the performance of RED depends on the parameters and RIO is a variation of RED algorithm, the performance of RIO also depends on the network conditions and parameter settings. It has been observed that the size of buffer depends on the amount of router capacity being reserved by the connections. Thus the network administrators are unable to provide bounds on delay experienced by the packets in the network. This is a serious shortcoming since many real-time applications require delay bounds.

Top

Since OUT packets are given lower priority, their parameters are 'harsh' as compared to those of IN packets. If the parameters are set statically, this causes the router to under-perform in the absence of IN flows. This reduces the throughput of the OUT packets. Since the connections come and go dynamically, and the network conditions change, the parameters cannot be set statically. Some methods have been proposed to set the parameters dynamically [10, 9]. Here the algorithm intuitively varies one of the parameters while keeping all other constant throughout, depending on the performance of the router. More than one variable, however, cannot be changed intuitively because the inter-relation among the various parameters and their combined effect on the performance is not known.

Since the relationship between the various parameters and performance is analytically not known, Black Box optimization technique is suitable for obtaining the optimal parameters. A similar approach was proposed in [12] for optimizing RED. However, the approach involved simulating the network for obtaining loss function measurements. Simulating the network would involve knowledge of the topology and behavior of all sources transmitting over the router. Moreover, simulating the network would lead to inaccuracies in the procedure. In our approach, we propose to run our algorithm directly over the DiffServ routers on-line. Here we change the code in the Network Simulator NS2 [13] for DiffServ routers to allow for the above and implement different settings.

In general any optimization technique to be used in conjunction with on-line measurements of loss function needs to possess the following properties [12]:

- High Efficiency: Since the optimization has to be carried out in real time, it is important that better parameters are obtained with limited computational resources.
- Scalability: The algorithm must be scalable since typically a large number of parameters need to be optimized.
- Robustness to noise: This is required since the estimates/observations of the various processes and/or performance metrics are in general noise corrupted.

In what follows, we present a steepest descent, gradient search based, two-timescale stochastic approximation algorithm that uses SPSA type gradient estimates with certain deterministic perturbation sequences. This algorithm is a robust variant of an algorithm in [14] and shows graceful convergence behavior while at the same time exhibits fast convergence with minimal computational overheads. The advantage with this algorithm is that it can be implemented on-line directly over the DiffServ gateway routers and hence completely eliminates the need for simulating the network. We call this algorithm Robust Two-Timescale SPSA. We also show that this algorithm possesses the properties mentioned above and hence is suitable for on-line optimization of RIO parameters. In a related recent work [11], a similar algorithm has been used for optimizing RED parameters for a single traffic class (not in the DiffServ setting considered here).

3 The Robust Two-Timescale SPSA Algorithm

Suppose for $n \geq 0$, $\zeta_n \in \mathcal{R}^p$ are independent and identically distributed (i.i.d.) random variables. Let $h : \mathcal{R}^N \times \mathcal{R}^p \rightarrow \mathcal{R}$ be an associated cost function that is assumed bounded and measurable. Let $J : \mathcal{R}^N \rightarrow \mathcal{R}$ be defined as:

$$J(\theta) = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^{n} h\left(\theta, \zeta_i\right) \tag{1}$$

Top

for any given $\theta \in \mathcal{R}^N$. We assume θ takes values in a compact set $C \subset \mathcal{R}^N$ that in particular has the form $C = \Pi_{i=1}^N [a_{i,\min}, a_{i,\max}]$, where $a_{i,\min} < a_{i,\max}$, $i = 1, \ldots, N$. The aim is to find a parameter $\theta^\star \in \mathcal{R}^N$ that minimizes $J(\theta)$ i.e.,

$$J(\theta^\star) = \min_{\theta \in C} J(\theta) \tag{2}$$

Note that here the function to be optimized in itself corresponds to the long-run average of certain 'noisy' function observations. Hence any algorithm that updates the parameter would have to compute $J(\theta)$ for any given θ before updating it. This in principle may require a substantial number of computations. In [15], two-timescale stochastic approximation algorithms were used for a similar objective. The idea in these is that data is aggregated on the faster timescale while parameter is updated on the slower one. Thus when viewed from the faster timescale, the parameter appears to be quasi-static while from the slower one, data appears to be essentially equilibrated. For superior performance, aggregation of data over an arbitrarily chosen fixed number of epochs in between successive parameter updates is advocated.

For finding the update direction in the case of an N-dimensional parameter vector, standard two-sided Kiefer-Wolfowitz gradient estimates require $2N$ loss function measurements. For loss functions such as $J(\theta)$ here, this would involve a lot of computation. The SPSA algorithm [16], on the other hand, requires only two measurements by randomly perturbing all parameter components simultaneously. An SPSA estimate of the ith partial derivative, $i = 1, \ldots, N$, has the form:

$$\nabla_i J(\theta) \approx \frac{J(\theta + \delta\Delta) - J(\theta - \delta\Delta)}{2\delta\Delta_i} \tag{3}$$

where $\delta > 0$ is a 'small' scalar. Also, Δ_i, $i = 1, \ldots, N$, are i.i.d., symmetric random variables satisfying P(Δ_i=0) = 0 and an inverse moment bound, and Δ is the vector, $\Delta = (\Delta_1, \Delta_2, \ldots, \Delta_N)^T$. In most applications, Δ_i are simply chosen to be i.i.d., Bernoulli distributed, random variables with $\Delta_i = \pm 1$ w.p. 1/2.

The two-timescale algorithms of [15] use SPSA type gradient estimates. In [14], variants of the algorithms in [15] were developed that also use SPSA type gradient estimates but with certain deterministic (in place of randomized) perturbation sequences. The use of deterministic sequences was found to considerably improve performance of these algorithms over those of [15] that use randomized sequences. The idea in deterministic perturbation sequences is that one first identifies a suitable set in which the N-dimensional Δ-sequences take values. Next, one cyclically moves the perturbation sequences through this set by picking one of the perturbation vectors at each update epoch. The above set containing say M perturbations should be such that the following holds: For every $i, j \in \{1, \ldots, N\}, i \neq j$, $\sum_{n=1}^{M} \frac{\Delta_j^n}{\Delta_i^n} = 0$. Thus the set of perturbations comprises of elements $\Delta^1, \ldots, \Delta^M$ with each $\Delta^i = (\Delta_1^i, \cdots, \Delta_N^i)^T, i = 1, \ldots, M$, and each Δ_j^i, $j = 1, \ldots, N$ being ± 1-valued. The perturbation sequence $\{\Delta(n)\}$ i.e., the sequence of perturbations corresponding to the sequence of parameter updates is then obtained as follows: Set $\Delta(0) = \Delta^1$(say), $\Delta(1) = \Delta^2, \cdots, \Delta(M-1) = \Delta^M$ and reset at the Mth update epoch, $\Delta(M) = \Delta^1$ and continue the above process. It was noted in [14] that the performance improvement above occurs because of the regular manner in which the perturbation sequence is moved. Specifically, two constructions for deterministic perturbations were given in [14]. We describe here the one using normalized Hadamard matrices since we use the same in our algorithm. In general an $m \times m$ ($m \geq 2$) matrix H is called a Hadamard matrix of order m if its entries belong to

$\{1, -1\}$ and $H^T H = mI_m$ where I_m is the $m \times m$ identity matrix. A Hadamard matrix is said to be normalized if all the elements in its first column are 1. For matrices of order $m = 2^k$, the construction in [14] is given as follows:

- For $k = 1$, let $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.
- For $k > 1$, $H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}$

The number of perturbations required in the perturbation space is $M = 2^{\lceil log_2 N \rceil}$. Thus for $N = 4$, one needs a 4×4 normalized Hadamard matrix. Note that from the above construction of normalized Hadamard matrices, $M \geq N$. The perturbation vectors $\Delta^1, \cdots, \Delta^M$ are now constructed as follows: Form a new matrix $\hat{H}$ from the normalized Hadamard matrix H by picking any N of the M columns and all M rows of H. Next set $\Delta^1, \cdots, \Delta^M$ as the M row vectors of $\hat{H}$. We now describe the following algorithm from [14] whose variant we shall later present.

Suppose $\{a(n)\}$ and $\{b(n)\}$ are two sequences of real numbers that satisfy

$$\sum_n a(n) = \sum_n b(n) = \infty,$$

$$\sum_n (a(n)^2 + b(n)^2) < \infty,$$

$$a(n) = o(b(n)).$$

The above sequences describe the two step-size schedules or timescales in these algorithms. The first two conditions above are standard requirements in stochastic approximation algorithms. The last condition implies that $a(n)$ goes to 0 faster than $b(n)$ does. Thus the recursion that is driven by $\{a(n)\}$ is slower than the one driven by $\{b(n)\}$.

Let $\pi_i : \mathcal{R} \rightarrow [a_{i,\min}, a_{i,\max}]$ be defined by $\pi_i(x) = \max(a_{i,\min}, \min(a_{i,\max}, x))$. Then π_i projects $x \in \mathcal{R}$ to the interval $[a_{i,\min}, a_{i,\max}]$. Next define $\pi : \mathcal{R}^N \rightarrow C$ by $\pi(y) = (\pi_1(y_1), \ldots, \pi_N(y_n))^T$, for $y = (y_1, \ldots, y_N)^T \in \mathcal{R}^N$. Then π projects $y \in \mathcal{R}^N$ to the projection set C.

3.1 Two-Timescale SPSA

Suppose $\theta(n)$ and $\Delta(n)$ correspond to the nth parameter and perturbation updates, respectively, for $n \geq 1$. Here $\Delta(n)$ is obtained using the normalized Hadamard matrix based construction described above. Let $\{\zeta_n^+\}$ and $\{\zeta_n^-\}$ be independent sequences of i.i.d. random variables as described previously. Also let $\{Z^+(n)\}$ and $\{Z^-(n)\}$ be two real valued sequences (defined below) that are used for averaging the cost function.

Suppose $L \geq 1$ is a given integer and $\delta > 0$ is a fixed small constant. The algorithm is as follows: Let $Z^+(0) = Z^+(1) = \ldots = Z^+(L-1) = Z^-(0) = Z^-(1) = \ldots = Z^-(L-1) = 0$. For all $i \in \{1, \ldots, N\}$,

$$\theta_i(n+1) = \pi_i\left(\theta_i(n) - a(n)\left[\frac{Z^+(nL) - Z^-(nL)}{2\delta\Delta_i(n)}\right]\right), \tag{4}$$

Top

where for $m \in \{0, 1, \ldots, L-1\}$,

$$Z^+(nL+m+1) = Z^+(nL+m) + b(n)\left(h(\theta(n) + \delta\Delta(n), \zeta^+(nL+m)) - Z^+(nL+m)\right), \quad \text{(5a)}$$

$$Z^-(nL+m+1) = Z^-(nL+m) + b(n)\left(h(\theta(n) - \delta\Delta(n), \zeta^-(nL+m)) - Z^-(nL+m)\right). \quad \text{(5b)}$$

In [14], it was observed that this algorithm using the normalized Hadamard matrix based construction for generating perturbations shows good performance except for some oscillations, i.e., exhibits non-smooth but fast convergence. We now present the robust variant of the above algorithm that exhibits smooth convergence behavior. This is desirable for real time optimization as with the RED setting considered here. The algorithm is based on the idea that median is a more robust estimator than mean and makes use of the $sgn(\cdot)$ function defined as

$$\begin{aligned} sgn(x) &= +1 \text{ if } x > 0 \\ &= -1 \text{ otherwise.} \end{aligned}$$

The use of $sgn(\cdot)$ function for smooth convergence has also been advocated in [17], [18]. In [17], this is used in certain Kiefer-Wolfowitz type algorithms. There are other potential advantages in using the $sgn(\cdot)$ function as well. For instance, one only needs to exchange one bit of information per parameter component using $sgn(\cdot)$ function in the case of distributed network implementations involving several RED routers with each router updating its own set of parameters and then transmitting this information to the other routers. We however do not consider this scenario (of distributed network implementations) in this paper.

3.2 Robust Two-Timescale SPSA

Suppose $\Delta(n), n \geq 0$, are defined using normalized Hadamard matrices described above. Let $\{\zeta_n^+\}$, $\{\zeta_n^-\}$, L and δ be as before. Also let $\{Z_n^+\}$ and $\{Z_n^-\}$ be real valued sequences defined as under. The algorithm is now as follows: Let $Z^+(0) = Z^+(1) = \ldots = Z^+(L-1) = Z^-(0) = Z^-(1) = \ldots = Z^-(L-1) = 0$. For all $i \in \{1, \ldots, N\}$,

$$\theta_i(n+1) = \pi_i\left(\theta_i(n) - a(n)sgn\left(\frac{Z^+(nL) - Z^-(nL)}{2\delta\Delta_i(n)}\right)\right), \quad (6)$$

where for $m \in \{0, 1, \ldots, L-1\}$,

$$Z^+(nL+m+1) = Z^+(nL+m) + b(n)\left(h(\theta(n) + \delta\Delta(n), \zeta^+(nL+m)) - Z^+(nL+m)\right), \quad \text{(7a)}$$

$$Z^-(nL+m+1) = Z^-(nL+m) + b(n)\left(h(\theta(n) - \delta\Delta(n), \zeta^-(nL+m)) - Z^-(nL+m)\right). \quad \text{(7b)}$$

Remark: Note that the factor 2δ in the denominator in (6) does not play any role. In fact, one can alternatively use the following iteration in place of (6) as we do in our experiments,

$$\theta_i(n+1) = \pi_i\left(\theta_i(n) - a(n)sgn\left(\left(Z^+(nL) - Z^-(nL)\right)\Delta_i(n)\right)\right) \quad (8)$$

However, it is simpler to analyze the algorithm using (6) since the argument of $sgn(\cdot)$ function there corresponds to the standard SPSA gradient estimate.

Top

4 Sketch of Convergence

We give here a brief sketch of the convergence analysis, the details of which can be found in [11]. Consider first the recursions (7a) -(7b) and assume $\theta(n) \equiv \theta$ and $\Delta(n) \equiv \Delta$ are fixed quantities. Then one can easily show using standard martingale arguments that

$$\|Z^+(nL+m) - J(\theta(n) + \delta\Delta(n))\|, \|Z^+(nL+m) - J(\theta(n) - \delta\Delta(n))\| \to 0 \tag{9}$$

as $n \to \infty$ for any $m \in \{0, 1, \ldots, L-1\}$. Note however that since we use two timescales or step size scheduled $\{a(n)\}$ and $\{b(n)\}$ respectively with $a(n) = o(b(n)$, recursions (7a)-(7b) proceed on a faster scale than recursions (6). Thus when viewed using the timescale of (6), recursions (7a)-(7b) appear to be essentially equilibrated, while when viewed from a corresponding timescale of (7a)-(7b), recursion (6) appears to be quasi-static. Thus when viewed from (7a)-(7b), $\theta \equiv \theta(n)$ and $\Delta(n) \equiv \Delta$ (i.e., quantities independent of n) is a valid assumption and (9) holds. One can now alternatively consider in place of (6), the recursion

$$\theta_i(n+1) = \pi_i \left(\theta_i(n) - a(n) sgn \left(\frac{J(\theta(n) + \delta\Delta(n)) - J(\theta(n) - \delta\Delta(n))}{2\delta\Delta_i(n)} \right) \right), \tag{10}$$

where $i = 1, \ldots, N$. Note now that performing appropriate Taylor series expansions of $J(\theta(n)+\delta\Delta(n)$ and $J(\theta(n) - \delta\Delta(n))$, respectively, around $\theta(n)$, one obtains

$$\frac{J(\theta(n) + \delta\Delta(n)) - J(\theta(n) - \delta\Delta(n))}{2\delta\Delta_i(n)} - \nabla_i J(\theta(n))$$

$$= \sum_{j=1, j\neq 1}^{N} \frac{\Delta_j(n)}{\Delta_i(n)} \nabla_j J(\theta(n)) + o(\delta)$$

Now since $\Delta(n)$ are generated using normalized Hadamard matrices, one can show [14] that

$$\sum_{l=n}^{n+m-1} \frac{\Delta_j(l)}{\Delta_i(l)} \nabla_j J(\theta(l)) \to 0$$

as $n \to \infty$ (see [14] for details of this argument).

Now recursion (6) without $sgn(\cdot)$ function viz. 4 can be approximated by

$$\theta_i(n+1) = \pi(\theta_i(n) - a(n)\nabla_i J(\theta(n))) \tag{11}$$

which can be seen using an ODE (ordinary differential equation) based argument to converge to a point in the set $\{\theta | \tilde{\pi}_i(\nabla J(\theta)) = 0\}$ where $\tilde{\pi}_i(\cdot)$ is defined according to

$$\tilde{\pi}_i(v(x)) = \lim_{\eta \to 0} \left(\frac{\pi_i(x + \eta v(x)) - x}{\eta} \right)$$

for a bounded continuous $v(x)$. For iteration 6 that we actually work with, one argues that instead of mean, one is lead to the median of the average cost (stochastic) gradient in the recursions, which however again converge to the minimum parameter values, see [11] for details of this argument.

Top

5 Optimized RIO Algorithm

The algorithms in [15], [14] have been designed for the setting of simulation based optimization. A direct implementation of these, however, would involve simulating the entire network. Note that the latter cannot be done accurately in our setting since connections come and leave at random times and traffic characteristics change significantly with time. We thus propose to use our Robust Two-Timescale SPSA Optimization Algorithm directly over DiffServ routers. Note that the algorithm of [14] (whose variant we use here) shows significant oscillations in the initial iterations as with any stochastic approximation scheme. Robust two-timescale SPSA removes transient oscillations, shows fast convergence, and is thus highly suited for implementation in a real network scenario. This algorithm is also highly suited for distributed network implementations (though not considered here) because of the use of the $sgn(\cdot)$ function as a result of which one would effectively require only one bit of information to move between routers, thereby significantly reducing computational overheads. We describe here the implementation of Robust Two-Timescale SPSA for optimizing RIO parameters. We call this algorithm 'Optimized RIO' in this setting. Even though the algorithm is implemented directly on the router, the basic principle remains the same as with simulation based optimization.

The Optimized RIO Algorithm is triggered when the performance degrades. As stated in equations (7a) and (7b), two parallel measurements with the parameter settings $\theta(n) + \delta\Delta(n)$ and $\theta(n) - \delta\Delta(n)$, respectively, are required to average data corresponding to these settings. Since performance is measured directly over the router, the measurements of quantities $Z^+(nL)$ and $Z^-(nL)$ in (7a) -(7b) cannot be done in parallel but is done over separate L-successive disjoint iterations. Over the L epochs when $Z^+(nL)$ (resp., $Z^-(nL)$) is updated, the router parameters are set to $\theta(n)+\delta\Delta(n)$ (resp., $\theta(n)-\delta\Delta(n)$). Parameter updates using (6) are performed once both $Z^+(nL)$ and $Z^-(nL)$ have been updated. Hence if we assume that averaging in (7a)-(7b) is done at packet arrival epochs, then parameter is updated in (6) after every $2L$ packet arrivals. The algorithm is as described in Algorithm 1 below.

Optimized RIO is disabled under the normal running of the router and is triggered only when performance degrades. Since Optimized RIO is executed mainly for small time durations, the computational overhead is negligible most of the time. One only requires here that the network is quasi-stationary, i.e., the network characteristics do not change too fast. Note that no assumptions are made as such about the network topology or characteristics. Also since the complexity of the algorithm is low, the parameter updates can be calculated directly on the router instead of in the 'control plane'. This can further reduce the network overhead of the algorithm. Also (as stated before), since the algorithm is designed to update parameters directly on the router, we require a robust algorithm as developed here for good performance and smooth convergence behavior as oscillations in parameter iterates during optimization can directly affect the router performance.

Note also that the objective function in the algorithm is determined depending on the priorities of the network administrator and it can be changed on-line if required, with minimal effort. In this paper, we consider experiments that minimize the oscillations of instantaneous queue sizes. In order to reduce oscillations of these, we consider in our experiments the objective function to be the mean squared difference of instantaneous queue length from a given threshold q_0. The cost incurred at the ith observation is thus $(q_i - q_0)$, where q_i is the instantaneous queue length as seen by the ith arriving packet.

Since minimizing the mean squared difference of instantaneous queue size with respect to a given threshold also reduces the oscillations in instantaneous queue size, this leads to more predictable queu-

Top

ing delays in the router and allows the network administrator to guarantee delay bounds on connections irrespective of the load. We now present the simulation results.

6 Simulation Results

Extensive simulation experiments were carried out to validate the effectiveness of our approach. We show in this section, results of simulation experiments for different network conditions. All the experiments show a significant improvement in performance over the traditional RIO mechanism. The Optimized RIO congestion avoidance algorithm was implemented in network simulator NS2 [13] by changing the network code for DiffServ routers and experiments were carried out. The network topology is as shown in Fig. 2.

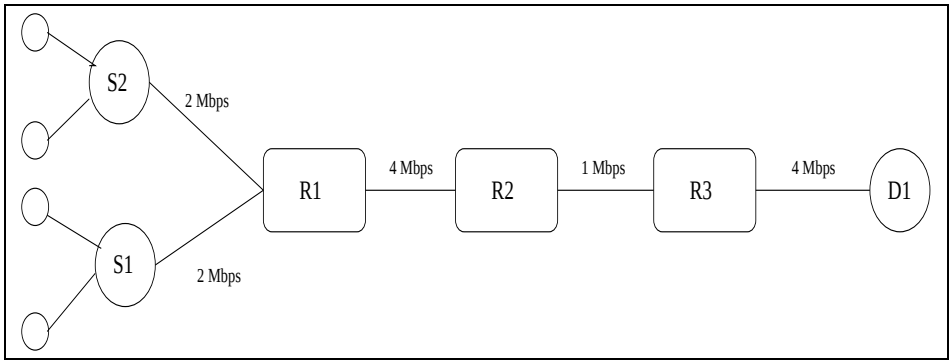


Figure 2: Network Topology for Simulation

FTP sources with infinite data are considered to be operating over TCP. These are equally distributed over nodes S1 and S2. A non-cooperative UDP source is connected to router S1. Destination of all the connections is D1. The link connecting routers R2 and R3 is the bottleneck link in the network. RIO is configured over R2 to manage a buffer of 100 packets. The RIO parameters are initialized to the recommended settings, i.e. max_{th} = 60, min_{th} = 20, w_q = 0.001 and max_p = 0.01 respectively for IN packets and max_{th} = 30, min_{th} = 10, w_q = 0.005 and max_p = 0.05 respectively for OUT packets. For the step size sequences, we chose $a(n) = a(0)/n$ and $b(n) = 1/n^{0.66}$, respectively for IN and OUT packets. The value of $a(0)$ was chosen to be 0.005 for w_q, 0.1 for max_p and 5 for max_{th} and min_{th}, respectively for IN and OUT packets. The values of the parameter δ for various parameter components were taken to be corresponding values of $a(0)$ for these. This was done for better convergence behavior. One can easily show that the analysis of convergence carries through with minor changes for different fixed values of $a(0)$ and δ for different parameter components. In particular, a different value of $a(0)$ for a given parameter component merely has the effect of changing the speed of convergence of that component to the optimum value.

In the first experiment, a number of TCP sources are connected to routers S1 and S2 (five each in our case). One non-cooperating source is connected to S1, occupying 20% of available bandwidth. Router R1 is the edge router and has SLA with the core router. Various experiments were carried out for different levels of SLA varying from 25% of bandwidth between R2 and R3 to 125% of the bandwidth being reserved by the sources. The objective for optimization was to keep the variation of instantaneous queue size from 60 packets to a minimum. The amount of throughput achieved by

Optimized RIO was consistently higher than the throughput achieved by the standard RIO algorithm. The comparison result is shown in Figure 3.

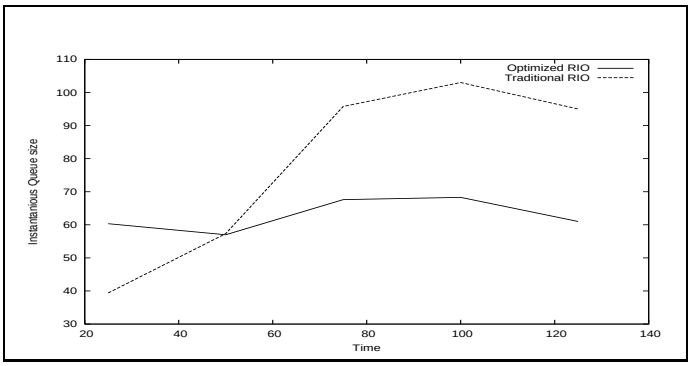


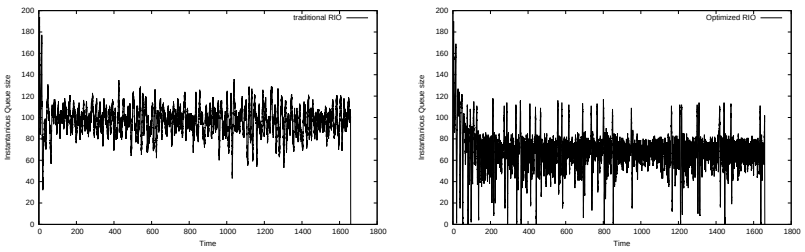
Figure 3: Average queuing delay

In Figure 4, we show the plot of instantaneous queue size with both RIO and Optimized RIO algorithms. We can clearly see that the optimization objective for optimized RIO for optimized RIO is achieved. Note that the mean of the instantaneous queue size has decreased from 100 with traditional RIO to near 60 in Optimized RIO.

In the second experiment, we considered only one source with no priorities. There was no higher priority traffic. This simulates the case of absence of priority flows. In the simulations we observed an increase of 18% throughput in the optimized RIO algorithm from 63 to 81% over traditional RIO.

7 Conclusion

For optimal performance, the parameters of RIO need to be dynamically tuned rather than set *a priori*, as was proposed previously. In this paper, we adapt a previously developed robust two-timescale simultaneous perturbation stochastic approximation algorithm for optimizing RIO parameters. This algorithm uses two different timescales and updates all parameter components simultaneously once every fixed number of packet arrivals. The algorithm uses a convolution of the $sgn(\cdot)$ function and



(a) Instantaneous Increase in Load for Traditional RIO

(b) Instantaneous Increase in Load for Optimized RIO

Figure 4: Performance Comparison of Optimized RIO with Traditional RIO

Top

the gradient estimate in the update step resulting in less oscillations. We gave a sketch of the convergence analysis of the algorithm and presented numerical results. Specifically, we showed performance comparisons using our algorithm with traditional RIO. We observed that our algorithm shows superior performance as compared to traditional RIO as it significantly reduces the instantaneous queue sizes and brings these near specified targets within a small time interval.Our algorithm reduces the variation in queue size as well. We proposed the use of our algorithm directly over DiffServ routers. The advantage of doing so is that unlike other approaches, one does not have to simulate the network at all. This can take care of changing network characteristics and/or dynamics as well, except that with any such change, the algorithm has to be reinitialized and rerun to obtain a new set of optimum parameter values.

The proposed algorithm is also adaptable to the changing needs of the network. The objective function can be redefined whenever required to obtain a different corresponding set of optimum parameter values. Moreover, there are no restrictions on the form of objective function as long as it is a measure of router performance. Also, we do not make any assumptions about the underlying protocol. Hence this approach can easily be extended to other protocols and AQM mechanisms as well. Further, since this is a model free approach, the proposed algorithm can be applied to protocols that are sensitive to network conditions without requiring that the interaction between parameters and protocols be known *a priori*. The proposed approach can be applied in various scenarios and need not be restricted to only the TCP protocol. Because of the direct measurement of the loss function on the router, our approach does not rely on sensitivity of the performance of the algorithms to network topology. Hence this approach can be applied to different protocols irrespective of their sensitivity to the parameters/network conditions.

Bibliography

[1] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[2] W. Feng, D. Kandlur, D. Saha, and K. G. Shin, "A self-configuring RED gateway," in *Proceedings of INFOCOM*, 1999, vol. 3, pp. 1320–1328.

[3] S. De Cnodder, O. Elloumi, and K. Pauwels, "RED behavior with different packet sizes," *Proceedings of Fifth IEEE Symposium on Computers and Communications*, pp. 792–799, 2000.

[4] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED," Technical report, The ICSI Center for Internet Research Berkeley, California, 2001.

[5] D. D. Clark and W. Fang, "Explicit allocation of best-effort packet delivery service," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362–373, 1998.

[6] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding PHB group," *Internet Standards Track RFC 2597, IETF*, 1999.

[7] N. Malouch and Z. Liu, "Performance analysis of TCP with RIO routers," in *GLOBECOM, Taiwan*, November 2002.

Top

Algorithm 1 Optimized RIO

for each packet arrival **do**
  Select the Parameters depending on priority in packet header
  Calculate the average queue size
  if $min_{th} \leq average \leq max_{th}$ **then**
    calculate probability p_a
    With probability p_a mark the packet
  else
    if $average \geq max_{th}$ **then**
      mark the arriving packet
    end if
  end if
  $m = m + 1$
  if $m < L$ **then**
    Parameter: $\theta(n) + \delta\Delta(n)$
    $Z^+(nL + m + 1) = Z^+(nL + m) + b(n)\,(qlen - Z^+(nL + m))$
  else
    if $L \leq m < 2L$ **then**
      Parameter: $\theta(n) - \delta\Delta(n)$
      $Z^-(nL + (m - L) + 1) = Z^-(nL + (m - L)) + b(n)\,(qlen - Z^-(nL + (m - L)))$
    end if
  end if
end for
for every $(2L)$th packet arrival **do**
  $\theta_i(n + 1) = \theta_i(n) - a(n)sgn\left[\frac{Z^+(nL) - Z^-(nL)}{2\delta\Delta_i(n)}\right]$
  if $\theta_i(n + 1) < a_{i,min}$ **then**
    $\theta_i(n + 1) = a_{i,min}$
  else
    if $\theta_i(n + 1) > a_{i,max}$ **then**
      $\theta_i(n + 1) = a_{i,max}$
    end if
  end if
  $m := 0$
  $n := n + 1$
  Generate new $\Delta(n) = (\Delta_1(n), \cdots, \Delta(n))^T$ using normalized Hadamard matrix.
end for
Variables:
average: average queue size
qlen: instantaneous queue size
$Z^+(l)$ and $Z^-(l)$: as given by Equation 7a, 7b
$\theta_i(n) : \theta_i(n)$ as given by Equation 6

Top

[8] G. Neglia, G. Bianchi, and M. Sottile, "Performance evaluation of a new adaptive packet marking scheme for TCP over DiffServ networks," in *GLOBECOM, San Fransisco*, December 2003, pp. 3652–3657.

[9] J. Orozco and D. Ros., "An adaptive RIO (A-RIO) queue management algorithm," in *Lecture Notes in Computer Science, Springer*, 2003.

[10] CH. Chien and W. Liao, "A self-configuring RED gateway for quality of service (QoS) networks," in *IEEE ICME*, July 2003.

[11] R. Vaidya and S. Bhatnagar, "Robust optimization of random early detection," http://keshava.csa.iisc.ernet.in/techreports/2004/ored.pdf, Indian Institute of Science, Bangalore, 2004.

[12] T. Ye and S. Kalyanaraman, "Adaptive tuning of RED using on-line simulation," *GLOBECOM*, vol. 3, pp. 2210–2214, 2002.

[13] NS, "Network Simulator," *http://www-mash.cs.berkley.edu/ns*.

[14] S. Bhatnagar, M. C. Fu, S. I. Marcus, and I. J. Wang, "Two timescale simultaneous perturbation stochastic approximation using deterministic perturbation sequences," *ACM Transactions on Modelling and Computer Simulation*, vol. 13, no. 4, pp. 180–209, 2003.

[15] S. Bhatnagar, M. C. Fu, S. I. Marcus, and S. Bhatnagar, "Two timescale algorithms for simulation optimzation of hidden Markov models," *IIE Transactions*, vol. 33, no. 3, pp. 245–258, 2001.

[16] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, pp. 332–341, 1992.

[17] B. Bharath and V. S. Borkar, "Robust parameter optimization of hidden Markov models," *Journal of Indian Institute of Science*, vol. 78, pp. 119–130, 1998.

[18] R. Y. Rubinstein, *Monte Carlo Optimization, Simulation and Sensitivity of Queueing Networks*, Wiley, 1987.

Top